VAX DATATRIEVE

Reference Manual

Order Number: AA–K079G–TE

May 1992

This manual contains general information on using VAX DATATRIEVE.

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

2 Using DATATRIEVE Variables

3 DATATRIEVE Functions

4 DATATRIEVE Commands, Statements, and Definition Clauses

Index

Figures

Tables

Preface

This manual describes the VAX DATATRIEVE software and provides reference information for terms, concepts, syntax elements, commands, statements, and definition clauses.

Intended Audience

This manual assumes you have a working knowledge of DATATRIEVE, or you know the basic concepts of data processing and are familiar with the VMS operating system.

If you have no prior experience with DATATRIEVE, the *VAX DATATRIEVE User's Guide* describes the tasks of managing information with DATATRIEVE.

Operating System Information

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

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 Release Notes*

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

- *VAX DATATRIEVE Installation Guide*

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

- *VAX DATATRIEVE User's Guide*

  Describes how to use VAX DATATRIEVE interactively.

- *VAX DATATRIEVE Guide to Interfaces*

  Includes information on using VAX DATATRIEVE to manipulate data and on using VAX DATATRIEVE with forms, relational databases, and database management systems.

- *VAX DATATRIEVE Guide to Programming and Customizing*

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

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

Syntax Diagrams

Table 1 explains the notation used in syntax diagrams.

Table 1 Notation Used in Syntax Diagrams

Element	Meaning	Do You Enter It?
WORD	An uppercase word is a keyword.[1]	Yes
word	A lowercase word indicates a syntax element.	Yes
separators (punctuation)	Separators are used to separate words (space), separate items you are listing (comma)[2], or tell DATATRIEVE you are finished with a clause (period) or a statement or command (semicolon).	Yes
{ } (braces)	Braces enclose a clause from which you must choose one alternative.	No
[ ] (brackets)	Brackets enclose optional clauses from which you can choose one or none.	No
... (horizontal ellipsis)	A horizontal ellipsis indicates you can repeat the part of the clause, statement, command, or expression immediately to the left of the ellipsis.	No
. . (vertical . ellipsis)	A vertical ellipsis indicates you can repeat the line of the clause, statement, command, or expression immediately above the ellipsis.	No

[1]Major keywords, such as READY, STORE, and FIND are always a required part of the input. Minor keywords, such as USING, AS, and ALL, sometimes let DATATRIEVE know what to expect next. In this case they are a required part of your input. Minor keywords can also be included to make input more like English. In this case they are enclosed in square brackets and you can omit them from your input. Do not use any DATATRIEVE keywords to name something you define. If you do, you might get either an error message or unexpected results.

[2]You are recommended to leave a space after a comma. A comma preceding a string of editing characters could be interpreted as part of the string if there were no intervening space.

1

Value Expressions and Boolean Expressions

This chapter describes the value expressions and Boolean expressions used in VAX DATATRIEVE statements.

A **value expression** is a string of symbols that specifies a value DATATRIEVE can use when executing statements. DATATRIEVE provides you with value expressions of the following types:

- Character string literals
- Numeric literals
- Qualified field names
- Local and global variables
- Date value expressions
- Prompting value expressions
- Values from dictionary and domain tables
- FROM value expression
- CHOICE value expression
- IF-THEN-ELSE value expression
- FORMAT value expression
- Statistical expressions
- Arithmetic expressions
- Concatenated expressions

In addition, certain DATATRIEVE functions are value expressions. They are described in Chapter 3.

A **Boolean expression** is the logical representation of a relationship between value expressions. The value of a Boolean expression is either true or false.

The Boolean expressions used in DATATRIEVE consist of value expressions, relational operators, and Boolean operators. The **relational operators** control the comparison of value expressions. The **Boolean operators** enable you to join two or more Boolean expressions and to reverse the value of a Boolean expression.

1.1 Value Expressions

Value expressions specify values that DATATRIEVE uses when executing statements.

1.1.1 Literals

The simplest way to specify a value is with a literal. A **literal** is either a character string enclosed in quotation marks or a number.

1.1.1.1 Character String Literals

A **character string literal** is a string of printing characters up to 253 characters long. The maximum size for an input line in DATATRIEVE is 255 characters, but in character string literals, two of those characters are used for the quotation marks. The printing characters consist of the uppercase and lowercase letters, numbers, and the following special characters:

! @ # $ % ^ & * ( ) - _ = + ‘ [

{] } ~ ; : ’ " \ | , < . > / ?

To type a literal on more than one line, enter a hyphen immediately before pressing the RETURN key. DATATRIEVE strips the hyphen from that part of the character string literal and waits for you to complete the literal by typing the closing quotation mark. As long as the total number of characters in the literal does not exceed 253, you can use any number of continuation characters between the quotation marks.

The maximum length of character strings used as the values of variables or fields is independent of the maximum length of character string literals. However, to assign as a value a character string exceeding 253 characters in length, you must use one of two methods:

- You can use concatenated expressions (described in the section in this chapter on concatenated expressions).
- For field values, you can define a very long field as a group field composed of elementary fields up to 253 characters long, and then store character string values in each elementary field.

Use pairs of either single or double quotation marks to enclose character string literals. You must use the same type of quotation mark to end a character string literal as you used to begin it.

To include one type of quotation mark in a character string literal, enclose the literal in quotation marks of the other type. For example, to include double quotation marks in a character string literal, enclose the character string in single quotation marks. In such cases, you can include unpaired quotation marks in the literals.

To include a quotation mark in a character string literal enclosed by the quotation marks of the same type, you must type two consecutive quotation marks for every one you want to include in the literal.

Table 1–1 shows some examples of character string literals and their values.

Table 1–1 Character String Literals and Their Values

Character String Literal	Value of String
"Invalid value for this field"	Invalid value for this field
"MAXIMUM PRICE IS $1400.- PLEASE RE-ENTER PRICE."	MAXIMUM PRICE IS $1400. PLEASE RE-ENTER PRICE.
"Capt. Jack says, ""Time to reorder.""""	Capt. Jack says, "Time to reorder."
"They said, ""We're going.""""	They said, "We're going."
'They said, "We''re going."'	They said, "We're going."

Although DATATRIEVE usually converts all lowercase letters of your input to uppercase, it preserves lowercase letters in character string literals. Because the case of the character string literals is preserved, comparisons using these literals are case sensitive.

1.1.1.2 Numeric Literals

A **numeric literal** is a string of digits that DATATRIEVE interprets as a decimal number. A numeric literal may contain a decimal point and up to 31 digits. The decimal point is optional and is not counted in the maximum number of digits.

A numeric literal can begin with a decimal point. Thus, for example, .5 is a valid numeric literal.

A numeric literal cannot end with a decimal point. For example, 123. is not a valid numeric, but 123.0 is.

If you use a numeric literal to assign a value to a field or a variable, the data type of the field or variable controls the maximum value you can assign.

Restrictions

- For a field or variable that belongs to the data type COMP, the limit on the number of digits you can specify in the PICTURE (PIC) clause is 18. If you specify a number of digits greater than 18 in the PIC clause, DATATRIEVE displays an error message.
- For a field or variable that belongs to the data type LONG, the largest number you can assign to that field or variable is 2,147,483,647. Any larger number with 31 digits or less results in a truncation error. The smallest number you can assign is -2,147,483,648. Any smaller number with less than 31 digits results in a truncation error.
- For all other numerical data types, the limit on digits you can assign and display is 31.
- The format specified in the PIC clause of a field or variable also limits the values you can assign with numeric literals. See the section on the PICTURE clause in Chapter 4 for more information.

1.1.2 Qualified Field Names

You can use elementary field names, virtual field names, and group field names as value expressions.

1.1.2.1 Elementary and REDEFINES Field Names

The value specified by an elementary field name is the value stored in a field of a record. If the field name you use refers to a REDEFINES field, the value associated with the name is determined by the clauses that define the REDEFINES field in the record definition.

1.1.2.2 COMPUTED BY Fields

COMPUTED BY fields are **virtual fields**. DATATRIEVE does not store the value of a COMPUTED BY field in a record. That value is calculated every time you refer explicitly or implicitly to a COMPUTED BY field.

The COMPUTED BY clause includes a value expression. The value specified by the virtual field is associated with the value expression in the COMPUTED BY clause. See the section on COMPUTED BY in Chapter 4 for more information.

1.1.2.3 Group Field Names

When you use group field names in assignment statements, the value of the group field depends on the type of assignment you specify. For example:

- group-field-name-1 = group-field-name-2

  In this type of assignment, the value of group-field-name-2 includes the values of all elementary fields, all REDEFINES fields, and all COMPUTED BY fields. The assignment of values is made on the basis of similar field names. For field names in group-field-1 that match names in group-field-2, DATATRIEVE assigns the values in group-field-2 to the appropriate fields in group-field-1.

  DATATRIEVE ignores any fields in group-field-2 whose names do not match any field names in group-field-1. For any elementary field in group-field-1 whose name matches none of the field names in group-field-2, DATATRIEVE assigns the MISSING or DEFAULT value if one is defined for the field, or a value of 0 if the field is numeric, or a blank if the field is alphabetic or alphanumeric.

  DATATRIEVE stores no values in any COMPUTED BY or REDEFINES fields in group-field-1, regardless of any matches between the names of those fields and fields of any type in group-field-2.

  The values of the elementary, REDEFINES, and COMPUTED BY fields associated with group-field-name-2 are the values stored in or associated with the fields that constitute group-field-2 of a record.

- elementary-field-name = group-field-name

  Because of problems that can arise from conflicting data types, assignments of this type are not recommended.

  If you make an assignment of this type, the value of the group field is the same value displayed when you enter a DISPLAY group-field-name statement. The value is the concatenation of the values in the elementary fields that constitute the group field. The REDEFINES and COMPUTED BY fields are ignored. You can reduce conflicts of data types by using an alphanumeric PICTURE string in the definition of the elementary field.

1.1.2.4 Query Names

If the record definition contains a query name for a field, you can use the query name exactly as you use the associated field name.

1.1.2.5 Qualifying Field Names

To clarify the context for recognizing names and associating values with those names, you can qualify field names with several optional elements:

- Context variables

  Distinguish between fields in different record streams.

- Collection names

  Distinguish between fields in selected records in different collections. You can use a collection name to modify a field from a selected record only.

- Domain names

  Distinguish between fields that have the same field name but are associated with different domains.

- Record names

  Distinguish between fields that have the same field name but are contained in different records.

- Group field names

  Distinguish between fields that have the same field name but are contained in different group fields.

Use the following format to specify field names:

[collection-name | context-variable] [domain-name] [group-field-name] [...] field-name

For duplicate field names, you must specify whatever option or combination of options is needed to make the name unique.

1.1.3 Variables

The DECLARE statement defines global and local variables for use as value expressions. When DATATRIEVE initializes a variable, it assigns the variable the MISSING VALUE or DEFAULT VALUE if one is specified in the DECLARE statement. DATATRIEVE assigns a value of zero to numeric variables and a space to alphabetic and alphanumeric variables if no MISSING VALUE or DEFAULT VALUE is specified.

1.1.3.1 Global Variables

You can define global variables only with DECLARE statements entered in response to the DTR> prompt of DATATRIEVE command level.

You can use a global variable as a value expression in any DATATRIEVE statement. Unless you define a global variable with a COMPUTED BY clause, the global variable retains the value you assign to it until you either assign it a new value or release it with the RELEASE command.

The value of a global variable defined with a COMPUTED BY clause depends on the value expression that controls the computation. For example, you can declare the value of the variable to be 1.2 times the price of a boat in the YACHTS domain. The value of the variable changes according to the value of the PRICE field for different records:

```
DTR> READY YACHTS
DTR> DECLARE VAR COMPUTED BY PRICE * 1.2.
DTR> FOR FIRST 5 YACHTS PRINT VAR USING $$$,$$$.99

   VAR

$44,341.20
$21,480.00
$33,000.00
$22,320.00
$11,874.00

DTR>
```

1.1.3.2 Local Variables

You can define local variables with DECLARE statements entered in BEGIN-END and THEN statements.

A local variable is released as soon as DATATRIEVE completes the execution of the clause or statement in which it was declared. Although a local variable stays in effect for subsequent statements of the compound statement in which it is declared, it has no meaning in any outer statements containing that compound statement.

The following example illustrates how a local variable declared in an inner statement can supersede one with the same name declared in an outer statement. Notice, however, that the value of the variable in the outer statement is not affected by the different data type or different value assigned to the inner one. Note also that neither local variable exists when DATATRIEVE completes the execution of the compound statements containing them both:

```
DTR> SET NO PROMPT
DTR> BEGIN
CON>    DECLARE X PIC XXX.
CON>    X = "TOP"
CON>    PRINT X
CON>            BEGIN
CON>               DECLARE X PIC 9V99 EDIT_STRING 9.99.
CON>               X = 1.23
CON>               PRINT X
CON>            END
CON>    PRINT X
CON> END

 X

TOP

 X

1.23

 X

TOP

DTR> PRINT X
Field "X" is undefined or used out of context
DTR>
```

To avoid problems resolving names for variables and fields, do not use variable names that duplicate field names of domains you have readied.

1.1.4 Date Value Expressions

If you define a field or a variable with a USAGE DATE clause, then you can assign a value with one of the four DATATRIEVE date value expressions:

- "TODAY" returns the value of the current system date.
- "NOW" returns the value of the current system date and time.
- "YESTERDAY" returns the value of one day before the current date.
- "TOMORROW" returns the value of one day after the current date.

Note that "NOW" is the only value expression that returns the time as well as the date. You can use the function FN$DATE to assign a date field a time that is not current.

The following DATATRIEVE session illustrates these expressions:

```
DTR> DECLARE X USAGE DATE.
DTR> X = "TODAY"
DTR> PRINT X

     X

28-Dec-1990

DTR> X = "TOMORROW"
DTR> PRINT X

     X

29-Dec-1990

DTR> X = "YESTERDAY"
DTR> PRINT X

     X

27-Dec-1990

DTR> DECLARE Y PIC X(23).
DTR> X = "NOW"
DTR> Y = X
DTR> PRINT Y

        Y

28-Dec-1990 13:35:11.54

DTR>
```

You can add or subtract dates. For example, you might want to know how many days you have to complete a project. Define variables for today's date and the project date. Then subtract today's date from the project due date:

```
DTR> DECLARE T USAGE DATE.
DTR> T = "TODAY"
DTR> DECLARE PROJECT_DUE DATE.
DTR> PROJECT_DUE = "21-AUG-90"
DTR> PRINT (PROJECT_DUE - T)
                      112

DTR>
```

DATATRIEVE indicates that the project is due in 112 days.

To use these value expressions, you must assign the DATE data type to the field or variable. Otherwise, DATATRIEVE treats the expression as a character string literal. For example:

```
DTR> PRINT "TODAY"
TODAY

DTR>
```

DATATRIEVE returns the value “TODAY” because the quoted expression is not associated with a variable or field of the DATE data type. However, if you supply an edit string containing date edit string characters, DATATRIEVE returns the current system date in the form specified by the edit string:

```
DTR> PRINT "TODAY" USING DD-MMM-YY
18-Jan-90

DTR>
```

1.1.5 Prompting Value Expressions

If you want DATATRIEVE to prompt you for a value, use a prompting value expression. This feature is especially useful in a procedure because it allows you to use a different value each time you invoke the procedure. DATATRIEVE recognizes two types of prompting expressions: the *.prompt and the **.prompt.

The *.prompt value expression has the following format:

```
*."prompt-name"
```

The prompt name is a character string literal. If the prompt name contains no blanks and conforms to the rules for DATATRIEVE names, you do not have to enclose the literal in quotation marks. If the prompt name contains blanks or does not conform to those rules, you must enclose it in quotation marks.

If you put a *.prompt value expression in a REPEAT loop or a FOR loop, DATATRIEVE prompts you for a value each time it executes the loop.

The **.prompt value expression has the following format:

```
**."prompt-name"
```

The same rules govern the use of quotation marks for the literal in the prompt name.

If you put a **.prompt value expression in a REPEAT loop or a FOR loop, DATATRIEVE prompts you for a value only the first time it executes the loop. If your **.prompt value expression assigns a value to a variable or a field, DATATRIEVE uses the same value each time through the loop. This feature is especially useful when storing or modifying a group of records that have a common value in one or more fields.

The following example shows the difference between the *.prompt and **.prompt value expressions:

```
DTR> READY PHONES WRITE
DTR> REPEAT 2
CON>    STORE PHONES USING
CON>    BEGIN
CON>            DEPARTMENT = **.DEPARTMENT
CON>            LOCATION = **.LOCATION
CON>            NAME = *.NAME
CON>            NUMBER = *.NUMBER
CON>    END
Enter DEPARTMENT: CED
Enter LOCATION: MK3
Enter NAME: Gardens, Marvin
Enter NUMBER: 555-1776
Enter NAME: D'Ecor, Espree
Enter NUMBER: 555-1812
DTR>
```

1.1.6 Values from a Table

You can use a value stored in a dictionary table or a domain table anywhere the syntax of a DATATRIEVE statement allows a value expression.

The format for retrieving a value from a dictionary table or a domain table is as follows:

```
value-expression VIA table-name
```

If the value expression you specify is stored as a code string in the table you specify, the value of the entire expression is the corresponding translation string in the table.

If the table contains an ELSE clause and the value expression you specify does not match any code string in the table, the value of the entire expression is the translation string stored in the ELSE clause of the table.

If the table contains no ELSE clause and the value expression you specify does not match any code string in the table, DATATRIEVE displays the following message:

```
Value not found from record or table.
```

See the *VAX DATATRIEVE User's Guide* for more information about the creation and use of dictionary tables and domain tables.

1.1.7 Statistical Expressions

Statistical expressions compute values based on a value expression evaluated for each record in a record stream.

1.1 Value Expressions

Format

To specify the average, maximum, minimum, standard deviation, or total:

{ AVERAGE | MAX | MIN | STD_DEV | TOTAL } value-expression [OF rse]

To specify the count:

```
COUNT  [OF rse]
```

To specify the running count:

```
RUNNING COUNT
```

To specify the running total:

```
RUNNING TOTAL value-expression
```

Arguments

value-expression

Is a DATATRIEVE value expression on which the statistical function operates.

OF rse

Is a record selection expression you can use to form a record stream of the records to which the statistical function applies.

Table 1–2 shows the operations performed by the DATATRIEVE statistical functions.

Table 1–2 Values Derived with Statistical Functions

Function	Value of Function
AVERAGE	The average value of the value expression
COUNT	The number of records in the CURRENT collection or in a specified collection or record stream
MAX	The largest value of the value expression
MIN	The smallest value of the value expression
RUNNING COUNT	The running count of the evaluations of the PRINT statement
RUNNING TOTAL	The running total of the value expression for each evaluation of the PRINT statement

(continued on next page)

Table 1–2 (Cont.) Values Derived with Statistical Functions

Function	Value of Function
STD_DEV	The standard deviation of the value expression
TOTAL	The total value of the value expression

Restrictions

- If you leave out the OF rse clause, you must have a current collection of records to which the value expression applies.
- Except for COUNT and RUNNING COUNT, you must supply a value expression when using the statistical functions. The value expression you usually supply is a field name or a variable name.
- Do not supply a value expression when using COUNT or RUNNING COUNT.
- Note that DATATRIEVE does not reset RUNNING COUNT and RUNNING TOTAL inside a compound statement:

```
DTR> FOR FIRST 3 YACHTS
CON>   BEGIN
CON>     PRINT "outside running count", RUNNING COUNT, BUILDER
CON>     FOR FIRST 3 YACHTS
CON>        PRINT "inside running count", RUNNING COUNT, BUILDER
CON>   END
```

  In this example, DATATRIEVE does not reset the running count for “inside running count.”

Results

- If you supply a record selection expression, DATATRIEVE uses all the records in the resulting record stream to compute the value of the function.
- If you do not supply a record selection expression in the OF clause, DATATRIEVE uses all records in the current collection to compute the value of the function.

Usage Notes

- When you specify two or more statistical expressions in the same PRINT statement, include an OF rse clause with each expression to identify the record stream relating to each statistical function. In the following example, the average is computed for the current collection because no RSE is specified, but the total is computed for the first five yachts only:

```
DTR> READY YACHTS
DTR> FIND YACHTS
[113 records found]
DTR> PRINT AVERAGE LOA, TOTAL LOA OF FIRST 5 YACHTS

AVERAGE  TOTAL
LENGTH   LENGTH
 OVER     OVER
  ALL     ALL

  30.        146
```

In the following case, the RSE is specified for both average and total. As a result, the statistical functions compute the average LOA and total LOA for the first five YACHTS records.

```
DTR> PRINT AVERAGE LOA OF FIRST 5 YACHTS,
CON>      TOTAL LOA OF FIRST 5 YACHTS

AVERAGE  TOTAL
LENGTH   LENGTH
 OVER     OVER
  ALL     ALL

  29.        146

DTR>
```

- When a value expression is more complex than a field name or variable name, enclose the expression with parentheses. For example:

```
DTR> PRINT TOTAL (BEAM + LOA) OF FIRST 5 YACHTS

  TOTAL

      194

DTR> PRINT AVERAGE (BEAM/2) OF FIRST 5 YACHTS

AVERAGE

   4.800

DTR>
```

- When you include a statistical expression in a PRINT statement or in Report Writer PRINT or AT statements, DATATRIEVE forms a column header by combining the name of the statistical function and the field name.
- When DATATRIEVE uses a statistical function to calculate a value based on the values of fields or variables, it does not include those values specified as MISSING values in the record definition or in the DECLARE statement. When DATATRIEVE omits records from a calculation because they have MISSING values in the specified field, it displays a message between the column header and the value. For example:

```
DTR> FINISH
DTR> READY YACHTS
DTR> FIND YACHTS
[113 records found]
DTR> PRINT AVERAGE PRICE

AVERAGE
 PRICE

[Function computed using 50 of 113 values.]
$25,388

DTR>
```

- When you specify statistical expressions, you can use them anywhere the syntax of a DATATRIEVE statement allows a value expression. A statistical expression can be the value expression that is part of another statistical expression.
- When DATATRIEVE encounters a RUNNING TOTAL or RUNNING COUNT, it increments it for each occurrence of a list field and for each iteration of a loop. For example:

```
DTR> FOR FIRST 2 FAMILIES
CON> BEGIN
CON>   PRINT SKIP 1, "Outer Running Count", RUNNING COUNT
CON>   PRINT PARENTS, SKIP 1
CON>   FOR KIDS
CON>     PRINT "Inner Loop", RUNNING COUNT,
CON>       EACHKID, RUNNING TOTAL AGE
CON> END

                    RUNNING
                     COUNT
Outer Running Count      1

  FATHER     MOTHER

JIM         ANN

                                    RUNNING
            RUNNING    KID           TOTAL
             COUNT     NAME    AGE   AGE

Inner Loop       1  URSULA      7        7
Inner Loop       2  RALPH       3       10

Outer Running Count      2

JIM         LOUISE
```

```
Inner Loop      3  ANNE       31       41
Inner Loop      4  JIM        29       70
Inner Loop      5  ELLEN      26       96
Inner Loop      6  DAVID      24      120
Inner Loop      7  ROBERT     16      136
```

- The TOTAL function is the only statistical function for which the default edit string of the field is not used. This is the expected behavior. The field's edit string is not used because the value generated by TOTAL is likely to be quite a bit larger than the individual field values. This larger value could overflow the edit string associated with the field. You can provide an edit string for the totaled value by specifying an edit string on the print statement as follows:

  ```
  DTR> PRINT TOTAL PRICE OF CURRENT USING $$$$$.99
  ```

Examples

AVERAGE

```
DTR> PRINT AVERAGE PRICE OF YACHTS WITH BUILDER = "AMERICAN"

AVERAGE
 PRICE

$14,395

DTR>
```

COUNT

```
DTR> PRINT COUNT OF FAMILIES WITH NUMBER_KIDS EQ 2 USING 9

COUNT

  6

DTR>
```

MAX

```
DTR> PRINT FAMILIES WITH
DTR>   NUMBER_KIDS EQ MAX NUMBER_KIDS OF FAMILIES

                     NUMBER    KID
  FATHER     MOTHER    KIDS     NAME    AGE

BASIL      MERIDETH      6   BEAU       28
                             BROOKS     26
                             ROBIN      24
                             JAY        22
                             WREN       17
                             JILL       20

DTR>
```

MIN

```
DTR> PRINT YACHTS WITH PRICE EQ MIN PRICE

                              LENGTH
                               OVER
MANUFACTURER    MODEL   RIG    ALL   WEIGHT BEAM  PRICE

 VENTURE       21       SLOOP   21     1,500  07   $2,823

DTR>
```

RUNNING COUNT

```
DTR> PRINT RUNNING COUNT, TYPE, BEAM, LOA OF YACHTS WITH
CON>    BUILDER = "PEARSON"

                                        LENGTH
RUNNING                                  OVER
 COUNT  MANUFACTURER   MODEL    BEAM   ALL

     1   PEARSON     10M         11    33
     2   PEARSON     26          08    26
     3   PEARSON     26W         09    26
     4   PEARSON     28          09    28
     5   PEARSON     30          09    30
     6   PEARSON     35          10    35
     7   PEARSON     36          11    37
     8   PEARSON     365         11    36
     9   PEARSON     39          12    39
    10   PEARSON     419         13    42

DTR>
```

RUNNING TOTAL

```
DTR> FOR FIRST 5 YACHTS PRINT TYPE, PRICE,
CON>    RUNNING TOTAL PRICE USING $$$$,$$$

                                 RUNNING
                                  TOTAL
MANUFACTURER   MODEL     PRICE    PRICE

 ALBERG      37 MK II   $36,951  $36,951
 ALBIN       79         $17,900  $54,851
 ALBIN       BALLAD     $27,500  $82,351
 ALBIN       VEGA       $18,600 $100,951
 AMERICAN    26          $9,895 $110,846

DTR>
```

STD_DEV

```
DTR> PRINT STD_DEV PRICE OF YACHTS USING $$$,$$$

STANDARD
DEVIATION
  PRICE

[Function computed using 50 of 113 values.]
$15,480

DTR>
```

TOTAL

```
DTR> PRINT TOTAL PRICE OF YACHTS WITH
CON>     BUILDER = "CHALLENGER" USING $$$$,$$$

 TOTAL
 PRICE

$122,278

DTR>
```

1.1.8 Arithmetic Expressions

An arithmetic expression consists of value expressions and arithmetic operators. The value expressions must all be numeric. You can use an arithmetic expression anywhere the syntax of a DATATRIEVE statement allows a value expression.

