VAX DATATRIEVE

Guide to Interfaces

Order Number: AA–PNY4A–TE

May 1992

This manual explains how to use the following interfaces with VAX DATATRIEVE.

- DECwindows
- Forms products: TDMS, FMS, DECforms
- VAX DBMS
- Rdb/VMS, Rdb/ELN, VIDA

Operating System:	VMS Version 5.4 or higher
Software Version:	VAX DATATRIEVE Version 6.0

This document was prepared using VAX DOCUMENT, Version 2.0.

Contents

3 Using DECforms with DATATRIEVE

4 Using DATATRIEVE with VAX DBMS

5 Using DATATRIEVE with Rdb/VMS, Rdb/ELN, or VIDA

6 Accessing Remote Data

Index

Figures

Tables

Preface

This manual describes how to use the following interfaces with VAX DATATRIEVE.

- DECwindows
- Forms products: TDMS, FMS, DECforms
- VAX DBMS
- Rdb/VMS, Rdb/ELN, VIDA

Intended Audience

This manual is intended for VAX DATATRIEVE users who know the basic concepts of data processing and are familiar with the VMS operating system.

Operating System Information

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

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

Related Documents

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

- *VAX DATATRIEVE User's Guide*

  Explains the concepts and terminology of DATATRIEVE.

- *VAX DATATRIEVE Reference Manual*

Contains reference information for DATATRIEVE.

References to Products

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

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

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

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

1

Using DATATRIEVE with DECwindows

This chapter explains how to use DATATRIEVE on a workstation that is running DECwindows.

Using DATATRIEVE with DECwindows provides you with an easy way to perform the following operations:

- Scroll through the commands, statements, and output of your DATATRIEVE session
- Execute command procedures
- Open and close log sessions
- Navigate through dictionaries
- Set column width and other DATATRIEVE setup parameters
- Display information on certain DATATRIEVE sources

This chapter covers the following topics:

- Preparing to use DATATRIEVE in a DECwindows environment
- Invoking DATATRIEVE in a DECwindows environment

Refer to DECwindows help for other DECwindows related material, or to the *VAX DATATRIEVE User's Guide* for general DATATRIEVE information.

While the DECwindows software will help guide you through some aspects of your DATATRIEVE session, you will find that many DATATRIEVE commands and statements must still be entered on the command line at the bottom of the main application window. If you are more comfortable using DATATRIEVE exclusively as a command line interface, the DTR> prompt at the bottom of the application window allows you to enter all DATATRIEVE commands and statements, as you have in the past.

1.1 Preparing to Use DATATRIEVE in a DECwindows Environment

Before you can begin using DATATRIEVE in a DECwindows environment, you must be certain that DECwindows and DATATRIEVE logicals have been set appropriately. You may also have to update the information under the Security option of the Customize menu in your Session Manager window.

The DECwindows logical DECW$DISPLAY must be defined when you invoke DATATRIEVE. To see how the logical has been defined, use the DCL command SHOW LOGICAL, as in the following example:

```
$ SHOW LOGICAL DECW$DISPLAY
   "DECW$DISPLAY" = "WSA1:"
```

The command takes the following form:

DEFINE DECW$DISPLAY target-node::0

In this syntax, target-node is the node name of the workstation that will display the DECwindows screen. It can be the same node as the one from which DATATRIEVE is invoked. You can also define the DECW$DISPLAY logical to point to the device specification of your workstation.

Even if the DECW$DISPLAY logical is correctly defined, DATATRIEVE does not invoke the DECwindows interface if the DTR$NOWINDOWS logical is set to true. By default, the logical DTR$NOWINDOWS has no value assigned to it; therefore, the DECwindows interface is automatically invoked if you are invoking DATATRIEVE from a workstation that is also running DECwindows.

To determine the value of DTR$NOWINDOWS, use the DCL command SHOW LOGICAL, as in the following example:

```
$ SHOW LOGICAL DTR$NOWINDOWS
%SHOW-S-NOTRAN, no translation for logical name DTR$NOWINDOWS
```

If the logical has been set, it must be deassigned:

```
$ DEASSIGN DTR$NOWINDOWS
```

When the DATATRIEVE DECwindows interface is invoked, DATATRIEVE looks for the file VAX_DATATRIEVE .UID. This file is supplied during installation and is found in SYS$COMMON:[SYSLIB].

1.2 Invoking DATATRIEVE in a DECwindows Environment

Invoking DATATRIEVE from DCL command level can be done in several ways. You can use the DCL command DATATRIEVE. This command lets you specify several optional qualifiers that let you customize your DATATRIEVE session. For example, you can use the DATATRIEVE command with the /INTERFACE= qualifier to specify whether you want to run DATATRIEVE as a command line interface or as a DECwindows interface. Use the /VARIANT= qualifier to specify which image of DATATRIEVE you want to run if your system has more than one image of DATATRIEVE installed. Use the /[NO]DEBUG qualifier to specify whether DATATRIEVE should display special debug messages.

You can also use the DCL command RUN SYS$SYSTEM:DTR32xx. The suffix xx is sometimes necessary to identify the image of DATATRIEVE that you want to run. You may have defined a symbol for this command in your LOGIN.COM file. You can use that symbol to invoke DATATRIEVE, as well. See the *VAX DATATRIEVE Reference Manual* for detailed information on the DATATRIEVE command and its qualifiers.

You can customize your FileView window to add DATATRIEVE as a menu item to an existing menu or to create a special menu just for DATATRIEVE. For more information on customizing your FileView window, see the DECwindows documentation.

When you invoke DATATRIEVE in a DECwindows environment, DATATRIEVE spawns a separate main application window. The main application window is described in the DECwindows help.

If you invoke DATATRIEVE and receive a message stating that the client is not authorized to access the server, check to be sure that the Security item of the Customize menu of your Session Manager window recognizes the node on which you are running your DATATRIEVE session and your username.

1.2.1 Restriction on Display of Tab Characters in DECwindows Environment

DATATRIEVE does not support tabbing logic in a DECwindows environment. This means that output displayed in one of the scrollable windows created by DATATRIEVE in a DECwindows environment does not reflect the appropriate number of characters specified by a tab character. Instead, DATATRIEVE converts each tab character to one ASCII space for online display purposes.

For example, if you enter a SHOW command for a procedure that contains a tab character to separate a command line from a comment, the resulting display shows only one space separating the command from the comment text.

The tabs remain in the actual text of the procedure. If you use the EXTRACT command to copy the procedure to a file, tab characters are correctly translated. This restriction affects only the online display of tab characters in a DECwindows environment. The restriction does not apply to tab characters displayed in DATATRIEVE when it is used as a command-line interface.

2

Using FMS and TDMS Forms with DATATRIEVE

A form is a terminal screen image used to display and collect information. You can use forms to display, modify, and store data managed by DATATRIEVE.

You can often format a data display more attractively using a form image. This is particularly true when you need to display records longer than the maximum number of characters your terminal screen can accommodate on one line.

DATATRIEVE allows you to use the following forms interfaces:

- VAX FMS
- VAX TDMS
- DECforms

DECforms is always available, while FMS and TDMS are optional and mutually exclusive, and are selected by the user at installation time.

This chapter deals with FMS and TDMS forms. For more information on DECforms, see Chapter 3.

To create a forms application, you need to take the following actions:

- Use the proper form editor to define a form
- Insert the form definition in a request library file (TDMS) or form library (FMS)
- Associate the form definition with a DATATRIEVE domain
- Make sure that the DATATRIEVE SET FORM command is in effect when your application executes

The following sections discuss each of these steps in greater detail. Because your form definition depends, at least partly, on how you plan to use the form, the following discussion begins with associating the form definition with a DATATRIEVE domain.

2.1 Associating a Form with a Domain

You can use a form to display data from RMS domains, view domains, VAX DBMS domains, relational domains, and remote domains.

When working with remote domains, you can use forms to store data in a remote domain and to display a selected record or a record from a record stream containing no other records. You cannot, however, use forms to display group fields from remote domains.

You can associate a form definition with a domain in two ways:

- Use the FORM IS clause within a domain definition to identify a form with a particular domain. DATATRIEVE uses that form with any STORE, MODIFY, DISPLAY, or PRINT statement that refers to that domain.
- Use the DISPLAY_FORM statement within a DATATRIEVE statement to map data to and from specific form and record fields. For example, include the DISPLAY_FORM statement within a FOR, STORE, or MODIFY statement.

When using either method, you must specify both the name of the form and the file specification of the library file containing the form. There are advantages to using each method. The FORM IS clause lets you use a form by specifying a single line of syntax in a domain definition. The DISPLAY_FORM statement lets you specify exactly which fields you want to map between a form and a record and lets you associate more than a single form with a domain.

Note

Use the DISPLAY_FORM statement if you intend to map numeric data between forms and records. Using the DISPLAY_FORM statement with the FORMAT value expression ensures that decimal points and signs are mapped correctly. FORMAT value expressions are discussed in the *VAX DATATRIEVE Reference Manual*.

You can see examples of what form displays look like with some of the sample DATATRIEVE domains. Note, however, that you must first make sure your system has a version of DATATRIEVE installed with the TDMS or the FMS forms interface. To see the examples of form displays, follow these steps:

1. Set your default VMS directory to one that contains the data files for the YACHTS, SAILBOATS, and FAMILIES domains. If you do not have these data files in one of your directories, copy them from the system directory DTR$LIBRARY:

2. Invoke DATATRIEVE and set the dictionary to the sample forms dictionary:

```
DTR> SET DICTIONARY CDD$TOP.DTR$LIB.FORMS
```

3. Make sure that the form setting is in effect for your session. (SET FORM is the DATATRIEVE default, but you may have run procedures that include the SET NO FORM command.)

```
DTR> SET FORM
```

4. Ready the YACHTS, SAILBOATS, or FAMILIES domain.
5. Print some records from the domain. Records appear one at a time. Press the RETURN key each time you want to display a new record. The DTR> prompt returns once all the records you requested have been displayed.
6. Press CTRL/C and then the RETURN key if you want to stop the display operation before all the records have been displayed.

2.1.1 The FORM IS Clause

If you include the FORM IS clause in a domain definition, you can have DATATRIEVE automatically use the form to display records.

The following examples show the use of the FORM IS clause in domain definitions. Note that file-name can be a TDMS request library file or an FMS forms library. The default file type for TDMS request library files is .RLB; the default file type for FMS form libraries is .FLB:

```
DEFINE DOMAIN YACHTS
 USING CDD$TOP.DTR$LIB.DEMO.YACHT ON YACHT.DAT
 FORM IS YACHT IN DTR$LIBRARY:FORMS;

DEFINE DOMAIN PERSONNEL_F
 USING PERSONNEL_REC ON [MORRISON]PERSON.DAT
 FORM IS PERSON IN [KELLER]FORMSLIB;

DEFINE DOMAIN REMOTE_FAMILIES USING FAMILIES AT NOVA"LINTER TAD"
 FORM IS FAM IN NOVA"LINTER TAD"::DB3:[LINTER]DTRTDMS;

DEFINE DOMAIN SAILBOATS
 OF CDD$TOP.DTR$LIB.DEMO.YACHTS_SEQUENTIAL, CDD$TOP.DTR$LIB.DEMO.OWNERS BY
 01 SAILBOAT OCCURS FOR YACHTS_SEQUENTIAL.
   03 BOAT FROM YACHTS_SEQUENTIAL.
   03 SKIPPERS OCCURS FOR OWNERS WITH BUILDER EQ BOAT.BUILDER.
      05 NAME FROM OWNERS.
      FORM IS SAIL IN DTR$LIBRARY:FORMS
;

DEFINE DOMAIN PART USING PART OF DATABASE PARTS_DB
 FORM IS PARTF IN PARTS.FLB;
```

There are some important points to note when using the FORM IS clause:

- The field names in the form definition must exactly match the corresponding field names in the record definition for the domain.
- You must define a form field for every field in the corresponding record to avoid unexpected mapping results.
- You cannot match form field names to names of REDEFINES fields or to query names.
- You may get unexpected results when mapping numeric fields.
- You cannot prevent a user from entering data in all the fields on the form when you specify the MODIFY or STORE statements, unless the fields were defined as Display Only by the form definer.
- When a user presses the RETURN key or ENTER key after a STORE or MODIFY operation, data is returned from all the fields on a form, including data that may have been previously mapped to a Display Only field. This may lead to unexpected input.
- You cannot use more than the single form that you specify in the domain definition to store or modify data associated with that domain.
- You cannot ready a domain if it contains a FORM IS clause and DATATRIEVE has not been installed with a forms package.
- You may have to take special steps to use the FORM IS clause with applications that use FMS and callable DATATRIEVE. See Section 2.6.1 for more information.
- You must exercise caution when your application modifies view domains that use the FORM IS clause. See Section 2.4.4 for more information on modifying data using view domains and FORM IS.

For more information on the FORM IS clause, see the DEFINE DOMAIN command in the *VAX DATATRIEVE Reference Manual*.

2.1.2 The DISPLAY_FORM Statement

The DISPLAY_FORM statement gives you four kinds of control not provided by the FORM IS clause of the domain definition:

- You can associate more than one form definition with a domain. Therefore, you can design a variety of forms to fit different purposes.
- You specify the record field values you want to map to or display on the form.

Only values from record fields you explicitly assign to form fields are displayed on the form. This allows you to mask data from different types of user. The USING clause gives you this control.

If you omit the USING clause of the DISPLAY_FORM statement, no record field values are displayed on their corresponding form fields. You may omit the USING clause if you are storing records rather than modifying records.

- You specify the form fields you want to return to the record.

  Only values from form fields explicitly assigned to record fields are returned to the record. The RETRIEVE clause gives you this control.

  If you omit the RETRIEVE clause, no form field values entered by the user or sent to the form by DATATRIEVE are returned to their corresponding record fields. Omit the RETRIEVE clause if you do not want to collect data from the form and you want to prevent inadvertent data modification during a display-only operation.

- You specify the format to display, store, and modify numeric fields correctly.

  Using the FORMAT value expression lets you control data transfer between numeric record fields and form fields. (See Section 2.2.2.1 for information on mapping numeric data types and Section 2.4.5 for information on using the FORMAT value expression.)

The only time you can use the DISPLAY_FORM statement without embedding it in another statement and without using the optional USING and RETRIEVE clauses is when you simply want to display the form itself. In this case, to avoid getting an error message, end the statement with a semicolon (;). For example:

```
DTR> DISPLAY_FORM YACHT IN DTR$LIBRARY:FORMS;
```

This type of DISPLAY_FORM statement lets you see the form before you use it without exiting from DATATRIEVE. You cannot enter data in the form that is displayed. You can use this particular DISPLAY_FORM format to display a form that is associated with a domain by the FORM IS clause. For more information on the DISPLAY_FORM statement, see the *VAX DATATRIEVE Reference Manual*.

2.2 Defining Forms

A form definition contains the following information:

- The screen image of the form. The screen image includes background text and the pictures of the fields in which data can be collected and displayed.
- The length and data type of each field.
- A set of attributes for each field.
- Video highlights for the form.

- The name of a help form that the user can display.

You can use the TDMS form editor or the FMS form editor to define a form. See the VAX TDMS documentation for more information on using the Form Definition Utility (FDU) to create and modify a TDMS form. See the VAX FMS documentation for information on creating and modifying an FMS form.

The sections that follow discuss some considerations you should keep in mind when creating a form to use with DATATRIEVE. The discussion assumes you are familiar with the process of creating a form definition. If you are unfamiliar with the forms product you intend to use with DATATRIEVE, you should create a few sample forms before reading these sections.

2.2.1 Defining Form Field Names

You specify form field names in the Assign phase of your form definition. Any names you specify during the Layout phase of your form definition are background text, not field names.

If the form you are creating will be referred to in the FORM IS clause of a domain definition, use the following guidelines when naming form fields:

- If the DATATRIEVE field name contains hyphens, they should be defined as underscores to FMS because DATATRIEVE converts all hyphens to underscores when it converts names to uppercase.
- Form field names must match the field names in the DATATRIEVE domain. Form field names can be up to 31 characters long.

  If you convert an FMS Version 1 form definition to FMS Version 2 or TDMS and you want to associate the converted form with a DATATRIEVE domain, make sure that the form field names match the record field names. Suppose, for example, you have an FMS Version 1 form with the form field MANUFA corresponding to the record field MANUFACTURER. If you convert that form to FMS Version 2 or TDMS, be sure to edit your new form definition and change the name of the form field to MANUFACTURER.
- Make sure that you define a form field for each record field. When the record field is a COMPUTED BY field, define a counterpart form field and select the Display Only attribute. The Display Only attribute prevents users from modifying the field's data.

  In FMS, if you do not define a form field for each record field, incorrect values may be mapped to record fields.

If the form you are creating will be referred to only in DISPLAY_FORM statements, the DATATRIEVE field names and form field names need not match. In addition, you do not have to define a form field for each record field.

When you are using the DISPLAY_FORM statement, you can define a form field for a DATATRIEVE variable. You may want to define a form field that maps to a variable to use in conditional statements. For example, you could collect an employee ID from a form field and return it to a variable. You could then use this ID to search a domain and display related employee data on the same form. Remember, however, that DATATRIEVE does not process form field values when the form user presses TAB to move to another field. The user must press the RETURN key before DATATRIEVE can evaluate and process data. In this example, the user would have to enter the employee identification code, ignore the remaining fields on the form, and press RETURN.

If you do not spell a record field name correctly in DISPLAY_FORM assignment statements or do not spell a form field name correctly either in the form definition referred to by the FORM IS clause or in the DISPLAY_FORM assignment statements, DATATRIEVE ignores the values in those fields. You receive no error messages to alert you to possible field name errors.

2.2.2 Defining Data Type and Length of Form Fields

When you define a form, match the form fields with the corresponding record fields in both of the following ways:

- Use a form field picture that describes the same data type as the record definition. The field pictures control the type of data a user enters in the form field at run time.

  For example, if you define a form field of all 9s, a user cannot move to the next form field if he enters anything but a digit in that form field.

  DATATRIEVE receives data from both TDMS and FMS as strings, however, so you cannot completely match form and record definition data types. For example, if your record definition defines a field as PIC 9(4) USAGE COMP, when you define a form field you can specify the form field characters 9999 but you cannot match COMP.

- Make sure your form field has the same length as your record field.

  For example, if the record field is defined as PIC X(25), use 25 Xs for the form field. If the form field is longer than the record field, the form user might receive a truncation error message and reenter prompt after pressing the RETURN key. If the form field is shorter than the record field, existing values for alphanumeric fields are truncated in the form display. A form field shorter than the record field might make it impossible for the form user to enter a valid value.

Note that regardless of the form pictures or record data types you assign, DATATRIEVE passes all values between forms and records as text strings. This affects how it handles numeric field values, as discussed in the following sections on data types requiring special treatment in form definitions.

2.2.2.1 Numeric Fields with Decimal Points or Signs

When matching numeric record fields that contain signs or decimal points with form field pictures, you may want to follow some of the guidelines in this section. These guidelines are based on the DISPLAY_FORM statement with the FORMAT value expression. This is the recommended method of handling numeric data types. (See Section 2.4.5 on handling numeric data for examples of the FORMAT value expression.)

- For numeric record fields that contain an implied decimal point (such as a V in the PICTURE clause or a SCALE clause) but not a sign, you can include a decimal field-marker character (FMS) or a decimal constant (TDMS) in the corresponding place on the form field.

  You do this during the Layout phase of defining your form.

- In the Assign phase of the form definition, you might decide to avoid the fixed decimal characteristic. Because values are passed from the form to DATATRIEVE as text strings, specifying the Fixed Decimal attribute does not scale numeric values for storage.

- If you assign the Fixed Decimal attribute to a numeric form field because the form might be used in applications other than DATATRIEVE, then:

  – Define the associated record field with an implied decimal field, for example, PIC 99V99.

  – Define the form field as 99.99 with Fixed Decimal, clear character 0 (if it is an FMS form), and Zero Fill attributes.

- If you want a field to have both a decimal point and a sign (either explicit or implicit), in FMS use N instead of 9 for the form field.

  In TDMS, you cannot combine a signed numeric field picture (N) with the Fixed Decimal attribute. To get the effect of a scaled signed number, you can use the N field without the Fixed Decimal attribute in TDMS. TDMS assigns a scale factor to fields you describe with an N picture based on the field picture. For a field you describe as NNN.NNN in the Layout phase of TDMS, the default uses a scale factor of –3, while a field you describe as NN.NN is assigned a scale factor of –2.

Table 2–1 provides examples that match form fields to numeric record fields when using the DISPLAY_FORM statement. Use the FORMAT value expression as illustrated in the example following Table 2–1 and in Section 2.4.5 on handling numeric data to ensure correct mapping between numeric fields.

Table 2–1 Matching Form Field Definitions to Numeric Record Fields

Record Field	Form Field	Form Field Attributes
PIC 999V99	999.99	You can use 9 or N because the data is unsigned.
PIC S9(4)V99	NNNNN.NN	An N is required for a signed field. The decimal point must agree with the record field.
WORD SCALE –3	NNN.NNN	The field must be large enough for the maximum value, sign, and decimal places.
REAL	NNNNNNNNN	The number of Ns is flexible, but no decimal point is included because the record field contains no implied decimal point.
LONG SCALE –4	NNNNNNN.NNNN	Similar to the preceding WORD example.

When you use the DISPLAY_FORM statement to map numeric fields, follow these guidelines:

- When mapping fields to a form, use the FORMAT VALUE expression with an edit string and multiply the data from the record field (as described by the edit string) by the scaling factor.
- When getting the fields from a form, divide the data from the record field by the scaling factor before storing the data.

The following procedure illustrates a typical method for mapping numeric fields using a form.

```
DEFINE PROCEDURE FORMAT
READY TEST_FORMAT WRITE
BEGIN
FOR TEST_FORMAT
BEGIN
 DISPLAY_FORM TEST_FORMAT_FORM IN
 [LINTOV]TESTFORM.RLB USING
  BEGIN
      PUT_FORM UNSIGNED_FIXED_DEC = -
         FORMAT(100 * UNSIGNED_FIXED_DEC) USING 99999
      PUT_FORM SIGNED_DECIMAL = -
         FORMAT(100 * SIGNED_DECIMAL) USING S999999
      PUT_FORM SCALE_FACTOR_3 = -
         FORMAT(1000 * SCALE_FACTOR_3) USING S99999
      PUT_FORM PLAIN_N = PLAIN_N
      PUT_FORM SCALE_FACTOR_4 = -
         FORMAT(10000 * SCALE_FACTOR_4) USING S99999
  END RETRIEVE USING

  BEGIN
      UNSIGNED_FIXED_DEC = -
         (GET_FORM UNSIGNED_FIXED_DEC/100)
      SIGNED_DECIMAL = -
         (GET_FORM SIGNED_DECIMAL/100)
      SCALE_FACTOR_3 = -
         (GET_FORM SCALE_FACTOR_3/1000)
      PLAIN_N = PLAIN_N
      SCALE_FACTOR_4 = -
         (GET_FORM SCALE_FACTOR_4/1000)
  END
END
END-PROCEDURE
```

2.2.2.2 Usage DATE Fields

You should define date fields as 11 Xs on your forms. DATATRIEVE displays the field in the default date format (DD-MMM-YYYY). Use 23 Xs to display both date and time.

If you want to display the date in other than the default format, you can do so with the DISPLAY_FORM statement. Define a variable that is computed by a format value for the date field and then display the variable rather than the date field on the form. In the following example, FIELD2, rather than FIELD1, can be mapped to the form:

```
Record field:

03  FIELD1 USAGE DATE.
```

Variable:

```
03  FIELD2 COMPUTED BY FORMAT(FIELD1) USING MMMBDDBYYYY.
```

You might want to add a COMPUTED BY date field such as FIELD2 to the record definition if your installation uses the specified date format as the company standard.

When the date field is defined as 11 Xs on your form, the form user can enter one of a variety of values and DATATRIEVE stores the date correctly. For example, DATATRIEVE stores the date value April 16, 1972 for any of these entries:

```
4/16/72

4 16 1972

APR 16 1972
```

You can also use both the PUT_FORM and GET_FORM components of the DISPLAY_FORM statement to map date fields. (See Section 2.4.3 and Section 2.4.4 on storing and modifying data with forms in this chapter.)

2.2.3 Specifying User Entry and Validation Criteria

DATATRIEVE validation clauses (in the record definition or DATATRIEVE statement) are not applied to any field until the form user finishes with a record and presses the RETURN key. After the RETURN key is pressed, DATATRIEVE looks at the data returned to the record fields and applies the record or statement validation clauses. If data in a field is invalid, the user is prompted again for valid data for that field.

Validation associated with the form fields can prevent users from entering incorrect data. Using 9s or Ns for numeric form fields, for instance, ensures that form users cannot tab to the next form field if they accidentally enter nonnumeric characters.

Even though the Fixed Decimal attribute is not passed to DATATRIEVE when field values return from the form, you can specify the attribute in your form definition if you decide it helps the user enter data correctly.

Choosing the Fixed Decimal, Right Justify, Left Justify, and Zero Fill attributes all affect how the user enters data and therefore can affect what data is returned to the record field.

Assign the Display Only attribute to any fields that the user cannot store into or modify. This will prevent the user from entering data in a field. Your DISPLAY_ FORM statement can prevent user modifications from reaching the stored record, but cannot prevent the user from entering that data in the form field. It is also good practice to prevent form users from entering changes that are not stored.

Note

TDMS allows users to assign field validators to form fields. DATATRIEVE ignores these validators, however, and does not use them in any DATATRIEVE application that uses TDMS.

2.2.4 Defining Multiple Screen Forms and Forms with Scrolled Areas

Neither FMS nor TDMS supports forms that span multiple screens. DATATRIEVE does not support forms that contain scrolled regions.

If you modify and store data using the DISPLAY_FORM statement, however, you can display a series of individual forms to collect and display data within a single procedure. Note that only one form can appear on the screen at a time.

2.2.5 Using Default Values

The default value and missing value for DATATRIEVE are displayed on the form fields when you use the FORM IS clause or the PUT_FORM component of the DISPLAY_FORM statement. The form user generally sees default and missing values only during a store operation. Do not specify default values in your form definition. DATATRIEVE does not store these values in the record.

2.2.6 Defining Forms for Domains That Contain Repeating Fields

You can define a matching form field for a repeating record field (one that includes an OCCURS clause). For example, a form for the FAMILIES domain could specify KID_NAME and AGE as indexed fields. You create an indexed element to match each occurrence of the record field.

If you are using TDMS, align all occurrences of the repeating fields vertically or horizontally in the Layout phase of your form definition. Then, in the Assign phase, specify the index attribute for the repeating items. If you are using FMS, follow these steps:

1. In the Layout phase of your form definition, specify the background text and field characters for the first occurrence of the repeating item.
2. Enter the Assign phase, and specify the attributes you want for the item. Type 1 for the index attribute.
3. Reenter the Layout phase and use the form editor's cut and paste function to create all additional occurrences of the repeating item. FMS automatically assigns the correct attributes for these additional occurrences.

Whether you are using TDMS or FMS, make sure you create enough repeating form fields to accommodate the maximum number of occurrences defined in the record definition.

2.3 Inserting Forms in Library Files

After you define a form with the TDMS or FMS editor, you must insert the form in a form library. For more information on this subject and on how to create and modify form libraries, refer to the VAX TDMS and VAX FMS documentation.

2.4 Using Forms to Display and Collect Data

After you define a form and insert the form definition in a form library, you can use the form. The following sections explain how to use DATATRIEVE commands and statements to display, collect, store, and modify data on a form.

2.4.1 Enabling and Disabling Form Use

The command SET [NO] FORM determines whether DATATRIEVE uses a form. If SET FORM is in effect and you ready a domain whose definition includes the FORM IS clause, or you use the DISPLAY_FORM statement, DATATRIEVE opens the form library specified.

If SET NO FORM is in effect, DATATRIEVE does not open a form library, and you cannot use a form. When you are using a domain whose definition includes a FORM IS clause, it is sometimes useful to review the contents of many records quickly. SET NO FORM allows you to override form display and use regular screen display.

Note that if you use a DATATRIEVE image installed without a forms package, using SET NO FORM does not let you use a domain definition that contains a FORM IS clause. You must edit the domain definition to remove the form reference or install DATATRIEVE with a forms package. The default is SET FORM.

You can see if SET FORM is in effect by using the SHOW SET_UP command. You can see which forms are currently loaded by entering SHOW FORMS. The SHOW FORMS command will display the type (FMS, TDMS or DECforms) for each of the forms in use. To release a form loaded with the DISPLAY_FORM statement from your workspace, use the RELEASE form-name command.

2.4.2 Displaying Data with Forms

When DATATRIEVE uses forms to display records, only one record at a time is displayed on the screen. To proceed to the next record, press the ENTER key or the RETURN key. If you want to skip the rest of the records, position the cursor on a nonnumeric field and press CTRL/C and then the RETURN key.

The following example shows how to display records from a domain whose definition includes the FORM IS clause:

```
DTR> SET FORM
DTR> READY YACHTS
DTR> PRINT FIRST 10 YACHTS
```

The PRINT statement in this example causes DATATRIEVE to use a form to display the first record in YACHTS. DATATRIEVE displays all the fields you included in your form definition. You can press the RETURN key to display the next record. Continue pressing the RETURN key until the tenth record is displayed. The next time you press the RETURN key, you get the DTR> prompt. If you want to display values for only a few fields, you can use a form with a DATATRIEVE view domain or you can use the DISPLAY_FORM statement. Your DISPLAY_FORM statement can refer to an entirely different form from the one specified in the FORM IS clause. It can also refer to the same form.

For example, suppose you want to display only the type and price of the first ten yachts on the YACHT form. The fields MANUFACTURER, MODEL, and PRICE in YACHTS correspond to the fields MANUFACTURER, MODEL, and PRICE in the YACHT form. Use the following statements:

```
DTR> FOR FIRST 10 YACHTS
CON> DISPLAY_FORM YACHT IN DTR$LIBRARY:FORMS USING
CON>    BEGIN
CON>       PUT_FORM MANUFACTURER = MANUFACTURER
CON>       PUT_FORM MODEL        = MODEL
CON>       PUT_FORM PRICE        = PRICE
CON>    END
```

With the DISPLAY_FORM statement, you can use a number of forms to display data in one domain. For example, suppose you create two forms for YACHTS: TYPE and SPECS. The form TYPE contains the fields MANUFACTURER and MODEL, corresponding to the MANUFACTURER and MODEL fields in YACHTS. The form SPECS contains fields that correspond to the remaining YACHTS fields.

The following procedure shows how to display each of the first 10 YACHTS records on two forms. For each record in the FOR loop, first the TYPE form is displayed and then the SPECS form is displayed.

```
PROCEDURE DISPLAY_TWO_FORMS
FOR FIRST 10 YACHTS
   BEGIN
   DISPLAY_FORM TYPE IN FORMSLIB USING
        BEGIN
        PUT_FORM MANUFACTURER = MANUFACTURER
        PUT_FORM MODEL        = MODEL
        END
   DISPLAY_FORM SPECS IN FORMSLIB USING
        BEGIN
        PUT_FORM LENGTH    = LOA
        PUT_FORM DISPLACE  = DISPLACEMENT
        PUT_FORM RIG       = RIG
        PUT_FORM BEAM      = BEAM
        PUT_FORM PRICE     = PRICE
        END
   END
END_PROCEDURE
```

2.4.3 Storing Data with Forms

To store records in a domain that is defined to include a FORM IS clause, use the STORE statement:

```
DTR> READY YACHTS FOR WRITE
DTR> STORE YACHTS
```

DATATRIEVE displays the YACHT form, including any default and missing values you specify in your record definition. Then DATATRIEVE waits for you to enter data.

While entering data, you can use the following:

- The TAB key to move to the next field
- The BACKSPACE key to move to the previous field
- The right and left arrow keys to move within a field
- The LINE FEED key to delete the contents of a field
- CTRL/C to stop storing and prevent the current record from being stored

When you finish entering data, press the ENTER or the RETURN key. DATATRIEVE attempts to store the record as it appears on the screen. If any validation errors occur, DATATRIEVE displays an error message at the bottom of the screen and lets you change the field that caused the error.

Note that the FORM IS clause does not give you field-level access in a store operation. If you enter a STORE USING statement and the FORM IS clause is your only association of a domain with a form, DATATRIEVE does not display the form. If you want a STORE USING statement to display a form, you must

include a DISPLAY_FORM statement. Interactive data entry on a form displayed with DISPLAY_FORM is the same as that described for FORM IS.

If you use the DISPLAY_FORM statement to store, you must use its RETRIEVE clause.

For example, suppose you receive information for YACHTS in parts. The first information you receive is the manufacturer, model, rig, and overall length. You want to store this information and later modify your records to include further data.

The following example shows how to use the form YACHT to store partial records in the domain YACHTS:

```
DTR> READY YACHTS WRITE
DTR> STORE YACHTS USING
CON>    DISPLAY_FORM YACHT IN DTR$LIBRARY:FORMS RETRIEVE USING
CON>        BEGIN
CON>          MANUFACTURER = GET_FORM VENDOR
CON>          MODEL        = GET_FORM MODEL
CON>          RIG          = GET_FORM RIG
CON>          LOA          = GET_FORM LENGTH
CON>        END
```

By using both the PUT_FORM and GET_FORM components of the DISPLAY_FORM statement, you can store values in fields without data entry from the form user. This is particularly useful when storing values into date fields and primary keys. If you specify the Display Only attribute for these fields in your form definition, you can prevent the form user from overriding the values you send to the form. By not selecting the Display Only attribute, you can allow the form user to modify such values.

For example, suppose you create the form PERSON to store data into PERSONNEL. Table 2–2 shows the fields in the form PERSON and corresponding fields in the domain PERSONNEL.

Table 2–2 Corresponding Fields in the Form PERSON and the Domain PERSONNEL

Field in the form PERSON	Field in the Domain PERSONNEL
ID	ID
STATUS	EMPLOYEE_STATUS
NAME	EMPLOYEE_NAME
	FIRST_NAME
	LAST_NAME
DEPT	DEPT
DATE	START_DATE
SALARY	SALARY
SUP_ID	SUP_ID

The following procedure, STORE_PERSON, shows how you can use the form PERSON to store PERSONNEL records:

```
! Task: store a new employee record. Generate a unique badge
! number, but allow the user to override it.
! The default starting date is "TODAY", but  the user can override
! that also.

PROCEDURE STORE_PERSON
STORE PERSONNEL USING
BEGIN

! Display the ID and DATE on the form.

  DISPLAY_FORM PERSON IN FORMSLIB USING
    BEGIN
        PUT-FORM ID    = 1 + MAX ID OF PERSONNEL
        PUT-FORM DATE  = FORMAT "TODAY" USING DD-MMM-YYYY
    END RETRIEVE USING

! The rest of the fields on the form are empty.
! Retrieve data the user enters on the form.

! The form PERSON has one field for a name. The domain
! PERSONNEL has fields for FIRST_NAME and LAST_NAME.
! Get the first and last name from the form name string.

    BEGIN

        DECLARE BLANK_POSITION WORD.
        BLANK_POSITION = FN$STR_LOCATE (GET_FORM NAME, " ")
        FIRST_NAME     = FN$STR_EXTRACT (GET_FORM NAME, 1,
                           BLANK_POSITION)
        LAST_NAME      = FN$STR_EXTRACT (GET_FORM NAME,
                           BLANK_POSITION + 1, 50)

        ID             = GET_FORM ID
        START_DATE     = GET_FORM DATE
        IF (GET_FORM STATUS  CONT "T")
            THEN STATUS = "TRAINEE" ELSE STATUS = "EXPERIENCED"
        DEPT           = GET_FORM DEPT
        SALARY         = GET_FORM SALARY
        SUP_ID         = GET_FORM SUP_ID
    END
END;
```

2.4.3.1 Storing Data in Hierarchical Records with Forms

To store data in a hierarchical record, use the STORE statement and the DISPLAY_FORM statement. The following example uses the DISPLAY_FORM statement to store data in the hierarchical record for the domain FAMILIES:

```
DTR> READY FAMILIES WRITE
DTR> !
DTR> ! The variable, A, is used to establish context for the
DTR> ! list field KIDS.
DTR> !
DTR> STORE A IN FAMILIES USING
CON> BEGIN
CON>        DISPLAY_FORM FAMILY IN DTR$LIBRARY:FORMS RETRIEVE USING
CON>        BEGIN
CON>               MOTHER = GET_FORM MOTHER
CON>               FATHER = GET_FORM FATHER
CON>               NUMBER_KIDS = GET_FORM NUMBER_KIDS
CON> !
CON> ! The MATCH statement transfers the data retrieved
CON> ! from the form with the GET_FORM KID_NAME and AGE value
CON> ! expressions to the KIDS fields.  In other words, each A.KIDS
CON> ! field value is transferred from the form to each KIDS record.
CON> !
CON>               MATCH KIDS, A.KIDS
CON>                 BEGIN
CON>                      KID_NAME = GET_FORM KID_NAME
CON>                      AGE = GET_FORM AGE
CON>                 END
CON>        END
CON> END
```

See the chapter on using hierarchies in the *VAX DATATRIEVE User's Guide* for more information.

2.4.4 Modifying Data with Forms

To modify records in a domain that uses a form, use the MODIFY statement:

```
DTR> READY YACHTS MODIFY
DTR> FOR YACHTS WITH PRICE > 30000
DTR> MODIFY
```

DATATRIEVE uses the form YACHT to display the record you specify. You can now modify the fields in that record. To move through the form, use the same keys you used while storing.

If any validation errors occur, DATATRIEVE displays the error message at the bottom of the screen and lets you change the field that caused the error. If you try to modify a key field that is defined with the NO CHANGE attribute, however, DATATRIEVE prints an error message and does not modify any fields in the record.

Because primary key fields are defined NO CHANGE by default, you lose all the modifications you make to a record when you try to modify its primary key field. If you do not change the primary key field, there is no problem. If you do try to modify a primary key field, DATATRIEVE returns an error and ignores the other modifications to the record. To avoid the error, you may want to define primary key form fields as Display Only form fields.

Note that the FORM IS clause does not give you field-level access in a modify operation. If you enter a MODIFY USING statement and the FORM IS clause is your only association of a domain with a form, DATATRIEVE does not display the form. If you want to use a MODIFY USING statement to display a form, you must include a DISPLAY_FORM statement.

You can include the DISPLAY_FORM statement within a MODIFY USING statement to modify data in a domain, whether or not a form is already assigned to the domain with the FORM IS clause.

If you use DISPLAY_FORM to modify, include all sections of the statement. The PUT_FORM statements display the data users are to modify, and the GET_ FORM value expressions store their changes.

In Section 2.4.3, the example showing how to store partial records in the domain YACHTS created records with data for the fields MANUFACTURER, MODEL, RIG, and LENGTH_OVER_ALL. Suppose you now have data for the rest of the fields and want to update the records you stored. You do not want to modify the data already entered; you want to enter values for the fields DISPLACEMENT, BEAM, and PRICE, thereby modifying only those fields.

One way to perform this task is to create a new form, YACHT_FORM2. This form contains fields that correspond to all the fields in YACHTS, as illustrated in Table 2–3.

Table 2–3 Modifying Data in Some Fields of a Form and a Domain

Field in the New Form YFRM2	Field in the Domain YACHTS
MANUFACTURER	MANUFACTURER
MODEL	MODEL
RIG	RIG
LENGTH	LENGTH_OVER_ALL
DISPLACE	DISPLACEMENT
BEAM	BEAM
PRICE	PRICE

When you create the form YACHT_FORM2, you can assign the Display Only attribute to the first four fields. With this qualifier in place, the form user can enter data only for the last three fields.

The following procedure, STORE_YACHTS_2, shows how to modify data on the form YACHT_FORM2, by entering values only for the three fields that do not have any data:

```
PROCEDURE STORE_YACHTS_2
READY YACHTS MODIFY

! Modify only YACHTS that are missing data.

FOR YACHTS WITH DISPLACEMENT = 0
MODIFY USING
   BEGIN

! Use the form YACHT_FORM2 to display entire records.

   DISPLAY_FORM YACHT_FORM2 IN FORMSLIB USING

! Display the fields for which data has been entered.
! These fields are defined as Display Only to TDMS or FMS.
! The user cannot enter data in them.
! (If you do not define these fields as Display Only, the
! user can change the form fields. However, DATATRIEVE
! ignores the changes.)

       BEGIN
          PUT_FORM MANUFACTURER = MANUFACTURER
          PUT_FORM MODEL    = MODEL
          PUT_FORM RIG      = RIG
          PUT_FORM LENGTH   = LOA
       END   RETRIEVE USING

! Get new data from the form.

       BEGIN
         DISPLACEMENT = GET_FORM DISPLACE
         BEAM         = GET_FORM BEAM
         PRICE        = GET_FORM PRICE
       END
   END
END_PROCEDURE
```

2.4.4.1 Modifying Data in Hierarchical Records with Forms

If you modify data in a hierarchical record, use the MODIFY statement and the DISPLAY_FORM statement. The following example uses the DISPLAY_FORM statement to modify data in the hierarchical record domain FAMILIES:

```
DTR> READY FAMILIES WRITE
DTR> FIND ALL FAMILIES WITH FATHER CONT JIM
CON> FOR CURRENT
CON> BEGIN
CON>        MODIFY USING DISPLAY_FORM FAMILY IN DTR$LIBRARY:FORMS USING
CON>        BEGIN
CON>               PUT_FORM FATHER      = FATHER
CON>               PUT_FORM MOTHER      = MOTHER
CON>               PUT_FORM NUMBER_KIDS = NUMBER_KIDS
CON> !
CON> ! The FOR loop establishes context for the KIDS field.
CON> !
CON>               FOR KIDS
CON>               BEGIN
CON>                      PUT_FORM KID_NAME = KID_NAME
CON>                      PUT_FORM AGE      = AGE
CON>               END
CON>        END   RETRIEVE USING
CON>        BEGIN
CON>               FATHER      = GET_FORM FATHER
CON>               MOTHER      = GET_FORM MOTHER
CON>               NUMBER_KIDS = GET_FORM NUMBER_KIDS
CON> !
CON> ! The FOR loop establishes the context for retrieving
CON> ! modified values from each occurrence of KIDS that
CON> ! satisfies the RSE and for storing those values in
CON> ! the KIDS record.
CON> !
CON>               FOR KIDS MODIFY USING
CON>               BEGIN
CON>                      KID_NAME = GET_FORM KID_NAME
CON>                      AGE      = GET_FORM AGE
CON>               END
CON>        END
CON> END
```

See the chapter on using hierarchies in the *VAX DATATRIEVE User's Guide* for more information.

2.4.5 Handling Numeric Data

Many numeric fields are stored with a fractional component and a sign. These fields may include a PICTURE string that explicitly specifies a sign and decimal point, for example, PIC S999V99. However, numeric values for some fields include a sign and fractional component in storage, even though there is no picture string to alert you to this fact. Some examples are record fields defined as USAGE REAL, USAGE WORD SCALE IS –2, or USAGE LONG SCALE IS –3. Section 2.2.2.1 on numeric fields with decimal points or signs explains how to define form fields that can contain both a sign and a decimal point.

If you use the FORM IS clause to transfer data to and from record fields that include a fraction and a sign in storage, results can be undesirable. As a general rule, never include the FORM IS clause in a domain definition when its record definition includes these numeric fields. Always use the DISPLAY_FORM statement to store, display, and modify such numeric fields. When you use the DISPLAY_FORM statement, you can assume control of the format and scale of the text string passed between the record field and the form field.

The following example illustrates how you handle numeric data that includes a decimal point and a sign. The CB_BALANCE field in the CHECKBOOK domain is defined as PIC S999V99. The associated BALANCE field on the CHECKS form is defined as NNNN.NN.

The DISPLAY_FORM statement transfers the value from the record field as a whole number and uses a FORMAT value expression to specify the text string as it should appear on the form field. The edit string in the example includes a minus sign (–) as a numeric insertion character so that only negative values are displayed with a sign. As an alternative, you can use a plus sign (+) if you want positive as well as negative values displayed with signs.

Because the value CB_BALANCE is stored with two decimal digits, the PUT_FORM Assignment statement must multiply the field value by 100 before transferring it to the form field. The Assignment statement in the RETRIEVE section divides the whole number value from the form field by 100 before storing it in the record field. Note that, in the following example, a FORMAT expression edit string is not necessary when the form field value returns to the record field:

```
MODIFY CHECKBOOK USING
DISPLAY_FORM CHECKS IN FORMSLIB USING
   PUT_FORM BALANCE =
      FORMAT (100 * CB_BALANCE) USING
         -99999 RETRIEVE USING
            CB_BALANCE = (GET_FORM BALANCE) / 100
```

You must multiply and divide values by the appropriate power of 10 for the record field definition. The example multiplies and divides values by 100 because the record field stores two decimal places. If, for example, the record field you are handling includes a SCALE IS –3 clause, you multiply and divide values by 1000.

Form fields that include a decimal point and are defined as 9s require the same treatment in a DISPLAY_FORM statement. Because such fields cannot include a sign character, however, you omit the plus (+) or minus (–) character from the edit string in the FORMAT value expression.

2.5 Using TDMS and FMS Forms in a DECwindows Environment

You can use DATATRIEVE with FMS or TDMS forms in a DECwindows environment as long as the following conditions exist:

- You have executed the SET FORM command.
- Your domains have been properly defined with the FORM IS clause of the DEFINE DOMAIN command.

When you ready a domain that uses a TDMS or FMS form, DATATRIEVE spawns a DECterm window. When you issue a statement, such as a print statement, the form is displayed in the DECterm window. The window remains on your screen until you either exit DATATRIEVE or until you use the FINISH command to end your access to the domain.

Note

Do not use the Quit option of the Commands menu in the DECterm window to dismiss the DECterm window unless you have done the following:

- Executed the SET NO FORM command
- Used the FINISH command to end your access to the domain or domains

If you ready more than one domain that uses a TDMS or FMS form, the DECterm window remains until you have used the FINISH command to end access to all of those domains.

Note

If you shrink your forms window to an icon and you perform some action from another window that affects your forms window, you may get unexpected results.

2.6 Restrictions on Using Forms

The following sections describe restrictions on using DATATRIEVE with forms. They also provide examples of alternatives to these restricted uses of forms.

2.6.1 DATATRIEVE and FMS

When you use the DISPLAY_FORM statement with FMS forms, DATATRIEVE passes a default field descriptor of 255 characters. If you try to concatenate fields from an FMS form, you get unexpected results. You can explicitly specify the description of a form field using the FORMAT value expression. Include an edit string in the USING clause of the FORMAT value expression so that DATATRIEVE does not use the 255-character default for an FMS form field.

The following example specifies field lengths for the fields MANUFACTURER and MODEL using FORMAT value expressions:

```
DECLARE FLD PIC X(30).
DISPLAY_FORM YACHT_FORM IN FORMSLIB;
  BEGIN
     PUT_FORM MANUFACTURER = MANUFACTURER
     PUT_FORM MODEL        = MODEL
  END RETRIEVE USING
     FLD = FORMAT (GET_FORM MANUFACTURER) USING X(20)| -
           FORMAT (GET_FORM MODEL) USING X(6);
```

When DATATRIEVE concatenates the MANUFACTURER and MODEL fields, it uses 20 characters for MANUFACTURER and 6 characters for MODEL instead of 255 characters for each field.

An additional restriction concerns the use of FMS forms with the DATATRIEVE Call Interface. If you have an application program that displays FMS forms and also calls upon the DATATRIEVE Call Interface to display forms using the FORM IS clause in a domain definition, you must save and restore the FMS terminal control areas after each call to DATATRIEVE. This restriction applies only to the use of the DATATRIEVE Call Interface with domains using the FORM IS clause. You can avoid the restriction by using DISPLAY_FORM instead of FORM IS.

See the VAX FMS documentation for more information about FMS terminal control areas and about the restriction on using the DATATRIEVE Call Interface.

2.6.2 DATATRIEVE Command Files and Forms Products

This restriction and its workaround apply only to Version 2 and later of FMS and Version 1 and later of TDMS.

DATATRIEVE uses SYS$INPUT to get its commands from VMS command files. Both FMS and TDMS also use SYS$INPUT to get terminal input. Therefore, you cannot use both DATATRIEVE and a forms product in the same command file unless you set up that file using the following steps:

1. The default interactive assignment for SYS$INPUT is your terminal. When you run a command file, SYS$INPUT is the command file itself. Therefore, your command file must assign SYS$INPUT to SYS$COMMAND (the default device name of your terminal) so that the forms product can get input from the terminal. That assignment must precede the invocation of DATATRIEVE.

   ```
   $ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
   ```

2. Then you must include all the DATATRIEVE commands and statements in a procedure that is invoked by the command file.

The resulting command file must include the following:

```
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$ DTR EXECUTE procedure-name
```

2.6.3 Modifying Data Using View Domains and FORM IS

You must be very careful when modifying records in a view based on more than one domain. The restriction documented here applies to modifying a field in a view when the modified field is the basis for selecting records from another domain. Although this restriction applies to views, it is included here to illustrate how the use of a form with views can mask modification errors.

You unintentionally modify data if you try to modify fields when all of the following conditions are true:

- You modify records in a view domain that uses FORM IS.
- The view selects records from another domain based on the value of a field in the view.
- You modify the field that forms the basis for selecting records from the second domain.

The following example shows what happens when you try to modify fields that refer to other domains in a view using a form. The sample view domain SAILBOATS has been edited to create the example.

The following is a view domain containing a FORM IS clause:

```
DTR> SHOW SAILBOATS

DOMAIN SAILBOATS
 OF CDD$TOP.DTR$LIB.DEMO.YACHTS_SEQUENTIAL, CDD$TOP.DTR$LIB.DEMO.OWNERS BY
01 SAILBOAT OCCURS FOR YACHTS_SEQUENTIAL.
   03 BOAT FROM YACHTS_SEQUENTIAL.
   03 SKIPPERS OCCURS FOR OWNERS WITH BUILDER EQ BOAT.BUILDER.
      05 NAME FROM OWNERS.
    FORM IS SAIL IN DTR$LIBRARY:FORMS
;
```

SAILBOATS refers to the OWNERS domain based on the value for BUILDER.

The following example shows the OWNERS records before the modification:

```
DTR> READY CDD$TOP.DTR$LIB.DEMO.OWNERS
DTR> PRINT OWNERS

OWNER
NAME      BOAT NAME      BUILDER     MODEL

SHERM MILLENNIUM FALCON ALBERG     35
STEVE DELIVERANCE       ALBIN      VEGA
HUGH  IMPULSE           ALBIN      VEGA
              .
              .
              .
```

Modify SAILBOATS as follows:

```
DTR> READY SAILBOATS
DTR> FOR FIRST 1 SAILBOATS WITH ANY SKIPPERS MODIFY
```

DATATRIEVE displays the first record that meets the criterion WITH BUILDER EQ BOAT.BUILDER on the form. Now modify the record to include a new value for BUILDER:

```
                    YACHT SPECIFICATION DATA

     Builder: ALBERG  ! Changed to ALBIN     Model: 37 MK II

     Length:  37          Beam:  12          Disp:  20000

     Rig:  KETCH                             PRICE: 36951

     Owners:  SHERM
```

DATATRIEVE updates all the fields on a form when you press the ENTER or RETURN key to complete data entry. Because you changed ALBERG to ALBIN, the owner name from the updated record appears in the next record for which there is a BUILDER named ALBIN.

SHERM now replaces STEVE in that OWNERS record.

```
DTR> READY CDD$TOP.DTR$LIB.DEMO.OWNERS
DTR> PRINT OWNERS

OWNER
NAME      BOAT NAME      BUILDER     MODEL

SHERM MILLENNIUM FALCON ALBERG     35
SHERM DELIVERANCE       ALBIN      VEGA
      IMPULSE           ALBIN      VEGA
JIM   EGRET             C&C        CORVETTE
                .
                .
                .
```

2.6.4 Special Graphics Characters in Forms

When you design a form using characters from the VT100 special graphics set, the characters might not automatically work from one form invocation to the next. If this problem occurs, press CTRL/W to repaint the form.

3

Using DECforms with DATATRIEVE

VAX DATATRIEVE uses DECforms as one of its forms interfaces (see Chapter 2 for more information about other DATATRIEVE's forms products). This chapter explains the DECforms concept of a form and the run-time interaction of DATATRIEVE with DECforms. This chapter does not describe DECforms features and terminology. For a complete description of DECforms, read the DECforms documentation.

3.1 Overview of DECforms

DECforms is a tool for integrating text and simple graphics into forms and menus. You can develop a DECforms form to create an interface to an application program such as DATATRIEVE.

A form is a collection of two basic types of structured information: background information that does not change (such as titles, headings, instructions), and changeable information (names, addresses, numbers, and so on...) that is entered by a user of the form, or displayed by an application program.

DECforms aids in the development of applications by separating forms from function. The form hides the application program from the user's view. The form takes care of the interaction between a terminal and the program, consequently the program remains unaware of the device it is dealing with. The program is concerned only about the data to be displayed or collected; it does not need to know where or in what format the data is placed on the screen. Therefore, using DECforms results in simpler programming and more maintainable applications.

You can see examples of DECforms forms by following the steps described in Section 2.1. Note, however, the following:

- To display a new record, press F10 instead of the RETURN key.
- To stop the display operation, press CTRL/C followed by F10.

3.1.1 Files Associated with DECforms Forms

There are four types of files associated with a form: the form source (.IFDL) file, the binary (.FORM) file, the object module (.OBJ) of the form, and the shareable image (.EXE) of the form. The binary and the shareable image files are both used at run time.

3.2 Form Management Combinations

In DATATRIEVE it is possible to use either FMS or TDMS in addition to DECforms in the same DATATRIEVE image. For more information on VAX FMS and VAX TDMS forms products, see Chapter 2. DECforms software is dynamically loaded at run-time and no specific operations are required at installation time. This means that DECforms, whenever installed on the target system, is always available, while FMS and TDMS are optional and mutually exclusive, and are selected by the user at installation time.

At execution, if the SET FORM option is set, DATATRIEVE performs the following actions with the form file specified in the syntax:

1. If DECforms is installed and the specified file is a DECforms form file, DECforms will be used with the specified form.
2. If the first step fails, DATATRIEVE tries to use VAX FMS or VAX TDMS software (if either of them is linked to the DATATRIEVE image).
3. If the second step fails, DATATRIEVE returns an appropriate error message.

3.3 The DECforms Interface to DATATRIEVE

The DECforms *Form Manager* is the interface between an application program—in this case DATATRIEVE—and a display device. It is a run-time system that controls form display and operation input on terminals. The Form Manager consists of six subroutines known as *requests*, they are:

- ENABLE— initializes a DECforms session.
- SEND— sends data (one or more records) to the form from the application program.
- RECEIVE— receives data (one or more records) from the form in the application program.
- TRANSCEIVE— sends data (one or more records) to the form and receives data (one or more records) from the form in one call.
- DISABLE— terminates a DECforms session.

- CANCEL— cancels all requests for a DECforms session.

DATATRIEVE automatically makes calls to the ENABLE and DISABLE requests. DATATRIEVE also makes calls to the SEND, RECEIVE, and TRANSCEIVE requests both when you PRINT, STORE, or MODIFY a domain record to which a form is linked (see Section 3.4.2.1 for more information) and when you use the SEND and RECEIVE clauses of the WITH_FORM statement (see Section 3.4.2.2 for more information).

The DECforms CANCEL request is not performed by DATATRIEVE.

Within DECforms forms you can perform DATATRIEVE queries. This is possible because DECforms allows you to call DATATRIEVE routines by using DATATRIEVE ports. See *VAX DATATRIEVE Guide to Programming and Customizing* for more information on DATATRIEVE routines and ports.

3.4 Creating an Application

To create a DATATRIEVE application that uses DECforms as an interface, you need to take the following actions:

- If you do not have a DECforms form, use a DECforms editor to create the form.
- Associate the DECforms form with a DATATRIEVE domain, or use a specific statement (WITH_FORM) to control interaction with the form.

Note

Make sure that the DATATRIEVE SET FORM command is in effect when your application executes.

Following sections discuss each of these steps in greater detail.

3.4.1 Creating a DECforms Form

You can create a DECforms form using the Form Development Environment (FDE) or the Panel Editor. For more information on how to create a DECforms form see the DECforms documentation.

Before you begin to work on a form you need to plan the DATATRIEVE program. First you must analyze the task that DATATRIEVE is to perform and determine what type of data it will be working with—the type of data the operator is expected to enter at the terminal. Then, you can prepare the source code to handle the form you will need for the application. Once you have decided what

data is to be transferred between DATATRIEVE and the form, and you have defined the DATATRIEVE records to transfer the data, you can create the form.

3.4.1.1 Transferring Records

The form communicates with DATATRIEVE through records. Records can be sent and received. When you define a record in DECforms make sure that the form record is logically equivalent to the DATATRIEVE record (the terminology used to define a DATATRIEVE record differs from the terminology used to define a DECforms record, see Section 3.4.1.2 for more information). In particular, make sure that the DATATRIEVE record has as many fields as the form record, and that each field in the DATATRIEVE record has the same data type, length, and position of the fields in the form record. The DATATRIEVE field names may differ from the DECforms field names.