DATATRIEVE provides four arithmetic operators. Table 1–3 shows the arithmetic operators and the operation each performs.

Table 1–3 Arithmetic Operators

Operator	Operation
+	Addition
–	Subtraction or negation
*	Multiplication
/	Division

You do not have to use spaces to separate arithmetic operators from value expressions, except in one case: if a DATATRIEVE name precedes a minus sign, you must put a space before the minus sign. Otherwise, DATATRIEVE interprets the minus sign as a hyphen and converts it to an underscore:

```
DTR> DECLARE X PIC 99.
DTR> X = 8
DTR> PRINT X-1
"X_1" is undefined or used out of context
DTR> PRINT 1-X
  -7

DTR>
```

You can use parentheses to control the order in which DATATRIEVE performs arithmetic operations. DATATRIEVE follows the normal rules of precedence when evaluating arithmetic expressions:

1. DATATRIEVE first evaluates any value expressions in parentheses.
2. DATATRIEVE then performs multiplications and divisions from left to right in the arithmetic expression.
3. Finally, DATATRIEVE performs additions and subtractions from left to right in the arithmetic expression.

The following examples show how DATATRIEVE evaluates arithmetic expressions:

```
DTR> PRINT (6 * 7) + 5
  47

DTR> PRINT 6 * (7 + 5)
  72

DTR> PRINT 6 + 7 * 5
  41

DTR> PRINT 12 - 6 * 2
   0
DTR> PRINT 5 + 10 / 2
   10

DTR>
```

1.1.9 Concatenated Expressions

DATATRIEVE allows you to join value expressions to form a concatenated expression. You can use a concatenated expression anywhere the syntax of a DATATRIEVE statement allows a character string literal. When DATATRIEVE concatenates value expressions, it converts their values to character string literals.

DATATRIEVE provides three types of concatenated expressions:

- value-expression | value-expression
- value-expression | | value-expression

- value-expression | | | value expression

In each case, DATATRIEVE converts the values of each value expression to a character string literal and joins the literals to form a longer literal. The differences among the three forms of concatenated expression lie in the way they treat trailing spaces of the first literal, and whether they add any spaces between the literals:

- A single bar leaves the literals as they are. For example:

```
"ABC"|"DEF"    "ABC  "|"DEF"    "ABC"|"  DEF"    "ABC  "|" DEF"
 ABCDEF         ABC  DEF         ABC  DEF         ABC    DEF
```

- A double bar suppresses trailing spaces of the first literal and does nothing to the leading spaces of the second literal. For example:

```
"ABC"||"DEF"   "ABC  "||"DEF"   "ABC"||"  DEF"  "ABC  "||"  DEF"
 ABCDEF         ABCDEF           ABC  DEF        ABC  DEF
```

- A triple bar suppresses trailing spaces of the first literal, inserts one space, and does nothing to the leading spaces of the second literal. For example:

```
"ABC"|||"DEF"  "ABC  "|||"DEF" "ABC"|||"  DEF"  "ABC  "|||"  DEF"
 ABC DEF        ABC DEF         ABC   DEF        ABC   DEF
```

You can use concatenated expressions for assigning values to lengthy fields or variables. Concatenated expressions provide the only method for assigning values to fields or variables whose lengths exceed 255 characters. The following example combines the T edit string, *.prompt value expressions, and character string literals to assign a value to a long variable. You can use similar assignment statements in the USING clauses of the STORE and MODIFY statements:

```
DTR> DECLARE STR PIC X(300) EDIT_STRING IS T(50).
DTR> STR = *.L1|*.L2|*.L3|*.L4|*.L5
Enter L1: This string contains the first part of a long character
Enter L2: string. This string is so long that you can't use just
Enter L3: one character string literal to assign a value to it.
Enter L4: You need concatenated expressions to increase the length
Enter L5: of this string beyond the limit of 255 characters.
DTR> PRINT STR

                        STR

This string contains the first part of a long
character string. This string is so long that you
can't use just one character string literal to
assign a value to it.  You need concatenated
expressions to increase the length of this string
beyond the limit of 255 characters.

DTR>
```

1.1.10 Conditional Value Expressions

The CHOICE and IF-THEN-ELSE value expressions return a value based on the evaluation of one or more Boolean expressions. These value expressions are useful when you need to assign values that depend on certain conditions. They can be used in any statement that accepts value expressions, as well as in COMPUTED BY clauses for variables or field definitions.

1.1.10.1 CHOICE Value Expression

Returns one of a series of values depending on the evaluation of a series of conditional (Boolean) expressions.

Format

```
CHOICE [OF]
     boolean-expression-1 [THEN] value-1
     [boolean-expression-2 [THEN] value-2]
             .              .     .
             .              .     .
             .              .     .
     ELSE value-n
END_CHOICE
```

Arguments

CHOICE
Marks the beginning of a CHOICE value expression.

OF
Is an optional word used to clarify syntax.

boolean-expression
Is a Boolean expression.

THEN
Is an optional language element you can use to clarify syntax.

value
Is the value returned by DATATRIEVE if the corresponding Boolean expression evaluates to true.

ELSE value-n
Is the value returned by DATATRIEVE if all the Boolean expressions evaluate to false.

Restriction

You must include the ELSE clause in the CHOICE value expression.

Results

- DATATRIEVE evaluates each Boolean expression in order. When a Boolean expression evaluates to true, DATATRIEVE returns the corresponding value in the THEN clause. DATATRIEVE then goes on to interpret the next language element after END_CHOICE.
- If all the Boolean expressions evaluate to false, DATATRIEVE returns the value specified in the ELSE clause.

Examples

Edit the record definition for YACHTS to add a new field DISCOUNT_PRICE. Calculate the price based on the discount that applies to a particular price range.

The following example shows the field definition for DISCOUNT_PRICE:

```
06 DISCOUNT_PRICE   COMPUTED BY
          CHOICE
             PRICE LT 20000 THEN (PRICE * .9)
             PRICE LT 30000 THEN (PRICE * .8)
             PRICE LT 40000 THEN (PRICE * .7)
             ELSE (PRICE * .6)
          END_CHOICE
          EDIT_STRING IS $$$,$$$.
```

The following example displays the expanded records:

```
DTR> READY YACHTS
DTR> FOR YACHTS WITH BUILDER = "ALBIN" OR
CON>     BUILDER = "AMERICAN" PRINT BOAT

                              LENGTH
                               OVER                          DISCOUNT
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE     PRICE

 ALBIN       79         SLOOP   26     4,200  09  $17,900 $16,110
 ALBIN       BALLAD     SLOOP   30     7,276  09  $27,500 $22,000
 ALBIN       VEGA       SLOOP   27     5,070  09  $18,600 $16,740
 AMERICAN    26         SLOOP   26     4,000  08   $9,895  $8,906
 AMERICAN    26-MS      MS      26     5,500  08  $18,895 $17,006
```

The following example declares a variable called DISCOUNT_PRICE that takes values depending on the value for PRICE. It then displays the field values for the records in YACHTS along with the value for DISCOUNT_PRICE:

```
DTR> DECLARE DISCOUNT_PRICE COMPUTED BY
CON>      CHOICE
CON>         PRICE LT 20000 THEN (PRICE * .9)
CON>         PRICE LT 30000 THEN (PRICE * .8)
CON>         PRICE LT 40000 THEN (PRICE * .7)
CON>         ELSE (PRICE * .6)
CON>      END_CHOICE
CON>      EDIT_STRING IS $$$,$$$.
DTR> FOR YACHTS WITH BUILDER = "ALBIN" OR BUILDER =
CON>      "AMERICAN" PRINT TYPE, PRICE, DISCOUNT_PRICE

                                 DISCOUNT
MANUFACTURER   MODEL     PRICE    PRICE

 ALBIN       79         $17,900 $16,110
 ALBIN       BALLAD     $27,500 $22,000
 ALBIN       VEGA       $18,600 $16,740
 AMERICAN    26          $9,895  $8,906
 AMERICAN    26-MS      $18,895 $17,006

DTR>
```

1.1.10.2 IF THEN ELSE Value Expression

Returns one of two values depending on the evaluation of a conditional (Boolean) expression.

Format

```
IF boolean-expression [THEN] value-1 ELSE value-2
```

Arguments

boolean-expression
Is a Boolean expression.

THEN
Is an optional language element you can use to clarify syntax.

value-1
Is the value returned by DATATRIEVE if the Boolean expression evaluates to true.

ELSE value-2
Is the value returned by DATATRIEVE if the Boolean expression evaluates to false.

Restriction
Unlike the IF-THEN-ELSE statement, the ELSE clause is required in the IF-THEN-ELSE value expression.

Results

- DATATRIEVE evaluates the Boolean expression. If the Boolean expression evaluates to true, DATATRIEVE returns the value-1.
- If the expression evaluates to false, DATATRIEVE returns the value-2.

Examples

You can use the IF-THEN-ELSE value expression to help avoid a dividing by zero error.

The following example specifies an IF-THEN-ELSE value expression as a print list item, assigning a value of -1 where the denominator is zero. It then displays data about Albin's and Ryder's yachts, and calculates the relationship of length (LOA) to width (BEAM). If the width is zero, it sets the value to -1:

```
DTR> FOR YACHTS WITH BUILDER = "ALBIN" OR BUILDER = "RYDER"
CON> PRINT TYPE, PRICE, LOA, BEAM,
CON>    IF BEAM = 0 THEN -1 ELSE LOA/BEAM ("LOA/"/"BEAM")

                                LENGTH
                                 OVER          LOA/
MANUFACTURER   MODEL     PRICE   ALL   BEAM    BEAM

 ALBIN       79         $17,900  26     10       2.600
 ALBIN       BALLAD     $27,500  30     10       3.000
 ALBIN       VEGA       $18,600  27     08       3.375
 RYDER       S. CROSS   $32,500  31     00      -1.000

DTR>
```

The following example declares a variable LOA_BEAM and uses the IF-THEN-ELSE value expression within an assignment statement. For yachts built by Albin or Ryder, it assigns a value of –1 to LOA_BEAM when the BEAM equals 0. Otherwise, it assigns LOA_BEAM the value of LOA divided by BEAM:

```
DTR> DECLARE LOA_BEAM PIC S9V99 EDIT_STRING +9V99.
DTR> FOR YACHTS WITH BUILDER = "ALBIN" OR BUILDER = "RYDER"
CON> BEGIN
CON>    LOA_BEAM =
CON>    IF BEAM = 0 THEN -1 ELSE LOA/BEAM
CON>    PRINT TYPE, PRICE, LOA, BEAM, LOA_BEAM
CON> END

                                LENGTH
                                 OVER        LOA
MANUFACTURER   MODEL     PRICE   ALL   BEAM  BEAM

 ALBIN       79         $17,900  26     10   +2.60
 ALBIN       BALLAD     $27,500  30     10   +3.00
 ALBIN       VEGA       $18,600  27     08   +3.38
 RYDER       S. CROSS   $32,500  31     00   -1.00
```

```
DTR>
```

1.1.11 FORMAT Value Expression

Specifies a value that is formatted according to the default edit string or the edit string you indicate.

Format

```
FORMAT value-expression [USING edit-string]
```

Arguments

value-expression
Is a field name or other DATATRIEVE value expression specifying a value that DATATRIEVE uses when executing statements.

edit-string
Is one or more edit string characters that determines the value of the value expression. For more information, see the section on the EDIT_STRING clause in Chapter 4.

Restriction

The edit string must conform to the conventions for specifying edit strings in DATATRIEVE. For more information, see the section on the EDIT_STRING clause in Chapter 4.

Results

- If a USING clause is present, DATATRIEVE returns a value according to the specified edit string.
- If no USING clause is present, DATATRIEVE returns a value according to the default edit string of the value expression. This edit string is based on the edit string or PICTURE clause for the field that is a part of the value expression.

Usage Notes

- You can use the FORMAT value expression as a sort key in a SORT statement or SORTED BY clause of an RSE. DATATRIEVE sorts the records according to the formatted value of the expression.
- You can use the FORMAT value expression as a reduce key in a REDUCE statement or REDUCED TO clause of an RSE. DATATRIEVE evaluates the expression for each record. Every time DATATRIEVE finds a new value, it retains the value of the field contained in the value expression.
- The maximum length of strings created using the FORMAT value expression is 65,535 characters.

Examples

You can use the FORMAT value expression to assist in data retrieval for fields with leading zeros. The following example shows the record definition for the PATIENT domain:

```
DTR> SHOW PATIENT_REC
RECORD PATIENT_REC USING
01 ACCOUNT.
     03 PATIENT_ID        PIC IS X(7).
     03 NAME.
            05 FIRST_NAME  PIC IS X(10).
            05 LAST_NAME   PIC IS X(10).
;
```

The records in this example are stored in an indexed file with PATIENT_ID as the key field. The following example shows the patient data:

```
DTR> PRINT PATIENT

PATIENT   FIRST       LAST
  ID      NAME        NAME

0000010 HANK       MORRISON
0000011 BILL       SWAY
0000012 SY         KELLER
0000013 WAYNE      SMITH
0000014 JOE        FREDERICK
```

The following statement attempts to find the first record:

```
DTR> FIND PATIENT WITH PATIENT_ID = 10
[0 records found]
```

In this case DATATRIEVE does not find the record, because the leading zeros are significant when the field is alphanumeric. (The PICTURE clause for PATIENT_ ID is X(7).)

Applying a FORMAT value expression to the field name overcomes the problem of the leading zeros:

```
DTR> FIND PATIENT WITH (FORMAT PATIENT_ID USING 99) = 10
[1 record found]
DTR> PRINT CURRENT

PATIENT   FIRST       LAST
  ID      NAME        NAME

0000010 HANK       MORRISON

DTR>
```

You can use the FORMAT value expression along with the REDUCED TO clause of the RSE to display the unique values in a record stream. The following procedure produces a report which displays data on all the yachts that have a price, divides the yachts into price categories that are multiples of $10,000, and specifies headings such as "YACHTS UNDER $10,000" and "YACHTS UNDER $20,000":

```
DTR> SHOW REDUCE_RPT
PROCEDURE REDUCE_RPT
READY YACHTS
FIND YACHTS WITH PRICE NE 0    1
FOR A IN CURRENT REDUCED TO (FN$FLOOR(PRICE/10000))   2
  BEGIN
    PRINT SKIP, COL 10,
       "YACHTS UNDER "(FORMAT (FN$FLOOR(A.PRICE/10000) + 1) USING  3
       $90,000), SKIP
    FOR YACHTS WITH PRICE NE 0 AND FN$FLOOR (PRICE/10000) =  4
       FN$FLOOR (A.PRICE/10000) SORTED BY PRICE
           PRINT PRICE, TYPE, RIG    5
  END
END_PROCEDURE

DTR> :REDUCE_RPT

         YACHTS UNDER $10,000

 PRICE  MANUFACTURER   MODEL     RIG

 $2,823  VENTURE     21         SLOOP
 $3,500  WINDPOWER   IMPULSE    SLOOP
   .       .          .           .
   .       .          .           .
   .       .          .           .

         YACHTS UNDER $90,000

$80,500  OLYMPIC     ADVENTURE  KETCH

DTR>
```

1 This statement forms a collection of all the yachts that have a price.

2 This statement determines the price categories that apply for the group of yachts. The expression FN$FLOOR (PRICE/10000) gives the value of the integer in the ten-thousands place of the price for a specific yacht. The expression is placed within a REDUCED TO clause to identify the unique digits in the ten-thousands place of the PRICE of all the yachts.

3 This statement prints headings for each price category of yachts, using the following expression for the price:

```
FORMAT (FN$FLOOR(A.PRICE/10000) + 1) USING  $90,000
```

This expression yields the value of the digit in the ten-thousands place plus 1, formatted according to the edit string of $90,000. For example, if the digit is "1" ("1" in the ten-thousands place), the value of the FORMAT value expression is $20,000.

4 This statement searches through the yacht records for each of the digits in the ten-thousands place, and finds records which match on the corresponding number in the price field. That is, it finds those yachts for which:

```
FN$FLOOR (PRICE/10000) = FN$FLOOR (A.PRICE/10000)
```

5 This statement prints the values for PRICE, TYPE, and RIG for each yacht that match on PRICE/10000.

1.1.12 FROM Value Expression

Allows you to perform complex retrievals of records from one or more domains or collections. You can include a FROM value expression in the Boolean of the RSE that forms the collection or record stream. The optional OTHERWISE clause lets you specify an alternative DATATRIEVE value expression if an initial search of data based on an RSE produces no records. If the number of records in a record stream equates to zero, the alternative specified by the OTHERWISE clause is returned.

Format

```
value-expression FROM rse [OTHERWISE value-expression]
```

Arguments

value-expression
Is a DATATRIEVE value expression

rse
Is a record selection expression

Results

When you use a FROM...OTHERWISE value expression, DATATRIEVE determines its value in the following way:

1. It forms a record stream as specified by the RSE.
2. If at least one record matches the RSE, DATATRIEVE uses the value stored in the first record of the record stream to evaluate the overall expression.
3. If the number of records in the record stream is zero, DATATRIEVE uses the value expression provided in the OTHERWISE clause to determine the value of the overall expression. This value expression is evaluated independently of the previously mentioned record stream; its value will be determined based on any context or record stream established outside of the FROM record stream.

Examples

The following example illustrates the use of the FROM...OTHERWISE value expression. In this example, the value expression specified by the OTHERWISE clause is the character string literal, "No builder."

```
DTR>PRINT MANUFACTURER FROM YACHTS WITH LOA > 55 OTHERWISE
[Looking for value expression]
CON>    "No builder"

No builder

DTR>
```

You can use other value expressions as well. The OTHERWISE clause at the end of the following example uses the MAX statistical operator to produce alternative information when the RSE in the FROM...OTHERWISE value expression finds no appropriate matches.

```
DTR>PRINT MANUFACTURER FROM YACHTS WITH LOA GT 55 OTHERWISE
[Looking for value expression]
CON>MAX LOA OF YACHTS
42

DTR>
```

You can also use the FROM...OTHERWISE value expression in a field definition based on a COMPUTED BY or a VALID IF clause. The following example illustrates such a definition:

```
DTR>SET DICTIONARY DISK$1:[KLAUS.DTR]
DTR>DEFINE RECORD SAMPLE_RECORD USING
DFN>01 SAMPLE_RECORD.
DFN>    03 SAMPLE_MODEL PIC X(10).
DFN>    03 SAMPLE_PRICE PIC 9(5).
DFN>    03 SAMPLE_BUILDER COMPUTED BY MANUFACTURER FROM YACHTS WITH
DFN>            PRICE GT 50000 OTHERWISE "NO BUILDER".
DFN> ;
[Record is 15 bytes long.]
```

Restriction

The following restrictions apply to the use of the FROM...OTHERWISE value expression:

- The keyword OTHERWISE must follow the RSE on the same line, or you must use a continuation character (-) to tell DTR that the expression is not yet complete. Because the part of the statement preceding the OTHERWISE clause can generally be construed by DATATRIEVE as a complete statement, DATATRIEVE prematurely terminates the search and produces a series of error messages. The following example illustrates this:

```
DTR> PRINT MANUFACTURER FROM YACHTS WITH LOA > 55
Value not found from record or table.
DTR> OTHERWISE NO BUILDER
OTHERWISE NO BUILDER

Expected statement, encountered "OTHERWISE".
DTR>
```

Proper use of the continuation character (-) corrects this problem:

```
DTR> PRINT MANUFACTURER FROM YACHTS WITH LOA GT 55 -
CON>    OTHERWISE "No builder"
No builder

DTR>
```

- The FROM...OTHERWISE value expression can be used in record definitions with COMPUTED BY and VALID IF fields; however, certain products with which DATATRIEVE interacts may not fully recognize the new syntax. For example, you can store the definition in either the DMU or the CDO format dictionary and you can use the definition from within DATATRIEVE; however, you cannot currently use the CDO SHOW RECORD/FULL command to display the user-specified attributes used in the record definition. In addition, other products such as Rdb/VMS or VAX DBMS may not be able to recognize the OTHERWISE clause in COMPUTED BY and VALID IF fields.
- While nesting of FROM...OTHERWISE clauses is allowed, you should consider the impact of such nesting on performance and on complications that could arise from multiple or complex nested statements.

1.2 Boolean Expressions

A Boolean expression is the logical representation of a relationship between value expressions. The value of a Boolean expression is either true or false.

You can use Boolean expressions in the following DATATRIEVE clauses and statements:

- WITH clause in a record selection expression
- WITH clause in a SELECT statement
- IF clause of an IF-THEN-ELSE statement
- IF clause of an IF-THEN-ELSE value expression
- CHOICE statement
- CHOICE value expression

- VALID IF clause in a record definition
- WHILE statement

Boolean expressions consist of value expressions, relational operators, and Boolean operators. **Relational operators** control the comparison of value expressions. **Boolean operators** enable you to join two or more Boolean expressions and to reverse the value of a Boolean expression. All Boolean expressions contain value expressions and relational operators, and some contain Boolean operators.

1.2.1 Relational Operators

Relational operators compare value expressions, check whether a code string is contained in a table, and check whether a record stream is empty or not. Most Boolean expressions contain a field name, a relational operator, and a value expression. Table 1–4 shows the format for using each relational operator.

Table 1–4 Relational Operators

Type of Comparison	Relationship of Values in Boolean	Relational Operator	Boolean Expression
PATTERN RECOGNITION	Exact match (case sensitive).	= EQUAL EQ	RIG = “YAWL” “YAWL” = RIG RIG = “MS”,“YAWL”
	No match (case sensitive).	NE NOT_EQUAL NOT EQUAL	RIG NE “YAWL” “YAWL” NE RIG RIG NE “MS”,“YAWL”
	Substring matches (not case sensitive).	CONT CONTAINING	RIG CONT “yawl” RIG CONT “ms”,“yawl”
	Beginning substring matches (case sensitive).	STARTING WITH [1]	RIG STARTING WITH “M” RIG STARTING WITH “M”,“Y”
	Substring does not match (not case sensitive).	NOT CONT NOT CONTAINING	RIG NOT CONT “ms” RIG NOT CONT “ms”,“ya”

[1]Using the STARTING WITH operator with numeric fields can produce inconsistent results; use only ASCII printable characters for the specified substring.