The DECforms record can be a record by itself or a list. A list is a DECforms structure, it is like a pointer to different records within the form itself. In some cases the DECforms record requires the same name as the top level field of the DATATRIEVE record. [1] This match is required when you use the FORM IS clause of the DEFINE DOMAIN command, but it is not required when you use the WITH_FORM statement.

If the FORM IS clause is the only association a domain has with a form, DATATRIEVE knows the form name, but needs in addition to know the form-record name. Since there is no way to explicitly specify the record name used by the form, the form-record name for a form associated with a domain must be the same as the name of the top-level domain-record. If the above condition is not satisfied, DATATRIEVE will give an error message. When an exchange record is specified, the matching must take place between the form reference and the name of the top level field of the exchange record (which actually must match the structure of the form record).

3.4.1.2 Data Types in DECforms and DATATRIEVE

In DATATRIEVE, every time you define an elementary field in your record definition, you must specify either a PICTURE or USAGE clause to tell DATATRIEVE what kind of characters are stored in the field and how many characters can fit.

In DECforms you have to follow the same process but the terminology is different. For instance if you want to declare an alphabetic field of 8 characters in a DATATRIEVE record, enter the following:

```
PIC X(8)
```

[1] In DATATRIEVE the top level field is equal to the record name.

But to declare the corresponding field in a DECforms record, enter the following:

```
CHARACTER (8)
```

Note that a DECforms field definition does not always match the corresponding DATATRIEVE field definition. For example, when you define a numeric field to be a signed integer, in the DATATRIEVE field the sign shares a character position with the field's rightmost digit, while in the DECforms field the sign has its own position. For this reason we suggest that you share record definitions between DATATRIEVE and DECforms using the CDD/Repository. See Section 3.4.1.3 for more information.

3.4.1.3 Using CDD/Repository to Store Record Definitions

To ensure a perfect match between DATATRIEVE and DECforms records, you should inherit the record definitions from the CDD/Repository. This method is easier and more reliable than defining one record in DATATRIEVE and the corresponding record in DECforms.

To take advantages of the CDD/Repository, do the following:

- If the record does not already exist in the CDD/Repository, define it with the CDD/Repository utilities (CDO) or with DATATRIEVE.
- Copy the record into DATATRIEVE using the FROM clause of the DEFINE RECORD command, or point to the CDD/Repository record using its path name.
- Copy the record into DECforms using the COPY statement of the IFDL language.

3.4.2 Data Interaction Between DATATRIEVE and DECforms

You can use a DECforms form to display, store, or modify data from an RMS domain, a view domain, a VAX DBMS domain, a relational domain, and a remote domain.

There are two basic mechanisms of interaction with DECforms:

- You can associate a form definition with a DATATRIEVE domain using the FORM IS clause of the DEFINE DOMAIN command.
- You can use the WITH_FORM statement by itself or within a DATATRIEVE statement to control data interaction between specific form records and DATATRIEVE records, and to exchange control items with the form. In this way it is possible to use a form without linking it to a domain, data may be records, parts of records or variables, and you can control, in a very detailed way, the interaction with a form.

3.4.2.1 The FORM IS Clause

The DEFINE DOMAIN command allows the user to define a direct link between a domain and a form through the FORM IS clause. If you include the FORM IS clause in a domain definition, DATATRIEVE automatically uses the form to display, store, or modify records. The syntax of the DEFINE DOMAIN command may slightly change according to the type of domain (see the *VAX DATATRIEVE Reference Manual* for more information on the DEFINE DOMAIN command).

The form file specified in the DEFINE DOMAIN command can either be a .FORM or a .EXE file. The default file extension is .EXE. The example below shows how DECforms objects can be used in the DATATRIEVE DEFINE DOMAIN command syntax:

```
DEFINE DOMAIN YACHTS
 USING CDD$TOP.DTR$LIB.DEMO.YACHT ON YACHT.DAT
 FORM IS YACHT IN DTR$LIBRARY:FORMS.EXE;
```

If you specify a form file with .FORM extension, the form name is syntactically required but ignored. If you specify a form file with .EXE extension then the form name is relevant and it must match the form name in the .EXE form file. See the documentation on DECforms for more information on .FORM and .EXE files.

The optional USING clause added to the FORM IS definition allows you to define an exchange record structure that will be used to send and receive data with DECforms. For more information on the exchange record structure see Section 3.5. If the form is not a DECforms one, the USING clause in the FORM IS definition is ignored at run time.

Note

When you define a view domain with the FORM IS clause and the form is a DECforms form, then the exchange record is required. If you omit it from the domain definition, DATATRIEVE returns an error message.

3.4.2.2 The WITH_FORM Statement

The WITH_FORM statement sends records to and receives records from a form without creating a link between form and domain. The WITH_FORM statement can only be used with DECforms forms.

You can use the WITH_FORM statement within a STORE or MODIFY statement to specify the actions needed to get the new values. For more information on the WITH_FORM statement see the *VAX DATATRIEVE Reference Manual*.

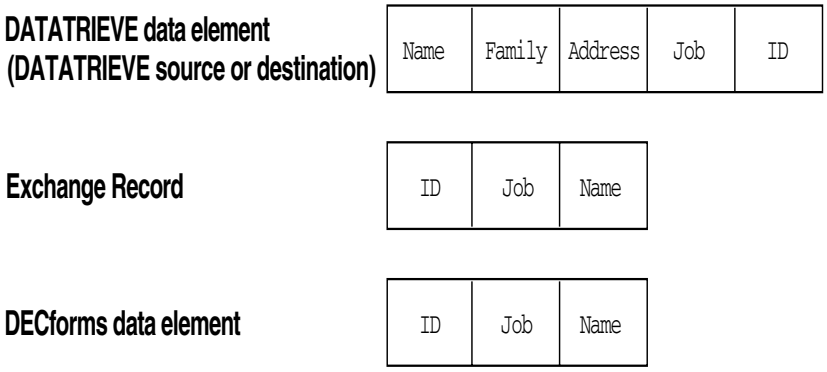
3.5 The Exchange Record

To exchange data, DATATRIEVE applications and DECforms forms must share a record definition. A DATATRIEVE data element[1] is used as the physical element to be sent to the form or as the storage area to receive data from the form. Sometimes the DATATRIEVE data element does not match a form record. For this reason an optional USING clause has been added, to allow you to transfer records that do not match between DECforms and DATATRIEVE.

The optional USING clause allows you to define an exchange record structure shared by DATATRIEVE and DECforms. DATATRIEVE uses a record buffer, structured along the exchange record definition, to move data to and from the form record.

The exchange record looks like a temporary record used as a filter for data to be sent or received from a form (see Figure 3–1).

Figure 3–1 The Exchange Record Mechanism



In a SEND operation, the data is transferred with a standard group-to-group assignment from the DATATRIEVE data element to the temporary record. The temporary record then becomes the actual record sent to the form. For more information on the mechanism of the group-to-group assignment, see the section on Assignment Statements in the *VAX DATATRIEVE Reference Manual*.

In a RECEIVE operation, data coming from a record form is collected in the temporary record which is then transferred with a standard group-to-group assignment to the final destination (a DATATRIEVE data element).

[1] A DATATRIEVE data element can be represented by a DATATRIEVE record, group field, field, or variable.

The exchange record does not require a full match with the DATATRIEVE data element. It is possible to define in the exchange record only the fields needed by the form (note that such fields, belonging to the DATATRIEVE data element, can be defined in the exchange record even with different data types). This is valid for both the FORM IS clause of the DEFINE DOMAIN command, and for the WITH_FORM statement.

If there is an exchange record linked to a record in the SEND or RECEIVE clauses, the field names of the exchange record must be identical to the field names of the DATATRIEVE record because this is needed by the group-to-group DATATRIEVE assignment. This is also valid for the FORM IS clause of the DEFINE DOMAIN command.

The optional USING clause of the FORM IS definition is supported only for a domain defined in the CDO dictionary (when the domain is defined in a DMU dictionary, the clause is stripped away from the definition). Nevertheless, the exchange record can exist either in DMU or CDO dictionary format.

You have to define the exchange record before you define the domain containing it even if DATATRIEVE opens it only when you ready the domain

When executing a WITH_FORM statement, DATATRIEVE does not convert the data sent to or received from DECforms. Only if there is an exchange record does DATATRIEVE perform the conversions between the two structures. All the other data conversions are done within DECforms.

Note

COMPUTED BY fields, VALID IF clauses, and MISSING VALUE clauses are ignored in the exchange record definition. If you want to use these fields and clauses, you have to define them in the actual DATATRIEVE data element.

3.5.1 When is the Exchange Record Needed ?

The exchange record is needed on the following occasions:

- When, in the DATATRIEVE data element, some fields are present logically but not physically, and the data type is not fully determined, as in the case of a record containing COMPUTED BY fields. The following example shows you the declaration of a DATATRIEVE record and its corresponding exchange record; note that the COMPUTED BY field has been declared as a physical field in the exchange record.

```
DEFINE RECORD RECORD_1
01 RECORD_1.
   10 PRODUCT_ID PIC X(5).
   10 PRICE PIC 9(10).
   10 DISCOUNT COMPUTED BY PRICE * 0.1.;

DEFINE RECORD EXCHANGE_1
01 EXCHANGE_1.
   10 PRODUCT_ID PIC X(5).
   10 PRICE PIC 9(10).
   10 DISCOUNT PIC 9(10).;
```

- When the DATATRIEVE data element has more fields than the DECforms record/list, or when some fields are either in a different position or have a different data type. The following example shows the declaration of a DATATRIEVE record, its corresponding exchange record, and a DECforms record.

```
DEFINE RECORD RECORD_2
01 RECORD_2.
   10 PRICE PIC S9(10).
   10 PRODUCT_ID PIC X(5).;

DEFINE RECORD EXCHANGE_2
01 EXCHANGE_2.
   10 PRODUCT_ID PIC X(5).
   10 PRICE PIC S9(10) SIGN LEADING SEPARATE.;

FORM RECORD FORM_REC_2
             PRODUCT_ID       CHARACTER (10)
             PRICE            INTEGER (9)
END RECORD
```

  In this example the exchange record is used to:

  – Invert the fields named price and product_id.

  – Get a field named price which exactly matches (in length and data type) the field price in the form record.

- When the DATATRIEVE data element is not represented by one physically contiguous storage area (a typical example is a record produced by a REDUCED TO clause), or when the DATATRIEVE data element belongs to a record stream composed of crossed sources. The following example shows you the declaration of a DATATRIEVE record, the domain definition, and how to send data to a DECforms form using a record reduced to few fields. Note that the exchange record and the form record are those used in the previous example.

```
DEFINE RECORD RECORD_3
01 RECORD_3.
   10 PRODUCT_ID PIC X(5).
   10 PRODUCER_ID PIC X(8).
   10 CLASS PIC X(2).
   10 PRICE PIC S9(10) SIGN LEADING SEPARATE.;

DEFINE DOMAIN DOMAIN_3 USING RECORD_3 ON FILE_2.DAT;

FOR X IN DOMAIN_3 REDUCED TO PRODUCT_ID, PRICE
   WITH_FORM FORM_2 IN FORM_FILE_2.EXE
      SEND FROM X USING EXCHANGE_2 TO FORM_REC_2;
```

- When the DATATRIEVE data element has an undefined number of fields, as in the case of a view record that has fields defined through the OCCURS clause. The following example shows you the declaration of a DATATRIEVE record, a view domain, the corresponding exchange record, and a DECforms record; it then shows you how to send data to a DECforms form using a view record that has fields defined through the OCCURS clause.

```
DEFINE RECORD RECORD_4
01 RECORD_4.
   10 PRODUCT_ID PIC X(5).
   10 PRODUCT_NAME PIC X(15).;

DEFINE DOMAIN DOMAIN_4 USING RECORD_4 ON FILE_4.DAT;
DEFINE DOMAIN DOMAIN_1 USING RECORD_1 ON FILE_1.DAT;
DEFINE DOMAIN DOMAIN_5 OF DOMAIN_1, DOMAIN_4
01 RECORD_5 OCCURS FOR DOMAIN_1.
   10 PRODUCT_ID FROM DOMAIN_1.
   10 PRODUCT_DESC OCCURS FOR DOMAIN_4 WITH DOMAIN_4.PRODUCT_ID
      EQ DOMAIN_1.PRODUCT_ID.
      20 PRODUCT_NAME FROM DOMAIN_4.;

DEFINE RECORD EXCHANGE_4
01 EXCHANGE_4.
   10 PRODUCT_ID PIC X(5).
   10 PRODUCT_DESC OCCURS 10 TIMES.
      20 PRODUCT_NAME PIC X(15).;

FORM RECORD FORM_RECORD_4
     PRODUCT_ID          CHARACTER (5)
     PRICE               INTEGER (10)
     GROUP PRODUCT_DESC  OCCURS 10
           PRODUCT_NAME  CHARACTER (15)
     END GROUP
END RECORD

FOR X IN DOMAIN_5 WITH_FORM FORM_4 IN FORM_FILE_4.EXE
   SEND FROM X USING EXCHANGE_4 TO FORM_RECORD_4;
```

3.6 Using DECforms to Display and Collect Data

The following sections explain how to use DATATRIEVE commands and statements to display, store, and modify data on a DECforms form.

3.6.1 Enabling and Disabling DECforms Use

DATATRIEVE enables a DECforms session when you issue the first PRINT, STORE, or MODIFY statement on a domain whose definition includes the FORM IS clause, or when you use the WITH_FORM statement.

The command SET [NO] FORM determines whether DATATRIEVE uses a form. You can see if SET FORM is in effect by using the SHOW SET_UP command. If SET NO FORM is in effect, DATATRIEVE does not use the form to perform the required operation. If SET FORM is in effect and you issue the first PRINT, MODIFY, or STORE statement on a domain whose definition includes the FORM IS clause, or if you use the WITH_FORM statement, DATATRIEVE enables a session for the form. Note that if you use the WITH_FORM statement or if you issue a PRINT, STORE, or MODIFY statement on a second domain that uses the same form name and form file, the existing session will be used.

You can see which forms are currently loaded by entering SHOW FORMS. The SHOW FORMS command will display the type (FMS, TDMS or DECforms) for each of the forms in use.

To disable a session, issue the FINISH command on a domain linked to a form. Note that if you have more than one domain linked to a form, finishing a single domain will not disable the session. You disable the session only when you issue the FINISH command on the last domain linked to the form. For more information on the FINISH command, see the *VAX DATATRIEVE Reference Manual*.

If a session was enabled with the WITH_FORM statement, to disable it you have to release the form loaded with the WITH_FORM statement from your workspace. Use the RELEASE form-name command. The form name (not the form file name) must be indicated. If more than one form has that name, only one is removed (the first in the list displayed by the SHOW FORMS command). For more information on the RELEASE command, see the *VAX DATATRIEVE Reference Manual*.

Note

If you want to use a form that has already been loaded on the system, you have to refresh your screen after you finish imputing your data (by pressing CTRL/W) in order to see the form on the screen.

3.6.2 Displaying Data with DECforms

There are two ways of displaying DATATRIEVE data using the DECforms interface, depending on whether the domain has a form associated with it or not.

3.6.2.1 The domain has a form associated with it

The following example shows how to display records from a domain whose definition includes the FORM IS clause:

```
DTR> SET FORM
DTR> READY YACHTS
DTR> PRINT YACHTS WITH LOA < 20
```

The PRINT statement in this example causes DATATRIEVE to use a form to display the first record in YACHTS. To display the next record, press the exit functional key defined by the form (usually F10 or CTRL/Z). Continue pressing that key until the last record is displayed. The next time you press the exit functional key, you get the DTR> prompt.

If you are displaying records and decide to skip the rest of the records, press CTRL/C then the exit functional key.

Note that since DECforms takes care of the interaction between the terminal and the program, DATATRIEVE has no control over the way data is displayed on the screen. In our examples DECforms displays one record, it then waits for a user action. This is obtained by specifying an ACTIVATE WAIT response in the SEND RESPONSE clause of the target form record.

3.6.2.2 No forms are associated with a domain

If you want to send data to a form without linking it to a domain or if you want to have more control of the data, you can use the WITH_FORM statement. Your WITH_FORM statement can refer to the same form specified in the FORM IS clause, or to an entirely different one.

The following example shows how records are sent to a form for display, by using WITH_FORM:

```
DTR> READY YACHTS
DTR> FOR X IN YACHTS
CON>    WITH_FORM YACHT IN DTR$LIBRARY:FORMS.EXE
CON>       SEND FROM X TO BOAT;
```

The WITH_FORM statement in this example causes DATATRIEVE to send the first record in the YACHTS domain to the form record BOAT. The form should respond to the send operation by displaying the form record data on the screen. Press the exit functional key to display the next record. Continue pressing the exit functional key until the last record is displayed. The next time you press that key, you get the DTR> prompt.

If you are displaying records and decide to skip the rest of the records, press CTRL/C then the exit functional key defined in the form.

Note that the SEND request does not always display data on the screen, since data sent to a form could be stored in form data without being displayed. The way to go back to DATATRIEVE from the form depends on the form (you have to see how the form is coded inside, if it displays or if it only stores data).

3.6.3 Storing Data with DECforms

The STORE statement enters a new data record in the DATATRIEVE domain. There are two different situations where the STORE statement is used, depending on whether the domain has a form associated with it or not.

3.6.3.1 The domain has a form associated with it

To store records in a domain that is defined to include a FORM IS clause, use the STORE statement:

```
DTR> READY YACHTS FOR WRITE
DTR> STORE YACHTS
```

DATATRIEVE calls the RECEIVE request specifying a receive record identifier that has the same name as the top level field of the domain record. The form should respond to this request by displaying the data entered by the user. While entering data, you must use the keys defined in the form that allow you to move through fields, enter data, or finish the store operation.

When you finish entering data, DATATRIEVE attempts to store the record as it is received from the form. If any validation errors occur, while checking the conditions given by the VALID IF clauses of the field definitions, DATATRIEVE displays an error message at the bottom of the screen and the statement fails. In this case the RECEIVE request is not repeated.

3.6.3.2 No forms are associated with the domain

If no forms are associated with the domain, and you want a STORE statement to display a form, or if the domain is linked to a form but you need more control (e.g. with control items), you must include a WITH_FORM statement.

Here is an example:

```
DTR> READY YACHTS WRITE
DTR> STORE X IN YACHTS USING
CON>    WITH_FORM YACHT IN DTR$LIBRARY:FORMS.EXE
CON>       RECEIVE FROM BOAT TO X;
```

In this example DATATRIEVE calls the RECEIVE request specifying the form record BOAT. The form should respond to this request by displaying the data entered by the user. Interactive data entry on a form displayed with WITH_ FORM is the same as that described for FORM IS.

The WITH_FORM statement can follow other statements, including other WITH_ FORM statements. But you cannot have a WITH_FORM statement inside a DISPLAY_FORM statement.

Note that the RECEIVE request is not equivalent to get-from-terminal, you may receive data from the form but nothing is displayed on the screen.

3.6.4 Modifying Data with DECforms

There are two different situations where the MODIFY statement is used, depending on whether the domain has a form associated with it or not.

3.6.4.1 The domain has a form associated with it

If you enter a MODIFY statement and the FORM IS clause is the only association a domain has with a form, the form connected to the domain will be used.

```
DTR> READY YACHTS MODIFY
DTR> MODIFY OF YACHTS WITH PRICE > 50000
```

DATATRIEVE calls the TRANSCEIVE request. The form should respond to this request by displaying the form panel. You can then modify the fields in that record. To move through the form, use the same keys you used while storing (they may vary, depending on how you defined them in the form).

Because primary key fields are defined NO CHANGE by default, when you try to modify them, DATATRIEVE returns an error message and you lose all the modifications you make to a record. To avoid the error, you may want to define primary key form fields as protected fields.

3.6.4.2 No forms are associated with the domain

If no forms are associated with the domain and you want to use a MODIFY USING statement to display a form, or the domain is linked to a form but you need more control (e.g.with control items), you must include a WITH_FORM statement.

You can include the WITH_FORM statement within a MODIFY statement to modify data in a domain, whether or not a form is already assigned to the domain with the FORM IS clause.

Just using the standard syntax and semantics for the MODIFY statement, the WITH_FORM statement may be used to SEND data to a form and RECEIVE it back, modified. Here is one example:

```
DTR> READY YACHTS MODIFY
DTR> MODIFY X IN YACHTS WITH BEAM = 13 USING
CON>    WITH_FORM YACHT IN DTR$LIBRARY:FORMS.EXE
CON>       SEND FROM X TO BOAT -
CON>          RECEIVE FROM BOAT TO X;
```

The form should react to the TRANSCEIVE operation, caused by a SEND and RECEIVE clauses in the same WITH_FORM statement, by displaying the form panel. It then waits for you to enter data. Interactive data entry on a form displayed with WITH_FORM is the same as that described for FORM IS.

3.7 Using DECforms in a DECwindows Environment

You can use DATATRIEVE with DECforms forms in a DECwindows environment as long as the following conditions exist:

- DECforms is available.
- You have executed the SET FORM command.

When you issue a PRINT, STORE, or MODIFY statement on a domain linked to a DECforms form, or when you use the WITH_FORM statement, DATATRIEVE spawns a DECterm window and the form is displayed in that window. Press the exit functional key defined in the form to display the next record. If you decide to skip the rest of the records, position the cursor on the main DATATRIEVE window and press CTRL/C, then move the cursor onto the DECterm window and press the exit functional key defined by the form (usually F10 or CTRL/Z).

The window remains on your screen until you either exit DATATRIEVE or use the FINISH command to end your access to the domain.

Note

Do not use the QUIT option of the Commands menu in the DECterm window to dismiss the DECterm window unless you used the FINISH command to end your access to the domain or domains.

If you associate more than one domain with that form, the DECterm window remains until you have used the FINISH command to end access to all of those domains.

When you issue the SHOW FORMS command in a DECwindows environment, DATATRIEVE spawns a Show window to display the command output. The SHOW FORMS command lists all the currently loaded forms, displaying also the forms type (FMS, TDMS, or DECforms).

3.8 Escape Routines

DECforms also supports a feature called escape routines. These are application program subroutines which can be called during form processing.

You use an escape routine when you need to do something you cannot do in a response. For example, you might write an escape routine that performs an arithmetic calculation or one that performs a file operation.

Escape routines can be implemented in two ways:

- Linked to a shareable image.
- Linked to an application program.

DATATRIEVE only supports the first option (but an application using callable DATATRIEVE can also use the second option). It is possible to use escape routines only when they are linked to a shareable image, and the name of the shareable image is assigned to the FORMS$IMAGE logical name. For a complete description of this feature see the DECforms documentation.

You can call DATATRIEVE from a DECforms form according to the rules described in the *VAX DATATRIEVE Guide to Programming and Customizing*.

4

Using DATATRIEVE with VAX DBMS

This chapter describes the commands, statements, and clauses that let you use VAX DATATRIEVE with VAX DBMS databases.

If you already use DATATRIEVE, you can skip Section 4.4 that deals with forming a DATATRIEVE query statement. This chapter discusses basic VAX DBMS concepts. For more information about VAX DBMS concepts, read the introductory documentation that accompanies VAX DBMS.

If you use VAX DBMS but are not familiar with DATATRIEVE, you can supplement this chapter by reading the chapter on writing record selection expressions in the *VAX DATATRIEVE User's Guide* and by using the VAX DATATRIEVE computer-based training package. You can also read the chapter on creating reports from a VAX DBMS database in the *VAX DATATRIEVE User's Guide*.

In this chapter, you learn to do the following:

- Create a database definition in DATATRIEVE that represents a VAX DBMS database
- Access the VAX DBMS database either by readying it directly or by defining domains for each VAX DBMS record and readying the domains
- Locate VAX DBMS records in a variety of ways
- Print whole records or parts of records
- Store and modify records
- Erase records
- Connect, reconnect, and disconnect records from sets
- Define and access view domains and hierarchical VAX DBMS records
- Create procedures and indirect command files that access VAX DBMS records and sets

This chapter uses examples in the PARTS sample database included with the DATATRIEVE User Environment Test Package (UETP) and installed on your system. The following command sets the dictionary default to the directory that contains the database domain definitions used in this chapter:

```
DTR> SET DICTIONARY CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS
```

Note

CDD$COMPATIBILITY is the logical name of the CDD/Repository compatibility dictionary. When you issue a SHOW command, you will see either the name of a physical disk or an anchor instead of the CDD$COMPATIBILITY logical name.

4.1 Advantages of Using DATATRIEVE

Although VAX DBMS has its own interactive query language to access VAX DBMS data, you may prefer to use DATATRIEVE. By using DATATRIEVE syntax with a few special VAX DBMS extensions, you can take the following actions:

- Find and manipulate groups of records
- Modify, update, and erase VAX DBMS data
- Create reports
- Plot graphs
- Relate data stored in RMS files and relational databases with data stored in VAX DBMS databases
- Use FMS, TDMS, or DECforms forms to access and change VAX DBMS data

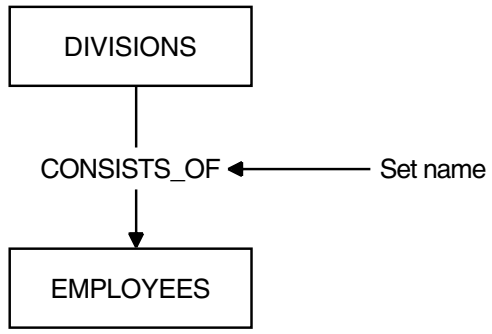
Using DATATRIEVE, you can access individual records, groups of records, and records related according to information in a VAX DBMS set. A **set** is a VAX DBMS data structure that establishes a relationship among records.

For example, the sample PARTS database contains a set named CONSISTS_OF. This set, illustrated in Figure 4–1, contains pointers that identify which employees (in the EMPLOYEES domain) work for which divisions (in the DIVISIONS domain).

A single record in the DIVISIONS domain is called a **single record occurrence**. A single record occurrence is the data of a single record in the database. For example, the SOFTWARE division is a single record occurrence.

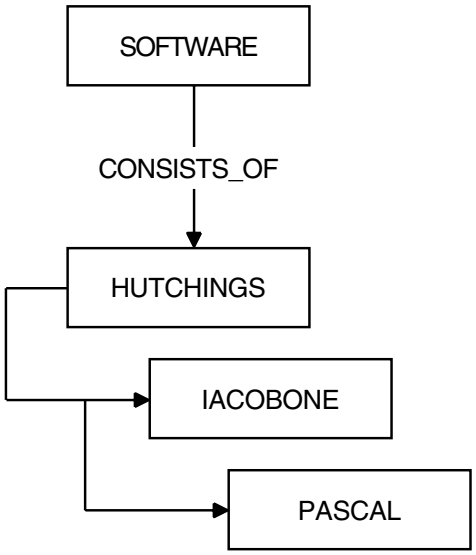
This single record occurrence is related to another set of records by the information in a **single set occurrence** of the set. A single set occurrence contains one owner record occurrence and zero or more member record occurrences.

Figure 4–1 VAX DBMS Set CONSISTS_OF



In a VAX DBMS database, relationships between records are always described in sets. For example, the single owner record occurrence of the SOFTWARE division is connected to employee records by the information in a single occurrence of the set CONSISTS_OF. Figure 4–2 shows this relationship.

Figure 4–2 Single Set Occurrence



The information that employees HUTCHINGS, IACOBONE, and PASCAL work for the SOFTWARE division is described in the single set occurrence of the CONSISTS_OF set.

In addition to manipulating data, a big advantage of using DATATRIEVE with VAX DBMS data is that you can format that data by using DATATRIEVE graphics and the Report Writer. See the chapter on using Plots and the chapter on using the Report Writer in the *VAX DATATRIEVE User's Guide* for information on using these features of DATATRIEVE. DATATRIEVE provides specialized syntax to work with VAX DBMS records and the following set relationships:

- Two define commands (DEFINE DATABASE and extensions to DEFINE DOMAIN)
- An extension to the READY command (READY database-path-name)
- Two SHOW commands (SHOW SETS and SHOW DATABASES)
- Three clauses that extend the record selection expression and refer to records as participants in sets (MEMBER, OWNER, and WITHIN)
- An extension to the STORE statement (CURRENCY clause)

- Three statements to work with sets (CONNECT, DISCONNECT, and RECONNECT)
- Two database commands (COMMIT and ROLLBACK)

4.2 Defining a Database: The DEFINE DATABASE Command

To access VAX DBMS records and sets, you must define the VAX DBMS database in DATATRIEVE terms. Using the DEFINE DATABASE command, you create a DATATRIEVE database instance for the VAX DBMS database.

The DEFINE DATABASE command performs the following functions:

- Defines a pointer to the VAX DBMS database and gives the database a unique DATATRIEVE name
- Identifies the VAX DBMS schema, subschema, and root file you want to access and associates it with the DATATRIEVE database name
- Stores the new database definition in the CDD/Repository dictionary

You must specify the name of a subschema, its schema, and the associated database root file, in that order. For more information on the DEFINE DATABASE command, see the *VAX DATATRIEVE Reference Manual*.