(continued on next page)

Table 1–4 (Cont.) Relational Operators

Type of Comparison	Relationship of Values in Boolean	Relational Operator	Boolean Expression
VALUE WITHIN A RANGE	First value is greater.	> GT GREATER_THAN	PRICE > 50000 50000 > PRICE
	First date value is later than the second expression.	AFTER	START_DATE AFTER "1-Jan-1990" "1-Jan-1990" AFTER START_DATE
	First value is greater than or equal.	GE GREATER_EQUAL	PRICE GE 50000 50000 GE PRICE
	First value is less.	< LT LESS_THAN	PRICE < 20000 20000 < PRICE
	First date value is earlier than the second expression.	BEFORE	START_DATE BEFORE "1-Jan-1990" "1-Jan-1990" BEFORE START_DATE
	First value is less than or equal.	LE LESS_EQUAL	PRICE LE 20000 20000 LE PRICE
	First value is between the two values or equal to one.	BT BETWEEN	PRICE BETWEEN 30000 AND 54000
FIELD VALUE MISSING	Field value is the MISSING VALUE.	MISSING	PRICE MISSING
	Field value is not the MISSING VALUE.	NOT MISSING	PRICE NOT MISSING
LOOK UP IN TABLE	Field value is in the table.	IN table-name	RIG IN RIG_TABLE
	Field value is not in the table.	NOT IN table-name	RIG NOT IN RIG_TABLE

(continued on next page)

Table 1–4 (Cont.) Relational Operators

Type of Comparison	Relationship of Values in Boolean	Relational Operator	Boolean Expression
RECORD STREAM EMPTY	Record stream is not empty.	ANY rse	FAMILIES WITH ANY KIDS
	Record stream is empty.	NOT ANY rse	FAMILIES WITH NOT ANY KIDS

The STARTING WITH relational operator is designed to work on text strings and can give inconsistent results when used with numeric fields.

When handling character string literals, DATATRIEVE considers lowercase letters to have a greater value than uppercase letters. Within each case, however, DATATRIEVE sorts the letters alphabetically, and considers the letters near the beginning of the alphabet to be smaller than those near the end. Consequently, "ALBIN" is less than "AMERICAN."

The order and value associated with alphanumeric characters is determined by the ASCII collating sequence. Lowercase letters have a higher ASCII value than uppercase letters. See the VAX/VMS documentation for more information on the DEC Multinational Character Set, which includes the ASCII character set.

In Boolean expressions using the relational operator CONTAINING, the comparison of the value expression and the field value is case insensitive. The comparison is case insensitive regardless of whether you enclose a character string literal within quotation marks.

The following examples show how to use relational operators that compare field values to value expressions:

```
DTR> FIND YACHTS WITH BEAM EQ 9,10,14
[50 records found]
DTR> PRINT CURRENT WITH RIG NE "SLOOP"

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 EASTWARD     HO        MS      24     7,000  09  $15,900
 FISHER       30        KETCH   30    14,500  09
 GRAMPIAN     34        KETCH   33    12,000  10  $29,675
```

The last PRINT statement can be written with the value expression preceding the field name. DATATRIEVE displays the same records:

```
DTR> PRINT CURRENT WITH "SLOOP" NE RIG

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 EASTWARD     HO         MS      24     7,000  09  $15,900
 FISHER       30         KETCH   30    14,500  09
 GRAMPIAN     34         KETCH   33    12,000  10  $29,675
```

You can also search for field values within a specified range:

```
DTR> PRINT YACHTS WITH LOA BT 30 AND 31

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 ALBIN        BALLAD     SLOOP   30     7,276  10  $27,500
 BOMBAY       CLIPPER    SLOOP   31     9,400  11  $23,950
 C&C          CORVETTE   SLOOP   31     8,650  09
                 .               .                  .
                 .               .                  .
                 .               .                  .
 SOLNA CORP   SCAMPI     SLOOP   30     6,600  10
```

The following examples show some other queries using Boolean expressions:

```
DTR> PRINT COUNT OF YACHTS WITH BUILDER CONTAINING "a"

COUNT

    78

DTR> FIND YACHTS WITH LOA < 20
[2 records found]
DTR> FIND YACHTS WITH PRICE MISSING
[63 records found]
DTR> FIND YACHTS WITH PRICE NOT MISSING
[50 records found]
```

The relational operator IN compares the contents of a field with the code strings in a dictionary table or domain table. This comparison is useful for validating data you assign to fields or variables. The following example shows how to write a record definition that uses a table to validate the data before it is stored:

```
DTR> DEFINE RECORD PHONE_REC USING
DFN> 01 PHONE.
DFN>    02 NAME PIC X(20).
DFN>    02 NUMBER PIC 9(7) EDIT_STRING IS XXX-XXXX.
DFN>    02 LOCATION PIC X(9).
DFN>    02 DEPARTMENT PIC XX
DFN>           VALID IF DEPARTMENT IN DEPT_TABLE.
DFN> ;
```

The relational operator ANY checks whether a record stream is empty or not. This operator is useful for work with lists in hierarchical records. The record selection expression following ANY generally specifies the name of a list or sublist. The following examples show how to use ANY. For more information on lists and hierarchies, see the *VAX DATATRIEVE User's Guide*.

```
DTR> READY FAMILIES
DTR> PRINT FAMILIES WITH ANY KIDS WITH AGE = 20

                       NUMBER    KID
  FATHER     MOTHER     KIDS     NAME     AGE

BASIL      MERIDETH       6    BEAU        28
                               BROOKS      26
                               ROBIN       24
                               JAY         22
                               WREN        17
                               JILL        20
JEROME     RUTH           4    ERIC        32
                               CISSY       24
                               NANCY       22
                               MICHAEL     20

DTR> PRINT FAMILIES WITH ANY KIDS WITH KID_NAME CONT "RAL"

                       NUMBER    KID
  FATHER     MOTHER     KIDS     NAME     AGE

JIM        ANN            2    URSULA       7
                               RALPH        3

DTR>
```

1.2.2 Boolean Operators

There are four Boolean operators: AND, OR, NOT, and BUT. With AND, OR, and BUT, you can join two or more Boolean expressions to form a single Boolean expression. NOT allows you to reverse the value of a Boolean expression.

The AND and BUT operators perform the same function. If you link Boolean expressions with either AND or BUT, the resulting Boolean expression is true only if all the Boolean operators linked with either AND or BUT are true.

If you link Boolean expressions with OR, the resulting Boolean expression is true if any one of the Boolean linked with OR are true.

If you precede a Boolean expression with NOT, the resulting Boolean expression is true if the Boolean expression following NOT is false.

The following examples show the use of Boolean operators:

```
DTR> READY YACHTS
DTR> PRINT YACHTS WITH BUILDER = "PEARSON" AND LOA = 30

                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 PEARSON     30         SLOOP   30     8,320  09

DTR> FIND YACHTS WITH BUILDER = "PEARSON" OR LOA = 30
[21 records found]

DTR> READY FAMILIES
DTR> PRINT FAMILIES WITH FATHER NOT EQ "JIM" AND
[Looking for Boolean expression]
CON> ANY KIDS WITH AGE GT 31

                      NUMBER    KID
  FATHER     MOTHER    KIDS     NAME    AGE

JEROME     RUTH          4    ERIC       32
                              CISSY      24
                              NANCY      22
                              MICHAEL    20
HAROLD     SARAH         3    CHARLIE    31
                              HAROLD     35
                              SARAH      27

DTR>
```

You can also use parentheses to group Boolean expressions. DATATRIEVE evaluates Boolean expressions in parentheses before evaluating other Boolean expressions. If a Boolean expression contains Boolean operators as well as parentheses, DATATRIEVE evaluates the Boolean expression in the following order:

1. Expressions enclosed in parentheses
2. Expressions preceded by NOT
3. Expressions combined with AND or BUT
4. Expressions combined with OR

Table 1–5 shows the use of parentheses and the evaluation of compound Boolean expressions.

Table 1–5 Compound Boolean Expressions

Expression	Value
bool-1 AND bool-2 AND bool-3	True if all three Boolean expressions are true.
bool-1 AND (bool-2 OR bool-3)	True if bool-1 is true and either bool-2 or bool-3 is true.
(bool-1 AND bool-2) OR (bool-3 AND bool-4)	True if both bool-1 and bool-2 are true or if both bool-3 and bool-4 are true.
NOT (bool-1 OR bool-2) AND bool-3	True if both bool-1 and bool-2 are false and bool-3 is true.

The following example illustrates compound Boolean expressions:

```
DTR> PRINT YACHTS WITH
[Looking for Boolean expression]
CON>    (MODEL = "BALLAD" AND BUILDER = "ALBIN") OR
[Looking for Boolean expression]
CON>    (BUILDER = "TANZER" AND MODEL = 28)

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 ALBIN       BALLAD     SLOOP   30     7,276  10  $27,500
 TANZER      28         SLOOP   28     6,800  10  $17,500

DTR>
```

2
Using DATATRIEVE Variables

You use variables in DATATRIEVE:

- To assign values to fields in STORE and MODIFY statements
- As counters in FOR, REPEAT, and WHILE loops
- As conditional values in Boolean expressions
- To specify field names that would otherwise be ambiguous

2.1 Declaring Variables

The statement for declaring a variable has the following format:

```
DECLARE variable-name variable-definition.
```

The variable name is the name you give to the variable you are creating. The variable definition consists of field definition clauses. When you declare a variable, you can use any of the DATATRIEVE definition clauses except OCCURS and REDEFINES. You must include at least one PIC, COMPUTED BY, or USAGE clause. You can also use the QUERY-HEADER, EDIT-STRING, SIGN, MISSING VALUE, and DEFAULT VALUE clauses.

To declare the variable A to be a three-digit numeric value with an initial value of zero, you use the variable name and variable definition as follows:

```
DTR> DECLARE A PIC 999.
DTR> A = 0
```

If you print the variable, it looks like this:

```
DTR> PRINT A

 A

000
```

The initial value for variables in numeric fields is zero. In alphanumeric strings, it is spaces. These are the default values if you do not specify a different default value or missing value.

You can also use a date field as a variable, as in this example:

```
DTR> DECLARE Y USAGE DATE EDIT_STRING DD-MMM-YY.
DTR> Y = "TODAY"
DTR> PRINT Y

    Y

19-May-87
```

You can define two kinds of variables:

- Local variables
- Global variables

2.2 Local Variables

You define local variables with DECLARE statements entered in BEGIN-END and THEN statements. The local variable has an effect only within the clause or statement in which you declare it.

In the following example, the local variable declared in the inner statement supersedes one with the same name declared in the outer statement. Notice that the different value or different data type assigned to the inner variable has no effect on the value of the variable in the outer statement. Note also that neither local variable exists when DATATRIEVE finishes executing the compound statements containing them both:

```
DTR> SET NO PROMPT
DTR> BEGIN
CON>   DECLARE X PIC XXX.
CON>   X = "TOP"
CON>   PRINT X
CON>     BEGIN
CON>       DECLARE X PIC 9.99.
CON>       X = 1.23
CON>       PRINT X
CON>     END
CON>   PRINT X
CON> END

 X

TOP

 X

1.23

 X

TOP
```

```
DTR>
```

2.3 Global Variables

Global variables are defined at the DATATRIEVE command level. They remain in your workspace until they are released or until you exit DATATRIEVE.

Suppose you want to assign to each boat in YACHTS a new price that is two-thirds of the present price. Using a COMPUTED BY clause in a global variable, you can apply a single formula to every yacht, as in the next example. Use the DECLARE statement to create the variable. Use a COMPUTED BY clause with a value expression to calculate the changed values.

```
DTR> READY YACHTS MODIFY
DTR> DECLARE FIRE_PRICE COMPUTED BY PRICE/1.5
CON> EDIT_STRING IS $99,999.99.
DTR> FOR FIRST 5 YACHTS PRINT BOAT, FIRE_PRICE

                             LENGTH
                              OVER                          FIRE
MANUFACTURER   MODEL    RIG   ALL  WEIGHT BEAM  PRICE     PRICE

 ALBERG       37 MK II  KETCH  37   20,000  12  $36,951 $24,634.00
 ALBIN        79        SLOOP  26    4,200  10  $17,900 $11,933.33
 ALBIN        BALLAD    SLOOP  30    7,276  10  $27,500 $18,333.33
 ALBIN        VEGA      SLOOP  27    5,070  08  $18,600 $12,400.00
 AMERICAN     26        SLOOP  26    4,000  08  $9,895  $06,596.67
DTR>
```

The variable FIRE_PRICE declared at DATATRIEVE command level remains in the workspace throughout the session. It changes its value whenever the value of PRICE changes. (See the *VAX DATATRIEVE User's Guide* for a discussion of context changes.) The variable remains in your workspace until you release the variable with a RELEASE statement or declare another global variable with the same name.

2.4 Using Variables to Assign Values to Fields

You can use variables to assign values to fields in the USING clauses of STORE and MODIFY statements. You cannot, however, use a variable to respond to a prompt for a field value, whether the prompt is the result of the STORE or MODIFY statement or of a prompting value expression in an Assignment statement.

In the USING clause of the STORE and MODIFY statements, you can supply values for fields by using value expressions on the right side of Assignment statements. In some circumstances, you can use variables in those assignments to control the uniformity of input data.

In this example, WORK is a domain you want to contain uniform names. The domain is indexed on WHO and allows duplicates:

```
DTR> SHOW WORK_REC
RECORD WORK_REC
 USING
01 TOP.
        03 JOB PIC X(15).
        03 RESPONSIBLE_PERSON PIC X(4)
           QUERY_NAME WHO.
;
```

NAME_TABLE translates the varying inputs into uniform values to store in the work domain:

```
DTR> SHOW NAME_TABLE
TABLE NAME_TABLE
EDIT_STRING IS X(16)
E               :       ED
ED              :       ED
EM              :       ED
M               :       ED
F               :       FRED
FH              :       FRED
FRED            :       FRED
H               :       FRED
L               :       RICK
R               :       RICK
RBL             :       RICK
RICK            :       RICK
RL              :       RICK
ELSE "NOT A VALID NAME"
END_TABLE
```

In the following STORE statement, the USING clause uses the variable PERSON with a prompting value expression for the responsible person. The table translates the value supplied to that prompt and stores the uniform results in the field WHO:

```
DTR> SET NO PROMPT
DTR> DECLARE PERSON PIC X(16).
DTR> READY WORK WRITE
DTR> REPEAT 3 STORE WORK USING
CON> BEGIN
CON>    JOB = *.JOB
CON>    PERSON = *.WHO
CON>    WHO = PERSON VIA NAME_TABLE
CON> END
Enter JOB: CLEANING
Enter WHO: E
Enter JOB: DRYING
Enter WHO: FR
Enter JOB: SELLING
Enter WHO: R
DTR> PRINT WORK

                     RESPONSIBLE
      JOB              PERSON

CLEANING           ED
DRYING             NOT A VALID NAME
SELLING            RICK

DTR>
```

2.5 Changing the Value of a Variable

You can change the value of a variable with an Assignment statement, using any DATATRIEVE value expression on the right side of the statement. You can also use a prompting value expression to change the value of a variable:

```
DTR> DECLARE X PIC XXX.
DTR> X = *."VALUE FOR X"
Enter VALUE FOR X: LIP
DTR> PRINT X
 X

LIP

DTR>
```

2.6 Using Context Variables

DATATRIEVE also provides **context variables** which serve as labels that identify a record stream to DATATRIEVE. You assign context variables to be temporary names of particular record streams. In this way you can make clear the domain from which a record stream originates, or you can create two different record streams based on the same domain.

In most cases, DATATRIEVE will know to what record stream a field name applies without needing context variables. For example, DATATRIEVE does not need context variables in the following store statement:

```
DTR> ! First, define a domain that will hold a subset
DTR> ! of YACHTS records, namely, those that
DTR> ! cost more than $20,000.
DTR> DEFINE DOMAIN RITZY_ONES USING YACHT ON RITZY;
DTR> DEFINE FILE FOR RITZY_ONES
DTR> READY RITZY_ONES WRITE
DTR> ! The FOR statement includes a STORE USING statement
DTR> ! to store the desired records in RITZY_ONES.
DTR> ! Note that you don't need context variables.
DTR> FOR YACHTS WITH PRICE > "$20,000"
CON> STORE RITZY_ONES USING
CON>   BEGIN
CON>       TYPE = TYPE
CON>       PRICE = PRICE
CON>   END
```

In this example, DATATRIEVE assigns the values of TYPE and PRICE of all boats in YACHTS that cost more than $20,000 to records in a new domain called RITZY_ONES.

Should you so choose, however, you can use context variables to identify record streams and to qualify field names. This makes your statements and procedures less ambiguous and easier to maintain. For example, you can perform the preceding store operation as follows:

```
DTR> FOR Y IN YACHTS WITH PRICE > "$20,000"
CON> STORE R IN RITZY_ONES USING
CON>   BEGIN
CON>       R.TYPE = Y.TYPE
CON>       R.PRICE = Y.PRICE
CON>   END
```

In certain cases, however, you must use context variables to identify a record stream explicitly. When you need to access the same domain two or more times in one statement, or when you need to compare record streams from the same domain, you must use context variables. In all other cases, you can use the domain name for qualifying field names, or else DATATRIEVE resolves the context automatically (as in the former example).

But when you must establish two record streams from the same domain, or when you cross a domain over itself, you use context variables to label different record streams. By qualifying each field name with the context variable and a period (.), you indicate clearly to DATATRIEVE how to evaluate field references. In the previous example, DATATRIEVE looks to the R stream to evaluate R.TYPE and to the Y stream to evaluate Y.TYPE. Thus, DATATRIEVE allows you to create

two (or more) record streams from the same domain without confusing or mixing records.

For example, assume that you wish to find how much of the company's workforce each department employs. You can perform the query in the following manner:

```
DTR> FOR D IN PERSONNEL REDUCED TO DEPT
CON> PRINT D.DEPT, (100 * (COUNT OF PER IN PERSONNEL -
CON> WITH PER.DEPT = D.DEPT)/ -
CON> (COUNT OF PERSONNEL)) ("PCT") USING Z9.9%

DEPT   PCT

C82   21.7%
D98   17.4%
E46    8.7%
F11   17.4%
G20   13.0%
T32   17.4%
TOP    4.3%
```

This statement sets up an inner loop of records from the PERSONNEL domain identified by the context variable PER and an outer loop of records from the same domain identified by the label D.

For each group of records with the same department, DATATRIEVE counts the records and divides the number by the total number of all records in PERSONNEL. The context variables D and PER allow you to refer to the same records in the PERSONNEL domain twice. Thus, you can print out all the department names and count all their members in the same statement.

The *VAX DATATRIEVE User's Guide* contains more examples of context variables and presents detailed information about how DATATRIEVE resolves context.

3
DATATRIEVE Functions

A DATATRIEVE function is a word you define and add to the DATATRIEVE language. By adding functions, you extend the capability of DATATRIEVE to efficiently perform specific tasks. To learn how to define functions for DATATRIEVE, see the *VAX DATATRIEVE Guide to Programming and Customizing*.

The DATATRIEVE installation kit provides some functions. You can use them to form value expressions or to set parameters for a process. These functions can be modified by users at your site. If they do not work as indicated in the examples, consult the person at your site responsible for DATATRIEVE for a list of the functions currently available.

DATATRIEVE functions have the following format:

```
function-name  [ (value-expression  [,...] ) ]
```

The remainder of this chapter is divided into three sections. The first section groups and lists the functions by type, with a brief description of the common features of each type. The second section presents all the functions (regardless of type) in alphabetic order, indicating the input and output of each function, as well as an example. The third section discusses optimizing function execution.

3.1 Functions Grouped by Type

The following sections group and list functions by type. The types include the following:

- Function value expressions
  - Functions using numeric data
  - Trigonometric functions
  - Functions using alphanumeric data
- Functions for keypad definitions

- Functions relating to processes
  - Functions for timing processes
  - Functions for using logical names
  - Functions for using symbols
  - Other functions relating to processes

Each section includes a description of the common features of each type.

3.1.1 Function Value Expressions

The following functions are value expressions, which you can use in any DATATRIEVE statement that permits a value expression. The functions are classified according to their input values. They take either numeric data, alphanumeric data, or dates as arguments.

3.1.1.1 Functions Using Numeric Data

The following functions are value expressions that take numbers as arguments and return numbers as values:

- FN$ABS—Calculates the absolute value of input
- FN$EXP—Calculates the value of **e** to a specified power
- FN$FLOOR—Truncates the decimal part of positive input or rounds negative input
- FN$LN—Calculates the natural log of input
- FN$LOG10—Calculates the base 10 log of input
- FN$MOD—Calculates the value of input according to a specified modulus
- FN$NINT—Calculates the integer nearest to input
- FN$SIGN—Indicates the sign of a number
- FN$SQRT—Calculates the square root of input

The following function differs from those listed above since it takes a number as input but returns a character string as value:

- FN$HEX—Calculates the hexadecimal character string equivalent of input

3.1.1.2 Trigonometric Functions

The following are trigonometric functions available with DATATRIEVE:

- FN$ATAN—Calculates the arctangent of input
- FN$COS—Calculates the cosine of input
- FN$SIN—Calculates the sine of input
- FN$TAN—Calculates the tangent of input

3.1.1.3 Functions Using Alphanumeric Data

The following functions use alphanumeric data as input. The default format for functions using alphanumeric data is PIC X(30). If a function returns a value that is longer than 30 characters, DATATRIEVE truncates the value. You can override the default and process a value longer than 30 characters by using a FORMAT value expression. Include the function in the FORMAT value expression and specify an edit string long enough for the value you want to process.

For more information about using FORMAT value expressions, see Chapter 1.

- FN$STR_EXTRACT—Extracts the substring from input
- FN$STR_LOC—Calculates the starting position of a substring in input
- FN$UPCASE—Changes the characters in a string to uppercase

3.1.1.4 Functions Using Dates

The following functions use dates as input:

- FN$JULIAN—Calculates the Julian date of input
- FN$WEEK—Calculates the number of weeks from start of year
- FN$YEAR—Extracts the year part of input
- FN$MONTH—Extracts the month part of input
- FN$DAY—Extracts the day part of input
- FN$TIME—Extracts the time part of input
- FN$HOUR—Extracts the hour part of input
- FN$MINUTE—Extracts the minute part of input
- FN$SECOND—Extracts the second part of input
- FN$HUNDREDTH—Extracts the hundredth-of-a-second part of input
- FN$DATE—Converts a date string to a 64-bit data value

3.1.2 Functions for Keypad Definitions

You can use the following functions to show and customize your keypad definitions:

- FN$COMMAND_KEYBOARD—Returns the current value of the COMMAND_KEYBOARD field in the DATATRIEVE Access Block (DAB)
- FN$DEFINE_KEY—Takes a key definition in DIGITAL Command Language (DCL) DEFINE/KEY syntax, then creates the defined key in DATATRIEVE
- FN$DELETE_KEY—Lets you delete a key definition currently in effect
- FN$KEYPAD_MODE—Lets you specify the terminal mode from within DATATRIEVE
- FN$KEYTABLE_ID—Returns the value of the KEYTABLE_ID field currently in the DAB
- FN$LOAD_KEYDEFS—Lets you define multiple keypad keys from a file containing DCL DEFINE/KEY commands
- FN$PROMPT_KEYBOARD—Returns the value of the PROMPT_KEYBOARD field currently in the DAB
- FN$SHOW_KEY—Shows the definition of a keypad key
- FN$SHOW_KEYDEFS—Shows the key definitions in all of the states

These functions are not available in a DECwindows environment.

3.1.3 Functions Relating to Processes

Most of the functions relating to processes are not value expressions; they initiate or affect various DATATRIEVE processes but have no output. The two exceptions are the FN$TRANSLOG and FN$OPENSLEFT functions, which generate output in providing information about a process.

3.1.3.1 Functions for Timing Processes

You can use the following functions to time processes:

- FN$INIT_TIMER—Initializes a timer and counter
- FN$SHOW_TIMER—Shows elapsed time since timer was last initialized

3.1.3.2 Functions for Using Logical Names

The following functions define, delete, and translate logical name synonyms for physical names like file specifications.

- FN$CREATE_LOG—Assigns a logical name as a synonym for a physical name
- FN$DELETE_LOG—Deletes the assignment of a logical name
- FN$TRANS_LOG—Translates a logical name

For more information about logical names, see Appendix C.

3.1.3.3 Function for Using Symbols

The following function returns the value of a DCL symbol.