The following example defines a database instance. The dictionary path name of the schema is CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS.PARTS. The name of the subschema is DTR_SUBSCHEMA, and the root file is DTR$LIBRARY:DTRPARTDB.

```
DTR> DEFINE DATABASE PARTS_DB
DFN>     USING SUBSCHEMA DTR_SUBSCHEMA
DFN>     OF SCHEMA PARTS
DFN>     ON DTR$LIBRARY:DTRPARTDB;
DTR>
```

4.3 Accessing the Database

After you define a database, you can access it in one of two ways:

- Ready it directly with the READY database-path-name command.
- Define domains for each record in the database with the DEFINE DOMAIN command and then ready each domain.

With the first method, you can ready all the records associated with that database using a single READY command. The advantage of using this method is that you need not define a domain for each record in the VAX DBMS database. The syntax is simpler and readying a database may be faster than readying the separate domains.

With the second method, in addition to defining the database, you need to define a domain for each VAX DBMS record. The advantage of using this method is that you can use view domains, and you can associate a form definition with a VAX DBMS record in the domain definition.

The next sections discuss these two methods.

4.3.1 Readying an Entire Database Directly

When you ready a database directly, you can access all the data from the following sources:

- All or selected VAX DBMS records associated with that database in the VAX DBMS subschema definition
- The sets in which those records participate

For example, if you ready PARTS_DB, you can access all the records in the PART subschema definition. When you enter the SHOW READY command, you see all the records made available by the READY command.

```
DTR> READY PARTS_DB
DTR> SHOW READY
Ready sources:
   CLASS:  Record, DBMS, shared read
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   PART:  Record, DBMS, shared read
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   COMPONENT:  Record, DBMS, shared read
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   VENDOR:  Record, DBMS, shared read
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   SUPPLY:  Record, DBMS, shared read
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   QUOTE:  Record, DBMS, shared read
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   EMPLOYEE:  Record, DBMS, shared read
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   DIVISION:  Record, DBMS, shared read
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
No loaded tables.
DTR>
```

For more information on the READY command, see the *VAX DATATRIEVE Reference Manual*.

4.3.2 Defining and Readying VAX DBMS Domains

You can also access VAX DBMS data by defining a domain for each VAX DBMS record you require. DATATRIEVE automatically maps database records to these domains.

In the following example, you take the following actions:

- Define a VAX DBMS domain (PART_S)
- Identify the associated VAX DBMS record type (PART)
- Identify the DATATRIEVE instance (PARTS_DB) of the VAX DBMS database

```
DTR> DEFINE DOMAIN PART_S USING
CON>   PART
CON>   OF DATABASE PARTS_DB;
DTR>
```

For the full syntax of DEFINE DOMAIN command VAX DBMS domains, see the *VAX DATATRIEVE Reference Manual*.

The following examples define domains for all the records (in addition to the PART record defined previously) in the VAX DBMS PARTS database. The domains are CLASSES, COMPONENTS, DIVISIONS, EMPLOYEES, QUOTES, SUPPLIES, VENDORS.

Note that you establish a domain definition for each record type you want to access in the PARTS_DB database. (Blank lines separate the definitions for clarity.)

```
DTR> DEFINE DOMAIN CLASSES USING
CON>   CLASS OF DATABASE PARTS_DB;
DTR>
DTR> DEFINE DOMAIN QUOTES USING
CON>   QUOTE OF DATABASE PARTS_DB;
DTR>
DTR> DEFINE DOMAIN SUPPLIES USING
CON>   SUPPLY OF DATABASE PARTS_DB;
DTR>
DTR> DEFINE DOMAIN VENDORS USING
CON>   VENDOR OF DATABASE PARTS_DB;
DTR>
DTR> DEFINE DOMAIN EMPLOYEES USING
CON>   EMPLOYEE OF DATABASE PARTS_DB;
DTR>
DTR> DEFINE DOMAIN COMPONENTS USING
CON>   COMPONENT OF DATABASE PARTS_DB;
DTR>
DTR> DEFINE DOMAIN DIVISIONS USING
CON>   DIVISION OF DATABASE PARTS_DB;
DTR>
```

As with the database or RMS domains, you must ready VAX DBMS domains before you can access data from those domains. When you use the READY command on a VAX DBMS domain, you get access not only to a record type, but also to the sets in which the record type participates.

For example, the following command readies the domains EMPLOYEES and DIVISIONS:

```
DTR> READY EMPLOYEES, DIVISIONS
```

It provides access to all the data from the following sources:

- The records associated with those domains (EMPLOYEE and DIVISION records)
- The sets in which those records participate (MANAGES, CONSISTS_OF, ALL_EMPLOYEES)

Note

You must ready all participants in a set (both owner and member domains) to access data identified by that set. The description of all the arguments to the READY command is in the *VAX DATATRIEVE Reference Manual*.

4.3.3 Results of the READY Command

When you ready a VAX DBMS database, a VAX DBMS record or a VAX DBMS domain, all the realms in which a VAX DBMS record participates are automatically readied. A **realm** is one or more schema areas and is defined in a subschema. A realm lets you restrict or grant access to sections of a database.

Ask your system administrator for a listing of the realms for the subschema you are using. The realms are ultimately associated with **storage areas**, VAX DBMS units contained in single files. It is actually these files that are opened, through a READY command that readies at least one of the domains in that file. You can ready a database or a domain as SHARED, PROTECTED, or EXCLUSIVE. The READY options you choose determine the level of VAX DBMS locking. Locking affects both you and other active users. You can also specify how you access the domains or records for READ, WRITE, MODIFY, or EXTEND access.

See the *VAX DATATRIEVE Reference Manual* and the VAX DBMS documentation for information on access options and modes.

When used with VAX DBMS domains, VAX DBMS records, or the entire VAX DBMS database, the READY command has the following effects:

- Each specified domain or VAX DBMS record is readied with the requested access.
- The default access mode is SHARED READ. If you do not specify EXCLUSIVE or PROTECTED access, DATATRIEVE always readies for SHARED access.
- When you ready more than a single domain or VAX DBMS record in the realm and enter a SHOW READY command, you see the VAX DBMS records and domains with the access option and mode you specified in the READY command. For example:

```
DTR> READY PARTS_DB USING SUPPLY, VENDOR EXCLUSIVE WRITE

DTR> SHOW READY
Ready sources:
   VENDOR:  Record, DBMS, exclusive write
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   SUPPLY:  Record, DBMS, shared read
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
```

  Note, however, other users' access to those records or domains is limited by the most restrictive access you specify in a READY command. This restrictive access applies until you ready the VAX DBMS domain or record again, or until you enter a final FINISH on the database. (The DATATRIEVE FINISH command ends access to VAX DBMS domains and records and executes a VAX DBMS COMMIT. See Section 4.10.4 on writing changes to the database for more information.)

  Thus, DATATRIEVE applies the access mode and option of the most restrictive domain or VAX DBMS record in the realm to all domains in that realm.

  In the preceding example, therefore, as long as VENDOR is readied with EXCLUSIVE WRITE, the access applied to SUPPLY is also EXCLUSIVE WRITE.

- To change access to a domain or a VAX DBMS record, you must ready the domain or VAX DBMS record again. Because DATATRIEVE will ready other domains or VAX DBMS records in the realm again if the new access is more restrictive, changing access to one domain or record may result in the entire realm being readied again.

- If you print, modify, or store data into a domain, record, or database, DATATRIEVE allows you to re-ready with a more restrictive access only after you enter a COMMIT, a FINISH on the record or domain, or a ROLLBACK statement. DATATRIEVE displays a message indicating it is releasing the collections automatically to allow such a re-ready.

EXCLUSIVE WRITE access lets you store and modify records but prevents other users from even retrieving records from the domain until you end your access to it or ready it again with a different access mode.

You can use the SHOW command to see the following information:

- The database records or domains that are readied (SHOW READY)
- The fields in the records that are readied (SHOW FIELDS)
- The sets that are made accessible, plus the access mode and access option with which you readied the database or domain (SHOW SETS)

4.3.3.1 The SHOW FIELDS Command

The following SHOW FIELDS command displays the fields of the record types EMPLOYEE and DIVISION:

```
DTR> SET DICTIONARY CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS
DTR> READY EMPLOYEES, DIVISIONS
DTR> SHOW FIELDS
DIVISIONS
   DIVISION
      DIV_NAME  <Character string>
EMPLOYEES
   EMPLOYEE
      EMP_ID (ID)       <Number>
      EMP_LAST_NAME (LAST_NAME) <Character string>
      EMP_FIRST_NAME (FIRST_NAME)       <Character string>
      EMP_PHONE (PHONE_NUMBER)  <Number>
      EMP_LOC (LOCATION)        <Character string>
No global variables are declared.
```

4.3.3.2 The SHOW SETS Command

Using the SHOW SETS command, you can see the sets in which the VAX DBMS domains and record types participate. If you defined domains, the SHOW SETS command identifies the domains that participate in the sets. Otherwise, it shows the VAX DBMS records that participate in each set. (In DATATRIEVE, because you can define a domain for each VAX DBMS record, a SHOW SETS command indicating the domains implicitly shows the VAX DBMS records that participate in the sets.)

For example, two domains, EMPLOYEES and DIVISIONS (and their associated records, EMPLOYEE and DIVISION), participate in the set MANAGES. When you have defined a domain and you enter a SHOW SETS command, you see the domain names you defined. The EMPLOYEES domain represents the EMPLOYEE record and the DIVISIONS domain represents the DIVISION record.

```
DTR> SHOW SETS
    Set: CONSISTS_OF
         Owner: DIVISIONS
         Member: EMPLOYEES, manual optional

    Set: MANAGES
         Owner: EMPLOYEES
         Member: DIVISIONS, automatic optional

    Set: ALL_EMPLOYEES
         Member: EMPLOYEES, automatic fixed
```

The terms member and owner refer to set characteristics that are described in the Section 4.5.1.1 on finding and printing VAX DBMS records.

The terms automatic, manual, optional, and fixed are discussed in Section 4.10.2 on erasing and disconnecting records and sets.

Section 4.5.1.1 on finding data in a VAX DBMS database discusses sets further.

4.4 Forming a DATATRIEVE Query

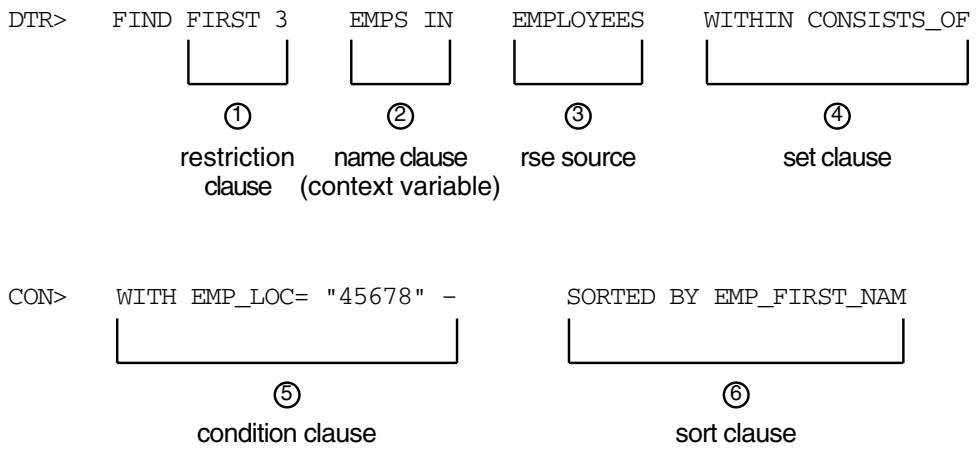
Once you have defined and readied your VAX DBMS database, you may want to take one of the following actions:

- Find and display a record or records from that database
- Find data related to that record through VAX DBMS set relationships

DATATRIEVE provides you with an English-like query language for VAX DBMS databases.

This section discusses how to form a simple DATATRIEVE query; it is for VAX DBMS users unfamiliar with DATATRIEVE. The next section discusses how to add the DATATRIEVE clauses (WITHIN, MEMBER, and OWNER) to find records related by a set. In DATATRIEVE, you can access a single record at a time or you can work with a group of records. Using the DATATRIEVE record selection expression (RSE), you can query, print, modify, or store data in a VAX DBMS database. A DATATRIEVE RSE identifies and limits the records you want to include in a record stream, as shown in Figure 4–3.

Figure 4–3 The Parts of an RSE



An RSE consists of the following components:

1 An optional restriction clause (ALL or a number) — to tell DATATRIEVE how many records to include in a record stream.

2 An optional name clause — to qualify or provide a name for the record source. You can use this name later to identify the record stream and to access records in that stream. The clause is also called a context variable when you use it to name a record stream or modify field names in a FOR loop.

3 A required record source — to identify an RMS, relational, or VAX DBMS domain name, a view or network domain, a VAX DBMS record name, a relation name, a collection, or a list.

4 A VAX DBMS set clause (WITHIN, OWNER, or MEMBER) — to identify a set name and the relationship to that set.

5 An optional selection condition (WITH Boolean expression) — to tell DATATRIEVE what values to look for in a record.

6 An optional sort clause (SORTED BY) — to sort the record stream in the order you specify.

The format for the RSE follows. Note that it includes the components just identified as well as the following two additional elements.

- An optional relational clause (CROSS) — to combine data from more than one domain

- An optional reduction clause (REDUCED TO) — to retain only unique values

```
[ FIRST n ]
[ ALL     ]  [context-var IN] rse-source

[CROSS [context-var IN] rse-source [OVER field-name]] [ . . . ]

[WITH boolean-expression] [REDUCED TO reduce-key [, . . . ]]

[SORTED BY sort-key [, . . . ]]
```

The rse-source has the following format:

```
{ domain-name
  collection-name
  list
  rdb-relation-name
  DBMS-record-name [ { MEMBER } [OF] [context-name.]set-name ] }
                   [ { OWNER  }                              ]
                   [ { WITHIN }                              ]
```

Note that the domain name in the preceding syntax includes VAX DBMS domains as well as relational, RMS, view, and remote domains. Note that the MEMBER, OWNER, and WITHIN set-name syntax is used only with a VAX DBMS domain name or VAX DBMS record name.

For a more complete explanation of RSE syntax and an example of the CROSS clause, see the *VAX DATATRIEVE Reference Manual* and the chapter on writing record selection expressions in the *VAX DATATRIEVE User's Guide*.

4.5 Forming a DATATRIEVE/VAX DBMS Query

Typically, you can use two methods to form a DATATRIEVE query that accesses VAX DBMS data:

- The FIND and SELECT statements, to establish a collection of records and point to a specific record from that collection
- The FOR statement, which uses an RSE to form a temporary record stream

These statements are discussed and illustrated in the next sections.

4.5.1 Forming a DATATRIEVE Collection of VAX DBMS Records

You can access VAX DBMS data in DATATRIEVE by forming a **collection**. A collection is a group of records that you can access until you take one of the following actions:

- Form a new collection (unless you assign a name to the new or old collection)
- Remove the collection with the RELEASE command
- Finish the domain that owns the collection using the FINISH command

4.5.1.1 Using the FIND Statement

You form a collection using the DATATRIEVE FIND statement. You can create collections from a readied VAX DBMS domain or from a VAX DBMS record in a database you have readied through DATATRIEVE.

When you form a collection, DATATRIEVE gives that collection the name CURRENT. You can access this collection with the name CURRENT, or you can assign an additional name for a collection and use this name to access the collection. You use the name clause as illustrated in the RSE format to name a collection.

If you name a collection, it is not deleted when you form another collection. Therefore, you can have several named collections of records available at any time in DATATRIEVE.

The following example forms a collection from the VAX DBMS record EMPLOYEE, which was readied when you readied the PARTS_DB database:

```
DTR> READY EMPLOYEE
DTR> FIND FIRST 5 EMP IN EMPLOYEE      1
[5 records found]
DTR> FIND EMP45 IN EMP WITH EMP_LOC = "45678"      2
[2 records found]
DTR> SHOW COLLECTIONS      3
Collections:
       EMP45      (CURRENT)
       EMP

DTR> PRINT CURRENT      4

                                        Phone
Ident  Last Name----------- First Name  Number   Loc

   998 HILL                OLA          124567 45678
 22234 HORTI               BRUCE        124567 45678

DTR> PRINT EMP      5
                                        Phone
Ident  Last Name----------- First Name  Number   Loc
```

```
 75624 FRASER              BOB         8902345 23456
 12333 HOFFMAN             MIKE        4568901 89012
   998 HILL                OLA          124567 45678
 22234 HORTI               BRUCE        124567 45678
 77777 PASCAL              RICHARD     4568901 89012
```

1 Form a collection of the first five employees and assign the name EMP to that collection.

2 Form a second collection by limiting the EMP collection to those employees with EMP_LOC = 45678 and assign the name EMP45.

3 Show that DATATRIEVE retains information about both collections EMP45 (also CURRENT) and EMP.

4 Print the CURRENT collection.

5 Print the EMP collection.

Note that you can still access the records in the EMP collection after you create a second collection with the FIND statement. As long as you name the collections, DATATRIEVE retains them when you create new collections.

If you name a collection, you can use this name as a context variable in an RSE to modify a record stream. Later sections contain examples showing how you can do this.

4.5.1.2 Using the SELECT Statement

After you form a collection, you use the SELECT statement to choose a record from that collection. This selected record is the target of other DATATRIEVE statements such as PRINT, MODIFY, and ERASE. The SELECT statement establishes VAX DBMS currency for a record.

You can specify the record you want to access either by using an integer (SELECT 5 gets the fifth record of the collection) or by using such syntax as FIRST, NEXT, LAST, and so on. The following example shows how the SELECT statement works:

```
DTR> SELECT      1
DTR> PRINT

                                    Phone
Ident  Last Name----------- First Name  Number   Loc
   998 HILL                 OLA          124567 45678


DTR> SELECT 2 EMP      2
DTR> PRINT

                                    Phone
Ident  Last Name----------- First Name  Number   Loc
```

```
 12333 HOFFMAN              MIKE        4568901 89012
```

1 Selects the first record in the EMP45 collection, which is also the CURRENT collection. When you do not specify a collection name, the SELECT statement defaults to the first record in the CURRENT collection.

2 Selects the second record in the EMP collection.

For a complete discussion of the FIND and SELECT statements, see the FIND and SELECT statements in the *VAX DATATRIEVE Reference Manual*.

4.5.2 Forming a Record Stream of VAX DBMS Records

The FOR statement differs from using collections in that it creates a temporary record stream that DATATRIEVE knows about only while it is executing that statement. It places locks on fewer VAX DBMS records than a FIND statement.

DATATRIEVE can access each record in that stream one by one, displaying, printing, or modifying each record according to your specifications. As each record is accessed, it becomes **current**.

The following example shows how to access VAX DBMS records using the FOR statement:

```
DTR> READY EMPLOYEES
DTR> FOR EMPLOYEES WITH EMP_LOC = "45678"      1
[Looking for statement]
CON> PRINT EMPLOYEE      2

                                        Phone
Ident  Last Name----------- First Name  Number   Loc

   998 HILL                 OLA          124567 45678
 22234 HORTI                BRUCE        124567 45678
 11141 SCHATZEL             BETH         124567 45678
 12322 THOMPSON             STEPHEN      124567 45678
 12345 HUNTER               BUTCH        124567 45678
          .
          .
          .

DTR> SHOW CURRENT      3
A current collection has not been established.
```

1 Form a temporary DATATRIEVE record stream of employee records with EMP_LOC = 45678.

2 Display only those employees in the record stream.

3 Try to show the CURRENT collection, which does not exist.

4.6 Forming a DATATRIEVE/VAX DBMS Query of Data Related by Sets

In the previous section, you used simple DATATRIEVE queries to access information from a single VAX DBMS record or domain.

Note that DATATRIEVE also makes set information available when you ready a domain. In the following sections, you access information from both the data in a single record and data in records related through information in a set.

As with VAX DBMS data in a single record, you can use the FIND and SELECT syntax to form collections, or you can use the FOR syntax to create a temporary record stream.

Note two important concepts about using DATATRIEVE statements to access VAX DBMS data related by set information:

- You must use either the FIND and SELECT or the FOR statement to establish the single record context, called currency, in VAX DBMS. VAX DBMS needs this single record context to find related records and sets.
- You must specify a VAX DBMS set name to identify the sets in which a record participates, unless you use the Context Searcher. (See Section 4.6.4 on using the SET SEARCH command to access sets.)

When you access information through VAX DBMS sets, you access:

- First, a particular record from a domain or record (for example, a department from the DIVISIONS domain)
- Second, data related to that record from other VAX DBMS domains or records through a set (for example, employees from the EMPLOYEES domain related through the set CONSISTS_OF)

The examples in this section use the set CONSISTS_OF. It represents the relationship between a department in an organization (DIVISIONS domain) and the employees that make up that department (EMPLOYEES domain). Figure 4–1 shows this relationship.

The following sections illustrate DATATRIEVE syntax you use to access information in sets.

4.6.1 Forming Collections of VAX DBMS Set Data

As with data in a single VAX DBMS record, you can form a collection of VAX DBMS data related through set information. You can then access those records by the collection name.

In the following example, for instance, you form a collection (DIV) from the DIVISIONS domain. You form a second collection (EMP) of employee records. The employee records in EMP collection are related to the selected record from the DIV collection. DATATRIEVE now knows about both collections, DIV and EMP. You can print records from both of these collections.

```
DTR> SET DICTIONARY CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS
DTR> READY DIVISIONS, EMPLOYEES

                    1
DTR> FIND DIV IN DIVISIONS WITH DIV_NAME = "SOFTWARE"
[1 record found]
DTR> SELECT

                    2
DTR> FIND EMP IN EMPLOYEES WITHIN CONSISTS_OF
[3 records found]
DTR> PRINT DIV_NAME, EMPLOYEE OF EMPLOYEES

                                                 Phone
Division Name---Ident  Last Name----First Name  Number   Loc

SOFTWARE         23451 HUTCHINGS      BRUCE       2346789 67890
SOFTWARE         43215 IACOBONE       ANTHONY      124567 45678
SOFTWARE         77777 PASCAL         RICHARD     4568901 89012
```

1 Use the FIND statement to create a collection of records from the DIVISIONS domain. The SELECT statement identifies a single record occurrence and establishes context (currency in VAX DBMS) with the set information.

2 Use the FIND statement again to establish a collection of employee information from the EMPLOYEES domain. By using the WITHIN set-name syntax, you identify the set that you want DATATRIEVE to use. This set identifies the employee records related to the SOFTWARE division.

When forming a collection of VAX DBMS set data, you must take the following actions:

- Identify a single record occurrence using FIND and SELECT so that DATATRIEVE can establish context (currency in VAX DBMS) for the related set information.

  In the first part of the example, DATATRIEVE uses the selected single record, SOFTWARE, to establish context for the set information.

- Specify the set that contains pointers relating the data from one domain (or VAX DBMS record) to another.

  In the second part of the example, the WITHIN clause tells DATATRIEVE to look at the single occurrence of the set CONSISTS_OF. The single set occurrence contains pointers that point from the single record occurrence DATATRIEVE currently knows about, the SOFTWARE division, to related records in the domain, EMPLOYEES.

4.6.2 Forming Record Streams of VAX DBMS Set Data

You can use the FOR loop to access the information from two domains. The nested FOR loop in the following example creates two record streams and allows you to access records from each stream:

```
DTR> FOR VID IN DIVISIONS WITH DIV_NAME CONT "VT"      1
CON>    FOR EMP IN EMPLOYEES WITHIN CONSISTS_OF      2
CON>       PRINT VID.DIV_NAME, EMP.EMPLOYEE      3

                                                      Phone
Division Name-------Ident  Last Name------ First Name  Number   Loc

VT100 DEVELOPMENT    65437 FRANK           BEBI        4568901 89012
VT100 DEVELOPMENT    12333 HOFFMAN         MIKE        4568901 89012
VT100 DEVELOPMENT    54332 IGLESIAS        RAFAEL      2346789 67890
VT52  DEVELOPMENT     9867 FLETCHER        BRUCE        124567 45678
VT52  DEVELOPMENT    43221 HYNES           RICH        8902345 23456
```

1 Form a temporary record stream of all records from the DIVISIONS domain with a DIV_NAME that contains the string VT (groups that develop video terminals). Assign that record stream a name (VID).

2 Form a second temporary record stream of the records from the related employee records in the EMPLOYEES domain identified by the set CONSISTS_OF.

3 Print the division name and the related employee information.

Notice that you can name the record stream in the FOR statement (VID) and use that name to qualify a field name (VID.DIV_NAME). In this example, the qualifying name is not necessary to identify fields uniquely.

FOR loops allow you to access records more quickly than FIND statements. Note that when you use a FOR loop, you need not use the SELECT syntax; DATATRIEVE selects a single record each time through the loop. For more information on the FOR loop, see the FOR statement in the *VAX DATATRIEVE Reference Manual*.

4.6.3 Using OWNER and MEMBER Clauses to Identify Sets

In Section 4.6.1, you used the WITHIN clause to identify:

- The domain name (EMPLOYEES) from which you wanted the employee data
- The set name (CONSISTS_OF) that identified the employee record you wanted

The WITHIN clause allows you to specify a set name without having to know if the domain is a member or owner of the set.

There are two additional clauses of the record selection expression that allow you to specify access to records through VAX DBMS sets:

- The MEMBER clause
- The OWNER clause

The OWNER and MEMBER clauses specify whether a record is a member or an owner of a set. In many cases, you can use the WITHIN clause in place of the MEMBER and OWNER clauses. The SHOW SETS command lets you see whether a domain is a member or an owner of a set.

```
DTR> SHOW SETS
    Set: CONSISTS_OF
         Owner: DIVISIONS      1
         Member: EMPLOYEES, manual optional

    Set: MANAGES
         Owner: EMPLOYEES      2
         Member: DIVISIONS, automatic optional

    Set: ALL_EMPLOYEES      3
         Member: EMPLOYEES, automatic fixed
```

The SHOW SETS command in this example provides the following information:

1 The EMPLOYEES domain (in VAX DBMS, the EMPLOYEE record) owns the set MANAGES. The DIVISIONS domain (in VAX DBMS, the DIVISION record) is a member of the set MANAGES.

2 The DIVISIONS domain (in VAX DBMS, the DIVISION record) owns the set CONSISTS_OF. The EMPLOYEES domain (in VAX DBMS, the EMPLOYEE record) is a member of the set CONSISTS_OF.

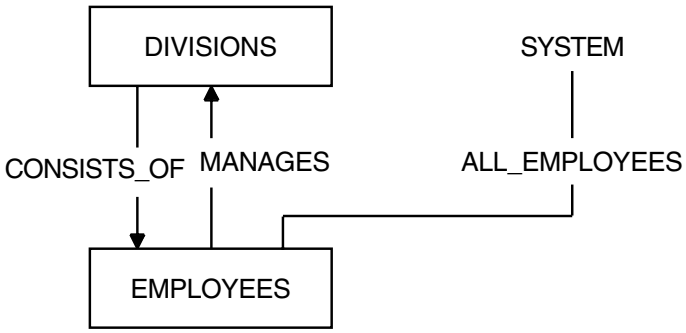
3 The EMPLOYEES domain is also a member of the ALL_EMPLOYEES set, a system-owned set.

Note

A system-owned set is a set owned by VAX DBMS instead of a user-defined record. System-owned sets have only one occurrence in the database and are used for relationships with a large number of member occurrences, or as entry points into a database. For more information on system-owned sets, see the VAX DBMS documentation.

Figure 4–4 shows all these set relationships.

Figure 4–4 Set Relationships in Sample VAX DBMS Database



The following sections show you how to use the MEMBER and OWNER clauses to access VAX DBMS records.

4.6.3.1 The MEMBER Clause

The MEMBER clause lets you access the member records of a set. Conceptually, you are telling DATATRIEVE to "look down" from a specified position (determined by a FIND/SELECT or a FOR statement) and find the members of the set that are linked to the selected record. For example, because EMPLOYEES is a member of the set CONSISTS_OF, you can use the MEMBER syntax rather than the WITHIN syntax you used in the previous section.

```
DTR> FIND DIV IN DIVISIONS WITH DIV_NAME = "SOFTWARE"
DTR> SELECT
DTR> FIND EMPLOYEES MEMBER OF CONSISTS_OF
DTR> PRINT ALL DIV_NAME, EMPLOYEE

                                                          Phone
Division Name------- Ident  Last Name----------- First Name  Number   Loc
```

```
SOFTWARE              23451 HUTCHINGS           BRUCE       2346789 67890
SOFTWARE              43215 IACOBONE            ANTHONY      124567 45678
SOFTWARE              77777 PASCAL              RICHARD     4568901 89012
```

4.6.3.2 The OWNER Clause

The OWNER clause tells DATATRIEVE to "look up" from a position in the database, thereby giving you access to the owner record of a set. The OWNER clause operates like the MEMBER clause; the only difference is in the direction that DATATRIEVE looks. In the following example, you want to know in which division employee Richard Pascal works. DIVISIONS is the owner of the set CONSISTS_OF.

```
DTR> FOR EMP IN EMPLOYEES WITH EMP_LAST_NAME= "PASCAL"      1
CON>  PRINT ALL EMPLOYEE, DIV_NAME OF
CON>  DIVISIONS OWNER OF CONSISTS_OF      2
DTR>

                                        Phone
Ident  Last Name----------- First Name  Number   Loc  Division Name---

77777 PASCAL               RICHARD     4568901 89012   SOFTWARE

DTR>
```

1 Form a record stream with the single employee record.

2 Use the OWNER clause with the CONSISTS_OF set to find the department in which the employee works.

4.6.4 Using the SET SEARCH Command to Access Sets

As a DATATRIEVE user, you can use the SET SEARCH command to establish context for list fields in records or for fields in hierarchical views.

In addition, as a DATATRIEVE user accessing a VAX DBMS database, you can use the SET SEARCH command to search for records related by set information. You use this command in place of the WITHIN, OWNER, or MEMBER clauses.

The SET SEARCH command instructs the DATATRIEVE Context Searcher to choose the shortest route between VAX DBMS record types when executing a PRINT statement. DATATRIEVE resolves the context for you so that you need not specify the set relationship.

The following example executes a SET SEARCH statement and prints the related division data without requiring you to specify the set CONSISTS_OF:

```
DTR> SET SEARCH
DTR> FIND FIRST  1 EMPLOYEES
[1 record found]
DTR> SELECT
DTR> PRINT DIV_NAME
Not enough context.  Some field names resolved by Context Searcher.

Division Name-------

ENG STOCKROOM

DTR>
```

The following example walks through all occurrences of the set type CLASS_PART, instructing DATATRIEVE to display on your terminal only the class code number, part identification numbers, and part descriptions of records owned by CLASSES:

```
DTR> SET SEARCH
DTR> PRINT CLASS_CODE, PART_ID, PART_DESC OF CLASSES
Not enough context. Some field names resolved by Context Searcher.

        Part
Code    Number    -----------------Part Description-----------------

 BR    BR-1234-56 LA34
       BR-3467-91 LA120
       BR-8901-23 LA36
 BT    BT-0456-78 VT52
       BT-1634-56 VT100
 BU    BU-0345-67 TERMINAL TABLE VT52
       BU-1045-68 FREE-STANDING FRAME ASSEMBLY
             .
             .
             .
       CG-3256-40 VT100 KEYBOARD KED ASSY
       CG-3454-38 PLASTIC KEY NUM. STYLE C
       CG-4567-89 PLASTIC KEY ALPHA. STYLE A
       CG-8767-78 VT100 SCREEN
       CG-8901-23 VT100 HOUSING
       CG-9435-61 KEY BASES
       CG-9562-13 VT100 NUMERIC KEY CAP SET
```

In this case, the Context Searcher found many records. You can find the owners and members of database sets by using the PRINT and SET SEARCH statements. It is important to remember that SET SEARCH always takes the shortest route in a set structure and, therefore, might not return the right answer.

Note that if you did not use the Context Searcher in the preceding example, the full DATATRIEVE query would use an inner print list. For the statement PRINT CLASS_CODE, PART_ID, PART_DESC OF CLASSES, you would need the following query:

```
DTR> PRINT CLASS_CODE, ALL PART_ID, PART_DESC OF
CON> PART_S MEMBER OF CLASS_PART OF CLASSES
DTR>
```

The expanded statement includes an inner print list.

The following example readies two more domains and instructs DATATRIEVE to display the name and description of the parts supplied by the vendor with the name QUALITY COMPS.

```
DTR> SET SEARCH
DTR> READY SUPPLIES, VENDORS
DTR> PRINT VEND_NAME, PART_DESC OF
CON>  VENDORS WITH VEND_NAME = "QUALITY COMPS"
Not enough context. Some field names resolved by Context Searcher.

--------------Vendor Name---------------

QUALITY COMPS
VT100 KEYBOARD ASSY
NUMERIC KEYPAD FRAME
VT52 HOUSING
```

This PRINT statement resulted in the display of all parts associated with the specified vendor. The PRINT statement is equivalent to the following:

```
DTR> PRINT VEND_NAME, ALL ALL PART_DESC OF PART_S OWNER OF
CON> PART_INFO OF SUPPLIES MEMBER OF VENDOR_SUPPLY OF VENDORS WITH
CON> VEND_NAME = "QUALITY COMPS"
```

4.7 Finding Data from Two or More Domains

In previous sections, you found records from several domains by first finding a single record in one domain and then related data in a second domain through a set relationship.

DATATRIEVE provides several ways for you to access records from two or more domains, in addition to using the simple DATATRIEVE queries shown in the previous sections. The following methods become particularly important when the data you want may reside in more than two domains:

- Combining the MEMBER and OWNER clauses to “walk the VAX DBMS sets”
- Using the CROSS clause of the RSE to join the data from several records or domains
- Defining a domain called a VIEW domain that lets you form simple queries, while keeping the complex set relationships in the domain definition

The example in the following sections uses the VENDORS, PART_S, and SUPPLIES domains. The data in the VENDORS and PART_S domains has what VAX DBMS calls a many-to-many relationship. For example:

- A single vendor might supply many different parts.
- A single part might be supplied by many different vendors.

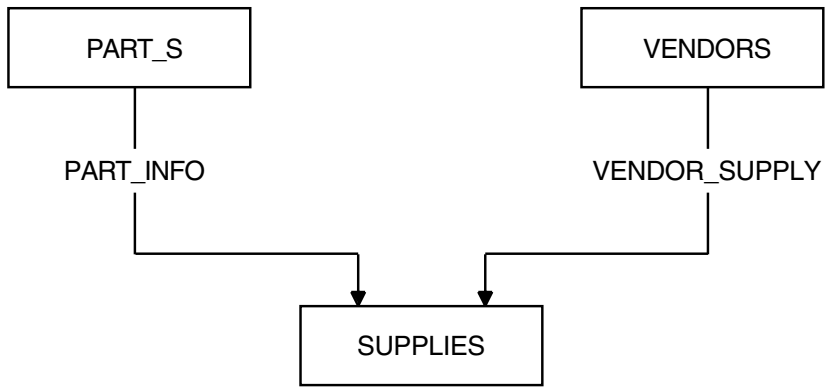
In a VAX DBMS database, a direct relationship cannot exist between records (DATATRIEVE domains) that have a many-to-many relationship. If, for example, you attempt to select a particular vendor from the VENDORS domain and then try to display an associated part from the PART_S domain, DATATRIEVE gives you an error message indicating you have not established the correct context for parts, as in the following example:

```
DTR> READY VENDORS, SUPPLIES, PART_S
DTR> FIND VENDOR WITH VEND_NAME = "QUALITY COMPS"
([1 record found])
DTR> SELECT
DTR> FIND PART_S WITHIN VENDOR_SUPPLY
Set "VENDOR_SUPPLY" is undefined or used out of context.
DTR>
```

To relate the parts and vendor data, you must go through a third domain or record that is owned by both the PARTS and VENDORS domains.

Figure 4–5 shows the VAX DBMS representative of this many-to-many relationship.

Figure 4–5 VAX DBMS Set Relating Three Domains



4.7.1 Walking the Sets

Suppose you want the following information that involves access to PARTS, VENDORS, and SUPPLIES:

- The names for a specific vendor (VENDORS domain)
- The types of parts that vendor supplies (PARTS domain)
- Delivery lag time for that part (SUPPLIES domain)

The following example shows how to do this:

```
DTR> SET DICTIONARY CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS
DTR> READY VENDORS, PART_S, SUPPLIES      1
DTR>
DTR> FIND VENDORS WITH VEND-NAME = "QUALITY COMPS"      2
[1 record found]
DTR> SELECT
DTR> PRINT VEND_ID, VEND_NAME

 Vendor
   ID    --------------Vendor Name---------------

55789012  QUALITY COMPS

DTR>
DTR> FIND SUP IN SUPPLIES MEMBER OF VENDOR_SUPPLY      3
[3 records found]
DTR> PRINT SUP
No record selected, printing whole collection.

Rtng Type  Lag Time

 F   MEMO 4-6 WEEKS
 0   REPR 1-2 MONTHS
 F   WSUP 7-8 WEEKS

DTR> FOR SUP      4
CON>  PRINT PART_ID, PART_DESC OF      5
CON>        PART_S OWNER OF PART_INFO

     Part
    Number   -----------------Part Description-----------------

CG-3161-34 VT100 KEYBOARD ASSY
CG-1052-00 NUMERIC KEYPAD FRAME
CG-0956-78 VT52 HOUSING
DTR>
```

1 Ready all three VAX DBMS domains. These READY commands also ready the set relationships among the domains.

2 Find a vendor (for example, the company called QUALITY COMPS).

3 Find the parts supplied by QUALITY COMPS. Because you cannot directly access part information in the PART_S domain from the VENDORS domain, you must go through the SUPPLIES domain. Find all the member records in SUPPLIES related to the selected VENDOR record.

4 Form a FOR loop that establishes a single record context for each record in the SUPPLIES domain. Note that when you use a FOR loop, you do not use the SELECT statement.

5 Print the three parts associated with that SUPPLIES record by using the OWNER clause to find related parts through the PART_INFO set.

Notice that you must continue to provide a single record context for DATATRIEVE with either the FIND and SELECT statements or the FOR loop.

4.7.2 Using the CROSS Clause

You can combine records from several VAX DBMS domains (or VAX DBMS records readied by the READY command for databases) with the CROSS clause from the record selection expression. The CROSS clause lets you compare and combine records from two or more sources into a single record stream. It forms temporary relationships between records stored in different domains (and lets you treat the data as though it were derived from one domain or record). You can also combine data from VAX DBMS records with relational or RMS records by using a CROSS clause.

In the following example, you use the three domains from the previous section. Using the CROSS clause, you combine several domains in one collection and write queries against that collection.

```
DTR> FIND QUAL_PART IN VENDORS WITH VEND_NAME = "QUALITY COMPS"
[1 record found] 1
DTR> FIND QUAL_PART CROSS    2
CON> SUPPLIES MEMBER VENDOR_SUPPLY  CROSS
CON> PART_S OWNER PART_INFO
[3 records found]
DTR> PRINT ALL VEND_ID, PART_ID, PART_DESC,SUP_TYPE    3

Vendor      Part
   ID       Number  -----------------Part Description---------Type

55789012 CG-3161-34 VT100 KEYBOARD ASSY                        MEMO
55789012 CG-1052-00 NUMERIC KEYPAD FRAME                       REPR
55789012 CG-0956-78 VT52 HOUSING                               WSUP
```

1 Form a named collection (QUAL_PART) of the vendor record with a vendor name of QUALITY COMPS.

2 Use the CROSS clause to join this vendor collection with the two other domains. You can do this with a FIND or FOR statement that uses two CROSS clauses.

3 Display the vendor and all the parts made by that vendor from the VENDORS and PART_S domains.

4.7.3 Using View Domains

If you define domains for each VAX DBMS record, you can use DATATRIEVE view domains to access data in those records. You cannot form views of VAX DBMS records readied by the READY command for databases.

You can define a simple DATATRIEVE view domain to see a subset of fields from a single domain. The following view, for example, lets you define a view domain that accesses only three fields in the PART_S domain. This view is further defined by the Boolean expression that limits the records to those with PART_SUPPORT = "FS".

```
DTR> DEFINE DOMAIN VIEW_PARTS_PUBLIC of PART_S USING
DFN> 01 PARTV OCCURS FOR PART_S WITH PART_SUPPORT = "FS".
DFN>     03 PART_ID FROM PART_S.
DFN>     03 PART_DESC FROM PART_S.
DFN>     03 PART_PRICE FROM PART_S.
DFN> ;
DTR>
```

4.7.3.1 Hierarchical Views

You can also use view domains to combine records from several VAX DBMS domains. A view domain that describes data from more than a single domain and does not use a CROSS clause is called a hierarchical view.

A hierarchical view domain, unlike a CROSS clause (which also combines data from two or more domains), lets you define the relationship of records from several domains and store it in the dictionary.

Once you define this relationship, you can display and modify data without having to consider set relationships. You can read and modify selected records from two or more domains as if the data were all in one domain.

Note that because data is not stored in a view, you cannot use a STORE statement with a view domain as your record source.

The following view combines the field DIV_NAME from the domain DIVISIONS and the fields EMP_ID and EMP_LAST_NAME from the EMPLOYEES domain. Note that once you define the relationship, you can ready the domains and print records from those domains.

```
DTR> SHOW DIV_VIEW
DOMAIN DIV_VIEW OF DIVISIONS, EMPLOYEES USING
01 GRP OCCURS FOR DIVISIONS.
  02 DIV_NAME FROM DIVISIONS.
  02 WORKERS OCCURS FOR EMPLOYEES WITHIN CONSISTS_OF.
     04 EMP_ID FROM EMPLOYEES.
     04 EMP_LAST_NAME FROM EMPLOYEES.
;
DTR> READY DIV_VIEW
DTR> PRINT FIRST 5 DIV_VIEW

Division Name------- Ident  Last Name-----------

LA34 DEVELOPMENT      65438 FRATUS
SOFTWARE              23451 HUTCHINGS
                      43215 IACOBONE
                      77777 PASCAL
RM05 DEVELOPMENT      99998 PAYNE
ENG BUILD & TEST      75624 FRASER
                      55675 HORYMSKI
                          0 HUMPHRY
                       9789 MASE
                      66666 PARVIAINEN
VT100 DEVELOPMENT     65437 FRANK
                      12333 HOFFMAN
                      54332 IGLESIAS
```

Note that when you define a view domain with two or more domains, the data is displayed in hierarchical form, unless you use a CROSS clause in the view definition.

In the previous example, the view of two domains displays one occurrence of the field DIV_NAME and a variable number of employees. In DATATRIEVE, to access individual fields in this hierarchical structure, you must use specialized DATATRIEVE syntax for retrieving values from list fields. See the chapter on using hierarchies in the *VAX DATATRIEVE User's Guide* for a complete explanation of this syntax.

4.7.3.2 Flat Views

When you combine a view domain with the relational CROSS clause, it flattens the hierarchical relationships. The following flat view combines fields from the domains VENDORS, SUPPLIES, and PART_S:

```
DTR> DEFINE DOMAIN FLAT_PART_VIEW OF PART_S, VENDORS, SUPPLIES USING
DFN> 01 A OCCURS FOR
DFN>    PART_S CROSS SUPPLIES MEMBER OF PART_INFO CROSS
DFN>    VENDORS OWNER OF VENDOR_SUPPLY.
DFN>         02 PART_ID FROM PART_S.
DFN>         02 SUP_TYPE FROM SUPPLIES.
DFN>         02 VEND_NAME FROM VENDORS.
DFN> ;
DTR>
```

The biggest advantage of defining a flat view is that you can refer to each of the fields more easily than in a hierarchical view. That is, you need not use an inner print list; you can access hierarchical fields as though they belong to a single record. The following example prints the first five records in a flat view and then displays a specific record from the VENDORS domain:

```
DTR> READY FLAT_PART_VIEW
DTR> PRINT FIRST 5 FLAT_PART_VIEW

   Part
  Number   Type --------------Vendor Name---------------

CE-3556-78 MEMO U.S. SEALS
AS-1110-85 REPR HIGH ENERGY CORP
SC-7896-12 REPR EMI TECHNOLOGY INC
CG-8767-78 WSUP ELECTRONIC SUPPLY CO.
CF-4058-32 CALL SYSTEMS HDWE REPS

DTR> PRINT VEND_NAME WITH PART_ID="CF405832"

--------------Vendor Name---------------

SYSTEMS HDWE REPS
```

4.8 Sample Procedures Using VAX DBMS Domains

You can define DATATRIEVE procedures that let you query a VAX DBMS database. A DATATRIEVE procedure is a fixed sequence of commands and statements that you create, name, and store in the dictionary. For almost any series of commands and statements you use repeatedly, you can save yourself time by defining a procedure.

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

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

To define a procedure, you enter the DEFINE PROCEDURE command at the DTR> prompt. DATATRIEVE prompts you with the DFN> prompt to indicate that you can enter a procedure definition. You end the procedure definition with an END_PROCEDURE keyword on a line by itself.

For example, the following procedure searches for a division associated with the employee name you specify:

```
DTR> DEFINE PROCEDURE EMPLOYEE_SEARCH
DFN> READY EMPLOYEES, DIVISIONS      1
DFN> PRINT "This procedure searches to find"
DFN> PRINT "the division associated with"
DFN> PRINT "the employee you specify."
DFN> PRINT SKIP
DFN> FOR EMPLOYEES WITH EMP_LAST_NAME =      2
DFN>  *."the employee's last name in capital letters"
DFN> PRINT DIVISIONS OWNER CONSISTS_OF      3
DFN> COMMIT EMPLOYEES, DIVISIONS
DFN> END_PROCEDURE
DTR>
```

1 Ready the domains EMPLOYEES and DIVISIONS.

2 Form a temporary record stream of the employee record you want. The prompt option (*) lets you specify the employee's name when you execute the procedure.

3 Display the owner of the CONSISTS_OF set of which the employee is a member.

To execute the procedure, enter the following:

```
DTR> :EMPLOYEE_SEARCH

This procedure searches to find
the division associated with
the employee you specify.

Enter the employee's last name in capital letters: ZOTTO

Division Name-------

RK05 DEVELOPMENT
```

For information on the COMMIT statement, see Section 4.10.4 on writing changes to the database. In the following example, you can define a procedure to find a vendor name and all the parts produced by that vendor.

```
DTR> DEFINE PROCEDURE VENDOR_PARTS
DFN> READY VENDORS, SUPPLIES, PART_S      1
DFN> PRINT "This procedure searches to find"
DFN> PRINT "the part associated with "
DFN> PRINT "the vendor you specify."
DFN> PRINT SKIP
DFN> FOR VENDORS WITH VEND_NAME =      2
DFN>   FN$UPCASE(*." Name of Vendor ")
DFN> FOR SUPPLIES MEMBER OF VENDOR_SUPPLY      3
DFN> PRINT ALL PART_ID OF PART_S OWNER OF PART_INFO      4
DFN> FINISH VENDORS, SUPPLIES, PART_S      5
DFN> END_PROCEDURE
DTR>
```

1 Readies the domains VENDORS, SUPPLIES, and PART_S.

2 Uppercases the vendor names you enter following the prompt and finds them in the VENDORS domain.

3 Finds the related records in the SUPPLIES domain.

4 Uses the context from the FOR statement in step 3 to print the related part number from the PART_S domain.

5 Finishes the readied domains.

To execute the procedure, enter the following:

```
DTR> :VENDOR_PARTS
This procedure searches to find
the part number associated with
the vendor you specify.

Enter Name of Vendor: quality comps
    Part
   Number

CG-3162-34
CG-1052-00
CG-0956-78

DTR>
```

4.9 Modifying Individual Fields in a Record

You can modify a field in a VAX DBMS record just as you do in an RMS domain using the DATATRIEVE MODIFY statement. The following example modifies the EMP_PHONE field of the EMPLOYEE record:

```
DTR> READY EMPLOYEES WRITE      1
DTR>
DTR> MODIFY EMP_PHONE OF        2
CON>  EMPLOYEES WITH EMP_ID = "53456"
Enter EMP_PHONE: 5345
DTR>
```

1 Ready the VAX DBMS EMPLOYEES domain for WRITE access.

2 Modify the field EMP_PHONE.

For a complete discussion of the MODIFY statement in DATATRIEVE, see the chapter on modifying data in the *VAX DATATRIEVE User's Guide*. Using the syntax described in that chapter, you can modify all or some fields within a single record occurrence or within a collection of records.

If you change a field that has a VAX DBMS CHECK clause, VAX DBMS checks the value you enter for that field. If the value violates the CHECK clause, DATATRIEVE returns a VAX DBMS error and does not prompt for the field.

In general, modifying a record affects at least the data portion of the record. If, however, you modify a field that is a sort key or a hash key for a set, VAX DBMS automatically reorders the members of the set. See the introductory documentation that accompanies VAX DBMS for more information about modifying sort or hash keys.

4.10 Storing VAX DBMS Records and Modifying Sets

When you add a record to a VAX DBMS database, you can affect other members of the sets in which the new record participates. You may also wish to disconnect a record from a particular set occurrence and perhaps reconnect it with another set occurrence.

DATATRIEVE provides you with several statements you can use with VAX DBMS domains to manipulate records as owners and members of sets:

- The STORE statement — adds a new record to the database and automatically connects the record to each set of which it is an automatic member.
- The CONNECT statement — connects a selected member record to a set.
- The DISCONNECT statement — disconnects a member record from each set you specify. (You cannot disconnect owner records.)
- The RECONNECT statement — disconnects a member record from each set occurrence you specify and connects the record to another set occurrence you specify. (You cannot reconnect owner records.)

4.10.1 Storing and Connecting Records

When you store a new record or when you want to connect a particular record to a set occurrence, the procedure you use depends on whether the set is an automatic or manual member of a set. Insertion into a set can be automatic or manual:

- Automatic

  DATATRIEVE automatically inserts the record into the set when you store the record.

- Manual

  After modifying or storing the record, you can connect it to the set of which it is a member or leave it unconnected in the database.

For example, when you use the SHOW SETS command you can see the characteristics identified by DATATRIEVE for member domains:

```
DTR> SET DICTIONARY CDD$COMPATIBILITY.DTR$LIB.DEMO.DBMS
DTR> READY VENDORS, SUPPLIES
DTR> SHOW SETS

Set: VENDOR_SUPPLY
       Owner:  VENDORS
       Member: SUPPLIES, automatic fixed

Set: ALL_VENDORS
      Member: Vendors, automatic fixed
```

SUPPLIES is an automatic member of the set VENDOR_SUPPLY. The automatic characteristic indicates that when you store a new supplies record, it is automatically connected to the set VENDOR_SUPPLY.

SUPPLIES is also a member of the system-owned set ALL_VENDORS. It automatically participates in this set.

The fixed characteristic means that the domain SUPPLIES must be a member of the set. In addition, the record occurrence cannot be connected to any set occurrence other than the one to which it belongs when it is stored. For example, a particular SUPPLY record cannot be connected to any other vendor through VENDOR_SUPPLY than the one it was associated with at the time the record was stored. This characteristic is discussed in Section 4.10.2 on erasing and disconnecting records from sets.

The following two sections discuss automatic and manual insertion in a set.

4.10.1.1 Automatic Insertion

If a record is an automatic member of a set, when you use the DATATRIEVE STORE statement to store new records in the VAX DBMS database, the record is automatically inserted into the set. If you are storing a new record and only want to connect it with a system-owned set, you do not have to establish context to insert a record. After readying the domain for WRITE access, all you need to store a record into a system-owned set is a STORE statement. For example:

```
DTR> READY EMPLOYEES WRITE
DTR> STORE EMPLOYEES
Enter EMP_ID: 53456
Enter EMP_LAST_NAME: WINSLEE
Enter EMP_FIRST_NAME: JOANNE
Enter EMP_PHONE: 5324
Enter EMP_LOC: AS

DTR>
```

The new record automatically becomes a member of system-owned sets in which the record (domain in DATATRIEVE) participates. For example, a new employee's record is automatically part of the ALL_EMPLOYEES set in this example.

If a newly stored record is an automatic member of a set not owned by the system, though DATATRIEVE automatically connects the record to the set in which the record participates, you must provide the context for the set occurrence to which you want to connect the record.

Therefore, when you use the STORE statement:

- You use the CURRENCY clause to provide context for automatic members of sets that are not owned by the system.
- The record is automatically connected to each set of which it is an automatic member.

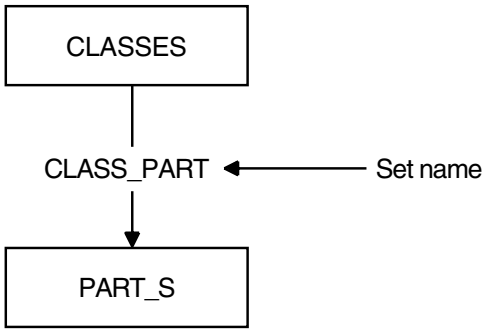
As with the display and print operations, DATATRIEVE uses the single-record context you supply to identify the set occurrence (and sometimes to select the position in that occurrence) when you modify sets or store records. If you fail to supply the single-record context, DATATRIEVE may insert a record in the wrong place, the record may not be moved, or you may receive an error message. Once you have established a single-record context, you can use this context to store many records without providing new single-record context each time.

The following example stores a new parts record in the PART_S domain and connects it to a specific occurrence of the CLASS_PART set. PART_S is an automatic member of the set CLASS_PART, shown in Figure 4–6. You:

- Establish a single-record context for the record *before* you store the record

- Store the record using the DATATRIEVE CURRENCY clause

Figure 4–6 VAX DBMS Set CLASS_PART




```
DTR> READY PART_S WRITE, CLASSES WRITE      1
DTR>
DTR> SHOW SETS      2
Set: CLASS_PART
       Owner: CLASSES
       Member: PART_S, automatic mandatory
       .
       .
       .
DTR> FIND BR IN CLASSES WITH CLASS_CODE = "BR"      3
[1 record found]
DTR> SELECT
DTR>
DTR> STORE PART_S CURRENCY BR.CLASS_PART      4
Enter PART_ID: BR902334
Enter PART_DESC: GUDGEON
Enter PART_STATUS: G
Enter PART_PRICE: 902.00
Enter PART_COST: 231.00
Enter PART_SUPPORT: X
DTR>
DTR> PRINT PART_ID, PART_DESC OF PART_S MEMBER      5
[Looking for set name]
CON> CLASS_PART

   Part
  Number   -----------------Part Description----------------

BR-1234-56 LA34
BR-3467-91 LA120
BR-8901-23 LA36
BR-9023-34 GUDGEON
```

```
DTR>
```

1 Ready the PART_S and CLASSES domains for WRITE access.

2 Use SHOW SETS to see that PART_S is an automatic member of the set CLASS_PART and that CLASS_PART is owned by another domain, CLASSES.

3 Establish a context with the single-set occurrence of the set CLASS_PART with which you want to connect the new part record.

You do this by using the FIND and SELECT statements to establish a single-record occurrence of the domain CLASSES with the class code of BR. When you select the record in the domain CLASSES, you establish context with the correct occurrence of the set CLASS_PART.

4 Store the new record in the PART_S domain and connect it to the current occurrence of the CLASS_PART set. DATATRIEVE prompts you for data values for the new record.

5 Check the new record. Because the CLASSES record BR is still the selected record, you can use the PRINT statement with the MEMBER clause of the record selection expression to see if the GUDGEON record is now a member of the class BR.

Note that you use the CURRENCY clause of the STORE statement to specify the exact set occurrence to which DATATRIEVE should connect the record. Note that the record being stored with a CURRENCY clause must be an automatic member of the set.

4.10.1.2 Manual Insertion

If a newly stored record is a manual member of a set, you must use the CONNECT statement to insert the record into the set. (You can also leave the record unconnected to any set.) You must establish the context when you connect the record to the set.

The following example connects an EMPLOYEE record to a DIVISIONS set. Because EMPLOYEES is a manual member of the set CONSISTS_OF, you must specifically insert the new employee record in the set CONSISTS_OF.

```
DTR> READY EMPLOYEES WRITE, DIVISIONS WRITE      1
DTR> STORE EMPLOYEES      2
Enter EMP_ID: 53456
Enter EMP_LAST_NAME: JANOV
Enter EMP_FIRST_NAME: LESLEY
Enter EMP_PHONE: 5324
Enter EMP_LOC: AS
```

```
DTR> FOR DIV IN DIVISIONS WITH DIV_NAME = "SOFTWARE"      3
CON>    FOR EMP IN EMPLOYEES WITH
CON>                        EMP_LAST_NAME = "JANOV"      4
CON>    BEGIN
CON>    CONNECT EMP TO DIV.CONSISTS_OF      5
CON>    PRINT EMP_LAST_NAME OF EMPLOYEES MEMBER      6
CON>      DIV.CONSISTS_OF
CON>    END

Last Name------

HUTCHINGS
IACOBONE
PASCAL
JANOV

DTR>
```

1 Ready both the EMPLOYEES and DIVISIONS domains for WRITE access.

2 Store the new employee record. (VAX DBMS automatically connects the record of the new employee to the system-owned set, ALL_EMPLOYEES, and to the sets owned by the employee record, MANAGES and RESPONSIBLE_ FOR.)