- FN$GET_SYMBOL—Returns the value of a DCL symbol

3.1.3.4 Other Functions Relating to Processes

The following functions allow you to control your process and customize your terminal:

- FN$DCL—Allows you to spawn directly from your main DATATRIEVE process to execute a specified DIGITAL Command Language (DCL) command
- FN$OPENS_LEFT—Calculates the number of additional files you can open
- FN$SPAWN—Creates a subprocess
- FN$WIDTH—Changes the character width of the terminal

3.2 DATATRIEVE Mathematical Functions Now Use G FLOATING Format Numbers

The DATATRIEVE functions that perform mathematical operations now use G-FLOATING format numbers for their internal operation. These functions include the following:

- FN$ABS
- FN$ATAN
- FN$COS
- FN$EXP
- FN$FLOOR
- FN$LN
- FN$LOG10

3.2 DATATRIEVE Mathematical Functions Now Use G FLOATING Format Numbers

- FN$MOD
- FN$NINT
- FN$SIN
- FN$SQRT
- FN$TAN

The use of G-FLOATING numbers ensures an improved precision, as these functions now provide 15 significant digits and an exponent range of ±1023.

If you require a different level of precision such as F-FLOATING or D-FLOATING, you can override these functions using the techniques described in the *VAX DATATRIEVE Guide to Programming and Customizing*.

3.3 Functions Listed Alphabetically

The following sections describe each function in alphabetical order. Each section includes the following information:

- A description of the function, its input, and its output.
- An example of the function's use.

FN$ABS

Calculates the absolute value of input.

Input

A signed decimal number.

Output

An unsigned decimal number.

Example

```
DTR> PRINT FN$ABS (-128)
  FN$ABS
 1.2800E+02
DTR>
```

FN$ATAN

Calculates the arctangent of input.

Input

A signed decimal number (radians).

Output

A signed decimal number.

Example

```
DTR> PRINT 4 * (FN$ATAN (1))
 3.1416E+00
DTR>
```

FN$COMMAND_KEYBOARD

Returns the current value of the COMMAND_KEYBOARD field in the data access block (DAB). COMMAND_KEYBOARD is the keyboard used for command input.

This function allows you to add other functions that access the features of the Screen Management Guidelines (SMG) of the Screen Management Facility.

This function is not available in a DECwindows environment.

Input

None.

Output

None.

Example

```
DTR> DECLARE COMMAND_KEYBOARD LONG.
DTR> COMMAND_KEYBOARD = FN$COMMAND_KEYBOARD
```

FN$COS

Calculates the cosine of input.

Input

A signed decimal number (radians).

Output

A signed decimal number.

Example

```
DTR> PRINT FN$COS (3.14159) using s9.999
FN$COS
-1.000
DTR>
```

FN$CREATE_LOG

Assigns a user mode logical name within the process logical name table as a synonym for a physical name. The logical name is deleted when you exit from DATATRIEVE.

Input

This function takes two parameters as input. The first is a logical name character string; the second is a physical name character string. Each string must be in quotation marks (unless you use variables previously declared). The two strings must be separated by a comma and enclosed in parentheses.

Output

None.

Example

```
DTR> FN$CREATE_LOG ("HANK", "DB0:[MORRISON.RW]LOG.RNO")
DTR>
```

Usage Note

Logical names are defined at execution time (runtime) and not during the compilation phase. If you define a logical name within a statement, the logical name is not translated during the compilation phase, consequently it is executed without being defined, which will lead to an error.

```
DTR> READY YACHTS
DTR> BEGIN
[Looking for statement]
CON> FN$CREATE_LOG("FORM_DIR","DTR$LIBRARY:FORMS");
CON> FOR X IN YACHTS
[Looking for statement]
CON> WITH_FORM YACHT IN FORM_DIR
[Looking for SEND or RECEIVE statement]
CON> SEND FROM X TO BOAT;
CON> END
Error opening DECforms form file DISK:[DALFY]FORM_DIR.EXE; .
```

FN$DATE

Converts a date string to a 64-bit data value.

Input

A complete date string formatted as "dd-MMM-yyyy hh:mm:ss.cc". Unpredictable results may occur if you supply only a portion of a date string, such as "21-MAR-1990". The month must be specified in uppercase.

Output

A date data value formatted as "dd-MMM-yyyy hh:mm:ss.cc".

Example

```
DTR> DECLARE X USAGE DATE EDIT_STRING X(23).
DTR> X = FN$DATE("30-AUG-1990 15:20:31.45")
DTR> PRINT X

            X

30-Aug-1990 15:20:31.45

DTR>

DTR> DECLARE Y USAGE DATE EDIT_STRING X(23).
DTR> DECLARE Z PIC X(23).
DTR> Z = "20-FEB-1990 14:54:29.83"
DTR> Y = FN$DATE (Z)
DTR> PRINT Y

            Y

20-Feb-1990 14:54:29.83

DTR>
```

FN$DAY

Extracts the day part of input (dd in dd-MMM-yyyy hh:mm:ss.cc).

Input

A date.

Output

An unsigned integer from 1 to 31.

Example

```
DTR> DECLARE CAL USAGE DATE EDIT_STRING X(23).
DTR> CAL = "NOW"; PRINT CAL

          CAL

 1-Feb-1990 08:51:11.55

DTR> PRINT FN$DAY (CAL)

  FN$DAY

          1

DTR>
```

FN$DCL

Allows you to spawn directly from your main DATATRIEVE process to execute a specified DCL command.

Input

Type FN$DCL at the DATATRIEVE prompt. On the same line, specify the DCL command argument you want to spawn to, such as the DCL print command. The DCL command argument can be any DCL command and must be placed inside quotation marks (unless you use variables previously declared), and enclosed in parentheses.

Output

Your DATATRIEVE process is suspended and the terminal is attached to the subprocess. The DCL command is executed.

Examples

```
DTR> FN$DCL ("PRINT REPORTACCOUNTS.RPT")
Job REPORTACCOUNTS (queue SYSTEMPRINT$QUEUE, entry 148)
started on PRINTER$LPA0
DTR>
```

In the above example the message appears indicating that the job has been added to the print queue. After the command has completed or you have exited from the program initiated by the command, you see the DTR> prompt. This shows that control has been returned to the original process in DATATRIEVE.

The following example shows how you can use variables with FN$DCL:

```
DTR> DECLARE ALP PIC X(30).
DTR> ALP = "DIR/COL=1"
DTR> FN$DCL (ALP)

Directory MY$DISK:[DALFY]

TEST.OBJ;1
ZTEST.FOR;3
```

Note that the FN$DCL process inherits attributes from the caller (that is, the main DATATRIEVE process from which it spawned). Refer to the VMS documentation on run-time library routines for more information.

Usage Notes

- The FN$DCL and FN$SPAWN functions are disabled by default for users in captive accounts.

  You can change this default and allow use of the FN$DCL and FN$SPAWN functions from captive accounts by using the logical name DTR$CAPTIVE_ALLOWED. Before you invoke DATATRIEVE, use the DCL ASSIGN or DEFINE command to define or assign the logical name DTR$CAPTIVE_ALLOWED to be either FN$DCL, FN$SPAWN, or both.

  ```
  $ DEFINE DTR$CAPTIVE_ALLOWED FN$DCL
  $ DEFINE DTR$CAPTIVE_ALLOWED FN$SPAWN
  $ DEFINE DTR$CAPTIVE_ALLOWED FN$DCL, FN$SPAWN
  ```

  You can type FN$DCL and FN$SPAWN in either uppercase or lowercase; the translation is not case sensitive. If you define DTR$CAPTIVE_ALLOWED to be both FN$DCL and FN$SPAWN and use quotation marks in the definition or assignment, you must use a set of quotation marks for each function and separate them with a comma:

  ```
  $ DEFINE DTR$CAPTIVE_ALLOWED "FN$DCL", "FN$SPAWN"
  ```

  Because DATATRIEVE checks the value of DTR$CAPTIVE_ALLOWED during startup, you cannot change the definition of DTR$CAPTIVE_ALLOWED from within DATATRIEVE using the function FN$CREATE_LOG.

  If the translation of DTR$CAPTIVE_ALLOWED is not valid, DATATRIEVE continues as if the logical name were not defined, but does not issue a message notifying you that it is ignoring the logical name.

  If your application allows users to enter commands at the DTR> prompt and you define DTR$CAPTIVE_ALLOWED as FN$DCL, be aware that users can spawn a process by entering SPAWN with the FN$DCL function:

  ```
  DTR> FN$DCL ("SPAWN")
  $
  ```

  You can allow a captive account to use the FN$DCL function to access DCL commands other than the DCL SPAWN command. To exclude SPAWN, set the AUTHORIZE parameter PRCLM to a value of 1 for the captive account.

- You cannot use FN$DCL in a DECwindows environment unless you have invoked DATATRIEVE from a DCL command line in a DECterm window.
- If you use FN$DCL in a DATATRIEVE DECwindows session, the subprocess is spawned to the DECterm window from which you invoked DATATRIEVE.

FN$DCL

The subprocess displays any output generated by the DCL command included as the argument to the FN$DCL function. This output appears in the DECterm window. To see the output, you may want to move the DECterm window from back to front. When the subprocess is complete, the DTR> prompt in the main application window is reactivated.

FN$DEFINE_KEY

Takes a key definition in DCL DEFINE/KEY syntax, then creates the defined key in DATATRIEVE.

This function is not available in a DECwindows environment.

Input

A quoted string containing the key definition in DCL DEFINE/KEY command syntax.

You must use single quotation marks for the outer pair of quotation marks. The inner pair of quotation marks must be double.

Output

None.

Example

```
DTR> FN$KEYPAD_MODE ("APPLICATION")
DTR> FN$DEFINE_KEY ('DEFINE/KEY/ECHO/NOTERMINATE KP7 "FIND" ')
```

Usage Note

The previous example will not work unless the device is set to APPLICATION. DATATRIEVE ignores the the terminal characteristics of your device, for this reason you must execute the FN$KEYPAD_MODE function before executing the FN$DEFINE_KEY function.

FN$DELETE_KEY

Deletes a key definition currently in effect.

This function is not available in a DECwindows environment.

Input

This function takes two parameters.

- The first parameter is the name of the key whose definition you want to delete.
- The second parameter is the state string. You must specify the state name DEFAULT when there is no alternate state.

Each parameter must be in matching quotation marks. The two parameters must be separated by a comma and enclosed in parentheses.

Output

None.

Example

```
DTR> FN$DELETE_KEY ("KP0","DEFAULT")
DTR> FN$DELETE_KEY ("KP0","GOLD")
```

FN$DELETE_LOG

Deletes the assignment of a logical name.

Input

The logical name you want to delete.

Output

None.

Example

```
DTR> FN$DELETE_LOG ("HANK")
DTR> PRINT FN$TRANS_LOG ("HANK") USING X(30)

          FN$TRANS
            LOG

DTR>
```

FN$EXP

Calculates the value of **e** to a specified power.

Input

A signed decimal number.

Output

A signed decimal number.

Example

```
DTR> PRINT FN$EXP (2)
          FN$EXP
 7.3891E+00
DTR>
```

FN$FLOOR

Truncates the decimal part of positive input or rounds negative input.

Input

A signed decimal number.

Output

A signed decimal number.

Example

```
DTR> PRINT FN$FLOOR (59.99)
  FN$FLOOR
 5.9000E+01
DTR> PRINT FN$FLOOR (-59.99)
  FN$FLOOR
 -6.0000E+01
```

FN$GET_SYMBOL

Returns the value of a DCL symbol.

Input

A character string that represents the symbol name.

Output

The value of a DCL symbol.

Example

```
DTR> PRINT FN$GET_SYMBOL("DTR$INVOKE") USING X(45)

                    FN$GET
                    SYMBOL

RUN DTR$DISK:[DTR.TESTS]DTRV4.EXE

DTR>
```

FN$HEX

Calculates the hexadecimal equivalent of input.

Input

A signed integer no larger than (2 ** 31) –1 (the maximum value that can be stored in a signed longword).

Output

A hexadecimal character string.

Example

```
DTR> PRINT FN$HEX(183)
 FN$HEX
      B7
DTR>
```

FN$HOUR

Extracts the hour part of input (hh in dd-MMM-yyyy hh:mm:ss.cc).

Input

A date.

Output

An unsigned integer from 1 to 24.

Example

```
DTR> DECLARE CAL USAGE DATE EDIT_STRING X(23).
DTR> CAL = "NOW"; PRINT CAL

            CAL

 1-Feb-1990 08:51:11.55

DTR> PRINT FN$HOUR (CAL)

  FN$HOUR

              8

DTR>
```

FN$HUNDREDTH

Extracts hundredth-of-a-second part of input (cc in dd-MMM-yyyy hh:mm:ss.cc).

Input

A date.

Output

An unsigned integer from 0 to 99.

Example

```
DTR> DECLARE CAL USAGE DATE EDIT_STRING X(23).
DTR> CAL = "NOW"; PRINT CAL

            CAL

 1-Feb-1990 08:51:11.55

DTR> PRINT FN$HUNDREDTH (CAL)

  FN$HUNDREDTH

            55

DTR>
```

FN$INIT_TIMER

Initializes a timer and counter.

Input

None.

Output

None.

Example

```
DTR> SHOW TIME_READY
PROCEDURE TIME_READY
FN$INIT_TIMER
READY OWNERS
FN$SHOW_TIMER
END_PROCEDURE

DTR> :TIME_READY

ELAPSED: 0 00:00:04.24 CPU: 0:00:00.61 BUFIO: 1 DIRIO: 42
                                                    FAULTS: 64

DTR>
```

FN$JULIAN

Calculates the Julian date of input.

(A Julian date is based on days of the year being numbered beginning with January 1st. The Julian date of January 6th is 6. The Julian date of February 2nd is 33, and so on.)

Input

A date.

Output

An unsigned integer from 1 to 366. (There are 366 days in a leap year.)

Example

```
DTR> DECLARE CAL USAGE DATE EDIT_STRING X(23).
DTR> CAL = "NOW"; PRINT CAL

           CAL

 1-Feb-1990 08:51:11.55
DTR> PRINT FN$JULIAN (CAL)

  FN$JULIAN

          32

DTR>
```

FN$KEYPAD_MODE

Lets you specify the terminal mode from within DATATRIEVE.

This function duplicates the ability of the SET [NO] APPLICATION_KEYPAD command. The function form of FN$KEYPAD_MODE, however, allows you to change the keypad mode inside compound statements.

This function is not available in a DECwindows environment.

Input

You must specify one of two parameters:

- APPLICATION specifies application keypad mode.
- NUMERIC specifies numeric keypad mode.

This function is not case sensitive; you can type APPLICATION or NUMERIC in either uppercase or lowercase. The words must be in either single or double quotation marks, and the spelling must be exact.

Output

None.

Example

```
DTR> FN$KEYPAD_MODE ("APPLICATION")
DTR> FN$KEYPAD_MODE ("NUMERIC")
```

FN$KEYTABLE_ID

Returns the value of the KEYTABLE_ID field currently in the data access block (DAB).

FN$KEYTABLE_ID allows you to add other functions that access other capabilities of the Screen Management Guidelines (SMG) of the Screen Management Facility.

This function is not available in a DECwindows environment.

Input

None.

Output

None.

Example

```
DTR> DECLARE KEYTABLE LONG.
DTR> KEYTABLE = FN$KEYTABLE_ID
```

FN$LN

Calculates the natural log of input.

Input

A signed number.

Output

A signed number.

Example

```
DTR> PRINT FN$LN (36)
  FN$LN
 3.5835E+00
DTR>
```

FN$LOAD_KEYDEFS

Lets you define multiple keypad keys from a file containing DCL DEFINE/KEY commands. This way you do not have to make multiple calls to the FN$DEFINE_KEY function.

This function is not available in a DECwindows environment.

Input

A quoted string with a DCL file specification indicating the file containing the DCL DEFINE/KEY commands.

Output

None.

Example

```
DTR> FN$LOAD_KEYDEFS ("APPL1.KEYS")
```

FN$LOG10

Calculates the base 10 log of input.

Input

A signed number.

Output

A signed number.

Example

```
DTR> PRINT FN$LOG10 (36)
  FN$LOG10
 1.5563E+00
DTR>
```

FN$MINUTE

Extracts the minute part of input (mm in dd-MMM-yyyy hh:mm:ss.cc).

Input

A date.

Output

An unsigned integer from 0 to 59.

Example

```
DTR> DECLARE CAL USAGE DATE EDIT_STRING X(23).
DTR> CAL = "NOW"; PRINT CAL

            CAL

 1-Feb-1990 08:51:11.55

DTR> PRINT FN$MINUTE (CAL)

  FN$MINUTE

          51

DTR>
```

FN$MOD

Calculates the value of input according to a specified modulus.

Input

This function takes two parameters: a signed number and a modulus. The two must be separated by a comma and enclosed in parentheses.

Output

A real number.

Example

```
DTR> PRINT FN$MOD (31,7)
  FN$MOD
 3.0000E+00
DTR>
```

FN$MONTH

Extracts the month part of input (MMM in dd-MMM-yyyy hh:mm:ss.cc).

Input

A date.

Output

An unsigned integer from 1 to 12.

Example

```
DTR> DECLARE CAL USAGE DATE EDIT_STRING X(23).
DTR> CAL = "NOW"; PRINT CAL

          CAL

 1-Feb-1990 08:51:11.55

DTR> PRINT FN$MONTH (CAL)

  FN$MONTH

            2

DTR>
```

FN$NINT

Calculates the integer nearest to input.

Input

A signed number.

Output

A signed integer.

Example

```
DTR> PRINT FN$NINT (59.99)
  FN$NINT
         60
DTR>
```

FN$OPENS_LEFT

Calculates the number of additional files you can open.

Input

None.

Output

An unsigned integer.

Example

```
DTR> PRINT FN$OPENS_LEFT

  FN$OPENS
    LEFT

           3

DTR>
```

FN$PROMPT_KEYBOARD

Returns the value of the PROMPT_KEYBOARD field currently in the data access block (DAB). PROMPT_KEYBOARD is the keyboard ID used for prompting.

FN$PROMPT_KEYBOARD allows you to add other functions that access other capabilities of the Screen Management Guidelines (SMG) of the Screen Management Facility.

This function is not available in a DECwindows environment.

Input

None.

Output

None.

Example

```
DTR> DECLARE PROMPT_KEYBOARD LONG.
DTR> PROMPT_KEYBOARD = FN$PROMPT_KEYBOARD
```

FN$SECOND

Extracts the second part of input (ss in dd-MMM-yyyy hh:mm:ss.cc).

Input

A date.

Output

An unsigned integer from 0 to 59.

Example

```
DTR> DECLARE CAL USAGE DATE EDIT_STRING X(23).
DTR> CAL = "NOW"; PRINT CAL

          CAL

 1-Feb-1990 08:51:11.55

DTR> PRINT FN$SECOND (CAL)

  FN$SECOND

          11

DTR>
```

FN$SHOW_KEY

Displays the definition of a keypad key.

This function is not available in a DECwindows environment.

Input

This function takes two parameters:

- The first parameter specifies the keypad key name.
- The second parameter is the state string. You must specify the state name DEFAULT when there is no alternate state.

Each parameter must be in matching quotation marks. The two quoted parameters must be separated by a comma and enclosed in parentheses.

Output

The definition of the specified key.

Example

```
DTR> FN$SHOW_KEY ("KP7","DEFAULT")

   KP7 = "SHOW ALL"
       (echo,terminate,noerase,nolock)
```

FN$SHOW_KEYDEFS

Displays the key definitions in all of the states. This function duplicates the ability of the DATATRIEVE SHOW KEYDEFS command.

The function form of FN$SHOW_KEYDEFS lets you show all keypad definitions inside compound statements.

This function is not available in a DECwindows environment.

Input

None.

Output

The definitions of all the defined keys.

Example

```
DTR> FN$SHOW_KEYDEFS

BLUE state keypad definitions:
    KP1 = "SHOW KEYDEFS"
        (noecho,terminate,noerase,nolock)
DEFAULT state keypad definitions:
    PF1 = " "
        (echo,noterminate,noerase,nolock,set_state=GOLD)
    PF4 = " "
        (echo,noterminate,noerase,nolock,set_state=BLUE)
    KP0 = "SHOW KEYDEFS"
        (echo,terminate,noerase,nolock)
    KP1 = "READY "
        (echo,noterminate,noerase,nolock)
    KP7 = "SHOW ALL"
        (echo,terminate,noerase,nolock)
    KP8 = "SHOW DOMAINS"
        (echo,terminate,noerase,nolock)
    KP9 = "SHOW RECORDS"
        (echo,terminate,noerase,nolock)
    ENTER = "SET APPLICATION_KEYPAD"
        (echo,terminate,noerase,nolock)
GOLD state keypad definitions:
    ENTER = "SET NO APPLICATION_KEYPAD"
        (echo,terminate,noerase,nolock)
    KP1 = "SHOW KEYDEFS"
        (noecho,terminate,noerase,nolock)
```

FN$SHOW_TIMER

Shows elapsed time since the timer was last initialized.

Note that DATATRIEVE does not include the information displayed by FN$SHOW_TIMER in a log file created with the OPEN command.

Input

None.

Output

Displays the elapsed time in this format:

```
D HH:MM:SS.SS
```

Example

```
DTR> SHOW TIME_READY
PROCEDURE TIME_READY
FN$INIT_TIMER
READY OWNERS
FN$SHOW_TIMER
END_PROCEDURE

DTR> :TIME_READY
ELAPSED: 0 00:00:04.24 CPU: 0:00:00.61 BUFIO: 1 DIRIO: 42
                                                    FAULTS: 64

DTR>
```

At the end of a 24-hour period, the timer begins displaying time in the number of days rather than the accumulated number of hours.

FN$SIGN

Indicates the sign of a number.

Input

A signed number.

Output

1, –1, or 0 (depending on the sign of the number).

Example

```
DTR> PRINT FN$SIGN (-4)
  FN$SIGN
          -1
DTR>
```

FN$SIN

Calculates the sine of input.

Input

A signed decimal number (radians).

Output

A signed decimal number.

Example

```
DTR> PRINT FN$SIN (3.14159/2)
FN$SIN
1.000
DTR>
```

FN$SPAWN

Creates a subprocess by calling the Run-Time Library (RTL) routine LIB$SPAWN.

Input

Type FN$SPAWN at the DTR> prompt to create the subprocess.

Output

Your default DCL prompt appears on the screen. You can then invoke utilities or enter commands.

Type LOGOUT after the DCL prompt to return to your original process in DATATRIEVE. A message is printed on the screen indicating that you have logged out of the subprocess. DATATRIEVE generates the DTR> prompt, showing that control has been returned to the original process in DATATRIEVE.

Example

```
DTR> FN$SPAWN
$ MAIL
  .
  .
  .
MAIL> EXIT
$ LOGOUT
  Process PROCESSNAME_1 logged out at 25-FEB-1990 09:47:09:27
DTR>
```

Usage Notes

- Use FN$SPAWN as a single and complete DATATRIEVE statement. Do not use this function within another simple or compound statement (unlike other functions).

  The FN$SPAWN function inherits attributes from the caller (the main DATATRIEVE process it spawned from). Refer to the VMS documentation on run-time library routines for more information.

- The FN$DCL and FN$SPAWN functions are disabled by default for users in captive accounts.

You can change this default and allow use of the FN$DCL and FN$SPAWN functions from captive accounts by using the logical name DTR$CAPTIVE_ALLOWED. Before you invoke DATATRIEVE, use the DCL ASSIGN or DEFINE command to define or assign the logical name DTR$CAPTIVE_ALLOWED to be either FN$DCL, FN$SPAWN, or both.

```
$ DEFINE DTR$CAPTIVE_ALLOWED FN$DCL
$ DEFINE DTR$CAPTIVE_ALLOWED FN$SPAWN
$ DEFINE DTR$CAPTIVE_ALLOWED FN$DCL, FN$SPAWN
```

You can type FN$DCL and FN$SPAWN in either uppercase or lowercase; the translation is not case sensitive. If you define DTR$CAPTIVE_ALLOWED to be both FN$DCL and FN$SPAWN and use quotation marks in the definition or assignment, you must use a set of quotation marks for each function and separate them with a comma:

```
$ DEFINE DTR$CAPTIVE_ALLOWED "FN$DCL", "FN$SPAWN"
```

Because DATATRIEVE checks the value of DTR$CAPTIVE_ALLOWED during startup, you cannot change the definition of DTR$CAPTIVE_ALLOWED from within DATATRIEVE using the function FN$CREATE_LOG.