3 Specify each record in the DIVISIONS domain to which you want to connect the new employee record. Specify the context variable DIV, so you can use it later to qualify the set name.

4 Specify the new employee record you stored in step 2.

5 Use the CONNECT statement to connect the new employee record to the SOFTWARE division through the set CONSISTS_OF.

6 Print the employees from the set occurrence to see that you made the correct set connection.

Note that you connect the employee named JANOV to the occurrence of the DIVISIONS domain (DIV) through the set CONSISTS_OF. Where the record is inserted into the set depends on the set-ordering criteria defined in the schema. See the VAX DBMS documentation for more information on set order.

4.10.2 Erasing, Disconnecting, and Reconnecting Records with Sets

You use the ERASE statement to remove a record. Because the ERASE statement can delete more than you intend, use it with caution. Accidental deletions can occur because of the ERASE statement's "cascading effect." This cascading effect can happen whenever the erased record is the owner of a set. Thus, if the current record is an owner of a set type, ERASE deletes all of the following:

- The current record
- All records in sets owned by the current record
- Any records in sets owned by those members, and so forth

You can remove a record from a set either by erasing it with the DATATRIEVE ERASE statement or disconnecting and reconnecting it with sets.

The removal characteristics of a record determine whether records can be removed from sets and the way in which they can be removed. The removal characteristic of a record is one of the following:

- Fixed

  You cannot disconnect the record from its set occurrence unless you erase the record from the database.

- Mandatory

  You cannot use DISCONNECT to remove the record from a set occurrence. However, you can use RECONNECT to move it from one occurrence of the set type to another.

- Optional

  You can use either DISCONNECT or RECONNECT to remove the record from a set occurrence.

The following sections discuss removal from a set in more detail.

4.10.2.1 Erasing VAX DBMS Records

Records that are fixed members of sets, once connected to a set occurrence, must be a member of that specific set occurrence until they are deleted from the database. They cannot be disconnected and remain in the database or reconnected to some other set occurrence.

The fixed characteristic is common with system-owned sets that are used to keep large numbers of records on file. For example, an organization usually keeps a generalized listing of all employees. Such a listing can be maintained by a system-owned set, as in the PARTS database with the ALL_EMPLOYEES set.

In a previous example, you added the record of the employee named JANOV to the ALL_EMPLOYEES set and connected it to the SOFTWARE division. The following example erases the record JANOV from the database, deleting it from the system-owned ALL_EMPLOYEES set. As a result of being erased, the record in the example is also disconnected from the SOFTWARE group.

```
DTR> READY EMPLOYEES WRITE, DIVISIONS WRITE,      1
CON>  PART_S WRITE
DTR>
DTR> FIND EMPLOYEES WITH EMP_LAST_NAME = "JANOV"      2
[1 record found]
DTR> SELECT;PRINT EMP_LAST_NAME      3

Last Name-----------

JANOV

DTR> ERASE
DTR> PRINT CURRENT      4
DTR>
DTR> FIND EMPLOYEES WITH EMP_LAST_NAME = "JANOV"      5
[0 records found]
DTR>
```

1 Erasing the record for JANOV from the database involves the domains EMPLOYEES, DIVISIONS, and PART_S. Ready for WRITE access all domains that are affected by the loss of an employee.

If you do not ready all necessary domains, you might encounter a problem when you attempt to erase the record. Realms are ultimately associated with storage areas, unless the files themselves are readied through a READY command that readies at least one of the domains in that file.

Therefore, you might receive an error from VAX DBMS stating that a particular storage area has not been readied.

2 Find and select the record you want to erase.

If the record you erase is the owner of any sets (EMPLOYEES is owner of the sets RESPONSIBLE_FOR and MANAGES), records in member domains (PART_S and DIVISIONS) are also erased. Therefore, be sure that you know exactly what you are erasing. Table 4–1 summarizes the effects of erasing a record on the record and its members.

3 Make sure that this is the record you want to erase and then erase the record.

4 Show that DATATRIEVE prints nothing in response to the PRINT CURRENT statement. The current collection is now empty.

5 Try to find the record you erased.

The employee entry JANOV is no longer in the database. As a result of being erased, his employee record has also been disconnected from all the sets of which it was a member.

4.10.2.2 Disconnecting and Reconnecting VAX DBMS Records from Sets

Records that are mandatory members of sets can move from one occurrence of a set to another. However, such records must always be members of some occurrence of that set type once they have been connected.

The advantage of such membership is the ability to change your mind about the attributes of a member record. For example, suppose you want to move a part record (terminal stands) from one class of the domain CLASSES (terminal assemblies) to another class in that same domain (video terminals). The PART_S domain, which contains the parts record, is a mandatory member of the set CLASS_PART.

In the following example, you use the set CLASS_PART:

```
DTR> READY PART_S MODIFY, CLASSES MODIFY      1

DTR> FIND PART_S WITH PART_ID CONT "BU"       2
[7 records found]
DTR> PRINT PART_ID, PART_DESC OF CURRENT

   Part
  Number   -----------------Part Description-----------------

BU-1045-68 FREE-STANDING FRAME ASSEMBLY
BU-2345-67 VIDEO TUBE
BU-3161-25 VT100 NUMERIC KEYPAD ASSY
BU-7014-68 VT100 MONITOR UNIT
BU-7014-65 VT100 KEYBOARD UNIT
BU-3456-70 TERMINAL TABLE VT100
BU-0345-67 TERMINAL TABLE VT52

DTR> FOR PART_S WITH PART_ID CONT "BU345670",      3
CON> "BU034567" PRINT CLASSES WITHIN CLASS_PART

Code -Class Description-- St

 BU  TERMINAL ASSEMBLIES  Y

DTR> FIND VIDEO_TERMS IN PART_S WITH PART_ID CONT "BT"      4
[2 records found]
DTR> PRINT PART_ID, PART_DESC OF VIDEO_TERMS

   Part
  Number   -----------------Part Description-----------------

BT-1634-56 VT100
BT-0456-78 VT52


DTR> FOR VIDEO_TERMS PRINT CLASSES WITHIN CLASS_PART       5

Code -Class Description-- St

 BT  VIDEO TERMINALS      G
 BT  VIDEO TERMINALS      G
```

```
DTR> FOR VT IN CLASSES WITH CODE = "BT"      6
CON> FOR TABLES IN PART_S WITH PART_ID = "BU345670" OR
CON> PART_ID = "BU034567" RECONNECT TABLES TO VT.CLASS_PART
DTR>

DTR> FIND CLASSES WITH CODE = "BT"      7
[1 record found]
DTR> SELECT
DTR> PRINT PART_ID, PART_DESC OF PART_S MEMBER OF CLASS_PART

   Part
  Number   -----------------Part Description-----------------

BT-0456-78 VT52
BT-1634-56 VT100
BU-0345-67 TERMINAL TABLE VT52
BU-3456-70 TERMINAL TABLE VT100

DTR>
```

1 Ready the necessary domains. Because you are disconnecting a PART_S record from an occurrence of CLASSES and then connecting it to another occurrence, you must ready both those domains for MODIFY access.

2 Display the PART_S records that contain the letters BU in their PART_ID number. You want to move the last two parts (the terminal tables for the VT52 and VT100) to the occurrence of CLASS_PART owned by BT.

3 For these two part records, show that they are connected to the particular occurrence of the set CLASS_PART that is owned by CLASSES record BU. Those two records belong to the class called TERMINAL ASSEMBLIES.

4 Determine where you would like to move these two records. First, find and display all the PART_S records that contain the letters BT in their PART_ID.

5 Using a FOR loop, find and display the record occurrence of the domain CLASSES to which these two records are related. This is the record occurrence, VIDEO TERMINALS, to which you want to reconnect your two terminal tables.

6 Create the necessary context to reconnect the terminal tables to the video terminal set. Use nested FOR loops to create the context and a RECONNECT statement to move the records.

7 Make sure that you actually moved the records. Find and select the CLASSES record with the value BT for CODE.

Note that the terminal tables are now members of a new set occurrence. Check the former location of those records to make sure they are no longer there:

```
DTR> FIND CLASSES WITH CLASS_CODE = "BU"
[1 record found]
DTR> SELECT
DTR> PRINT CURRENT

Code -Class Description-- St

 BU  TERMINAL ASSEMBLIES  Y

DTR> PRINT PART_ID, PART_DESC OF PART_S MEMBER CLASS_PART

   Part
  Number   -----------------Part Description-----------------

BU-1045-68 FREE-STANDING FRAME ASSEMBLY
BU-2345-67 VIDEO TUBE
BU-3161-25 VT100 NUMERIC KEYPAD ASSY
BU-7014-65 VT100 KEYBOARD UNIT
BU-7014-68 VT100 MONITOR UNIT
DTR>
```

The terminal tables are no longer in the occurrence of CLASS_PART owned by BU.

4.10.2.3 Disconnecting and Connecting VAX DBMS Records from Sets

Records that are **optional members** of sets can belong to an occurrence of a set type or not belong to any occurrence at all. For example, the PARTS database has two system sets, ALL_PARTS and ALL_PARTS_ACTIVE. The ALL_PARTS set describes all parts cataloged by a firm. The ALL_PARTS_ACTIVE set consists of all parts currently in production or inventory. As a part is retired, it may be removed from the ALL_PARTS_ACTIVE set but retained in the ALL_PARTS listing of everything ever made.

You can use the DISCONNECT statement to remove OPTIONAL members of sets from those sets. You can later use the CONNECT statement to insert the disconnected record into another occurrence of the same set type or you can let that record remain disconnected in the database.

The following example disconnects a part from the ALL_PARTS_ACTIVE set:

```
DTR> READY PART_S WRITE      1
DTR>
DTR> PRINT PART_S MEMBER ALL_PARTS_ACTIVE WITH      2
CON> PART_ID = "BU104568"

   Part                                            Unit
  Number   -----------Part Description------------ST  Price

BU-1045-68 FREE-STANDING FRAME ASSEMBLY           G   $305
```

```
DTR> FOR P IN PART_S WITH PART_ID = "BU104568"      3
CON> DISCONNECT P FROM ALL_PARTS_ACTIVE
DTR>
DTR> PRINT PART_S MEMBER ALL_PARTS_ACTIVE WITH
CON> PART_ID = "BU104568"      4
DTR>
```

1 Ready the necessary domains. Because you are removing a PART_S record from a system-owned set, you need only ready PART_S for WRITE access.

2 Display the record that you want to remove from the ALL_PARTS_ACTIVE set.

Because ALL_PARTS_ACTIVE is a system-owned set, you do not have to establish context with the SELECT statement when you want to display the members of the set.

3 Establish the context you need to disconnect a record by specifying in a FOR loop the record you want removed from the set. Then use the DISCONNECT statement to remove it.

4 Check the ALL_PARTS_ACTIVE set to make sure the record is no longer there. DATATRIEVE responds with the DTR> prompt rather than a display of the record; the part record is no longer a member of the ALL_PARTS_ ACTIVE set.

4.10.3 Summary of Membership Characteristics

The record membership criteria limit the changes you can make to a database. Table 4–1 and Table 4–2 summarize the effects of various statements on the record being modified.

Table 4–1 Effect of Insertion, Retention, and Database Operations on Target Record

	Effect on Target Record					
INSERTION RETENTION	**CONNECT**	**DISCONNECT**	**ERASE**	**MODIFY**	**RECONNECT**	**STORE**
AUTOMATIC FIXED	Not Possible	Not Allowed	Erase	Reorder	Not Allowed	Insert
AUTOMATIC MANDATORY	Not Possible	Not Allowed	Erase	Reorder	Move Reorder	Insert
AUTOMATIC OPTIONAL	Insert	Remove	Erase	Reorder	Move Reorder	Insert
MANUAL FIXED	Insert	Not Allowed	Erase	Reorder	Not Allowed	No Effect
MANUAL MANDATORY	Insert	Not Allowed	Erase	Reorder	Move Reorder	No Effect
MANUAL OPTIONAL	Insert	Remove	Erase	Reorder	Move Reorder	No Effect

Notes to Table

Insert — Connects a record into an occurrence of the given set type.
Move — Reconnects a record from one occurrence of the given set type to another occurrence of the same set type.
Remove — Disconnects a record from an occurrence of the given set type.
Reorder — Can affect set ordering.
No Effect — Does not affect set membership.
Not Allowed — Returns an exception.
Not Possible — Cannot be done.

Table 4–2 Effect of Insertion, Retention, and Database Operations on Member

	Effect on Member	
INSERTION RETENTION	**ERASE ALL**	**ERASE**
AUTOMATIC FIXED	Erase	Erase
AUTOMATIC MANDATORY	Erase	Not Allowed
AUTOMATIC OPTIONAL	Erase	Remove
MANUAL FIXED	Erase	Erase
MANUAL MANDATORY	Erase	Not Allowed
MANUAL OPTIONAL	Erase	Remove

Notes to Table

Remove — Disconnects a record from an occurrence of the given set type.
Not Allowed — Return an exception.

4.10.4 Writing Changes to the Database

To write the changes you made to the database, you must enter a COMMIT statement. If, however, you do not want to save the changes you made, you can enter a ROLLBACK statement and leave the database as it was. The following statement rolls back any changes:

```
DTR> ROLLBACK
ROLLBACK executed; collection CURRENT automatically released
DTR>
```

DATATRIEVE automatically readies database domains again after you have committed or rolled back.

- A COMMIT statement performs a VAX DBMS COMMIT RETAINING.
- A ROLLBACK statement is equivalent to a DATATRIEVE ABORT.

The DATATRIEVE FINISH and EXIT commands end access to domains or VAX DBMS records. The FINISH command executes a VAX DBMS COMMIT (without the RETAINING argument) when you finish the last readied domain or record, or finish them all at once. The EXIT command also executes a VAX DBMS COMMIT statement.

Note

There is an important difference between the DATATRIEVE EXIT command and the DBQ EXIT command: The DATATRIEVE EXIT command executes a VAX DBMS COMMIT statement; the DBQ EXIT command issues a VAX DBMS ROLLBACK.

To write changes to the database, end access to the domains or VAX DBMS records, and remain in DATATRIEVE, use the FINISH command. To write changes to the database and end your DATATRIEVE session as well as access to domains and records, use the EXIT command:

```
DTR> FINISH
DTR> EXIT
```

5

Using DATATRIEVE with Rdb/VMS, Rdb/ELN, or VIDA

You can use VAX DATATRIEVE to access the Digital family of relational database management systems. These systems include Rdb/VMS, Rdb/ELN, and the VIDA facility. The VIDA facility is a relational database facility that provides access to data stored on IBM systems. VIDA users have transparent access to IBM data through VIDA with any of the following:

- DATATRIEVE
- The Relational Database Operator (RDO)
- The SQL interface to Rdb/VMS
- A third generation language supported by an Rdb/VMS data manipulation language precompiler

Using DATATRIEVE with VAX RMS files provides excellent data access when your database contains fewer than 5,000 records. If your database is larger than that, using a relational database product for data storage optimizes response time for your DATATRIEVE queries, data maintenance, and report-writing tasks.

The Digital family of relational database products provides the advantages of a database management system, including data integrity and in some cases, data security and journaling. At the same time, the relational model of data organization is easier to understand and to use than the network (CODASYL-style) model of data organization.

In this chapter, the terms relational database product or relational database refer to all three of these Digital relational database products: VAX Rdb/VMS, VAX Rdb/ELN, and VIDA. A specific product name is used for cases in which the information applies to that product alone.

The DATATRIEVE statements you use for data queries and report writing are the same, whether you are accessing an RMS file-structured database or a relational one. If you currently use DATATRIEVE to create and maintain file-structured databases, you need to learn some extensions to the DATATRIEVE language to access and maintain a relational database.

5.1 Getting Started with DATATRIEVE and Relational Databases

In a relational database, data is organized into **relations**. Relations are tables. A table has a horizontal dimension (rows) and a vertical dimension (columns). A row in a relation is a set of data fields, analogous to a record in a file. The fields in each row define the columns. In this chapter, the term **record** refers to an entire row in a database relation.

Figure 5–1 shows part of the structure of a relation called DEPARTMENTS in the PERSONNEL database installed with the DATATRIEVE User Environment Test Package (UETP).

Figure 5–1 Sample Rdb/VMS Relation

	Column 1 DEPARTMENT_CODE	Column 2 DEPARTMENT_NAME	Column 3 MANAGER_ID	
Row 1 →	ADMN	Corporate Administration	00225	. . .
Row 2 →	ELEL	Electronics Engineering	00397	
Row 3 →	ELGS	Large Systems Engineering	00369	
Row 4 →	ELMC	Mechanical Engineering	00215	
Row 5 →	ENG	Engineering	00435	

Figure 5–2 shows the relations and fields for the sample PERSONNEL database. Most examples in this chapter refer to the relation and field names in the PERSONNEL database. The data shown in the examples may be different from the data that appears on your screen.

Figure 5–2 Sample Rdb/VMS Database

EMPLOYEES EMPLOYEE_ID LAST_NAME FIRST_NAME MIDDLE_INITIAL ADDRESS_DATA STREET TOWN STATE ZIP SEX BIRTHDAY SOCIAL_SECURITY STATUS_CODE	DEGREES EMPLOYEE_ID COLLEGE_CODE YEAR_GIVEN DEGREE DEGREE_FIELD	JOBS JOB_CODE WAGE_CLASS JOB_TITLE MINIMUM_SALARY MAXIMUM_SALARY
	JOB_HISTORY EMPLOYEE_ID DEPARTMENT_CODE JOB_CODE JOB_START JOB_END SUPERVISOR_ID	COLLEGES COLLEGE_CODE COLLEGE_NAME ADDRESS_DATA STREET TOWN STATE ZIP
SALARY_HISTORY EMPLOYEE_ID SALARY_AMOUNT SALARY_START SALARY_END	DEPARTMENTS DEPARTMENT_CODE DEPARTMENT_NAME MANAGER_ID BUDGET_PROJECTED BUDGET_ACTUAL	WORK_STATUS STATUS_CODE STATUS_NAME STATUS_TYPE

The following command sets the dictionary default to the directory that contains the database definition and the domain definitions used in this chapter.

```
DTR> SET DICTIONARY CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB
```

Note

CDD$COMPATIBILITY is the logical name of the CDD/Repository compatibility dictionary. When you issue a SHOW command, you will

see either the name of a physical disk or an anchor instead of the CDD$COMPATIBILITY logical name.

The examples and references in this chapter primarily refer to Rdb/VMS, with some examples for Rdb/ELN and VIDA. For additional examples of using DATATRIEVE with Rdb/VMS, Rdb/ELN, and VIDA, see the documentation that accompanies each of those products.

5.2 Defining the Database

To access a relational database with DATATRIEVE, you need to do both of the following:

- Define the database, if the database does not already exist.

  When you define an Rdb/VMS or Rdb/ELN database, you specify the structure of the database by defining individual relations, fields for these relations, and key fields that link one relation with another.

  You must use either the Relational Database Operator (RDO) or the SQL interface to Rdb/VMS to define an Rdb/VMS database. The VAX Rdb/VMS documentation provides information on using both of these utilities.

  To define an Rdb/ELN database, use the Rdb/ELN Data Definition Language Compiler (ERDL). The VAX Rdb/ELN documentation tells you how to use ERDL to define databases.

  When you use VIDA to access a database, the database already has been defined on a remote IBM system. The VIDA documentation explains how the database must be defined so that VIDA can access it.

- Define the database in DATATRIEVE.

  You can use DATATRIEVE to access a relational database only if the database has a dictionary path name. You can create a dictionary path name by defining the database in DATATRIEVE. When you define the database, you store the definition of the database, or the location of the database file, in the CDD/Repository dictionary. DATATRIEVE uses this definition when you access the database.

  Rdb/ELN and VIDA databases do not have dictionary path names so you must create these with the DEFINE command in DATATRIEVE.

  You may have specified a dictionary path name when you created an Rdb/VMS database. In that case, you do not need to define the database in DATATRIEVE. DATATRIEVE uses the existing Rdb/VMS path name to access the database. (However, if you want to access the database from a

VMS directory other than the default, the Rdb/VMS database must be defined in RDO or SQL, using the full file specification.)

If you want to access an Rdb/VMS, Rdb/ELN, or a VIDA database that does not have a path name, you can create a path name with the DATATRIEVE DEFINE DATABASE command (see the *VAX DATATRIEVE Reference Manual* for more information).

5.2.1 Defining an Rdb/VMS Database in DATATRIEVE

The following examples illustrate three ways to create a dictionary path name for the Rdb/VMS PERSONNEL database:

```
DTR> DEFINE DATABASE CDD$COMPATIBILITY.DEPT29.PERSONNEL ON
DFN>                DBA2:[D29.DAT]PERSONNEL.RDB;
DTR>

DTR> DEFINE DATABASE CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL
DFN> ON DTR$LIBRARY:PERSONNEL;
DTR>

DTR> DEFINE DATABASE PERSONNEL ON PERSONNEL;
DTR>
```

The first example specifies the full path name and file specification for a user database in directory D29.DAT on disk DBA2. The second example specifies the full path name and file specification for the DATATRIEVE demonstration database. The third example relies entirely on current defaults, both for the path name and the file specification.

5.2.2 Defining an Rdb/ELN Database in DATATRIEVE

Because VAX Rdb/ELN does not use the dictionary, you must use the DATATRIEVE DEFINE command to establish a path name for the Rdb/ELN database. The DATATRIEVE DEFINE command creates a dictionary path name that is actually a pointer to the Rdb/ELN database file; it does not cause data definitions for the Rdb/ELN database to be stored in the dictionary. The following example defines a dictionary path name for the Rdb/ELN demonstration database.

```
DTR> DEFINE DATABASE PERS_ELN ON [V_WRITER.ELN]RDBDEMO.RDB
DFN> ;
DTR>
```

Attaching to an Rdb/ELN database requires SYSLCK privilege. In some cases, DATATRIEVE returns a database format error if you do not have SYSLCK privilege. See the Rdb/ELN documentation for more information.

5.2.3 Defining a VIDA Database in DATATRIEVE

To access a VIDA database with DATATRIEVE, you must define a dictionary path name for the VIDA database name. A VIDA database name is essentially a set of values that allows VIDA to get the data that a VAX user wants to access. You define a dictionary path name for a VIDA database with the DATATRIEVE DEFINE command. A VIDA database name specifies:

- The database type; for example:

  /TYPE=VIDA2
- A database name; for example:

  /DATABASE= PERSONNL
- Optional qualifiers, such as an access name; for example:

  /ACCESS=CICS

The dictionary path name CDD$TOP.PENDALTOWN.VIDA in the following example is associated with a VIDA database name. The VIDA database name in the example includes only basic qualifiers:

```
DTR> DEFINE DATABASE CDD$TOP.PENDALTOWN.VIDA ON
DFN>   "/TYPE=VIDA2/DATABASE=PERSONNL";
DTR>
```

This command associates the path name CDD$TOP.PENDALTOWN.VIDA with the VIDA database name. The DEFINE command does not create data definitions for the VIDA database in the dictionary.

VIDA also permits you to use logical names to define some qualifier values. The VIDA documentation explains VIDA access parameters.

You may want to define a logical name for the entire VIDA database name and then use this logical in the DEFINE command. (Quotation marks in the VIDA database name must be doubled when they appear in a logical name.) The following example shows how to define a logical name for the VIDA database name and how to use the logical name in the DEFINE command:

```
$ DEFINE VIDA_DATABASE
_Equ name: "/TYPE=VIDA2/DATABASE=PERSONNL"
$ DTR
DTR> DEFINE DATABASE CDD$TOP.PENDALTOWN.VIDA ON VIDA_DATABASE;
DTR>
```

In this example, the user first defines the logical name VIDA_DATABASE to be equivalent to the text string containing the required /TYPE qualifier and the qualifier /DATABASE. The user then uses this logical name in place of the database name in the DATATRIEVE DEFINE DATABASE statement.

Note

You must use quotation marks around the VIDA database name. You must use an additional set of quotation marks around qualifier values when they contain special characters, such as a space.

5.3 Accessing the Database

After you create a path name for your relational database, you can access it with DATATRIEVE in either of two ways:

- Ready the database directly.
- Define a domain for each relation that you want to access and then ready the domains you want to use.

5.3.1 Readying a Relational Database Directly

You can ready an Rdb/VMS, an Rdb/ELN, or a VIDA database without defining DATATRIEVE domains for any relations.

After you ready a VIDA database, you can work with it just as you would an Rdb/VMS or Rdb/ELN database. Because VIDA is a read-only database, however, you cannot use statements that write to VIDA sources. For more information on the READY command, see the *VAX DATATRIEVE Reference Manual*.

The following sections explain the types of access you can have to data in relational databases.

5.3.1.1 Access Modes and Options

The access mode and option you choose determine what operations you can perform on the data in the database and how current your data is. The default access option, SNAPSHOT, is called a read-only or "snapshot" ready. The SNAPSHOT access option implicitly includes the READ access mode; when you specify SNAPSHOT as the access option, DATATRIEVE readies the database as SNAPSHOT READ. When you specify CONCURRENCY with SNAPSHOT for Rdb/ELN or VIDA databases, DATATRIEVE behaves as though the access were SHARED READ. You will see changes that other users make to the data.

The SNAPSHOT access option lets you read data without locking other users out of the database. You cannot update data when you access it with the SNAPSHOT access option. Therefore, other users can have READ, WRITE, MODIFY, or EXTEND access to the same database, relation, or domain. VIDA databases are read-only; therefore, users cannot have WRITE, MODIFY, or EXTEND access.

In Rdb/VMS and Rdb/ELN databases, the SNAPSHOT access option lets you see a "picture" of the database, exactly as it was when you readied the relation, domain, or database. When other users change the data in the sources you have readied, you do not see these changes until you end your access to that data or reready the database for another operation.

In VIDA databases, you do see changes that other users make to the data you access. VIDA does not provide the same degree of data consistency as Rdb/VMS and Rdb/ELN. Thus, when you use the SNAPSHOT option to access a VIDA database, you do not have a snapshot "picture" of the database. Instead you see the database as it is being updated.

VIDA databases are read-only; you can ready a VIDA database only with SNAPSHOT, SNAPSHOT READ, or READ access.

In Rdb/VMS and Rdb/ELN databases, the PROTECTED and EXCLUSIVE access options ensure that other users cannot make changes to data while you are updating that data. The WRITE, MODIFY, and EXTEND access modes signal your intention to perform an update operation and cause Rdb/VMS or Rdb/ELN to use write locks. For more information on locking, see the VAX Rdb/VMS documentation on database administration and maintenance.

5.3.1.2 Consistency Options

Consistency options control whether you see changes made by other users to the data you are accessing. CONSISTENCY guarantees that while you are accessing data, you do not see any changes made by other users. CONSISTENCY is the DATATRIEVE default for the first ready of a relational source. For subsequent readies, the DATATRIEVE default is the consistency option used for the most recent ready still in effect for a relational source.

CONCURRENCY, on the other hand, allows you to see updates made by others while you are accessing the database. For example, with VIDA you might print out a report in which the individual line items do not add up to the final total. This can occur because another user is making changes to the line item data while you are creating the report.

Currently, the relational database products implement CONSISTENCY and CONCURRENCY in different ways. You can access:

- Rdb/ELN databases in both CONSISTENCY and CONCURRENCY modes
- Rdb/VMS databases in CONSISTENCY mode

  Even though you can specify CONCURRENCY mode, you get CONSISTENCY mode
- VIDA databases in both CONSISTENCY and CONCURRENCY mode

The first time you ready a relation or a database in your DATATRIEVE session, DATATRIEVE uses CONSISTENCY as the default consistency option. However, at any other time, when you ready the same relational source and do not specify a consistency option, DATATRIEVE uses the option which is in effect for that database. For example, if you have readied a VIDA database for CONCURRENCY and do not specify a consistency option in your next READY command for that database, DATATRIEVE uses the CONCURRENCY option.

5.3.1.3 Examples

The following examples illustrate various ways to ready a relational database directly.

To ready an entire Rdb/VMS database:

```
DTR> READY PERSONNEL
DTR>

DTR> READY CDD$COMPATIBILITY.DEPT39.PERSONNEL MODIFY CONSISTENCY
DTR>
```

To ready an entire Rdb/ELN database:

```
DTR> READY PERS_ELN CONCURRENCY
DTR>

DTR> READY CDD$COMPATIBILITY.V_WRITER.PERS_ELN MODIFY CONSISTENCY
DTR>
```

To ready an entire VIDA database:

```
DTR> READY CDD$TOP.PENDALTOWN.VIDA
DTR>

DTR> READY CDD$TOP.PENDALTOWN.VIDA READ CONCURRENCY
DTR>
```

To ready selected Rdb/VMS relations:

```
DTR> READY CDD$COMPATIBILITY.DEPT39.PERSONNEL USING EMPLOYEES
DTR>

DTR> READY PERSONNEL USING
CON> EMPLOYEES, SALARY_HISTORY WRITE CONSISTENCY
DTR>
```

To ready selected Rdb/ELN relations:

```
DTR> READY CDD$COMPATIBILITY.V_WRITER.PERS_ELN USING EMPLOYEES CONCURRENCY
DTR>

DTR> READY PERS_ELN USING
CON> EMPLOYEES, SALARY_HISTORY MODIFY CONSISTENCY
DTR>
```

To ready selected VIDA relations:

```
DTR> READY CDD$TOP.PENDALTOWN.VIDA USING EMPLOYEES SNAPSHOT
DTR>

DTR> READY CDD$TOP.PENDALTOWN.VIDA USING EMPLOYEES,
CON> SALARY_HISTORY READ CONCURRENCY
DTR>
```

When you specify CONCURRENCY with SNAPSHOT, DATATRIEVE behaves as though the access were SHARED READ. You will see changes that other users make to the data. When you issue the READY command to ready a database directly and specify more than one access mode, access option, or consistency option in the command, DATATRIEVE returns an error.

5.3.2 Defining and Readying Relational Domains

You can define a DATATRIEVE domain for each relation that you want to access. Then, you use the domain names to ready the database relations you want to access.

Accessing the database through domains may slightly slow the performance of the DATATRIEVE READY command; accessing a database directly works more quickly. However, you may want to define domains because:

- You can define DATATRIEVE views of Rdb/VMS, Rdb/ELN, and VIDA domains
- You can link a form definition with a domain definition using the FORM IS syntax (see Chapter 2 and Chapter 3 for more information about form products.

For example, the following commands define domains for the EMPLOYEES and SALARY_HISTORY relations in the Rdb/VMS PERSONNEL database:

```
DTR> DEFINE DOMAIN EMPLOYEES USING EMPLOYEES OF
DFN>   DATABASE CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL;
DTR>

DTR> DEFINE DOMAIN SALARY_HISTORY USING SALARY_HISTORY OF
DFN>   DATABASE CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL
DFN>   FORM IS SALARYHST IN FORMSLIB;
DTR>
```

After you define a domain for a relation in a relational database, you can ready it.

You do not supply a DATATRIEVE definition of the fields and indexes associated with each domain. DATATRIEVE retrieves this information from the relational database product when you ready the domains:

```
DTR> READY SALARY_HISTORY, EMPLOYEES
DTR>
```

When you issue a READY command for domains, DATATRIEVE readies the database that contains the relation on which the domain is based. Thus, when you ready a domain based on a relation in a database, use the access modes and options that you would use to ready the database itself. If you ready both a database and a domain based on a relation in that database, use the same consistency option for both readied sources. In some cases, DATATRIEVE will return an error if you do not.

Remember that any restriction that applies to readying a database relation also applies to readying a domain that is based on the relation. For more information on the READY command, see the *VAX DATATRIEVE Reference Manual*.

See Section 5.3.1.1 on access modes for a discussion of product specific implementations of the READY command options.

5.4 Using Views

A view is a "virtual" relation. Its definition specifies fields from one or more source relations. A view contains no data. You cannot always use views for store or modify operations, but you can use them to display records.

You might want to create a view that is a subset of the fields in a relation when you do not want certain users to see confidential data. You would give these users access to the view but not to the relation itself.

You might want to create a view that joins fields from more than one relation when you know that users will often request that combination of fields. It is simpler for users to access and display the view than it is for them to repeat a query that accomplishes the same relational join.

You can create views in two ways:

- Using an Rdb/VMS or Rdb/ELN utility

  In this case, DATATRIEVE treats the view as it does any other relation in the Rdb/VMS or Rdb/ELN database. (VIDA database systems do not include views; but you can define views of VIDA data using DATATRIEVE.)

- Using DATATRIEVE

  In this case, you must first define a DATATRIEVE domain for each relation in the view. Then you can define a DATATRIEVE view domain of the DATATRIEVE domains.

5.4.1 Creating Rdb/VMS and Rdb/ELN Views

You create an Rdb/VMS or Rdb/ELN view relation using the Rdb/VMS RDO utility, the SQL interface to Rdb/VMS, or the Rdb/ELN ERDL compiler. After you create a view with RDO, SQL, or ERDL, you can access it with the READY command, just as you can a simple relation. If you want your view to use a form automatically, define a DATATRIEVE domain for it.

There are advantages to using Rdb/VMS and Rdb/ELN views rather than view domains:

- The DATATRIEVE response time is faster when you use Rdb/VMS or Rdb/ELN views.
- You can store or modify records using Rdb/VMS or Rdb/ELN views if they contain fields from only one source relation. You cannot store records using DATATRIEVE view domains, even when those views access fields from only one relation.

5.4.2 Defining and Using DATATRIEVE View Domains

DATATRIEVE view domains provide the following advantages:

- You can create views that combine fields from an Rdb/VMS, an Rdb/ELN, or a VIDA database with fields from RMS (file-structured) and VAX DBMS domains. The ability to combine fields from different types of databases is a powerful feature of DATATRIEVE view domains.
- Your database administrator might decide that creating an Rdb/VMS or Rdb/ELN view does not benefit enough database users to warrant creating one. In this case, you can create a DATATRIEVE view that is useful for your application.
- You can create views of VIDA data. VIDA does not permit you to define views; instead, you can use DATATRIEVE to create the view.
- You can refer to DATATRIEVE domain tables with a view domain. This allows you to establish constraints that are not defined for the relational database, but that are appropriate for your personal application.

In such cases, create a view domain with the DEFINE DOMAIN command.

Note

– You cannot base view domains directly on relations. You must first define a DATATRIEVE domain for each relation the view accesses.

– Views defined in SQL are usable from DATATRIEVE only if they identify physical records.

Refer to the *VAX DATATRIEVE Reference Manual* for a detailed explanation of the arguments and restrictions that apply to view domains.

The following examples define view domains for the Rdb/VMS PERSONNEL database. The view domain MAILING_INFO uses a subset of fields from the EMPLOYEES domain.

```
DTR> DEFINE DOMAIN MAILING_INFO OF EMPLOYEES
DFN> 01 NAME_AND_ADDRESS OCCURS FOR EMPLOYEES.
DFN>    03 FIRST_NAME        FROM EMPLOYEES.
DFN>    03 MIDDLE_INITIAL    FROM EMPLOYEES.
DFN>    03 LAST_NAME         FROM EMPLOYEES.
DFN>    03 ADDRESS_DATA      FROM EMPLOYEES.
DFN>    03 STREET            FROM EMPLOYEES.
DFN>    03 TOWN              FROM EMPLOYEES.
DFN>    03 STATE             FROM EMPLOYEES.
DFN>    03 ZIP               FROM EMPLOYEES.
DFN> FORM IS MAILFORM IN FORMSLIB;
DTR>
```

The view domain MANAGER_NAMES uses fields from both the EMPLOYEES and DEPARTMENTS domains and a domain table, MANAGERS_TABLE, that is based on current values in the DEPARTMENTS domain. HIS_NIBS is defined as a query name for MANAGER_ID in the DEPARTMENTS domain. MANAGERS_ TABLE pairs the query name HIS_NIBS with the associated DEPARTMENT_ CODE:

```
DTR> DEFINE DOMAIN MANAGER_NAMES OF DEPARTMENTS, EMPLOYEES USING
DTR> 01 DEPARTMENT OCCURS FOR DEPARTMENTS.
DFN>    03 DEPARTMENT_CODE   FROM DEPARTMENTS.
DFN>    03 DEPARTMENT_NAME   FROM DEPARTMENTS.
DFN>    03 MANAGED_BY OCCURS FOR EMPLOYEES WITH
DFN>                          HIS_NIBS IN MANAGERS_TABLE.
DFN>       06 FIRST_NAME     FROM EMPLOYEES.
DFN>       06 MIDDLE_INITIAL FROM EMPLOYEES.
DFN>       06 LAST_NAME      FROM EMPLOYEES.
DFN> ;
DTR>
```

You access a view domain by readying it, just as you would a simple domain. Remember that any restriction that applies to readying a database relation also applies to readying a view that contains a domain based on that relation. Refer to the READY command section of the *VAX DATATRIEVE Reference Manual* for restrictions on readying relations.

5.5 Displaying Information About Readied Relations and Domains

The SHOW READY and SHOW FIELDS commands provide information about the relations and fields you can access. The SHOW READY command displays the sources that are readied and their access modes, access options, and consistency options. The following example readies the domain SALARY_HISTORY, displays information about that domain, and shows the fields you can access:

```
DTR> READY SALARY_HISTORY
DTR> SHOW READY
Ready sources:
  SALARY_HISTORY: Domain, Rdb, snapshot read, consistency
            <CDD$COMPATIBILITY.DEPT29.PERSONNEL.SALARY_HISTORY;1>
No loaded tables.
DTR> SHOW FIELDS FOR SALARY_HISTORY
  SALARY_HISTORY
      EMPLOYEE_ID       <Character string>
      SALARY_AMOUNT     <Number>
      SALARY_START      <Date>
      SALARY_END        <Date>
```

If you do not specify an access mode or an access option for a relational database, domain, or relation, the default access is SNAPSHOT. (Remember that you can ready VIDA databases only in SNAPSHOT READ mode.) Other users can have READ, WRITE, MODIFY, or EXTEND access to the database, domain, or relation. (VIDA databases are read-only; therefore, users cannot have WRITE, MODIFY or EXTEND access.) The following example shows the result of readying an Rdb/VMS database directly. There are no domains defined for the relations in this example:

```
DTR> READY PERSONNEL
DTR> SHOW READY
Ready sources:
   WORK_STATUS:  Relation, Rdb, snapshot read, consistency
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL;1>
   DEGREES:  Relation, Rdb, snapshot read, consistency
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL;1>
   COLLEGES:  Relation, Rdb, snapshot read, consistency
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL;1>
   DEPARTMENTS:  Relation, Rdb, snapshot read, consistency
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL;1>
   JOBS:  Relation, Rdb, snapshot read, consistency
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL;1>
   SALARY_HISTORY:  Relation, Rdb, snapshot read, consistency
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL;1>
   JOB_HISTORY:  Relation, Rdb, snapshot read, consistency
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL;1>
   EMPLOYEES:  Relation, Rdb, snapshot read, consistency
           <CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL;1>
No loaded tables.
```

5.6 Ending Access to Domains, Relations, and Views

Use the FINISH command to end access to relations.

DATATRIEVE does not finish a database until you have finished every readied relation (or domain based on a relation) in that database. You can finish individual relations or domains without affecting access to the database or the locking specified by your access mode. However, when you finish the last relation or domain from a database, DATATRIEVE releases all the locks or resources you held and ends access to the database as a whole.

5.7 Storing and Maintaining Data in an Rdb/VMS or Rdb/ELN Database

Two extensions to the DATATRIEVE language apply only when your data is managed by a relational database management system. These extensions are the COMMIT and ROLLBACK statements. To understand what these statements do, you must understand the way Rdb/VMS or Rdb/ELN stores and updates information in your database and how this differs from storing and updating information in a file-structured database.

Because VIDA databases are read-only, this discussion does not include them. If you do try to commit VIDA data, DATATRIEVE executes the COMMIT statement and starts a transaction.

5.7 Storing and Maintaining Data in an Rdb/VMS or Rdb/ELN Database

When you ready a file-structured domain, DATATRIEVE opens the data file associated with the domain and makes any changes to the data file as you enter them. Once the changes are made to the file, you can consider them permanent.

When you ready an Rdb/VMS or Rdb/ELN database directly or when you ready Rdb/VMS or Rdb/ELN domains, DATATRIEVE makes any changes to the database as you enter them. However, those changes are not permanent until one of the following occurs:

- You enter a COMMIT statement, either interactively or as part of a procedure.
- You cause DATATRIEVE to perform a commit operation in one of these ways:
  - You enter a final FINISH command for the last readied domain or the last readied relation (depending upon your chosen method of database access).
  - You exit from DATATRIEVE.

If you decide that you do not want your entries to take effect, you can enter a ROLLBACK statement. When you use the ROLLBACK statement, all the changes made since execution of the last COMMIT or ROLLBACK statement are undone. If neither statement executed, then the ROLLBACK statement undoes all the changes made since the beginning of your session.

If your system fails, an implicit ROLLBACK executes. In this case, Rdb/VMS or Rdb/ELN undoes all changes made to the database since execution of the last implicit or explicit commit or rollback. If you open a DATATRIEVE log file at the beginning of your session, it is easy to find out what changes need reentry after a system failure.

On a more sophisticated level, Rdb/VMS and Rdb/ELN provide journaling and other facilities that help you recover from an accident. In any event, do not rely on reports that were generated before the system failure to determine what changes have been permanently stored.

Once you perform an operation on data in a domain or relation (display, store, or modify it), you must enter a COMMIT, ROLLBACK, or FINISH command before you can ready the database again. If, for instance, you have displayed data from a relation and you want to change the access mode to modify the relation, first use the COMMIT or ROLLBACK statement. If you do not, DATATRIEVE issues an error. You must enter a COMMIT, ROLLBACK, or FINISH command before you can ready the database again.

If you want to ready another domain or relation in a different access mode or option, you should enter a COMMIT or ROLLBACK statement before the READY command. You should also explicitly commit or roll back when you issue a READY command with a new LOCK_WAIT setting. Make sure to use the SET LOCK_WAIT command before the READY, or the new setting will not take effect until the next time you ready a source.

Another way of thinking about COMMIT and ROLLBACK statements is to understand that they end transactions. When working with relational databases, it is important to think in terms of **transactions**. Because Rdb/VMS and Rdb/ELN give many users access to a database at the same time, they control user activities to avoid access conflicts and data inconsistencies. Therefore, Rdb/VMS and Rdb/ELN require each user to identify a unit of database activity, called a **transaction**.

A transaction is an operation on the database that must complete as a unit or not complete at all. In DATATRIEVE, a transaction on a relational database begins with a READY command and ends with either a COMMIT, FINISH, or ROLLBACK statement. Transactions cannot be nested; they can only be performed consecutively.

Because you cannot selectively commit or roll back some parts of a transaction and not others, it is important to keep transactions short. In addition, you should try to conduct transactions with SNAPSHOT (read-only) or SHARED access if possible.

The next two sections discuss the COMMIT and ROLLBACK statements in greater detail and present some examples of their use.

5.7.1 Using the COMMIT Statement

When you enter a COMMIT statement, you make permanent all the changes made to the database since execution of the last COMMIT or ROLLBACK statement and release all locks held during the transaction. If neither of these statements has been executed since the beginning of your session, entering COMMIT means that you wish to make permanent all the database changes you have made during your DATATRIEVE session.

Note that when you issue an explicit COMMIT statement, it affects all databases that may be readied, including other relational or VAX DBMS databases. When DATATRIEVE performs an implicit commit because you ready a relational database with a different access mode or LOCK_WAIT setting, it affects only the database you ready.

When you issue a COMMIT statement, DATATRIEVE maintains all collections of records from relational sources. However, when you issue a COMMIT statement, or DATATRIEVE performs a commit operation implicitly, Rdb/VMS, RDB_ELN, and VIDA start a new transaction. In this new transaction, your Rdb/VMS or Rdb/ELN collection includes committed changes other users have made to the records in the collection since you formed the collection.

The following examples illustrate explicit and implicit execution of the COMMIT statement.

The first example illustrates the use of an explicit COMMIT statement. One record is permanently stored in the relation JOB_HISTORY following the COMMIT statement.

```
DTR> READY PERSONNEL USING JOB_HISTORY WRITE
DTR> STORE JOB_HISTORY
Enter EMPLOYEE_ID: 00166
Enter JOB_CODE: APGM
Enter JOB_START: 11-Nov-1979
Enter JOB_END: 8-Aug-1981
Enter DEPARTMENT_CODE: PRMG
Enter SUPERVISOR_ID: 00319
DTR> COMMIT
```

The next example shows both an explicit commit and implicit commit. The user readies the Rdb/VMS PERSONNEL database for WRITE access and stores a record in the DEGREES relation. DATATRIEVE requires the user to issue a COMMIT, ROLLBACK, or FINISH command before the user can reready the database.

The user commits the transaction and rereadies the PERSONNEL database and then readies a VIDA database in a new LOCK_WAIT setting, access mode, and consistency option. DATATRIEVE performs an implicit commit and does not prompt for a COMMIT, ROLLBACK, or FINISH command because the user has not performed any data operations since the last commit.

```
DTR> SET DICTIONARY CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB
DTR> SHOW DATABASES
Databases:
        PERSONNEL

DTR> SET NO LOCK_WAIT
DTR> READY PERSONNEL USING DEGREES WRITE
```

```
DTR> STORE DEGREES
Enter EMPLOYEE_ID: 00406
Enter COLLEGE_CODE: STAN
Enter YEAR_GIVEN: 1985
Enter DEGREE: PhD
Enter DEGREE_FIELD: Arts
DTR> READY PERSONNEL USING EMPLOYEES WRITE
Source cannot be readied unless a COMMIT, ROLLBACK or FINISH is issued.
DTR> COMMIT
DTR> READY PERSONNEL USING EMPLOYEES WRITE
DTR> SET LOCK_WAIT
DTR> READY CDD$TOP.PENDALTOWN.VIDA CONCURRENCY
```

In the next example, the FINISH command does not specify any domain or relation name and so ends access to the entire database, not just JOB_HISTORY. It therefore results in an implicit COMMIT statement. In this case, two records are permanently stored in JOB_HISTORY after the FINISH command:

```
DTR> READY PERSONNEL USING JOB_HISTORY WRITE
DTR> REPEAT 2 STORE JOB_HISTORY
Enter EMPLOYEE_ID: 00164
Enter JOB_CODE: DMGR
Enter JOB_START: 9-Sept-1981
Enter JOB_END: 18-Feb-1983
Enter DEPARTMENT_CODE: MBMN
Enter SUPERVISOR_ID: 00359
Enter EMPLOYEE_ID: 12487
Enter JOB_CODE: SPGM
Enter JOB_START: 07-Jul-1980
Enter JOB_END: 9-Sep-1981
Enter DEPARTMENT_CODE: MCBM
Enter SUPERVISOR_ID: 04164
DTR> FINISH
```

In the following example, the EXIT command (or CTRL/Z entered at the DTR> prompt) implicitly executes a COMMIT statement. The results are the same as in the preceding example, except that five records are permanently stored in JOB_HISTORY:

```
DTR> SHOW STORE_JOB_HISTORY
PROCEDURE STORE_JOB_HISTORY
SET ABORT
READY PERSONNEL USING JOB_HISTORY WRITE
DECLARE REC_NUM PIC 999.
REC_NUM = *."number of records you are adding"
REPEAT REC_NUM STORE JOB_HISTORY
END_PROCEDURE
DTR> :STORE_JOB_HISTORY
Enter number of records you are adding: 5
Enter DEPARTMENT_CODE: ELGS
                  .
                  .
                  .
DTR> EXIT
```

Note

The use of CTRL/Y causes you to exit from DATATRIEVE but signals abnormal termination. When you enter CTRL/Y, Rdb/VMS, Rdb/ELN, and VIDA execute a ROLLBACK command. However, DATATRIEVE does not display a message informing you of the rollback. In the case of VIDA databases, which are read-only, the rollback is meaningless.

Remember that it is always better to end a transaction explicitly with a COMMIT or ROLLBACK statement than to rely on DATATRIEVE to interpret a statement as an implicit end to a transaction. That way, you can be sure which operations are included in each transaction.

5.7.2 Using the ROLLBACK Statement

When you enter a ROLLBACK statement, you undo all the changes made to the database since execution of the last implicit or explicit COMMIT or ROLLBACK statement. If none of these was executed since the beginning of your session, entering ROLLBACK means that you wish to undo all the database changes you have made during your DATATRIEVE session.

The ROLLBACK statement releases all collections of relational records and VAX DBMS records. Even though VIDA databases are read-only and cannot have changes rolled back, the ROLLBACK statement also releases VIDA collections.

Note that when you issue a ROLLBACK statement, it affects all readied databases, including VAX DBMS and other relational databases.

ROLLBACK has the same effect as an ABORT statement. This means that it can alter the flow of execution of procedures, command files, and nested statements. If you embed a ROLLBACK statement in a procedure, you might wish to begin the procedure with an explicit COMMIT statement. This ensures that if the procedure rolls back, DATATRIEVE does not roll back more operations than you expected.

If you make a mistake when entering data for one of the records you are storing, you can still use CTRL/Z to keep that record from being stored. When you use CTRL/Z in this way, you affect only the record on which you are working, not any other record entries you might have made.

If you are designing procedures to be used by people unfamiliar with DATATRIEVE, you probably want to include the FINISH command in your procedure. This is especially useful if the procedure uses the default NO LOCK_ WAIT setting or if the procedure accesses a database, relation, or domain in PROTECTED or EXCLUSIVE modes. Otherwise, the domain or relation remains locked to all users after your procedure executes. If your procedure includes a ROLLBACK option, however, make sure that the FINISH command does not execute before the ROLLBACK statement. Otherwise, if the FINISH command ends user access to the last readied domain (or to all of them at once), a COMMIT statement executes before the ROLLBACK statement does.

5.8 Querying the Database, Writing Reports, and Using Collections

The statements you use for queries and report writing are the same for Rdb/VMS, Rdb/ELN, and VIDA relations or domains as they are for other domains. Here are some reminders if you plan to use collections:

- The COMMIT statement maintains collections of records.
- The ROLLBACK statement releases any collections that contain relational database records.
- The FINISH command releases any collections containing records from the domain, relation, or database being finished.

Make sure you ready domains, relations, or the database with the access you need for the collections you plan to create. If you inadvertently ready the database for READ access, form a large collection, print some records, and then try to modify records, you get an error message. At this point, you must enter COMMIT before you can ready the database for MODIFY access. Although a COMMIT statement maintains collections, if other users have made changes to records in the collection since you formed it, you will see these changes after you issue the COMMIT statement.

When designing procedures that produce reports or displays that include database changes, you might want to enter a COMMIT statement before the PRINT or REPORT statement. This ensures that you are using the most current data for your report or display and that your session will not inadvertently roll back changes that currently appear in your report. For the same reason, you might use a COMMIT statement before any query that is based on assumed database modifications. However, you should remember that when other users are accessing the database in the same DATATRIEVE session, a COMMIT statement makes their changes to the data permanent and visible to your application.

Note also that DATATRIEVE uses the relation name as the top-level group field. For this reason you cannot use the top-level group field name as a print-list element in a Report Writer PRINT statement or as a header or summary element in an AT TOP or AT BOTTOM statement when you create reports.

5.9 Using the Rdb/VMS or Rdb/ELN Segmented String Data Type in DATATRIEVE

DATATRIEVE provides limited support for a special Rdb/VMS an Rdb/ELN data type called **segmented string**.

Fields defined with the segmented string data type can contain completely unstructured data. Segmented string fields have the following special characteristics:

- You can store any type of data in a segmented string field. Segmented strings can contain ASCII text, binary code, Remote Graphics Instruction Set (ReGIS) graphics, or any other data type.
- You do not have to specify the length of data in a segmented string. This makes segmented strings useful for storing data that is arbitrarily long, such as text files or graphic data.
- Rdb/VMS and Rdb/ELN do not allocate any storage space for segmented string fields unless you actually store data in the field. This makes segmented string fields a good choice for optional comments or descriptions associated with a record.

Using DATATRIEVE, you can display, modify, and store data in segmented string fields. Although Rdb/VMS and Rdb/ELN allow you to store any type of data in a segmented string field, DATATRIEVE currently limits you to character input. You cannot access segmented string data in VIDA because VIDA databases do not use the segmented string data type.

5.9 Using the Rdb/VMS or Rdb/ELN Segmented String Data Type in DATATRIEVE

5.9.1 Defining Segmented String Fields in Rdb/VMS or Rdb/ELN

You cannot use DATATRIEVE to define segmented string fields. To define segmented string fields, use the Rdb/VMS RDO utility, the SQL interface to Rdb/VMS, or the Rdb/ELN ERDL compiler.

The following example shows how to use the Rdb/VMS RDO utility to define a segmented string field and a relation that uses it. Once you define the relation, you can use it as part of the sample Rdb/VMS database, PERSONNEL, created during installation of DATATRIEVE.

The segmented string field RESUME contains the resume for an employee in the PERSONNEL database. The EMPLOYEE_ID field links the RESUME relation with the EMPLOYEES relation in the PERSONNEL database:

```
$ RUN SYS$SYSTEM:RDO
RDO> SET DICTIONARY CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB
RDO> ! First invoke the sample PERSONNEL database:
RDO> INVOKE DATABASE PATHNAME PERSONNEL
RDO> ! Define a segmented string field called RESUME:
RDO> DEFINE FIELD RESUME
cont>   DATA TYPE IS SEGMENTED STRING.
RDO> ! Define a relation using RESUME and EMPLOYEE_ID, which
RDO> ! is based on an already-defined field, ID_NUMBER:
RDO> DEFINE RELATION RESUMES.
cont>   EMPLOYEE_ID
cont>           BASED ON ID_NUMBER.
cont>   RESUME.
cont> END RESUMES RELATION.
RDO> ! Display the fields for the relation RESUMES:
RDO> SHOW FIELDS FOR RESUMES
 Fields for relation  RESUMES
     EMPLOYEE_ID                    text size is  5
       based on global field  ID_NUMBER
     RESUME                         segmented string
                                      segment_length 512
RDO> ! Use the COMMIT statement to store the
RDO> ! new field and relation for PERSONNEL:
RDO> COMMIT
RDO> EXIT
```

5.9.2 Displaying Segmented String Fields in DATATRIEVE

To display data in segmented string fields from within DATATRIEVE, follow these steps:

- Ready the database. For best performance, use only the relations that you need to work with (in this example, the RESUMES relation defined in the preceding section).

- Use DATATRIEVE PRINT or LIST statements to display data in the segmented string field.

The following example illustrates these steps:

```
DTR> SET DICTIONARY CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB
DTR> SHOW DATABASES
Databases:
        PERSONNEL
DTR> ! Ready the database:
DTR> READY PERSONNEL USING RESUMES
DTR> ! Check that RESUMES is ready:
DTR> SHOW READY
Ready sources:
   RESUMES:  Relation, Rdb, snapshot read, consistency
          <CDD$COMPATIBILITY.DTR$LIB.DEMO.RDB.PERSONNEL>
No loaded tables.

DTR> ! Show the fields for RESUMES:
DTR> SHOW FIELDS FOR RESUMES
RESUMES
   RESUMES
      EMPLOYEE_ID       <Character string>
      RESUME    <Segmented string>

DTR> PRINT RESUMES WITH EMPLOYEE_ID = 99800

EMPLOYEE
   ID
                                     RESUME

 99800
                Frank B. Harold
                1492 County Road
                Hicktown, US 54321

OBJECTIVE       Junior Lab Technician

EDUCATION       B.S. Chemical Engineering, Quinnipiac College, 1983.
                Hicktown Senior High School, 1979

DTR> LIST RESUMES WITH EMPLOYEE_ID = 99800

EMPLOYEE_ID : 99800
RESUME      :
                Frank B. Harold
                1492 County Road
                Hicktown, US 54321

OBJECTIVE       Junior Lab Technician

EDUCATION       B.S. Chemical Engineering, Quinnipiac College, 1983.
                Hicktown Senior High School, 1979
```

```
DTR>
```

Note

You cannot use relational operators or the SORTED BY clause in RSEs that refer to segmented string fields. See Section 5.9.4 on restrictions and usage notes for more information.

5.9.3 Storing and Modifying Segmented String Fields in DATATRIEVE

The unstructured nature of segmented strings requires different conventions in STORE or MODIFY statements than are used in updating other fields in DATATRIEVE.