If the translation of DTR$CAPTIVE_ALLOWED is not valid, DATATRIEVE continues as if the logical name were not defined, but does not issue a message notifying you that it is ignoring the logical name.

If your application allows users to enter commands at the DTR> prompt and you define DTR$CAPTIVE_ALLOWED as FN$DCL, be aware that users can spawn a process by entering SPAWN with the FN$DCL function:

```
DTR> FN$DCL ("SPAWN")
$
```

You can allow a captive account to use the FN$DCL function to access DCL commands other than the DCL SPAWN command. To exclude SPAWN, set the AUTHORIZE parameter PRCLM to a value of 1 for the captive account.

- You cannot use FN$SPAWN in a DECwindows environment unless you have invoked DATATRIEVE from a DCL command line in a DECterm window.
- If you use FN$SPAWN during a DATATRIEVE DECwindows session, the subprocess is spawned to the DECterm window from which you invoked DATATRIEVE.

  You must input any commands to your FN$SPAWN process from the DCL prompt of the DECterm window. When you have completed your subprocess, enter the LOGOUT command at the DCL prompt to return control to your DATATRIEVE session.

FN$SQRT

Calculates the square root of the input number.

Input

Zero or a positive decimal number.

Output

Zero or a positive decimal number.

Example

```
DTR> PRINT FN$SQRT (196)
  FN$SQRT
 1.4000E+01
DTR>
```

FN$STR_EXTRACT

Extracts a substring from the input character string using a default edit string of 30 characters.

Input

This function takes three parameters:

- A character string
- An ordinal number of starting character (numerical position within string)
- The length of desired substring

Output

A substring.

Example

```
DTR> DECLARE WOMBAT PIC X(25).
DTR> WOMBAT = "Wombats have sharp claws."
DTR> PRINT FN$STR_EXTRACT (WOMBAT,9,4)

           FN$STR
           EXTRACT

have

DTR>
```

FN$STR_LOC

Calculates the starting position of the specified substring in the input character string.

Input

This functions takes two parameters: a character string and a substring.

Output

An unsigned integer. If the string is undefined, FN$STR_LOC returns the value 0.

Example

```
DTR> DECLARE WOMBAT PIC X(25)
DTR> WOMBAT = "Wombats have sharp claws."
DTR> PRINT FN$STR_LOC (WOMBAT,"claws")

 FN$STR
  LOC

          20

DTR>
```

FN$TAN

Calculates the tangent of the input number.

Input

A signed decimal number (radians).

Output

A signed decimal number.

Example

```
DTR> PRINT FN$TAN (3.14159/4)
  FN$TAN
 1.0000E+00
DTR>
```

FN$TIME

Extracts the time part of input (hh:mm:ss.cc in dd-MMM-yyyy hh:mm:ss.cc).

Input

A date.

Output

The time in VMS format.

Example

```
DTR> DECLARE CAL USAGE DATE EDIT_STRING X(23).
DTR> CAL = "NOW"; PRINT CAL

          CAL

 1-Feb-1990 08:51:11.55

DTR> PRINT FN$TIME (CAL)

  FN$TIME

 08:51:11.5

DTR>
```

FN$TRANS_LOG

Translates a logical name.

Input

A character string containing the logical name you want to translate.

Output

A character string.

Example

```
DTR> FN$CREATE_LOG ("HANK", "DB0:[MORRISON.RW]LOG.RNO")
DTR> PRINT FN$TRANS_LOG ("HANK") USING X(30)

             FN$TRANS
               LOG

DB0:[MORRISON.RW]LOG.RNO

DTR>
```

FN$UPCASE

Changes the characters in a string to uppercase.

Input

A character string.

Output

The input character string, all in uppercase.

Example

```
DTR> DECLARE WOMBAT PIC X(25).
DTR> WOMBAT = "Wombats have sharp claws."
DTR> PRINT FN$UPCASE (WOMBAT)

          FN$UPCASE

WOMBATS HAVE SHARP CLAWS.

DTR>
```

Usage Note

DATATRIEVE truncates the output string at 30 characters. To display a string longer than 30 characters, specify the length of the string in a USING clause.

FN$WEEK

Calculates the week number for a date you enter.

(The week number is an integer from 1 to 52. A week number is assigned sequentially to each week, beginning with the first week of the year. The first week of January is week number 1, the second week of January is week number 2, and so on.)

Input

A date.

Output

An unsigned integer from 1 to 52.

Example

```
DTR> DECLARE CAL USAGE DATE EDIT_STRING X(23).
DTR> CAL = "NOW"; PRINT CAL

          CAL

 1-Feb-1983 08:51:11.55

DTR> PRINT FN$WEEK (CAL)

  FN$WEEK

         5

DTR>
```

Usage Note

The FN$WEEK function does not use Sunday as the default beginning of each week (unlike most calendars); instead, it begins its calculations with the first day of each year. It uses the day of the week on which January 1st occurs as the first day of each subsequent week that year. For example, January 1st occurred on a Wednesday in 1986, so FN$WEEK calculates all week numbers in 1986 as though each week begins on Wednesday. Thus, the first Sunday, Monday, and Tuesday in 1986 are still part of week number 1.

Use the following procedure and table to print the week of the year with Sunday as the first day of the week:

```
DTR> DEFINE PROCEDURE OTHER_WEEK
DFN> !
DFN> !  declare variables for the various operations
DFN> !
DFN> DECLARE DATE USAGE DATE EDIT_STRING X(23).
DFN> DECLARE YEAR PIC 9(4).
DFN> DECLARE TEMP PIC 9(3).
DFN> DECLARE WEEK PIC 9(2).
DFN> !
DFN> DATE = *."date"
DFN> YEAR = FN$YEAR(DATE)
DFN> TEMP = YEAR VIA FIRST_DAY_TBL
DFN> WEEK = FN$WEEK(DATE + TEMP)
DFN> PRINT "The week of the year is ", WEEK (-) USING X9
DFN> END_PROCEDURE
DTR> DEFINE TABLE FIRST_DAY_TBL
DFN> 1984:0
DFN> 1985:2
DFN> 1986:3
DFN> 1987:4
DFN> 1988:5
DFN> 1989:6
DFN> 1990:7
DFN> ELSE 0
DFN> END_TABLE
DTR>
```

The procedure calls a number from the table. This number represents the difference between the actual first day of the year and Sunday. The procedure adds this number to the date you have entered. The first week will then begin on Sunday; the OTHER_WEEK procedure will use Sunday as the default beginning of subsequent weeks.

Note that the numbers are stored in the table by year. The years shown in the table are only a sample; you can include as many years as you need.

FN$WIDTH

Changes the character width of the terminal.

Input

An integer.

Output

None.

Example

```
DTR> SET COLUMNS_PAGE = 132
DTR> FN$WIDTH (132)
```

FN$WIDTH (132) sets the terminal's width at 132 characters or columns. The SET COLUMNS_PAGE command ensures that any output produced by REPORT, PRINT, SUM, or LIST statements is spaced across the 132 columns.

Usage Note

You should not use the FN$WIDTH function to adjust terminal width in a DECwindows environment. DATATRIEVE continues to function if you use FN$WIDTH in a DECwindows environment; however, you may get unexpected output.

FN$YEAR

Extracts the year part of input (yyyy in dd-Mmm-yyyy hh:mm:ss.cc).

Input

A date.

Output

An unsigned integer for years 1858 to 9999.

Example

```
DTR> DECLARE CAL USAGE DATE EDIT_STRING X(23).
DTR> CAL = "NOW"; PRINT CAL

          CAL

 1-Feb-1990 08:51:11.55

DTR> PRINT FN$YEAR (CAL)

  FN$YEAR

       1990

DTR>
```

3.4 Optimizing Function Execution

DATATRIEVE executes functions within loops in two ways:

- For some functions, DATATRIEVE optimizes their execution within loops. Optimizing means that DATATRIEVE factors the function out of the loop and executes it once at the top of the loop.
- For the remainder of the functions, DATATRIEVE does not optimize their execution within loops. Instead, DATATRIEVE executes these functions for each iteration of the loop.

You can determine the default for each function by looking at the function definitions. Function definitions for all functions provided with the DATATRIEVE kit are found in DTR$LIBRARY:IDTRFND.MAR. Site specific or user-defined functions are found in DTR$LIBRARY:DTRFNDxx.MAR. [1]

- If the function definition does not include the DTR$FUN_NOOPTIMIZE statement, DATATRIEVE optimizes the function execution in loops. In the following example, FN$COS (3.14159) is computed only once, since the result does not vary between iterations of the FOR loop:

```
READY VALUES
FOR VALUES
    PRINT X * FN$COS (3.14159)
```

- If a function definition does include the DTR$FUN_NOOPTIMIZE statement, DATATRIEVE does not optimize the execution of the function in loops. The following functions are not optimized because, in most instances, they should be executed for each iteration of a loop:

```
FN$CREATE_LOG
FN$DELETE_LOG
FN$INIT_TIMER
FN$SHOW_TIMER
FN$TRANS_LOG
FN$WIDTH
```

You can control the optimization of function execution for user-defined functions by editing the function definitions in DTR$LIBRARY:DTRFNDxx.MAR to include or to delete the DTR$FUN_NOOPTIMIZE statement.

See the *VAX DATATRIEVE Guide to Programming and Customizing* for more information about changing function definitions and adding your own functions to DATATRIEVE.

[1] xx represents the 1 to 26-character suffix that may be added at installation.

4

DATATRIEVE Commands, Statements, and Definition Clauses

This chapter describes all DATATRIEVE commands, statements, and record and field definition clauses, and presents them in alphabetical order. Before you use a command, statement, or clause, read its description completely.

Structure of DATATRIEVE Command Descriptions

The description or each DATATRIEVE command, statement, or clause is divided into the following categories of information:

- **Format**

  The format of the command, statement, or clause includes the spelling and placement of keywords and the placement of required and optional syntax elements. As a rule, command and statement names and other keywords cannot be abbreviated.

  The sequence of command, statement, or clause elements is also critical. You must follow the sequence shown in the format. If you omit an optional element, leave its relative position in the command or statement empty and proceed to the remaining elements in a left-to-right order.

  Statements cannot use path names in place of given names. You can use path names with commands but not with statements. The FINISH command is the only exception to this rule.

- **Arguments**

  The arguments section explains each element of the syntax in greater detail.

- **Restrictions**

  The restrictions tell you what requirements and limits there are on the use and action of a command, statement, or clause. They also list the access privileges you must have to enter a command or statement.

- **Results**

  Results tell you what action DATATRIEVE takes when you use the command or statement and its various options.

- **Usage Notes**

  Usage Notes present some common uses of the command, statement, or clause and its elements and indicate what other commands and statements you can use in conjunction with the command or statement.

- **Examples**

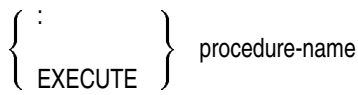
  The examples show the use of representative sequences of commands and statements. In the case of clauses, they show sample record or field definitions.

Symbols and conventions used in syntax formats are listed at the beginning of this manual.

: (EXECUTE)

Invokes a DATATRIEVE procedure.

Format



Argument

procedure-name
Is the given name, full dictionary path name, or relative dictionary path name of the DATATRIEVE procedure you want to invoke.

Restrictions

- To invoke a procedure, you must have P (PASS_THRU), S (SEE), and E (DTR EXECUTE/EXTEND) access to it.
- You cannot invoke a procedure during an ADT, EDIT, or Guide Mode session.
- You cannot include the invocation of a procedure in the definition of a domain, record, or table.
- Do not allow a procedure to invoke itself, either directly or indirectly; you may create an infinite loop.
- If the procedure contains any commands, you cannot include an invocation of the procedure within a BEGIN-END block.

Results

- If the procedure consists of full commands or statements, DATATRIEVE executes each command or statement in the procedure in order.
- If the procedure contains only a clause or an argument from a command or statement, DATATRIEVE includes the procedure within the command or statement containing the procedure invocation.

: (EXECUTE)

Usage Notes

- You can use either a colon (:) or EXECUTE before the procedure name. The results are the same.
- You can nest procedures by invoking a procedure within another procedure, but you must be careful not to allow the procedure to invoke itself.
- You can invoke a procedure in a REPEAT statement to execute it a number of times or in a FOR statement to apply it to a collection of records. You must, however, use care when invoking a procedure in these statements. For example, the following syntax can be used, but the results may be unexpected:

  ```
  REPEAT n :procedure-name
  ```

  This statement does not execute the procedure n times. When DATATRIEVE encounters the first complete statement in the procedure, it assumes that the REPEAT statement is also complete. Therefore, it executes the first command or statement in the procedure n times. DATATRIEVE then executes the remaining commands or statements in the procedure once.

  To repeat the entire procedure n times, enclose the procedure invocation in a BEGIN-END block:

  ```
  REPEAT n
      BEGIN
        :procedure-name
      END
  ```

  Use a similar technique for procedure invocations controlled by a FOR loop. For example:

  ```
  FOR rse
      BEGIN
         :procedure-name
      END
  ```

  Remember that if you use a procedure in this way, it cannot include a FIND, SELECT, or DROP statement because these statements cannot be used in BEGIN-END blocks.

Examples

The following example invokes a procedure to find the employee in PERSONNEL with the largest salary. It uses EXECUTE to invoke the procedure from the DIGITAL Command Language (DCL) level. In this example, DTR is the global symbol for invoking VAX DATATRIEVE.

```
DTR> SHOW MAX_SALARY
PROCEDURE MAX_SALARY
READY PERSONNEL
PRINT PERSONNEL WITH SALARY = MAX SALARY OF PERSONNEL
END_PROCEDURE

DTR> EXIT
$ DTR EXECUTE MAX_SALARY

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

00012 EXPERIENCED CARLA  SPIVA  TOP 12-Sep-1972  $75,892 00012

$
```

The following example invokes a procedure three times. The procedure displays employees in a given department with salaries greater than $40,000.

```
DTR> SHOW BIG_SALARY
PROCEDURE BIG_SALARY
FOR PERSONNEL WITH DEPT  = *."the department"
BEGIN
  IF SALARY GT 40000
  THEN PRINT ID, NAME, DEPT,START_DATE, SALARY
END
END_PROCEDURE

DTR> REPEAT 3
CON> BEGIN
CON>    :BIG_SALARY
CON> END
Enter the department: F11

        FIRST       LAST            START
 ID     NAME        NAME    DEPT    DATE     SALARY

00891 FRED       HOWL       F11   9-Apr-1976 $59,594
78923 LYDIA      HARRISON   F11  19-Jun-1979 $40,747
Enter the department: T32
38462 BILL       SWAY       T32   5-May-1980 $54,000
83764 JIM        MEADER     T32   4-Apr-1980 $41,029
Enter the department: TOP
00012 CHARLOTTE  SPIVA      TOP  12-Sep-1972 $75,892

DTR>
```

: (EXECUTE)

The following example invokes a procedure to specify an edit string clause for a variable:

```
DTR> DEFINE PROCEDURE E_S
DFN> EDIT_STRING IS $$,$$$.99
DFN> END_PROCEDURE
DTR> DECLARE PRICE_PER_FT COMPUTED BY PRICE/LOA :E_S.
DTR> PRINT TYPE, PRICE_PER_FT OF FIRST 5 YACHTS

                          PRICE
                           PER
MANUFACTURER   MODEL        FT

 ALBERG      37 MK II     $998.68
 ALBIN       79           $688.46
 ALBIN       BALLAD       $916.67
 ALBIN       VEGA         $688.89
 AMERICAN    26           $380.58

DTR>
```

@ (Invoke Command File)

Invokes a command file.

Format

@ file-spec

Argument

file-spec
Is the file specification for the file you want to execute.

Restrictions

- When you invoke a command file, it must be on a line by itself.
- You cannot invoke command files while you are using the Application Design Tool (ADT) or Guide Mode.
- You cannot invoke a command file in a loop created by the FOR, REPEAT, or WHILE statements.
- You cannot include a command file invocation within a BEGIN-END block.
- Do not allow a command file to invoke itself, either directly or indirectly; you can create an infinite loop.
- When you invoke a command file using the DCL DATATRIEVE command, you must use double quotation marks around the @ command and the command file. This is to prevent the operating system from interpreting the commands contained in the command file as a DCL commands.

Results

- If you do not include a file type in the file specification, DATATRIEVE uses a .COM file type as a default. If there is no file in the specified VMS directory with a .COM file type, DATATRIEVE uses a .DTR file type as a default.
- If the command file consists of full commands or statements, DATATRIEVE executes each command or statement in the command file in order.
- If the command file ends with an incomplete DATATRIEVE command or statement, DATATRIEVE returns a CON> prompt, waiting for you to supply the missing elements.

Usage Notes

- You can include comments in a command file by placing an exclamation mark (!) before each comment line.
- To display the lines of a command file when you invoke it, enter the SET VERIFY command either from DCL or DATATRIEVE command level.

Example

The following example invokes a command file, BUILDER.COM, from DCL to produce a report on yachts by a specified builder.

```
$ DATATRIEVE "@BUILDER"
READY YACHTS
REPORT YACHTS WITH BUILDER = *.BUILDER
SET COLUMNS_PAGE = 70
PRINT BOAT
END_REPORT
Enter BUILDER: ALBIN

                                                  11-Oct-1982
                                                  Page 1

                                LENGTH
                                 OVER
MANUFACTURER    MODEL    RIG    ALL    WEIGHT   BEAM    PRICE

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

$
```

ABORT Statement

Stops the execution of a single statement, an entire procedure, or a command file.

Format

ABORT [value-expression]

Argument

value-expression
Is a DATATRIEVE value expression, usually a character string literal.

Restriction

When DATATRIEVE processes the ABORT statement, ABORT affects the outermost statement that contains it.

When a statement contains nested FOR loops, you cannot use an ABORT statement to transfer control from an inner loop to an outer loop. Similarly, when a statement contains nested BEGIN-END blocks, you cannot use an ABORT statement to transfer control from an inner block to an outer one.

Results

- If the abort occurs during an interactive session, DATATRIEVE stops executing the statement containing the ABORT statement and returns control to the DATATRIEVE command level (indicated by the DTR> prompt). The ABORT statement does not end the interactive session.
- If the abort occurs in a procedure or command file, the result depends on whether SET ABORT is in effect:
  - If SET ABORT is in effect, DATATRIEVE returns to command level without executing the rest of the procedure or command file.
  - If SET NO ABORT is in effect, DATATRIEVE aborts the current statement and then processes any statements and commands remaining in the procedure or command file.
- If the abort occurs within a STORE or MODIFY statement, DATATRIEVE does not store a new record or modify the target record.
- If the abort occurs during an interactive session, DATATRIEVE displays a message containing the value of the value expression you specify in the ABORT statement:

```
DTR> ABORT "Bad value for LOA."
ABORT: Bad value for LOA.
DTR> ABORT 3+2
ABORT: 5
DTR>
```

- If you invoke DATATRIEVE with an invocation command line (for example, $ DTR32 @BIGJOB), the effect of an ABORT statement depends on whether SET ABORT is in effect:
  - If SET ABORT is in effect, DATATRIEVE returns to the DCL command level (indicated by the dollar sign prompt) without executing the rest of the command file.
  - If SET NO ABORT is in effect, DATATRIEVE aborts the statement containing the ABORT statement and then processes any statements and commands remaining in the command file.
- When you submit a batch stream containing a DATATRIEVE command file, any ABORT statements in the command file behave as they would if you had entered them during an interactive session:
  - Whether SET ABORT is in effect or not, DATATRIEVE prints the abort message containing the value of the specified value expression in the log file of the batch job.
  - If SET ABORT is in effect and DATATRIEVE processes an ABORT statement in a procedure or command file, DATATRIEVE returns to the DATATRIEVE command level without executing the rest of the procedure or command file.
  - If SET NO ABORT is in effect and DATATRIEVE processes an ABORT statement in a procedure or command file, DATATRIEVE aborts the current statement and then processes any statements and commands remaining in the procedure or command file.
  - If you invoke DATATRIEVE with an invocation command line in the batch stream and DATATRIEVE processes an ABORT statement, SET ABORT ends the DATATRIEVE session and returns control of the process to the batch stream. SET NO ABORT aborts the current statement, and DATATRIEVE processes the remaining statements in the command file.

  If the batch stream contains other command files queued after the DATATRIEVE command file, the processing of a DATATRIEVE ABORT statement does not end the batch job.

Usage Note

Use an IF-THEN-ELSE statement to establish the conditions for the abort (see the section on the IF-THEN-ELSE statement in this chapter). The Boolean expression in the IF clause establishes the conditions that control the ABORT statement. An abort occurs when:

- The Boolean expression is true and the ABORT statement is in the THEN clause.
- The Boolean expression is false and the ABORT statement is in the ELSE clause.

Examples

The following example shows how to store a record in the YACHTS domain. If the value of the BEAM field is 0, the operation is aborted and a message is displayed.

```
DTR> STORE YACHTS VERIFY USING
[Looking for statement]
DTR> IF BEAM EQ 0 THEN ABORT "Bad value for BEAM"
Enter MANUFACTURER: AMERICAN
Enter MODEL: 1980
Enter RIG: SLOOP
Enter LENGTH_OVER_ALL: 25
Enter DISPLACEMENT: 7500
Enter BEAM: 0
Enter PRICE: 10000
ABORT: Bad value for BEAM
DTR>
```

The following example shows how to define a procedure to write a report on the current collection and abort the entire procedure if there is no current collection to report on.

```
DTR> DEFINE PROCEDURE YACHT_REPORT
DFN> SET ABORT
DFN> PRINT "HAVE YOU ESTABLISHED A CURRENT COLLECTION?"
DFN> IF *."YES or NO" CONTAINING "N" THEN
DFN>      ABORT "SORRY--NO COLLECTION, NO REPORT."
DFN> REPORT
DFN> PRINT BOAT
DFN> AT BOTTOM OF REPORT PRINT COUNT, TOTAL PRICE
DFN> END_REPORT
DFN> END_PROCEDURE
DTR> :YACHT_REPORT
```

```
HAVE YOU ESTABLISHED A CURRENT COLLECTION?
Enter YES or NO: NO
ABORT: SORRY--NO COLLECTION, NO REPORT.
DTR>
```

In the following example, a procedure is defined to update the OWNERS domain after the sale of a boat. The boat is checked against the inventory in the YACHTS domain and the procedure is aborted if the boat is not in YACHTS. Then, the OWNERS domain is checked for a record of the boat. If a record exists, the owner's name is changed and the boat name is updated, if desired. If no record exists, a new record is stored in the OWNERS domain. The procedure requires MODIFY or WRITE access to OWNERS for the update and EXTEND or WRITE access for the entry of a new record.

The example aborts because there is no YACHTS record for an ALBERG 42. The value expression specified in the ABORT statement includes the variables BLD and MOD.

```
DTR> SHOW SALE_BOAT
PROCEDURE SALE_BOAT
READY YACHTS, OWNERS WRITE
SET ABORT
DECLARE BLD PIC X(10).
DECLARE MOD PIC X(10).
BLD = *."BUILDER'S NAME"
MOD = *."MODEL"
IF NOT ANY YACHTS WITH (BUILDER = BLD AND
       MODEL = MOD) THEN
       BEGIN
          PRINT SKIP, "RECORD NOT FOUND IN YACHTS.", SKIP
          ABORT "POSSIBLE INVENTORY ERROR FOR--"||BLD|||MOD
       END ELSE
       PRINT SKIP, "YACHTS RECORD FOUND FOR "|BLD|||MOD
IF ANY OWNERS WITH (BUILDER = BLD AND
       MODEL = MOD) THEN
       BEGIN
          FOR OWNERS WITH (BUILDER = BLD AND
             MODEL = MOD)
          MODIFY USING
             BEGIN
                PRINT COL 10, NAME, SKIP
                NAME = *."NEW OWNER'S NAME"
                PRINT COL 10, BOAT_NAME, SKIP
                IF *."CHANGE BOAT NAME? Y/N"
                   CONTAINING "Y" THEN PRINT SKIP THEN
                   BOAT_NAME = *."NEW BOAT NAME"
                END
       END ELSE
```

```
       STORE OWNERS USING
       BEGIN
          NAME = *."NEW OWNER'S NAME"
          BOAT_NAME = *."BOAT NAME"
          BUILDER = BLD
          MODEL = MOD
       END
END_PROCEDURE
DTR> :SALE_BOAT
Enter BUILDER'S NAME: ALBERG
Enter MODEL: 42
RECORD NOT FOUND IN YACHTS.
ABORT: POSSIBLE INVENTORY ERROR FOR--ALBERG 42
DTR>
```

ADT Command

Invokes the Application Design Tool (ADT), an interactive aid that helps you define a domain, its associated record, and its data file.