In STORE or MODIFY statements that prompt for input, DATATRIEVE repeats the prompt for a segmented string field until you press the TAB key followed by the RETURN key in response to the prompt. Pressing only the RETURN key causes DATATRIEVE to redisplay the prompt for another segment of the field. (Pressing the TAB and RETURN keys after entering text also redisplays the prompt for another segment.)

```
DTR> STORE RESUMES
Enter EMPLOYEE_ID: 23456
Enter RESUME: This is the first line of the RESUME field. RET
Enter RESUME: This is the second line. RET
Enter RESUME: To end a segmented string, press the TAB key RET
Enter RESUME: then the RETURN key at the "Enter" prompt. RET
Enter RESUME:  TAB RET

DTR> ! Now print the RESUMES record just stored:
DTR> PRINT RESUMES WITH EMPLOYEE_ID = 23456

EMPLOYEE
   ID
                                     RESUME

 23456
This is the first line of the RESUME field.
This is the second line.
To end a segmented string, press the TAB key
then the RETURN key at the "Enter" prompt.

DTR>
```

In STORE or MODIFY statements with a USING clause, repeat Assignment statements within a BEGIN-END block for each line of the segmented string field:

```
DTR> MODIFY RESUMES WITH EMPLOYEE_ID = 23456 USING
CON> BEGIN
CON> RESUME = "This example modifies the RESUME field"
CON> RESUME = "of the same record we stored in the "
CON> RESUME = "previous example.  You use as many assignment"
CON> RESUME = "statements as you need.  End the modify or"
CON> RESUME = "store operation with an END statement."
CON> END
DTR> PRINT RESUMES WITH EMPLOYEE_ID = 23456

EMPLOYEE
   ID
                                     RESUME

 23456
This example modifies the RESUME field
of the same record we stored in the
previous example.  You use as many assignment
statements as you need.  End the modify or
store operation with an END statement.

DTR>
```

Note

You cannot store or modify only part of a segmented string. Although you create separate "segments" of the field with STORE or MODIFY statements, you cannot store or modify an individual segment. You must store the entire segmented string or none of it. In the same way, you cannot modify portions of a segmented string field. The MODIFY statement creates an entire segmented string and uses this to replace the old segmented string. It does not update an existing segmented string.

The maximum length of an Rdb/VMS or Rdb/ELN segmented string segment is 64K bytes. However, the maximum length of a segmented string segment that you can store in DATATRIEVE, interactively or with a procedure, is 255 characters. This is because 255 is the largest field DATATRIEVE can accept as screen input.

The previous examples showed that you can store or modify segmented string fields interactively with DATATRIEVE. You can also create a domain, record, and procedure to simplify storing entire files in a segmented string field. You can use the following domain and record definitions to store files in the segmented string field RESUMES:

```
DTR> SHOW TEMP_RESUME
DOMAIN TEMP_RESUME ! TEMP_RESUME is a temporary domain used to
                   ! associate the file you want to store in the
                   ! RESUME segmented  string field with a general
                   ! record definition that divides the file into
                   ! records that DATATRIEVE can store.
                   !
        USING TEMP_RESUME_REC ON
        TEMP_RESUME.DAT
                   ! TEMP_RESUME.DAT is the file you want to
                   ! store in the segmented string field.
;

DTR> SHOW TEMP_RESUME_REC
RECORD TEMP_RESUME_REC USING
01 TOP PIC X(255).  ! TEMP_RESUME_REC only has one
                    ! field of 255 characters (the maximum
                    ! size DATATRIEVE allows for a segment.)
                    ! Its only purpose is to separate the
                    ! file you wish to store in a segmented
                    ! string field into records. You can then
                    ! use DATATRIEVE to store the records as
                    ! individual segments in the segmented string.
;

DTR>
```

For example, suppose you want to add to the resume stored in the RESUMES record for employee number 99800. The following example displays the resume, then writes it to TEMP_RESUME.DAT:

```
DTR> PRINT RESUMES WITH EMPLOYEE_ID = 99800

EMPLOYEE
   ID
                                     RESUME

 99800
                Frank B. Harold
                1492 County Road
                Hicktown, US 54321

OBJECTIVE       Junior Lab Technician

EDUCATION       B.S. Chemical Engineering, Quinnipiac College, 1983.
                Hicktown Senior High School, 1979

DTR> PRINT RESUME (-) OF RESUMES WITH
DTR>     EMPLOYEE_ID = 99800 ON TEMP_RESUME.DAT
DTR>
```

Exit DATATRIEVE and make any changes you want to TEMP_RESUME.DAT:

```
$ ! You have edited TEMP_RESUME.DAT and this is how it looks:
$ TYPE TEMP_RESUME.DAT
                Frank B. Harold
                1492 County Road
                Hicktown, US 54321

OBJECTIVE       Junior Lab Technician

EDUCATION       B.S. Chemical Engineering, Quinnipiac College, 1983.
                Hicktown Senior High School, 1979

EMPLOYMENT      Chemistry Tutor, Quinnipiac College (9/82 - 5/83)
                Helped freshman chemistry students learn the
                concepts of atomic weights, valence and
                covalence bonding, and empirical formulas

REFERENCES      Available upon request

$ ! To store the file TEMP_RESUME.DAT into a segmented string,
$ ! you need to know the length of the longest record in the
$ ! file.  Use the command ANALYZE/RMS:
$ ANALYZE/RMS TEMP_RESUME.DAT

Check RMS File Integrity                    24-JUN-1985 06:28:28.09   Page 1
USER$DISK:[SERLE.RDBDEMO]TEMP_RESUME.DAT;2
        .
        .
        .

RMS FILE ATTRIBUTES

        File Organization: sequential
        Record Format: variable
        Record Attributes:   carriage-return
        Maximum Record Size: 255
        Longest Record: 62
        .
        .
        .

$
```

Finally, you can use a procedure, MODIFY_ANY_RESUME, to update the RESUME field for employee number 99800. The procedure takes the following actions:

- Readies the RESUMES relation of the PERSONNEL database for MODIFY access
- Readies TEMP_RESUME
- Creates a DATATRIEVE command file called SEGMENT.COM

- Prints statements in SEGMENT.COM to store the file TEMP_RESUME.DAT in the RESUME segmented string field:
  - A MODIFY USING statement that prompts the user for the employee number of the employee whose resume is to be updated.
  - An Assignment statement (RESUME =) for each line of the file TEMP_RESUME.DAT. The clause FORMAT (TOP) USING X(62) in the Assignment statement uses the length of the longest record, determined from the ANALYZE/RMS command, for the length of the segments in the segmented string.
- Executes the DATATRIEVE command file SEGMENT.COM

This example displays the procedure MODIFY_ANY_RESUME and executes it to store TEMP_RESUME.DAT in the RESUME field of employee number 99800:

```
DTR> SHOW MODIFY_ANY_RESUME
PROCEDURE MODIFY_ANY_RESUME
!
! Take the file named in the domain TEMP_RESUME
! and store it in the RESUME field of the record
! specified by the employee number entered by the user.
!
SET COLUMNS_PAGE = 132;
READY PERSONNEL SHARED MODIFY USING RESUMES;
READY TEMP_RESUME;
ON SEGMENT.COM
BEGIN
  PRINT "MODIFY RESUMES WITH EMPLOYEE_ID = *.'employee_id' USING BEGIN";
  FOR TEMP_RESUME
     PRINT "RESUME  ="||"'"||FORMAT (TOP) USING X(62)||"'";
  PRINT "END";
END;
@SEGMENT
COMMIT
FINISH TEMP_RESUME
END_PROCEDURE

DTR> :MODIFY_ANY_RESUME
Enter employee_id: 99800
DTR> PRINT RESUMES WITH EMPLOYEE_ID = 99800

EMPLOYEE
   ID
                                      RESUME

 99800
                Frank B. Harold
                1492 County Road
                Hicktown, US 54321
```

```
OBJECTIVE       Junior Lab Technician

EDUCATION       B.S. Chemical Engineering, Quinnipiac College, 1983.
                Hicktown Senior High School, 1979

EMPLOYMENT      Chemistry Tutor, Quinnipiac College (9/82 - 5/83)
                Helped freshman chemistry students learn the
                concepts of atomic weights, valence and
                covalence bonding, and empirical formulas

REFERENCES      Available upon request

DTR> COMMIT
DTR> FINISH
```

The DATATRIEVE command file SEGMENT.COM created by the procedure looks like this:

```
MODIFY RESUMES WITH EMPLOYEE_ID = *.'employee_id' USING BEGIN
RESUME  ="              Frank B. Harold"
RESUME  ="              1492 County Road"
RESUME  ="              Hicktown, US 54321"
RESUME  ="              "
RESUME  ="              "
RESUME  ="OBJECTIVE     Junior Lab Technician"
RESUME  ="              "
RESUME  ="              "
RESUME  ="EDUCATION     B.S. Chemical Engineering, Quinnipiac College, 1983."
RESUME  ="              Hicktown Senior High School, 1979"
RESUME  =""
RESUME  =""
RESUME  ="EMPLOYMENT    Chemistry Tutor, Quinnipiac College (9/82 - 5/83)"
RESUME  ="              Helped freshman chemistry students learn the"
RESUME  ="              concepts of atomic weights, valence and"
RESUME  ="              covalence bonding, and empirical formulas"
RESUME  =""
RESUME  ="REFERENCES    Available upon request"
END
```

Note

Source files for TEMP_RESUME that contain single quotation marks require special treatment. This is because the procedure MODIFY_ANY_RESUME uses single quotation marks as delimiters for the character string literals stored by the MODIFY USING statement. For the procedure to work, you must make sure the file contains two consecutive single quotation marks for every one you want stored.

5.9.4 Restrictions and Usage Notes for Segmented String Fields

The following restrictions and usage notes concern defining segmented string fields in RDO, SQL, or ERDL:

- When you define a segmented string field in RDO, SQL, or ERDL with the DATA TYPE IS SEGMENTED STRING clause, the SUB_TYPE designation, if any, is ignored by DATATRIEVE.
- You cannot specify a MISSING_VALUE IS clause when you define a segmented string field in RDO, SQL, or ERDL. This is an Rdb/VMS and Rdb/ELN restriction.
- You cannot specify a DEFAULT_VALUE FOR DTR clause when you define a segmented string field in RDO, SQL, or ERDL. DATATRIEVE gives a warning message when you ready the relation and any such default value is ignored.
- DATATRIEVE does allow the QUERY_HEADER FOR DATATRIEVE and the QUERY_NAME FOR DATATRIEVE clauses when you define a segmented string field in RDO, SQL, or ERDL.
- If you use an EDIT_STRING FOR DTR clause when you define a segmented string field in RDO, SQL, or ERDL, you can only specify a T edit string. If you use any other type of edit string, DATATRIEVE issues a warning message when you ready the relation containing the field. The length of the T edit string defaults to the current setting of COLUMNS_PAGE. (The default for DATATRIEVE is 80.)

  For example, if COLUMNS_PAGE is set to 80, the default edit string for DATATRIEVE is T(80). The T(80) edit string is also used if no EDIT_STRING has been defined in the RDO field definition.

  Therefore, the SET COLUMNS_PAGE command in DATATRIEVE can be used to change where a segment line breaks when printed if no edit string was defined for the field in RDO, SQL, or ERDL. (The change takes effect on READY, COMMIT, or ROLLBACK.) Changing the EDIT_STRING clause in the Rdb/VMS or Rdb/ELN field definition also changes the position where a segment line breaks when printed in DATATRIEVE.

The following restrictions and usage notes concern the use of segmented string fields within DATATRIEVE:

- When you ready a domain or relation containing segmented string fields, DATATRIEVE places the segmented string fields last in the relation, regardless of the position of the segmented string when the relation was defined in RDO, SQL, or ERDL. If more than one segmented string field is defined in a relation, the segmented string fields are placed last, in the order in which they were defined in the relation.

The SHOW FIELDS command displays the order of the fields within DATATRIEVE and denotes which fields are segmented string fields.

- A segmented string field has no data type. DATATRIEVE cannot convert or validate the contents of a segmented string field.
- If a segmented string segment is longer than 255 bytes, it is output in subsegments of 255 bytes that break at the COLUMNS_PAGE setting (or at the position in the T edit string, if one is specified in the field definition in RDO or SQL). If a segment is shorter than the current COLUMNS_PAGE setting, the segment is left-justified and blank-filled.
- When you use a PRINT statement to display a segmented string field, the header for the segmented string field begins on a separate line, following the header lines of fields of other data types. If more than one segmented string field is in the domain, each field has its own header. DATATRIEVE centers the header based on the current COLUMNS_PAGE setting.

  Each segment of a segmented string starts in the first column. The T edit string defined for each segmented string field (or the COLUMNS_ PAGE setting, if none is defined) controls where a segmented string line breaks. DATATRIEVE displays a blank line for a missing segmented string. Individual segments cannot be displayed separately.
- When you use a LIST statement to display a segmented string field, DATATRIEVE prints the segments of the field below the field header instead of beside it. DATATRIEVE displays a blank line for a missing segmented string.
- You cannot use the DISPLAY statement with a segmented string field.
- You cannot store segments larger than 255 bytes in a segmented string field with DATATRIEVE. This is due to a general DATATRIEVE limit on the number of characters that can be input on a line. If this segment exceeds 255, the segment will be truncated.
- You must enter all segments of a segmented string in a single STORE or MODIFY statement. Individual segments cannot be entered or retrieved separately.

  In STORE or MODIFY statements that prompt for input, DATATRIEVE repeats the prompt for each segment to be entered in the segmented string.

  Pressing the TAB key followed by the RETURN key in response to a prompt for a segmented string field terminates the segmented string if data has already been entered in response to previous segment prompts. Pressing the TAB key followed by the RETURN key in response to the initial prompt for a segmented string field results in nothing being stored in the field.

- You cannot modify portions of a segmented string field. The MODIFY statement creates an entirely new segmented string for a record. It does not update an existing segmented string.
- Data can be stored in segmented strings only in response to segment prompts or as character string literals assigned to a segmented string field in a STORE or MODIFY USING statement. You cannot assign values of fields or declared variables to a segmented string field.

  If data is input as a character string literal in a STORE or MODIFY USING statement, each character string represents one segment. The field name must be repeated for each character string literal being assigned to a segment of the segmented string field. As soon as a new field name is encountered, the string is terminated.
- You cannot retrieve or store segmented string fields from remote domains. When you ready a remote domain, DATATRIEVE ignores the segmented string fields and gives a warning message naming the fields. DATATRIEVE does ready the remote domain, however, and you can retrieve or store fields of other data types.
- Segmented string fields cannot be used with forms or plots.
- You cannot refer to segmented strings with the following elements of a record selection expression:
  - Relational operators
  - Boolean operators (AND, OR, NOT, BUT)
  - SORTED BY clause
  - REDUCED TO clause
  - CROSS clause

5.10 Modifying the Structure of Relational Sources

You cannot modify the structure of a relation or view relation using DATATRIEVE. If you want to add, change, or delete fields or indexes, you must restructure the Rdb/VMS, Rdb/ELN or VIDA database using techniques described in the documentation for those products.

5.11 Ensuring Data Security

Like any other domain, each relational domain has an associated DATATRIEVE access control list (ACL) that specifies access privileges. There is also an ACL associated with the database path name.

In addition, Rdb/VMS provides an access control list for each relation in the database. (Rdb/ELN and VIDA databases do not provide access control lists.) When you ready a domain, DATATRIEVE first checks the domain's ACL, then checks the database path ACL, and finally checks Rdb/VMS access privileges for the associated relation. Users are denied the requested access if any of these access control lists denies them the required privilege.

If you want the Rdb/VMS access control list to be the main means of access control for Rdb/VMS domains, you might consider opening up the DATATRIEVE ACLs to allow all users READ, WRITE, MODIFY, and EXTEND privileges. This allows users to "pass through" to the Rdb/VMS access control list.

If you are not using domains to access the database, the ACL associated with the database path name and the Rdb/VMS access control list are the only means of access control. In this case, you have to maintain only two access control lists.

5.12 Validating Data for Rdb/VMS or Rdb/ELN Relations and Domains

Keep in mind that when you are storing into or modifying fields whose definitions are common to more than one relation, you affect values only in the relations that you specify. You do not automatically change data values in the same fields of any other relations.

The EMPLOYEE_ID field definition, for example, is common to several relations in the PERSONNEL database. A user could assign an EMPLOYEE_ID value for Jack Jones in the EMPLOYEES relations and give him a different EMPLOYEE_ ID value in the JOB_HISTORY relation. You should design validation procedures to protect against such an occurrence. Design your database so that the minimum number of identical fields exist from one relation to another. Limit common fields to the keys you need in order to match records.

One way to help ensure that fields common to several relations remain consistent is to use the Rdb/VMS or Rdb/ELN DEFINE CONSTRAINT statement. Using RDO, SQL, or ERDL, you can define a constraint that requires, for example, that an EMPLOYEE_ID must already exist in the EMPLOYEES relation before it is stored in the JOB_HISTORY relation. If a user violates this constraint by entering an ID for a nonexistent employee, the relational database product returns an error and does not store the value.

Note

One option of the DEFINE CONSTRAINT statement specifies that the validation criteria is not evaluated until a COMMIT statement is issued. If a database is set up to check constraints at the time a COMMIT statement executes, validation errors can occur later than interactive DATATRIEVE users expect to receive them.

In addition, you can check data for validity with the VALID IF clause of the Rdb/VMS and Rdb/ELN DEFINE FIELD statement. When you define the field in RDO, SQL, or ERDL, you specify the range of values that the field must have by using the VALID IF clause. The relational database product returns an error when a user enters any value outside this range.

5.13 Optimizing Performance

To optimize DATATRIEVE performance, keep these points in mind when using DATATRIEVE with Rdb/VMS, Rdb/ELN, or VIDA:

- DATATRIEVE cannot use the Rdb/VMS or Rdb/ELN index structure to search the data contained in a collection. Data retrieval using keyed fields to search relations is faster than an exhaustive search of large collections. Avoid using the FIND statement unless the resulting collection contains a small number of records.

  Because of the loss of keyed retrieval, forming a new collection from another collection is also likely to be time-consuming. If a collection contains a small number of records, however, you may find that DATATRIEVE responds more quickly when you specify that collection as a record source.

- DATATRIEVE and the relational database products use different default settings for waiting for locked records. Using the DATATRIEVE default may cause Rdb/VMS or Rdb/ELN to generate error messages you do not expect. In DATATRIEVE, when your relational transaction encounters a locked record, the DATATRIEVE default setting (SET NO LOCK_WAIT) causes Rdb/VMS or Rdb/ELN to generate an error message and return you to the DTR> prompt.

  In Rdb/VMS or Rdb/ELN, the default setting (START_TRANSACTION WAIT) causes a transaction to wait until a locked record is released and then continue the operation.

  The following example shows the error message generated when a user tries to update a locked record of the sample EMPLOYEES domain with the default DATATRIEVE setting in effect:

```
DTR> MODIFY FIRST_NAME
Enter FIRST_NAME: Norman
%RDB-E-LOCK_CONFLICT, NO WAIT request failed because resource was locked
-RDMS-F-LCKCNFLCT, lock conflict on area 25
DTR>
```

To change the setting in DATATRIEVE, issue the SET LOCK_WAIT command. Instead of generating the error messages and returning to the DTR> prompt, your relational transaction waits until the data operation of the other user is completed and the record is released. Rdb/VMS implements adjustable locking levels, which allow faster access to records that are concurrently readied by several users. With adjustable locking, you can frequently access data readied by another user before that user's transaction ends.

Changing the default setting to LOCK_WAIT will affect performance in the sense that your transaction must wait for a locked record to be released. At the same time, however, the LOCK_WAIT setting allows your transaction to take advantage of adjustable locking levels and gives you greater access to data in the database. In addition, you do not lock other users out of the database as you do when you use the default NO LOCK_WAIT setting.

6
Accessing Remote Data

This chapter explains how to define network domains and access distributed domains.

6.1 Defining Network Domains and Accessing Remote Domains

With VAX DATATRIEVE, you can access domains defined on other systems linked to yours by the DECnet network. Each system must have DATATRIEVE installed.

The term **network domain** refers to the domain you define at your local node containing a network address. The term **remote domain** or **distributed domain** refers to the domain located at the remote node.

To access a remote domain, you must tell DATATRIEVE the network address of the remote domain. You can do that in one of two ways:

- You can include the network address of the remote domain in the READY command:

```
DTR> READY CDD$TOP.DEPT32.SMITH.PERSONNEL AT BIGVAX
```

- At your local node, you can define a domain (called a network domain) that contains the address of the remote domain. Then you ready the network domain at your local node just as you would any domain definition:

```
DTR> DEFINE DOMAIN REM_PERSONNEL
DFN>    USING CDD$TOP.DEPT32.SMITH.PERSONNEL
DFN>    AT BIGVAX"SMITH PASSMETHROUGH";
DTR> READY REM_PERSONNEL
```

When you ready a remote domain, either directly using the network address in the READY command or by readying a local domain that contains a network address, DATATRIEVE does the following:

- Invokes the DATATRIEVE Distributed Data Manipulation Facility (DDMF) at the remote node to process the DATATRIEVE statements that refer to the domain at that node
- When you finish the domain, it terminates the remote process

The DDMF keeps a trace file of your requests and its responses. It writes this file to the login directory of the remote process. If the remote node is a VAX/VMS system, the trace file is NETSERVER.LOG. If the remote node is a PDP–11 system, the trace file is DDMF.LOG.

If the DDMF is handling more than one domain, the remote process ends when you finish the last domain.

The following sections explain the process of defining network domains and how to access remote domains.

6.1.1 Defining Network Domains

To define a network domain, you define a DATATRIEVE domain at the local node that specifies the link with the domain definition at the remote node.

The following example defines a network domain for a domain on a remote VAX/VMS system:

```
DTR> DEFINE DOMAIN PERSONNEL
DFN>    USING CDD$TOP.DTR$USERS.CUVERDALE.PERSONNEL
DFN>    AT BIGVAX"CUVERDALE SESAME";
DTR>
```

See the *VAX DATATRIEVE Reference Manual* for the full syntax.

The following example defines a network domain for a domain on a remote PDP–11 system:

```
DTR> DEFINE DOMAIN CDD$TOP.DEPT39.PERSONNEL USING PERSONNEL
DFN>    AT ELEVEN"*.USERNAME *.PASSWORD";
DTR>
```

6.1.2 Accessing Remote Domains

As you have seen, you can access a remote domain by either of the following methods:

- Readying a network domain (the domains you learned to define in the previous section)
- Including the network address in a READY command

The next two sections show how to access a remote domain both ways.

6.1.2.1 Readying a Network Domain

Readying a network domain makes distributed processing transparent to the DATATRIEVE user. Depending upon how you specify login requirements in the network address of the domain definition, the user may have to enter a user name or password. The user need not be concerned with the actual location of the domain, however, as that is already defined in the network domain definition.

The following example readies the network domain REM_PERSONNEL, whose definition explicitly specifies user name and password in the network address:

```
DTR> READY REM_PERSONNEL
DTR>
DTR> SHOW REM_PERSONNEL
DOMAIN REM_PERSONNEL USING CDD$TOP.DEPT32.SMITH.PERSONNEL
AT BIGVAX"SMITH PASSMETHROUGH";
```

The following example readies the network domain REM_SALES whose definition specifies prompting value expressions for user name and password in the network address:

```
DTR> READY REM_SALES
Enter USERNAME: GREEB
Enter PASSWORD:
DTR>
DTR> SHOW REM_SALES
DOMAIN REM_SALES USING SALES AT ANODE"*.USERNAME *.PASSWORD";
DTR>
```

6.1.2.2 Readying a Remote Domain Directly

When you ready a remote domain directly, you specify the name of the remote domain and its network address in the READY command. The formats and results for specifying a network address in the READY command are the same as those for including the network address in a network domain definition.

The following example readies a remote domain on a VAX/VMS system. The command specifies a full dictionary path name and uses a default DECnet account in the network address:

```
DTR> READY CDD$TOP.DEPT32.PERSONNEL AT BIGVAX
DTR>
```

The next example readies a remote domain on a PDP–11 system. The network address specifies the login account assigned to the user VOJTEK. The prompting value expression for the password ensures that the password is not displayed on the terminal as it is entered:

```
DTR> READY PERSONNEL AT ELEVEN"VOJTEK *.PASSWORD"
DTR> Enter PASSWORD:
DTR>
```

6.1.3 Results of Accessing Remote Domains

There are some facts common to both ways of accessing remote domains. Whichever method you choose, the remote process running DATATRIEVE executes a login command file, just as you do when you log in to your local system. Depending on how you specify a network address, the remote process can log in to a specific account or it can log in to a default DECnet account.

For example, the command READY PERSONNEL AT BIGVAX does not specify a user name. Therefore, it starts a remote process using a default DECnet account. (The *VAX DATATRIEVE Installation Guide* explains how to set up a default DECnet account for DATATRIEVE on a VAX/VMS system.) In this case, the login procedure executes the login command file for the default DECnet account.

On the other hand, the READY PERSONNEL AT BIGVAX"SWAZY ITSME" command starts a remote process using the login account assigned to user Swazy. The login procedure executes the login command file for Swazy's account.

In addition to providing security information such as user name and password, you must make sure the remote process uses the correct dictionary and system directories when it invokes DATATRIEVE and readies the domain for you.

On a VAX/VMS system, the login command file for the account used by the remote process can include commands that set any defaults not included in the network address and needed by the remote process running DATATRIEVE. However, the login file should not execute commands that are appropriate for interactive mode and that might cause a network process to fail. (Assignments to TT: can fall into this category.)

The following example illustrates some helpful commands that you might want to include in a login file for an account on a remote VAX/VMS system. In the example, there are no commands preceding the first line that might cause a network process to fail.

```
$ IF F$MODE() .EQS. "NETWORK" THEN GOTO NETWORK_PROCESS
                    .
                    .
                    .
$ NETWORK_PROCESS:
$ SET DEFAULT [HAMOND.PERSONNEL.DATA]
$ DEFINE CDD$DEFAULT "CDD$TOP.HAMOND.PERSONNEL"
$ EXIT
```

The SET DEFAULT command moves the network process to the VMS directory that contains the data file. This is necessary if the login directory does not contain the data file and if the definition of the domain being readied does not contain a full file specification for the data file.

The DEFINE command sets the logical name CDD$DEFAULT to the dictionary containing the definition of the domain being readied. This is necessary if this information is not included in the network address.

The EXIT command exits the login command file so that subsequent commands inappropriate for a network process are not executed.

For a PDP–11 system, you can use a SET DICTIONARY command in a DATATRIEVE QUERY.INI file to access the dictionary directory you want to use when the remote process invokes DATATRIEVE. This is unnecessary, of course, if the dictionary file containing the domain definition is in the login directory.

6.1.4 Restrictions on Using Remote Domains

When you access data located on remote systems, note the following restrictions:

- Using a simple DATATRIEVE domain that includes a remote node name in the specification for the data file can slow DATATRIEVE response time. Avoid using this method to access data on other systems. Instead, use the methods explained in this chapter to ensure optimum DATATRIEVE performance.
- The CONTAINING operator and the STARTING WITH operator do not work with a prompt for remote domains. For example, the following PRINT statement does not work:

  ```
  PRINT REMOTE_YACHTS WITH BUILDER CONTAINING *.BUILDER
  ```

  DATATRIEVE uses a fixed-length field to transmit the prompt value to the remote server. Trailing spaces are appended to any value shorter than the length of this field.
- You cannot use a Boolean expression containing the relational operators IN or ANY in a record selection expression with a remote domain or collection as its source.
- You can use remote domains or collections in a CROSS clause only if both of the following conditions are met:
  - The remote domains or collections reside on the same remote node.
  - You access the remote domains or collections with the same account, user name, and password.
- When validation checks are made for data entered in response to a prompt, distributed DATATRIEVE does not reprompt when validation errors occur. A validation error causes the statement being processed to abort.

- In a record selection expression with a remote domain as its source, you cannot use value expressions that require computations involving remote data.
- If you access a DATATRIEVE domain containing COMPUTED BY fields from a remote domain and execute a SHOW FIELDS command on the domain, the computed fields are identified as character strings. In addition, DATATRIEVE may also improperly format the printed output of domains that include such fields.

  You can avoid the printed output problems by explicitly defining the COMPUTED BY fields with EDIT_STRING clauses. With edit strings explicitly specified in the record definition, DATATRIEVE produces identical formatting at both the local and the remote nodes.
- You cannot ready a remote relational database by specifying a database path name in a READY command that uses the AT syntax. You must first create domain definitions on the remote node for each relation you want to access. You can then ready the database using these domains.

  The following example does not work:

```
DTR> READY CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL AT
[Looking for Node Specification]
CON> DEPT42"SMITH PASSWORD"
[DDMF] You can not READY a database with an alias.
[DDMF] (Statement abandoned due to error.)
```

  If you define a DATATRIEVE domain on the remote node for each relation in PERSONNEL, you can ready and access each relation in the remote relational database PERSONNEL. In the following example, the user defines a DATATRIEVE domain on the remote node DEPT42 for the relation SALARY_HISTORY in the database PERSONNEL.

```
DTR> DEFINE DOMAIN SALARY_HISTORY USING SALARY_HISTORY OF
DFN> DATABASE CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL;
DTR>
```

  Then the user can ready the remote domain from the local node:

```
DTR> READY SALARY_HISTORY AT DEPT42"SMITH PASSWORD"
```

6.1.5 Accessing IBM Data from a VAX

VAX users can access data that resides on an IBM system. The VIDA facility provides VAX/VMS users with transparent access to data on IBM systems. Users retrieve IBM data through VIDA with DATATRIEVE, RDO, SQL, or a third generation language supported by a relational database data manipulation language precompiler. Chapter 5 contains information about defining and accessing VIDA databases. For additional information, see the VIDA documentation.

Index

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

V

W