Format

```
ADT
```

Arguments

None.

Restrictions

- You cannot invoke ADT from an application program using the DATATRIEVE Call Interface.
- Do not use the ADT command in a command file.
- The record definition and data file definition created by ADT do not contain all the features, clauses, and options available when using the DATATRIEVE DEFINE or REDEFINE commands.
- The length of the domain name cannot exceed 27 characters.
- See the following commands for the access privileges needed to add the domain and record definitions to the data dictionary and to define the data file for the domain: DEFINE DOMAIN, DEFINE RECORD, DEFINE FILE, and DELETE.
- You cannot use ADT to create a new version of a domain or record definition.

Results

- DATATRIEVE invokes the Application Design Tool, which asks you a series of questions about the domain, the size and type of the fields you want in the record, and the name and type of the data file. On VT100-, VT200-, and VT300- family terminals and workstations, the current state of the record definition is displayed in the upper part of the screen or window. There is a reverse facility, enabling you to return to an earlier stage of your dialogue with ADT. See the *VAX DATATRIEVE User's Guide* for additional information on ADT.
- ADT gives you the option of having ADT enter your definitions immediately into the CDD/Repository data dictionary or having ADT write a command file you can invoke at a later time.

- If you respond to any of ADT's questions with a CTRL/Z, DATATRIEVE returns control to command level (indicated by the DTR> prompt), and displays the following message:

```
ADT exited by user request
DTR>
```

Example

Invoke the Application Design Tool:

```
DTR> ADT
Do you want detailed prompts? (YES or NO) :
```

See the *VAX DATATRIEVE User's Guide* for a sample ADT session.

ALLOCATION Clause

Specifies the type of word boundary alignment DATATRIEVE uses when storing records in a data file associated with a record definition. It also controls the way DATATRIEVE retrieves data from files created by user programs or other applications software.

Format

```
ALLOCATION [IS]  { MAJOR_MINOR           }
                 { ALIGNED_MAJOR_MINOR   }
                 { LEFT_RIGHT            }
```

Arguments

MAJOR_MINOR

Causes DATATRIEVE to use MAJOR_MINOR alignment when storing or retrieving data in a data file. MAJOR_MINOR forces word boundary alignments according to data types for elementary fields defined with the SYNCHRONIZED clause and forces group fields to the maximum alignment of the elementary fields they contain. MAJOR_MINOR is the default for DATATRIEVE, VAX COBOL, and COBOL–81.

ALIGNED_MAJOR_MINOR

Causes DATATRIEVE to use ALIGNED_MAJOR_MINOR alignment when storing or retrieving data in a data file. ALIGNED_MAJOR_MINOR forces word boundary alignments according to data types for all elementary fields in the record and group fields to the maximum alignment of the elementary fields they contain.

LEFT_RIGHT

Causes DATATRIEVE to use LEFT_RIGHT alignment when storing or retrieving data in a data file. LEFT_RIGHT forces word boundary alignment for elementary fields defined as COMP, COMP_1, COMP_2, and DATE. LEFT_RIGHT is the default alignment for DATATRIEVE, COBOL-11, and COBOL-74.

Restriction

When defining a record for an existing data file, the alignment type of the record definition must match that of the data file.

Results

- For records with ALIGNED_MAJOR_MINOR allocation and for fields with SYNCHRONIZED clauses in records with MAJOR_MINOR allocation, DATATRIEVE aligns fields on word boundaries. When DATATRIEVE shifts a field to the next word boundary, it considers the bytes it skips as part of the record. These bytes are called filler bytes.
- The number of filler bytes DATATRIEVE inserts when MAJOR_MINOR and ALIGNED_MAJOR_MINOR are in effect depends on the data type of the field.
- Fields begin on quadword boundaries if they are declared DATE, COMP_2 (or DOUBLE), and QUAD (COMP for numbers from PICTURE 9(10) to 9(18)).
- Fields begin on longword boundaries if they are declared LONG (COMP for numbers from PICTURE 9(5) to 9(9)), and COMP_1 (or REAL).
- Fields begin on word boundaries if they are declared WORD (COMP for numbers from PICTURE 9(1) to 9(4)).
- When the allocation is either ALIGNED_MAJOR_MINOR or MAJOR_ MINOR, the group field boundaries are aligned according to the maximum alignment of the elementary fields that comprise the group.
- When the allocation is LEFT_RIGHT, fields begin on word boundaries if they are declared DATE, QUAD, COMP_2 (or DOUBLE), LONG, COMP_1 (or REAL), and WORD.

Usage Notes

- The ALLOCATION clause is an optional record definition clause. If you include it in your record definition, it can affect the way DATATRIEVE handles the internal storage and retrieval of data in some or all of the fields in your data file.
- If you want to use VAX DATATRIEVE on data files created with DATATRIEVE, you must add an ALLOCATION LEFT-RIGHT clause to the DATATRIEVE record definition to ensure that VAX DATATRIEVE can interpret your data correctly.

Example

The following example shows the use of the ALLOCATION clause with the YACHT record:

```
DELETE YACHT;
REDEFINE RECORD YACHT OPTIMIZE
ALLOCATION IS LEFT_RIGHT
01 BOAT.
  03 TYPE.
     06 MANUFACTURER PIC X(10)
        QUERY_NAME IS BUILDER.
     06 MODEL PIC X(10).
  03 SPECIFICATIONS
     QUERY_NAME SPECS.
     06 RIG PIC X(6)
        VALID IF RIG CONT "SLOOP","KETCH","MS","YAWL".
     06 LENGTH_OVER_ALL PIC XXX
        VALID IF LOA BETWEEN 15 AND 50
        QUERY_NAME IS LOA.
     06 DISPLACEMENT PIC 99999
        QUERY_HEADER IS "WEIGHT"
        EDIT_STRING IS ZZ,ZZ9
        QUERY_NAME IS DISP.
     06 BEAM PIC 99 MISSING VALUE IS 0.
     06 PRICE PIC 99999
        MISSING VALUE IS 0
        VALID IF PRICE>DISP*1.3 OR PRICE EQ 0
        EDIT_STRING IS $$$,$$$.
;
```

Assignment Statements

An Assignment statement assigns a value to an elementary field, a group field, or a variable.

Assigning a Value to an Elementary Field

Assigns a value to an elementary field in a MODIFY or STORE statement.

Format

```
field-name = value-expression
```

Arguments

field-name
Is the name of an elementary field.

=
Is an equal sign indicating assignment; it is not equivalent to the relational operator EQ or EQUAL.

value-expression
Is the value to be assigned to the field. This argument can be any DATATRIEVE value expression.

Restrictions

- An elementary field Assignment statement can be used only in a MODIFY . . . USING or STORE . . . USING statement. See the sections on MODIFY and STORE in this chapter for more information.
- To use an Assignment statement inside a STORE . . . USING statement, you must ready the domain for WRITE or EXTEND access. To use this statement inside a MODIFY . . . USING statement, you must ready the domain for WRITE or MODIFY access. See the section on READY in this chapter for more information.
- You cannot assign a value to a COMPUTED BY field.

Results

- DATATRIEVE stores the value of the value expression in the specified field, performing any data type conversions necessary.
- If the value expression is a prompting value expression (*.prompt-name), DATATRIEVE prompts for the value of the field. DATATRIEVE rejects your input and reprompts for the value if any of the following occurs:
  - Truncation error

    You have entered more characters than the field definition allows.
  - Conversion error

    You have entered a character that is inappropriate for the field, such as a letter for a numeric field.
  - Sign error

    You have entered a minus sign for an unsigned numeric field.
  - VALID IF failure

    You have entered an invalid value for the field. That is, the value results in a Boolean expression (specified in a VALID IF clause in the record definition) that is "False." See the chapter on record definitions in the *VAX DATATRIEVE User's Guide* for more information on record definitions and field definition clauses.
- If the value expression is not a prompting value expression and truncation, conversion, or sign errors occur, DATATRIEVE accepts the value with a warning:
  - If you enter too many digits for a numeric field, DATATRIEVE truncates the high-order digits, stores the remaining digits in the field, and issues an error message.
  - If you enter too many characters for an alphanumeric field, DATATRIEVE truncates the rightmost characters, stores the remaining characters in the field, and issues an error message.
  - If a VALID IF failure occurs, DATATRIEVE does not execute the Assignment statement and does not execute the STORE or MODIFY statement containing the Assignment statement.

Usage Notes

- When using this type of Assignment statement, you may want DATATRIEVE to check your values for validity and give you the opportunity to recover from an error caused by invalid input. The prompting value expression, shown in the Assignment statement that follows, provides the only means for you to recover from truncation, conversion, or sign errors, or VALID IF failures. This facility is especially useful when you are creating records with the STORE . . . USING statement.

  ```
  field-name = *.prompt-name
  ```

- If you are unsure of the name of a field or the general type of data it contains, use the SHOW FIELDS command (see the section in this chapter on the SHOW command). SHOW FIELDS displays a brief description of each field in a readied domain.

Examples

In the following example, the YACHTS domain is readied for WRITE access, and a record for the manufacturer CHALLENGER is stored in the domain:

```
DTR> READY YACHTS WRITE
DTR> STORE YACHTS USING MANUFACTURER = "CHALLENGER"
DTR> PRINT YACHTS WITH BUILDER EQ "CHALLENGER"

                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG     ALL   WEIGHT BEAM  PRICE

 CHALLENGER                                  0  00
 CHALLENGER  32         SLOOP   32    12,800  11  $31,835
 CHALLENGER  35         SLOOP   35    14,800  12  $39,215
 CHALLENGER  41         KETCH   41    26,700  13  $51,228

DTR>
```

In the following example, the YACHTS domain is readied for MODIFY access, and the value of the PRICE field is changed to a new value specified by the user:

```
DTR> READY YACHTS MODIFY
DTR> FIND YACHTS
[113 records found]
DTR> SELECT
DTR> PRINT PRICE THEN MODIFY USING PRICE = *."NEW PRICE"

 PRICE

$36,951
Enter NEW PRICE: 39000
DTR> PRINT PRICE
```

```
 PRICE
$39,000
DTR>
```

Assigning a Value to a Group Field

Assigns a value to a group field in a MODIFY or STORE statement.

Format

group-field-name-1 = group-field-name-2

Arguments

group-field-name-1
Is the name of a group field to which you want to assign a value.

=
Is an equal sign indicating assignment; it is not equivalent to the relational operator EQ or EQUAL.

group-field-name-2
Is the name of a group field containing the values you want to assign to group-field-1. This group field must have at least one elementary field with the same name as an elementary field in group-field-1.

Restrictions

- A group field Assignment statement can be used only in a MODIFY . . . USING or STORE . . . USING statement. See the sections in this chapter on the MODIFY and STORE statements for more information.
- To use an Assignment statement in a MODIFY . . . USING statement, you must ready the domain containing group-field-1 for MODIFY or WRITE access. To use it in a STORE . . . USING statement, you must ready the domain for WRITE or EXTEND access. See the section in this chapter on the READY command for more information.
- Both group-field-1 and group-field-2 must have at least one elementary field with the same field name or query name.

Results

DATATRIEVE changes the values of all fields in group-field-1 to the values of identically named fields in group-field-2. All other elementary fields in group-field-1 are set to zero or blank in a STORE statement or remain unchanged in a MODIFY statement.

Examples

The following example shows how to store a record in the YACHTS domain with the same values in the group field SPECIFICATIONS as the selected record:

```
DTR> SET NO PROMPT
DTR> READY YACHTS WRITE
DTR> FIND YACHTS
[113 records found]
DTR> SELECT
DTR> PRINT SPECS

       LENGTH
        OVER
 RIG    ALL   WEIGHT BEAM  PRICE

KETCH   37    20,000  12  $36,951

DTR> STORE YACHTS USING
CON> BEGIN
CON>     BUILDER = *.BUILDER
CON>     SPECS = SPECS
CON> END
Enter BUILDER: HUGHES
DTR> PRINT YACHTS WITH BUILDER = "HUGHES"

                               LENGTH
                                OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 HUGHES                KETCH   37    20,000  12  $36,951

DTR>
```

The following example shows how to store records into a new domain PRICED_ YACHTS. Only the boats that have a price are stored.

Assignment Statements

```
DTR> DEFINE DOMAIN PRICED_YACHTS USING YACHT ON PYACHT.DAT;
DTR> DEFINE FILE FOR PRICED_YACHTS
DTR> READY PRICED_YACHTS WRITE
DTR> SET NO PROMPT
DTR> FOR YACHTS WITH PRICE NE 0
CON>      STORE PRICED_YACHTS USING BOAT = BOAT
DTR> FIND PRICED_YACHTS
[50 records found]
DTR>
```

Assigning a Value to a Variable

Assigns a value to a variable.

Format

variable-name = value-expression

Arguments

variable-name
Is the name of a variable that has been defined with a DECLARE statement.

=
Is an equal sign indicating assignment; it is not equivalent to the relational operator EQ or EQUAL.

value-expression
Is a value expression, the value of which you want to assign to the specified variable.

Restrictions

- You can use a variable Assignment statement anywhere a DATATRIEVE statement is allowed.
- The variable-name must be defined with the DECLARE statement.

Results

- DATATRIEVE assigns the specified value to the variable.
- If the value expression is a prompting value expression (*.prompt-name) DATATRIEVE prompts for the value of the field. DATATRIEVE rejects your input and reprompts for the value if any of the following occurs:
  - Truncation error

    You have entered more characters than the field definition allows.

  - Conversion error

    You have entered a character that is inappropriate for the field, such as a letter for a numeric field.

  - Sign error

    You have entered a minus sign for an unsigned numeric field.

  - VALID IF failure

    You have entered an invalid value for the field. That is, the value results in a Boolean expression (specified in a VALID IF clause in the record definition) that is false. See the chapter on record definitions in the *VAX DATATRIEVE User's Guide* for more information on record definitions and field definition clauses.

- If the argument value expression is not a prompting value expression and truncation, conversion, or sign errors occur, DATATRIEVE accepts the value with a warning:
  - If you enter too many digits for a numeric field, DATATRIEVE truncates the high-order digits, stores the remaining digits in the field, and issues an error message.
  - If you enter too many characters for an alphanumeric field, DATATRIEVE truncates the rightmost characters, stores the remaining characters in the field, and issues an error message.
  - If a VALID IF failure occurs, DATATRIEVE does not execute the Assignment statement.

Usage Note

When using this type of Assignment statement, you may want DATATRIEVE to check your values for validity and give you the opportunity to recover from an error caused by invalid input. The prompting value expression shown in the following Assignment statement provides the only means for you to recover from truncation, conversion, or sign errors or VALID IF failure:

```
variable-name = *.prompt-name
```

Examples

In the following example, the global variable NEW_PRICE is declared and a value is assigned to it:

```
DTR> DECLARE NEW_PRICE PIC 9(5) EDIT_STRING IS $$$,$$$.
DTR> NEW_PRICE = "$25,000"
DTR> PRINT NEW_PRICE

  NEW
 PRICE

$25,000

DTR>
```

In the following example, two global variables, NEW_PRICE and OLD_PRICE, are declared. The value in the PRICE field of the first boat in YACHTS is printed and then assigned to OLD_PRICE. Then the value of OLD_PRICE is assigned to NEW_PRICE:

```
DTR> DECLARE NEW_PRICE PIC 9(5).
DTR> DECLARE OLD_PRICE PIC 9(5).
DTR> FIND YACHTS; SELECT; PRINT PRICE

 PRICE

$36,951
DTR> OLD_PRICE = PRICE
DTR> PRINT OLD_PRICE

 OLD
 PRICE

$36,951

DTR> NEW_PRICE = OLD_PRICE
DTR> PRINT NEW_PRICE

 NEW
 PRICE

$36,951

DTR>
```

AT Statements (Report Writer)

The Report Writer AT statements display header and summary lines at the top and bottom of the report, and at the top and bottom of pages and control groups (groups of sorted records with the same value in one or more fields).

The AT TOP statement summarizes information for the entire group of records in the current collection, not the records of the page, group, or report that you specify in the statement.

The AT BOTTOM statement calculates the summary information based on the records of the page, group, or report that you specify in the statement.

With the two forms of the AT statement, you can specify the value, position, and format of the print objects in the header and summary lines:

The AT statements accept the same types of print objects as a PRINT statement.

Format

AT BOTTOM OF { REPORT | PAGE | field-name } **PRINT** summary-element [, ...]

AT TOP OF { REPORT | PAGE | field-name } **PRINT** { header-element | summary-element } **[, ...]**

Arguments

field-name
Is a field from the record definition for the report's record stream or a variable.

header-element
summary-element
Specifies the value, position, and format of the fields. These elements are summarized in Table 4–1.

Table 4–1 AT Statements Summary Elements

Summary Element	Function	Usage Notes
AVERAGE value expression	Displays the average value of the value expressions.	Calculated for the detail lines of the report, page, or group specified. Does not indicate the average for the detail lines of the report, page, or control group.
ATT name	Sets print attributes for the following print list elements.	Same usage as in the PRINT statement (Report Writer).
COL n	Specifies where the output of the next header or summary element begins.	Same usage as in the PRINT statement (Report Writer).
COLUMN_ HEADER	Displays the column headers.	Overrides the suppression of column headers for AT TOP OF REPORT or PAGE.
COUNT	Displays the number of detail lines.	Calculated for the detail lines of the report, page, or group specified.Does not indicate the number of detail lines of control group specified.
field-name [modifier]	In AT BOTTOM OF field-name and AT TOP OF field-name, prints the common field value for each control group.	Same usage as in the PRINT statement (Report Writer).
MAX value expression	Displays the maximum value of the value expression.	Calculated for the detail lines of the report, page, or group specified.Does not indicate the maximum value for the detail lines of the report, page, or control group specified.

(continued on next page)

Table 4–1 (Cont.) AT Statements Summary Elements

Summary Element	Function	Usage Notes
MIN value expression	Displays the minimum value of the value expression.	Calculated for the detail lines of the report, page, or group specified.Does not indicate the minimum value for the detail lines of the report, page, or control group specified.
NEW_PAGE	Causes the Report Writer to start a new page before the output of the next summary element.	Same usage as in the PRINT statement (Report Writer). Cannot be used in neither AT BOTTOM OF PAGE nor AT TOP OF PAGE.
NEW_SECTION	Starts a new page and a new sequence of page numbers, beginning with Page 1.	Cannot be used in neither AT BOTTOM OF PAGE, nor AT TOP OF PAGE.
REPORT_ HEADER	Displays the report header, including the report name, date, and page number.	Overrides the suppression of a report header in AT TOP OF REPORT or PAGE.
RUNNING COUNT	In AT BOTTOM OF field-name and AT TOP OF field-name, prints the number of control breaks for that field to that point in the report.	In AT BOTTOM OF PAGE and AT TOP OF PAGE, prints the number of pages to that point in the report.
SKIP [n]	Prints the next header or summary element n lines from the current line.	Same usage as in the PRINT statement (Report Writer).
SPACE [n]	Leaves spaces between the output of the preceding and following elements.	Same usage as in the PRINT statement (Report Writer).

(continued on next page)

Table 4–1 (Cont.) AT Statements Summary Elements

Summary Element	Function	Usage Notes
STD_DEV value expression	Displays the standard deviation of values for the value expression.	Calculated for the detail lines of the report, page, or group specified.Does not indicate the standard deviation for the detail lines of the report, page, or control group specified.
TAB [n]	Inserts the space of one or more tabs before the output of the next element.	Same usage as in the PRINT statement (Report Writer).
TOTAL value expression	Displays the total of values for the expression.	Calculated for the detail lines of the report, page, or group specified.Does not indicate the total for the detail lines of the report, page, or control group.
value expression	Displays the value in a header or summary line.	Same usage as in the PRINT statement (Report Writer).

Restrictions

- AT TOP OF PAGE and AT BOTTOM OF PAGE are ignored in DTIF reports.
- Elements NEW_PAGE, NEW_SECTION, and REPORT_HEADER are ignored in AT statements in DTIF format.
- You cannot specify the summary elements NEW_PAGE and NEW_SECTION in neither an AT BOTTOM OF PAGE PRINT statement, nor an AT TOP OF PAGE PRINT statement.
- For relational sources, DATATRIEVE uses the relation's name as the top-level group field. This top-level field cannot be used in AT TOP or AT BOTTOM statements. For example, with the EMPLOYEES relational domain, you cannot use this statement:

  ```
  AT BOTTOM OF EMPLOYEES PRINT EMPLOYEE_ID
  ```

- You cannot use OCCURS fields from views in AT TOP OF or AT BOTTOM OF statements. Using view domain fields defined with OCCURS clauses in Report Writer AT statements causes a NODATA error message.

- You cannot specify a statistical expression in an AT TOP statement unless you have already formed a current collection. VAX DATATRIEVE evaluates the statistical expression based on the records in the current collection.
- Under certain conditions, specifying an AT TOP OF clause may have an effect on the subsequent column positioning of the field if it is also specified in the PRINT detail line. This occurs under the following combination of circumstances:
  - You specify the field in an AT TOP OF statement.
  - You specify a column number for the start of the field value in the print detail line.
  - You do not specify a header for the column in the print detail line.

  To print the field value in the column you specify, you must explicitly suppress the header for the individual field using a hyphen in parentheses (-).

Results

- The AT BOTTOM OF field-name statement and the AT TOP OF field-name statement establish a pattern of control breaks for the entire report, dividing the report into groups of records with a common value for the field. DATATRIEVE displays the summary line at the bottom of the control group for which the field name is the sort key. Statistical expressions are computed for the records for the detail lines in that control group.

  When you specify AT BOTTOM OF field-name PRINT field-name, the Report Writer prints the value in the specified field of the last detail line in the control group.
- If you use AT BOTTOM OF field-name, DATATRIEVE checks the sort order of the collection or record stream you want to report. You can establish a sort order with the FIND, SORT, or REPORT statements.
- If AT BOTTOM OF REPORT is included in a report specification, DATATRIEVE displays the header line or summary line below the detail lines on the last page of the report. Statistical expressions are computed for the records for all of the detail lines in the report.
- If an AT TOP OF REPORT is included in a report specification, DATATRIEVE displays the header line or summary line above the detail lines on the first page of the report and suppresses the report header on the first page of the report. The report header is displayed on the following pages of the report, and the page numbers begin with Page 1 on the second physical page of the report. To specify a title page for your report, end the print list with NEW_PAGE or NEW_SECTION.

- If an AT BOTTOM OF PAGE statement is included in a report specification, DATATRIEVE displays the header line or summary line at the bottom of each page of the report. Statistical expressions are computed for the records for the detail lines on that page.
- If an AT TOP OF PAGE is included in a report specification, DATATRIEVE displays the header line or summary line at the top of each page of the report and replaces the report header on every page.

Usage Notes

- The header and summary elements may contain modifiers to specify edit strings or to suppress headers.
- If you use AT BOTTOM OF field-name without having sorted the records, the Report Writer divides the detail lines into control groups anyway. A new control group is formed every time the value in the specified field changes. If no values in that field are repeated in consecutive detail lines, the Report Writer treats each line as a separate control group and prints the header and summary lines above and below each line as indicated in the record specification.

  You may want to follow this approach in the following situation. For example, the records may already be sorted according to an indexed key field. If there is only one level of control break, the Report Writer divides the control groups at the appropriate places. However, if you want multiple levels of control groups, you must sort the records in advance with a FIND, SORT, or REPORT statement.
- In table formats AT TOP OF PAGE and AT BOTTOM OF PAGE are ignored.

Examples

The following example shows the use of both the AT BOTTOM and AT TOP statements:

```
DTR> SHOW SALARY_REPORT
```

```
PROCEDURE SALARY_REPORT
REPORT PERSONNEL WITH DEPT = "D98","E46","T32" SORTED BY DEPT
SET REPORT_NAME = "SALARY REPORT"
SET COLUMNS_PAGE = 60
AT TOP OF DEPT PRINT DEPT
PRINT ID, FIRST_NAME|||LAST_NAME ("NAME"), SALARY
AT BOTTOM OF DEPT PRINT SKIP,
      COL 36, DEPT|||"TOTAL:",
      TOTAL SALARY USING $$$$,$$$, SKIP, COL 13,
      "**********************************",
      SKIP, COL 32, "OVERALL TOTAL:", COL 50,
      RUNNING TOTAL (TOTAL SALARY) USING $$$$,$$$, SKIP
END_REPORT
END_PROCEDURE
```

Figure 4–1 shows the report produced by this specification.

Figure 4–1 Control Group Report Based on One Sort Key

```
DTR> :SALARY_REPORT

                    SALARY REPORT           14-May-1984
                                            Page 1

  DEPT        ID               NAME              SALARY

  D98
            02943      CASS TERRY                $29,908
            39485      DEE TERRICK               $55,829
            49843      BART HAMMER               $26,392
            84375      MARY NALEVO               $56,847

                                D98 TOTAL:    $168,976
            **********************************
                           OVERALL TOTAL:    $168,976

  E46
            38465      JOE FREIBURG              $23,908
            48475      GAIL CASSIDY              $55,407

                                E46 TOTAL:     $79,315
            **********************************
                           OVERALL TOTAL:    $248,291

  T32
            34456      HANK MORRISON             $30,000
            38462      BILL SWAY                 $54,000
            48573      SY KELLER                 $31,546
            83764      JIM MEADER                $41,029
```

(continued on next page)

Figure 4–1 (Cont.) Control Group Report Based on One Sort Key

```
                          T32 TOTAL:    $156,575
**********************************
                       OVERALL TOTAL:    $404,866
```

See the *VAX DATATRIEVE User's Guide* for more examples of the AT TOP statement.

BEGIN END Statement

Groups DATATRIEVE statements into a single compound statement called a BEGIN-END block.

Format

```
BEGIN
    statement-1

    [statement-2]
        .
        .
        .
END
```

Arguments

BEGIN

Marks the beginning of a BEGIN-END block.

statement

Is a DATATRIEVE statement. Within the BEGIN-END block, end each statement with a semicolon, a RETURN, or both.

END

Marks the end of the BEGIN-END block.

Restrictions

- You must observe all restrictions on the statements included in the BEGIN-END block. This manual lists these restrictions in the descriptions of the various statements.
- Do not use FIND, SELECT, or DROP statements in a BEGIN-END block.
- You cannot use DATATRIEVE commands in a BEGIN-END block. Consequently, a BEGIN-END block cannot include a procedure that contains DATATRIEVE commands.
- You cannot invoke a command file in a BEGIN-END block. That is, you cannot have a line in a BEGIN-END block that contains an at sign (@) followed by a file specification. DATATRIEVE treats a command file invocation as a command and does not execute command files specified within a BEGIN-END block.

Results

- Within a BEGIN-END block, DATATRIEVE executes the statements in the order you entered them.
- When the BEGIN-END block includes a DECLARE statement, the variable it defines is a local variable. You cannot refer to a local variable from outside the BEGIN-END block. DATATRIEVE automatically releases all local variables when it completes the BEGIN-END block in which they are defined.

  If a BEGIN-END block that defines a local variable is in a FOR loop or REPEAT statement, the local variable is initialized each time DATATRIEVE executes the BEGIN-END block. The variable is initialized to zero, space, the DEFAULT value, or the MISSING value depending on the way you defined the variable in the DECLARE statement. See the sections on the DECLARE statement and the DEFAULT VALUE and MISSING VALUE clauses for more information.
- When you create a single BEGIN-END block interactively, DATATRIEVE prompts you, after you press the RETURN key, for the elements needed to complete an individual statement or to complete the block. The CON> prompt indicates that DATATRIEVE is ready for you to continue entering statements or elements of statements into the BEGIN-END block.

  After you enter END, DATATRIEVE executes all the statements in the BEGIN-END block. When DATATRIEVE completes the last statement in the BEGIN-END block, you see the DTR> prompt, indicating you have returned to DATATRIEVE command level. If you are entering nested BEGIN-END blocks, DATATRIEVE continues to prompt you with the CON> until you have entered the END that completes the outermost BEGIN-END block.

  DATATRIEVE does not execute any of the statements in the nested blocks until you enter the END that completes the outermost BEGIN-END block.
- If you enter CTRL/C while DATATRIEVE is executing the statements in a BEGIN-END block, DATATRIEVE does not execute any of the remaining statements that follow in the block.

  DATATRIEVE treats the whole BEGIN-END block as one statement, regardless of the number of statements or nested BEGIN-END blocks it contains. Because CTRL/C cancels the execution of the current statement, it cancels the execution of the remainder of the entire, outermost BEGIN-END block, regardless of its complexity or the number of levels of other BEGIN-END blocks nested in it.

If a statement in a BEGIN-END block prompts you for a value and you enter a CTRL/C, DATATRIEVE reprompts you for the value; it does not end the execution of the BEGIN-END block. To end the execution of a BEGIN-END block when DATATRIEVE is prompting you for a value, enter CTRL/Z. DATATRIEVE then ends the execution of the outermost BEGIN-END block and returns you to DATATRIEVE command level.

Usage Notes

- You can use a BEGIN-END block anywhere you can use a DATATRIEVE statement.
- You can nest BEGIN-END blocks. There is virtually no limit on the levels of nested BEGIN-END blocks you can form.
- To repeat an entire sequence of DATATRIEVE statements, put the statements in a BEGIN-END block, and put the BEGIN-END block in a REPEAT statement.
- If you invoke a procedure in a REPEAT statement (using the form REPEAT n :procedure-name), DATATRIEVE repeats the first statement of the procedure n times and then executes the other statements in the procedure once each.

  To repeat all the statements of a procedure you invoke in a REPEAT statement, put the procedure invocation in a BEGIN-END block, and put the BEGIN-END block in the REPEAT statement.
- Use a BEGIN-END block to include more than one DATATRIEVE statement in a FOR loop.
- Use a BEGIN-END block to include more than one DATATRIEVE statement in the THEN and ELSE clauses in an IF-THEN-ELSE statement.
- When storing or modifying records, use BEGIN-END blocks to include more than one statement in the USING and VERIFY USING clauses of the STORE and MODIFY statements.
- If you write a procedure that uses consecutive PRINT statements to format your output, the line spacing changes when you put the procedure in a BEGIN-END block. The one blank line between the result of each PRINT statement disappears. To preserve that spacing when you include the procedure in a BEGIN-END block, edit the procedure and insert a SKIP print-list element at the beginning of the second and each succeeding PRINT statement.

Examples

The following example shows how to store five records in the domain PHONES, each having LOCATION MB1-H2 and DEPARTMENT CE. The SET NO PROMPT command suppresses the "[Looking for . . .]" prompts preceding each CON> prompt. This example also shows how DATATRIEVE responds to CTRL/C and CTRL/Z when it is prompting you for input.

```
DTR> READY PHONES WRITE
DTR> SET NO PROMPT
DTR> REPEAT 5 STORE PHONES USING
DTR> BEGIN
CON>      NAME=       *.NAME
CON>      NUMBER=     *.NUMBER
CON>      LOCATION=   "MB1-H2"
CON>      DEPARTMENT= "CE"
CON> END
Enter NAME: FRED
Enter NUMBER: 555-1234
Enter NAME: GERRY
Enter NUMBER: CTRL/C
^C
Enter NUMBER: CTRL/Z
Execution terminated by operator
DTR>
```

The following example shows how to use a BEGIN-END block to put three statements in the USING clause of a MODIFY statement, print a YACHTS record, modify the price, and print the result of the modification:

```
DTR> READY YACHTS WRITE
DTR> SET NO PROMPT
DTR> FOR YACHTS WITH PRICE = 0
CON> MODIFY USING
CON> BEGIN
CON>          PRINT
CON>          PRICE = *."NEW PRICE"
CON>          PRINT
CON> END

                                LENGTH
                                 OVER
MANUFACTURER  MODEL     RIG     ALL    WEIGHT BEAM  PRICE

 BLOCK I.     40        SLOOP    39     18,500  12
Enter NEW PRICE: 30000

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

```
 BLOCK I.    40         SLOOP   39    18,500  12  $30,000
 BUCCANEER   270        SLOOP   27     5,000  08
Enter NEW PRICE: CTRL/Z
Execution terminated by operator

DTR>
```

The following example shows how a BEGIN-END block is treated as a single statement:

```
DTR> DEFINE PROCEDURE LOOP_EXAMPLE
DFN> PRINT "SHOW HOW A BEGIN-END WORKS WITH REPEAT"
DFN> PRINT "AND MORE THAN ONE STATEMENT"
DFN> END_PROCEDURE
DTR>
DTR> REPEAT 2 :LOOP_EXAMPLE
SHOW HOW A BEGIN-END WORKS WITH REPEAT
SHOW HOW A BEGIN-END WORKS WITH REPEAT
AND MORE THAN ONE STATEMENT

DTR> REPEAT 2 BEGIN :LOOP_EXAMPLE; END
SHOW HOW A BEGIN-END WORKS WITH REPEAT
AND MORE THAN ONE STATEMENT
SHOW HOW A BEGIN-END WORKS WITH REPEAT
AND MORE THAN ONE STATEMENT

DTR>
```

CDO Command

Passes a command line to the Common Dictionary Operator (CDO) Utility.

Format

CDO cdo-command-line

Argument

cdo-command-line
Is a command line to be passed to the CDO.

Result

DATATRIEVE passes the command line to the CDO. DATATRIEVE does not validate your command line or modify it in any way.

Usage Notes

- The DATATRIEVE CDO command is the same as spawning a CDO command but is more efficient because no subprocess is created.
- The DATATRIEVE SET DICTIONARY command controls the default dictionary directory setting used by the CDO command. When you enter a SET DICTIONARY command, the default dictionary used by the CDO command is also changed. However, a CDO SET DEFAULT command does not change your current DATATRIEVE default dictionary directory setting.

Example

In the following example, a change in field definition FLD_A causes a message to be displayed when you READY MY_DB. The message is displayed and cleared and the CLEAR NOTICES command is verified using the CDO command.

```
DTR> CDO SHOW NOTICES MY_DB
USR$DISK:[RICK.TEST]GLOBAL.MY_DB;1 is possibly invalid, triggered
by CDD$DATA_ELEMENT USR$DISK:[RICK.TEST]GLOBAL.FLD_A;1

DTR> CDO CLEAR NOTICES MY_DB
DTR> CDO SHOW NOTICES MY_DB
%CDO-I-NOMESSAGES, USR$DISK:[RICK.TEST]GLOBAL.MY_DB;1 has no notices
DTR>
```

CHOICE Statement

Causes DATATRIEVE to execute one of a series of statements or compound statements, depending on the evaluation of a series of conditional (Boolean) expressions. The CHOICE statement is a convenient substitute for nested IF-THEN-ELSE statements.

Format

```
CHOICE   [OF]
        boolean-expression-1 [THEN] statement-1
        [boolean-expression-2 [THEN] statement-2]
                    .          .       .
                    .          .       .
                    .          .       .
        [ELSE statement-n]
END_CHOICE
```

Arguments

CHOICE

Marks the beginning of a CHOICE statement.

OF

Is an optional language element you can use to clarify syntax.

boolean-expression

Is a Boolean expression. (See Chapter 1.)

THEN

Is an optional language element you can use to clarify syntax.

statement

Is a simple or compound statement you want DATATRIEVE to execute if the corresponding Boolean expression evaluates to true.

ELSE statement-n

Specifies the statement you want DATATRIEVE to execute if all the preceding Boolean expressions evaluate to false.

END_CHOICE

Marks the end of the CHOICE statement.

Restrictions

- You must observe all restrictions on the statements used in the CHOICE statement. This manual lists these restrictions in the descriptions of the various statements.
- DATATRIEVE does not evaluate view domain fields defined with an OCCURS clause when you use such fields in CHOICE statements.

Results

- DATATRIEVE evaluates each Boolean expression in order. When a Boolean expression evaluates to true, DATATRIEVE executes the corresponding statement in the THEN clause. Then DATATRIEVE goes on to execute the next command or statement after END_CHOICE.
- If you specify an ELSE clause and all the preceding Boolean expressions evaluate to false, DATATRIEVE executes statement-n in the ELSE clause.
- If you do not specify an ELSE clause and all the preceding Boolean expressions evaluate to false, DATATRIEVE does not execute any of the statements in the THEN clauses and is ready to execute the next command or statement it encounters.

Usage Note

If any of the statements in the THEN clauses are compound statements, you must enclose the set of statements within a BEGIN-END block.

Examples

The following example shows how to define a procedure MODIFY_YACHTS, prompt the user to identify a record by specifying the BUILDER and MODEL of a yacht, and print the record, prompting the user to modify a field from YACHTS. This process is continued until the user replies to a prompt that no further changes need to be made.

```
DTR> SHOW MODIFY_YACHTS
PROCEDURE MODIFY_YACHTS
READY YACHTS WRITE
FOR YACHTS WITH BUILDER = *."the builder" AND
      MODEL = *."the model"
BEGIN
  PRINT
  PRINT SKIP
  PRINT "The fields you can modify are: RIG,LOA,DISP,BEAM,PRICE"
  MODIFY USING
  WHILE *."Y to modify a field, N to exit" CONT "Y"
    BEGIN
      DECLARE FLD PIC X(5).
      FLD = *."field to modify"
      CHOICE
           FLD = "RIG" THEN RIG = *.RIG
           FLD = "LOA" THEN LOA = *.LOA
           FLD = "DISP" THEN DISP = *.DISP
           FLD = "BEAM" THEN BEAM = *.BEAM
           FLD = "PRICE" THEN PRICE = *.PRICE
           ELSE PRINT "That's not a field in YACHTS."
      END_CHOICE
      PRINT
      PRINT SKIP
    END
  PRINT SKIP, "No more changes."
END
END_PROCEDURE

DTR> :MODIFY_YACHTS
Enter the builder: ALBIN
Enter the model: 79

                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 ALBIN       79         SLOOP   26     4,200  10  $17,900

The fields you can modify are: RIG, LOA, DISP, BEAM, PRICE
Enter Y to modify a field, N to exit: Y
Enter field to modify: RIG
Enter RIG: KETCH

                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 ALBIN       79         KETCH   26     4,200  10  $17,900

Enter Y to modify a field, N to exit: Y
Enter field to modify: RUG
That's not a field in YACHTS.
 ALBIN       79         KETCH   26     4,200  10  $17,900
```

```
Enter Y to modify a field, N to exit: N

No more changes.

DTR>
```

The following example shows how to print the TYPE and PRICE of the yachts by ALBERG and ALBIN, indicating whether the price is inexpensive, moderate, or expensive. Column headers are suppressed:

```
DTR> READY YACHTS
DTR> FOR YACHTS WITH BUILDER = "ALBERG" OR BUILDER = "ALBIN"
CON> CHOICE
CON> PRICE LT 20000 THEN PRINT TYPE(-),PRICE(-),"INEXPENSIVE"
CON> PRICE LT 30000 THEN PRINT TYPE(-),PRICE(-),"MODERATE"
CON> ELSE PRINT TYPE(-), PRICE(-), "EXPENSIVE"
CON> END_CHOICE

ALBERG     37 MK II   $36,951 EXPENSIVE
ALBIN      79         $17,900 INEXPENSIVE
ALBIN      BALLAD     $27,500 MODERATE
ALBIN      VEGA       $18,600 INEXPENSIVE

DTR>
```

CLOSE Command

Closes a Record Management System (RMS) trace file you created with an OPEN command as a log for your interactive dialogue with DATATRIEVE.

Format

```
CLOSE
```

Arguments

None.

Restriction

The CLOSE command can be entered at DTR> prompt or, if you are using the DATATRIEVE DECwindows interface, choose the Close Log item of the File menu.

Usage Note

The CLOSE command allows you to close the log file without ending your DATATRIEVE session.

Examples

The following example shows a CLOSE command:

```
DTR> CLOSE
```

COMMIT Statement

Makes permanent all the changes you made to relational and VAX DBMS databases since the most recent COMMIT or ROLLBACK statement or, if you have not performed a COMMIT or ROLLBACK, since the first database READY command.

The COMMIT statement does not affect collections; collections are maintained. For VAX DBMS databases, the COMMIT statement performs a COMMIT RETAINING. For relational databases, the COMMIT statement starts a new transaction that gives you a new look at the database.

When you have both relational and VAX DBMS databases readied, the COMMIT statement commits all VAX DBMS and relational databases, regardless of whether you made any changes to their data.

RMS domains are not affected by the COMMIT statement.

Format

```
COMMIT
```

Arguments

None.

Restriction

Do not include COMMIT statements in compound statements containing RSEs that operate on data from relational sources. You must keep the COMMIT statement independent of the compound statement containing the RSE in order for it to work.

Usage Note

The COMMIT statement for both relational and VAX DBMS databases starts a new transaction. COMMIT maintains collections. After a COMMIT, a relational database collection will include changes other users have made to the records in your collection since you formed the collection.

Examples

The following VAX DBMS example connects an employee named Hill to a part LA36 in the RESPONSIBLE_FOR set. The COMMIT statement makes this change permanent.

```
DTR> FIND E IN EMPLOYEES WITH EMP_LAST_NAME = "HILL"
DTR> SELECT 1
DTR> FOR P IN PART WITH PART_DESC = "LA36"
CON>   CONNECT P TO E.RESPONSIBLE_FOR
DTR> COMMIT

DTR>
```

The following relational database example stores a record in the relation DEPARTMENTS. The COMMIT statement makes this change permanent:

```
DTR> READY DATABASE PERSONNEL USING DEPARTMENTS WRITE
DTR> STORE DEPARTMENTS
Enter DEPARTMENT_CODE: SENG
Enter DEPARTMENT_NAME: SOFTWARE ENGINEERING
Enter MANAGER_ID: 87215
DTR> COMMIT

DTR>
```

COMPUTED BY Clause

Describes a COMPUTED BY field.

Format

COMPUTED BY value-expression

Argument

value-expression
Is a DATATRIEVE value expression.

Restrictions

- This clause is valid for elementary fields only.
- You cannot use an OCCURS clause or a VALID IF clause in a COMPUTED BY field.
- Because it does not exist in the record, you cannot assign a value to a COMPUTED BY field. STORE and MODIFY statements cannot refer to a COMPUTED BY field.
- You cannot redefine a COMPUTED BY field with the REDEFINES clause.
- Do not use a MISSING VALUE, PICTURE, or USAGE clause when defining a COMPUTED BY field. Because a COMPUTED BY field is a virtual expression, you cannot use a clause that specifies how the value is stored.

Result

When you refer to the COMPUTED BY field in a statement, DATATRIEVE resolves the field name to the nearest single record context and evaluates the value expression using the field values in that record.

Usage Notes

- COMPUTED BY fields are useful if you display arithmetic computations frequently. COMPUTED BY fields allow you to specify the computation once in the record definition, and then to print its value simply by referring to its field name. Generally, the computation includes the name of one or more fields in the record definition.
- A COMPUTED BY field does not occupy space in a record. It exists solely in the record definition. The value of a COMPUTED BY field is calculated only when you use it in a statement.

- You can use the CHOICE or IF-THEN-ELSE value expression in defining a COMPUTED BY field. This is useful if the value of the field depends on other field values.
- You can use a COMPUTED BY field as a sort key in SORT and SUM statements and in the SORTED BY clause of record selection expressions. You can also use a COMPUTED BY field to form control groups in the AT TOP and AT BOTTOM statements of the DATATRIEVE Report Writer.
- If you include a COMPUTED BY clause in a field definition, you do not have to include any other field definition clause. However, you may want to use an EDIT_STRING clause to format the computed value when it is printed.
- Edit strings are not inherited from the fields that make up a COMPUTED BY value. If you want a specific representation for the COMPUTED BY field, explicitly specify an edit string for that field.
- The value expression of a COMPUTED BY field in a record definition cannot refer to a list field in the same domain. To compute the value of a list field in the same domain, ready the domain containing the list field and then declare a variable outside the record definition to include the COMPUTED BY value expression.
- When a COMPUTED BY field refers to a list field in another domain, you must ready the domain with the list field before you ready the domain with the COMPUTED BY field. You must finish the domains in the reverse order: the domain with the COMPUTED BY field first and the domain with the list field last.

Examples

In the following example, the price per pound of a yacht is computed as the price divided by the displacement. In this case, both the PRICE and DISP fields are defined in the record definition.

```
06 PRICE_PER_POUND
   EDIT_STRING $$$,$$9.99
   COMPUTED BY PRICE/DISP.
```

When the PRICE_PER_POUND field is used in a command or statement, DATATRIEVE divides the value of the record's PRICE field by the value of its DISP field. The result of the computation is the value of the PRICE_PER_ POUND field.

In the following example, the discount price of a yacht is computed. The amount of the discount varies with the price of a yacht. The field PRICE is defined in the record definition for YACHTS:

```
06 DISCOUNT_PRICE   COMPUTED BY
          CHOICE
             PRICE LT 20000 THEN (PRICE * .9)
             PRICE LT 30000 THEN (PRICE * .8)
             PRICE LT 40000 THEN (PRICE * .7)
             ELSE (PRICE * .6)
          END_CHOICE
          EDIT_STRING IS $$$,$$$.
```

When DISCOUNT_PRICE is used in a command or statement, DATATRIEVE evaluates each Boolean expression in order until one evaluates to true. Then it performs the corresponding computation on PRICE.

In the following example, the value of the SALESFORCE field is derived from a dictionary or domain table named SALES_TABLE:

```
06 SALESFORCE
   EDIT_STRING IS X(20)
   COMPUTED BY MANUFACTURER VIA SALES_TABLE.
```

In this example, DATATRIEVE uses the value of the MANUFACTURER field in the current record to search the dictionary or domain table SALES_TABLE for a matching code. If one is found, DATATRIEVE uses its translation as the value of the SALESFORCE field.

CONNECT Statement

Makes explicit connections between a record and the VAX DBMS sets you specify in the TO list. Before establishing the connections, DATATRIEVE sets up a currency indicator for each set specified in the TO list.

Format

```
CONNECT   context-name-1

          [TO] {[context-name-2.] set-name-1} [,...]
```

Arguments

context-name-1

Is the name of a valid context variable or the name of a collection with a selected record. It must identify a record occurrence of a domain with a record type that participates in the specified set type. That record must not be a member of the sets specified by the TO list, but its record type must be a valid member type in the specified set types.

context-name-2

Is the name of a valid context variable or the name of a collection with a selected record. It must identify a record that participates in the specified set. If the SYSTEM owns the set, you do not need to establish a context for the set. If the set is not owned by the SYSTEM and the context name is not present, DATATRIEVE uses the most recent single record context of a domain with a record type that participates in the specified set type.

set-name

Is the name of a set type.

Example

The following example connects an employee named Hill to a part LA36 in the RESPONSIBLE_FOR set. P and E in the following example are context names. E refers to a collection with a selected record and P to a record stream. The records in these contexts are all owner records or member records in the set.

```
DTR> FIND E IN EMPLOYEES WITH EMP_LAST_NAME = "HILL"
DTR> SELECT 1
DTR> FOR P IN PART WITH PART_DESC = "LA36"
CON>   CONNECT P TO E.RESPONSIBLE_FOR
DTR>
```

CROSS Clause

You use the CROSS clause to perform the following tasks:

- Compare records from one domain or collection
- Combine records from two or more domains or collections
- Flatten hierarchical domains to ease access to the list items

Format

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

Arguments

context-variable
Is the name of a valid context variable. A context variable is a temporary name that identifies a record stream to DATATRIEVE.

rse-source
Is the record selection expression (RSE) that identifies the records you want to work with.

field-name
Is a field whose values form the basis for the cross operation.

Restriction

You cannot reference a group field name in an OVER clause if the group field includes a COMPUTED BY field. If you cross over a group field containing a COMPUTED BY field, DATATRIEVE includes too many records in the record stream. Instead, you should explicitly list each individual field of the group.

Result

When you include the CROSS clause in an RSE, DATATRIEVE uses two or more sources to form combinations of records you could otherwise form only with nested FOR loops or view domains. You can join these records in combinations based on the relationship between the values of fields in the sources. This combining of records is called a *relational join*.

Usage Note

You can use key optimization to improve the way DATATRIEVE performs the join. See the *VAX DATATRIEVE User's Guide* for guidelines on how to improve DATATRIEVE performance.

Example

Crossing a Domain with Itself

When you want to compare records within the same domain, use the CROSS clause to cross the domain with itself. For example, suppose you want to display information about manufacturers who build boats with more than one type of rig. The following query uses one RSE that includes five out of the six optional elements of an RSE. DATATRIEVE processes the query far faster than if nested FOR loops were used:

```
DTR> DEFINE PROCEDURE CROS
DFN> READY YACHTS
DFN> PRINT BUILDER, RIG, A.RIG OF
DFN>   A IN YACHTS CROSS YACHTS OVER BUILDER WITH
DFN>   RIG GT A.RIG REDUCED TO BUILDER, RIG, A.RIG
DFN> END_PROCEDURE

DTR> :CROS

MANUFACTURER  RIG    RIG

 AMERICAN     SLOOP  MS
 CHALLENGER   SLOOP  KETCH
 GRAMPIAN     SLOOP  KETCH
 IRWIN        SLOOP  KETCH
 ISLANDER     SLOOP  KETCH
 NORTHERN     SLOOP  KETCH
 PEARSON      SLOOP  KETCH

DTR>
```

Crossing Two Domains

When you want to join records from two domains, use the CROSS clause. If the records share a common field, you may want to add OVER followed by the field name.

For example, suppose you want to join the names of boat owners from the OWNERS domain with information on individual boats you have in the YACHTS domain. You want each OWNERS record paired only with the YACHTS record having the same MANUFACTURER and MODEL. The group field TYPE, which includes both MANUFACTURER and MODEL, is the primary key for the YACHT.DAT file. It is defined as NO DUP and, as a result, no two boats can

have the same value for TYPE. The RSE that forms this temporary combination of records is on the second input line of the following PRINT statement:

```
DTR> PRINT NAME, YACHTS.TYPE, PRICE OF
CON>     YACHTS CROSS OWNERS OVER TYPE

   NAME    MANUFACTURER   MODEL     PRICE

STEVE       ALBIN       VEGA       $18,600
HUGH        ALBIN       VEGA       $18,600
JIM         C&C         CORVETTE
ANN         C&C         CORVETTE
JIM         ISLANDER    BAHAMA      $6,500
ANN         ISLANDER    BAHAMA      $6,500
STEVE       ISLANDER    BAHAMA      $6,500
HARVEY      ISLANDER    BAHAMA      $6,500
TOM         PEARSON     10M
DICK        PEARSON     26
JOHN        RHODES      SWIFTSURE

DTR>
```

Crossing More Than Two Domains

You can join records from more than two domains, collections, or lists. In fact, you can join two domains that share a common field and then join the expanded "records" with a third domain by means of a different field.

For example, suppose that you have three domains A, B, and C. A and B share a common field (FOO) that is a key field for B. You can join records from A and B by specifying the following RSE:

```
A CROSS B OVER FOO
```

But you may want to join this record stream with records from C. Assume that C has a key field (BAR) that is also found in either A or B. You specify the following RSE to join records from the three sources:

```
A CROSS B OVER FOO CROSS C OVER BAR
```

The CROSS clause also simplifies your work with hierarchical records. One record in FAMILIES can have data on as many as 10 children.

The following example forms a collection of the records with the largest number of children:

```
DTR> FIND FAMILIES WITH NUMBER_KIDS = MAX NUMBER_KIDS OF FAMILIES
DTR> PRINT CURRENT

                       NUMBER    KID
  FATHER     MOTHER     KIDS     NAME    AGE
```

```
BASIL      MERIDETH      6   BEAU       28
                             BROOKS     26
                             ROBIN      24
                             JAY        22
                             WREN       17
                             JILL       20
```

The record selection expression CURRENT CROSS KIDS flattens the hierarchy, simplifying the retrieval of list items. For example:

```
DTR> PRINT FATHER, MOTHER, NUMBER_KIDS,
DTR>     EACH_KID OF CURRENT CROSS KIDS

                      NUMBER    KID
  FATHER     MOTHER    KIDS     NAME    AGE

BASIL      MERIDETH      6   BEAU       28
BASIL      MERIDETH      6   BROOKS     26
BASIL      MERIDETH      6   ROBIN      24
BASIL      MERIDETH      6   JAY        22
BASIL      MERIDETH      6   WREN       17
BASIL      MERIDETH      6   JILL       20

DTR>
```

DATATRIEVE Command

The DCL command DATATRIEVE starts a DATATRIEVE session. This DCL command lets you specify several optional qualifiers that let you customize your DATATRIEVE session.

Format

DATATRIEVE [/INTERFACE= { DECWINDOWS / CHARACTER_CELL } /[NO]DEBUG /VARIANT=image-suffix] ["DATATRIEVE"]

Arguments

/INTERFACE = { DECwindows / CHARACTER_CELL }

Specifies the display interface to be used by DATATRIEVE. If your default environment is the command line interface, then CHARACTER_CELL is the default. If your system is properly defined to invoke the DECwindows interface, then DECwindows is the default. This qualifier overrides the setting specified by the DTR$NOWINDOWS logical. (See the *VAX DATATRIEVE Guide to Interfaces* for more information on the DTR$NOWINDOWS logical.)

/[NO]DEBUG

Specifies whether DATATRIEVE should display special debug messages. The default is NODEBUG.

/VARIANT = image-suffix

Indicates the 1- to 26-character suffix applied to a DATATRIEVE image upon installation to uniquely identify that image. This qualifier is needed only if you invoke a DATATRIEVE image that was not selected as the default image at installation time. If the default image you are running includes a suffix and you want to run an image that does not include a suffix, use the /VARIANT qualifier alone, as in the following example:

```
$ DATATRIEVE/VARIANT
```

"DATATRIEVE"

Specifies a DATATRIEVE command or statement that is to be executed by DATATRIEVE. It allows the user to execute a DATATRIEVE command, statement, or procedure without entering interactive DATATRIEVE. If the

DATATRIEVE argument contains more than one word, it must be placed in quotation marks.

Usage Notes

- Invoking DATATRIEVE with the DATATRIEVE command is particularly useful if you are working in a DECwindows environment and you want to submit a DATATRIEVE command procedure from a foreign command line. Normally, if your system is defined to invoke the DECwindows interface and you try to execute a DATATRIEVE command from a foreign command line, the DATATRIEVE main application window appears on your screen. By using the DCL DATATRIEVE command with the /INTERFACE=CHARACTER_CELL qualifier, you can enter the command on a foreign command line and the main application window will not be displayed.
- You can abbreviate this DCL command to the shortest form that is unique among other DCL commands.
- If you are not certain about the default image of DATATRIEVE that was installed on your system, see your system manager or the person in charge of DATATRIEVE on your system.

Example

The following example shows how to execute the DATATRIEVE SHOW DICTIONARY command without entering interactive DATATRIEVE.

```
$ DATATRIEVE/INTERFACE=CHARACTER_CELL "SHOW DICTIONARY"
The default directory is _CDD$TOP.DTR$LIB.DEMO

$
```

DECLARE Statement

Defines a global or a local variable.

Format

DECLARE variable-name variable-definition .

Arguments

variable-name
Is the name of the variable being defined. The name must conform to the rules for names listed in the *VAX DATATRIEVE User's Guide*.

variable-definition
Is the definition of the variable, which consists of field definition clauses. If you include more than one such clause, separate them with spaces, tab characters, or carriage returns. Refer to the chapter on record definitions in the *VAX DATATRIEVE User's Guide* for information on field definition clauses.

. (period)
Ends the DECLARE statement.

Restrictions

- In the variable definition, you must include at least one COMPUTED BY, PICTURE (PIC), or USAGE clause to specify the data type, length, scale, and edit string of the variable.
- Although you can include other field definition clauses, you cannot use an OCCURS clause or REDEFINES clause in the variable definition. A variable can have the properties of only an elementary, nonrepeating field in a DATATRIEVE record definition.

Results

- When you use a DECLARE statement at the DATATRIEVE command level, DATATRIEVE creates a global variable. A global variable exists until you end the DATATRIEVE session, until you explicitly release the variable with the RELEASE command, or until you enter a DECLARE statement for a variable of the same name.

- Unless you define a global variable with a COMPUTED BY clause, the global variable retains the value you assign it until you explicitly release the variable with a RELEASE command, assign a new value to the variable, or end your DATATRIEVE session. The value of a global variable defined with a COMPUTED BY clause depends on the value expression that controls the computation. If the value expression is based on the value of a field in a record, the variable has a value only when there is a valid single record context in which to resolve the value expression.
- When you use a DECLARE statement as part of any other DATATRIEVE statement, such as a BEGIN-END or THEN statement, DATATRIEVE creates a local variable. A local variable exists only within the statement in which it is defined. DATATRIEVE releases all local variables created within a statement when it encounters the end of that statement.

  For example, if you define a local variable in the third BEGIN-END block in a series of four nested BEGIN-END blocks, you can refer to, assign values to, and retrieve values from the variable in the third and fourth blocks. That is, after DATATRIEVE executes the DECLARE statement in the third block, the value of the variable can be changed or retrieved by any of the remaining statements in the third block, including any of the statements in the fourth and innermost block. That local variable, however, has no meaning for any statement outside those two inner blocks. No statement in the first or second block and outside the two inner blocks can refer to the local variable defined in the third one.
- If the statement that defines a local variable is in a FOR loop or REPEAT statement, the local variable is initialized each time DATATRIEVE executes the statement.

  The initial value assigned to the variable by DATATRIEVE depends on the field definition clauses included in the DECLARE statement in which the variable was defined. DATATRIEVE assigns the DEFAULT value to the variable if one is defined. If no DEFAULT value is defined but a MISSING value is, DATATRIEVE assigns the MISSING value to the variable.

  If neither a DEFAULT nor MISSING value is defined, numeric variables are initialized as zero, and alphanumeric and alphabetic variables are initialized as spaces.

Usage Notes

- To assign a value to the variable, use the following format of the Assignment statement:

  ```
  variable-name = value-expression
  ```

  See Chapter 1 and the section on the DECLARE statement for more information on assigning values to variables.

- Two SHOW commands let you display the names and types of the global variables currently defined: SHOW FIELDS and SHOW VARIABLES. See the section on the SHOW command for more information.

```
DTR> DECLARE X REAL.
DTR> DECLARE Y PIC 9(5).
DTR> DECLARE A PIC XX.
DTR> DECLARE B PIC AAA.
DTR> SHOW VARIABLES
Global variables
   X    <Number>
   Y    <Number>
   A    <Character string>
   B    <Character string>

DTR>
```

- Use the RELEASE command (see the section on the RELEASE command) to remove global variables from your workspace and from the context stack:

```
DTR> SHOW VARIABLES
Global variables
   X    <Number>
   Y    <Number>
   A    <Character string>
   B    <Character string>

DTR> RELEASE X, A
DTR> SHOW VARIABLES
Global variables
   Y    <Number>
   B    <Character string>

DTR>
```

Examples

Declare the global variable NEW_BEAM as a 2-digit numeric field with a DEFAULT value of 10:

```
DTR> DECLARE NEW_BEAM PIC 99 DEFAULT VALUE IS 10.
DTR> PRINT NEW_BEAM

NEW
BEAM

 10

DTR>
```

Declare the global variable X as a single-precision floating-point number, with a MISSING VALUE of 36:

```
DTR> DECLARE X REAL MISSING VALUE IS 36.
DTR> PRINT X

    X

36.0000000

DTR> SHOW VARIABLES
Global variables
   X    <Number>

DTR> RELEASE X
DTR> SHOW FIELDS
No ready sources or global variables declared.
DTR>
```

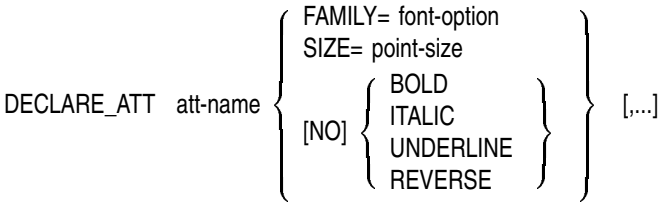
Declare the variable DUE as a date. Assign today's date to DUE and suppress the header with a hyphen in parentheses:

```
DTR> DECLARE DUE USAGE IS DATE.
DTR> DUE = "TODAY"
DTR> PRINT DUE (-)

22-Sep-90

DTR>
```

DECLARE_ATT Statement (Report Writer)

Specifies text control elements (font family, character size, weight, slant, underline) overriding or complementing the default attributes for the ATT argument used by the Report Writer's PRINT and SET statements.

Format



Arguments

att-name

Is the user-defined name of the attribute list. This name is used by the ATT clause in the Report Writer Print statement.

Table 4–2 shows the options which can be controlled by the DECLARE_ATT statement.

Table 4–2 DECLARE_ATT Print Attributes

Attribute	Options	Header default	Body default
Family	HELVETICA TIMES COURIER AVANTGARDE SYMBOL LUBALIN_GRAPH NC_SCHOOLBOOK SOUVENIR	HELVETICA	HELVETICA
Point Size	8 10 12 14 18 24 36 48 72 96	14	10
Bold	BOLD NO BOLD	BOLD	NO BOLD
Italic	ITALIC NO ITALIC	NO ITALIC	NO ITALIC
Underline	UNDERLINE NO UNDERLINE	NO UNDERLINE	NO UNDERLINE
Reverse	REVERSE NO REVERSE	NO REVERSE	NO REVERSE

font-option
Is the font family chosen. Available options are listed in Table 4–2.

point-size
Is the point size for the chosen font family. Available options are listed in Table 4–2.

Restrictions

- DECLARE_ATT is legal within the REPORT statement only.
- DECLARE_ATT statements must precede all other statements in the report specification.
- RESET is not a valid name for a DECLARE_ATT statement.

Result

The DECLARE_ATT statement creates attributes that can be referred to using the ATT clause.

Usage Notes

- Attributes declared through DECLARE_ATT can be used in subsequent SET and PRINT statements.
- If there are multiple DECLARE_ATTs with the same name, latter ones override the earlier ones.
- Font size is specified in typographical points.
- Text attributes are ignored when a DTIF format is chosen.
- In DDIF and PS formats, all attributes are encoded.
- Note that certain devices or receiving applications may not process correctly the attributes set. For example, certain terminals do not support ANSI escape sequences.
- For TEXT format reports, attributes are applied through the introduction of standard ANSI escape sequences in the text file. The following table defines the applicability of text attributes to TEXT format:

Attribute	Applicable to TEXT
FAMILY	N
SIZE	N
BOLD	Y
ITALIC	N
UNDERLINE	Y
REVERSE	Y

- There are different default text attributes for the headers (report title, column headers) and body (all other components) of the report. The default text attributes are shown in Table 4–2.

Example

The following example shows a number of DECLARE_ATT statements:

```
DTR> REPORT PAYABLES WITH INVOICE_DUE NOT MISSING -
RW>           SORTED BY AGE ON AGING_REPORT.PS FORMAT PS
RW>
RW> DECLARE_ATT TITLE      FAMILY = TIMES, SIZE = 24, BOLD, NOITALIC
RW> DECLARE_ATT DATE_PAGE  FAMILY = TIMES, SIZE = 14, NOBOLD, ITALIC
RW> DECLARE_ATT COL_HDR    FAMILY = TIMES, SIZE = 12, BOLD, ITALIC
RW> DECLARE_ATT DETAIL     FAMILY = NC_SCHOOLBOOK, SIZE = 10
RW> DECLARE_ATT TOT_ACCNTS FAMILY = HELVETICA, SIZE = 12, NOBOLD
RW> DECLARE_ATT TOTAL      FAMILY = HELVETICA, SIZE = 14, BOLD
RW> DECLARE_ATT ULINE      UNDERLINE
RW> DECLARE_ATT NO_ULINE   NOUNDERLINE
RW> DECLARE_ATT GRAND_TOTAL FAMILY = HELVETICA, SIZE = 14,
                                          BOLD, ITALIC
```

DECLARE PORT Statement

Creates a temporary DATATRIEVE port with the name you specify and readies the port for WRITE access. DATATRIEVE does not enter a definition of the port in the CDD/Repository data dictionary.

Format

```
DECLARE PORT   port-name USING

        level-number-1 field-definition-1.

        [level-number-2 field-definition-2.]
                .               .
                .               .
                .               .
        [level-number-n field-definition-n.]
  ;
```

Arguments

port-name
Is the name of the port. It cannot duplicate a DATATRIEVE keyword or the given name of any domain you may bring into your workspace.

level-number
Is the level number for the field in the port declaration. It indicates the relationship of the field to the other fields of the port.

field-definition
Is a field definition. A port declaration must have at least one field definition. Each field definition ends with a period. See the chapter on record definitions in the *VAX DATATRIEVE User's Guide* for descriptions of the field definition clauses.

; (semicolon)
Ends the port declaration.

Restrictions

- You cannot invoke a procedure in a port declaration.
- In the port declaration, you must include at least one PICTURE (or PIC) clause or one USAGE clause.
- You cannot ready a port that is already declared. When you declare a port, DATATRIEVE automatically readies the port for WRITE access.

Results

- If you press the RETURN key before typing the semicolon, DATATRIEVE prompts you to continue the declaration with the CON> prompt. DATATRIEVE continues to prompt you with the CON> prompt until you type a semicolon and press the RETURN key, or until it detects a syntax error.

  If you make a syntax error while entering the port declaration, DATATRIEVE returns to command level (indicated by the DTR> prompt) without creating the port.
- If your application program declares a port at the DATATRIEVE command level, DATATRIEVE creates a port that is available to your application program until you end your access to it with the FINISH command or until you exit from your session. Such a port is a **global port** and, in this respect, is similar to a global variable.
- If your application program declares the port at some level in a compound statement, the port is valid only for the remainder of the statement in which it is declared and for those statements contained in the statement in which it is declared. Such a port is a **local port** and, in this respect, is similar to a local variable.

Usage Note

See the *VAX DATATRIEVE Guide to Programming and Customizing* for information about using a port to transfer data between DATATRIEVE and your application program.

Example

The following example shows a port declared for YACHTS records in a FORTRAN program:

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

DECLARE SYNONYM Command

Defines a synonym for a DATATRIEVE keyword.

Format

DECLARE SYNONYM { synonym [FOR] keyword } [,...]

Arguments

synonym

Is the name of the synonym being defined. The name must conform to the rules for names listed in the *VAX DATATRIEVE User's Guide*.

FOR

Is an optional word to clarify syntax.

keyword

Is the name of a DATATRIEVE keyword for which a synonym is being defined. (See the *VAX DATATRIEVE User's Guide*.)

Restrictions

- You cannot define a synonym for the keyword CURRENT.
- You cannot use the DECLARE SYNONYM command within a BEGIN-END block.
- You cannot use the DECLARE SYNONYM command within a FOR loop.
- You cannot define a keyword as a synonym for another keyword. For example, you cannot define PRINT as a synonym for FIND.
- You cannot define the same synonym for two different keywords. For example, if you have already defined A as a synonym for ALL, you cannot define A as a synonym for AVERAGE.
- You cannot use the DTR$SYNONYM file for creating synonyms for user-defined keywords (UDK). If you want to create synonyms for user-defined keywords, you can use the startup command file DTR$STARTUP.

Results

- DATATRIEVE interprets the synonym in exactly the same way as the keyword with which you associated it.
- This synonym is in effect for the current DATATRIEVE session only. When you exit from DATATRIEVE, all the synonyms you defined with DECLARE SYNONYM are released.

Usage Notes

- Use DECLARE SYNONYM to define temporary synonyms during a particular DATATRIEVE session.
- You can define a synonym for a user-defined keyword.
- You can define a synonym for another synonym. For example:

```
DTR> DECLARE SYNONYM P FOR PRINT
DTR> DECLARE SYNONYM PR FOR P
```

  Now PR is a synonym for the keyword PRINT. You can use PR, PRINT, or P interchangeably.
- If you want to define synonyms that will remain in effect in future DATATRIEVE sessions, include the assignment of synonyms in your DTR$STARTUP command file.
- You can also assign synonyms using a DTR$SYNONYM file. (The DECLARE SYNONYM command replaced this method for assigning synonyms in Version 2.0 and is still the recommended method for users of Version 2.0 or later.)

  Create a file containing a list of keywords and their corresponding abbreviations. Use one keyword/abbreviation pair per line. Capitalize all keywords and definitions and leave at least one space between the keyword and the abbreviation. Assign the logical name DTR$SYNONYM to this file. When you invoke DATATRIEVE, it uses this file to find any synonyms you have defined. If you make an incorrect synonym file entry, DATATRIEVE does not produce an error message, but the synonym does not work.

Examples

Define synonyms for the keywords FIND and PRINT. Use the synonyms to form a collection of the first two yachts, and then print the collection.

```
DTR> READY YACHTS
DTR> DECLARE SYNONYM FD FOR FIND, PT FOR PRINT
DTR> FD FIRST 2 YACHTS; PT CURRENT

                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 ALBERG      37 MK II   KETCH   37    20,000  12  $36,951
 ALBIN       79         SLOOP   26     4,200  10  $17,900

DTR>
```

Calculate the price per foot and per pound for the first five yachts. To save typing, define synonyms for COMPUTED, EDIT_STRING, and PRINT.

```
DTR> SET NO PROMPT
DTR> DECLARE SYNONYM CD FOR COMPUTED
CON>     E_S FOR EDIT_STRING, PT FOR PRINT
DTR> DECLARE PER_FT CD BY (PRICE/LOA) E_S $$$$.99.
DTR> DECLARE PER_LB CD BY (PRICE/DISP) E_S $$.99.
DTR> FOR FIRST 5 YACHTS
CON>     PT TYPE, PRICE, PER_FT, PER_LB

                                    PER     PER
MANUFACTURER   MODEL     PRICE     FT      LB

 ALBERG      37 MK II   $36,951   $998.68  $1.85
 ALBIN       79         $17,900   $688.46  $4.26
 ALBIN       BALLAD     $27,500   $916.67  $3.78
 ALBIN       VEGA       $18,600   $688.89  $3.67
 AMERICAN    26          $9,895   $380.58  $2.47

DTR>
```

DEFAULT VALUE Clause

Specifies a default value for initializing a field to which you do not make a direct assignment in STORE or Restructure statements.

Format

```
DEFAULT   [VALUE]   [IS]   literal
```

Argument

VALUE IS

Are optional keywords you can use to clarify the syntax of the clause.

literal

Is either a numeric or character string literal. (See Chapter 1 for a discussion of these two types of literals.)

Restriction

This clause is valid for elementary fields only.

Results

DATATRIEVE uses the DEFAULT VALUE clause to initialize fields when you create records in the following ways:

- When you use Assignment statements in the USING clause of a STORE statement but make no assignment to the field with the default value specified in the DEFAULT VALUE clause
- When you respond either to prompts from a STORE statement without a USING clause or to prompting value expressions in the USING clause of a STORE statement and, in response to a prompt, you press the TAB key and then the RETURN key

Examples

Define a field in a student record that sets a default value for tuition owed at registration.

```
09 TUITION_DUE PIC Z(4)9.99
   EDIT_STRING IS $$$,$$$.99
   DEFAULT VALUE IS 4800.
```

Define a date field with the default value of the day you store the record:

```
03 DATE_IN USAGE DATE DEFAULT "TODAY".
```

DEFINE DATABASE Command

Defines a path name for a relational database (Format 1) or a path name for a VAX DBMS database instance (Format 2).

Format 1

```
DEFINE DATABASE  rdb-database-path  ON  root-file-spec;
```

Format 2

```
DEFINE DATABASE  dbms-database-path
    [USING]  [SUBSCHEMA]  subschema-name
    [OF]  [SCHEMA]  schema-path-name
    ON  root-file-spec;
```

Arguments

rdb-database-path
Is the CDD/Repository path name of the relational database you want to define. The path name can be either a DMU or a CDO style path name.

root-file-spec
Is the name of the database root file. For relational databases, the default file type is .RDB; for VAX DBMS, the default file type is .ROO.

dbms-database-path
Is the CDD/Repository path name you choose for the VAX DBMS database instance. The path name can be either a DMU or a CDO style path name. Although a DMU style path name is still accepted, you are recommended to use a CDO path name.

subschema-name
Is the name of a subschema for the specified schema.

schema-path-name
Is the CDO path name of a VAX DBMS schema.

;(semicolon)
Ends the DEFINE DATABASE command.

Restrictions

- Relational and VAX DBMS database path names do not have version numbers.
- To define a DMU format database you must have the following access privileges:
  - P (PASS_THRU) access to all ancestors of the dictionary directory you want to create
  - P (PASS_THRU) and X (EXTEND) access to the parent directory
- To define a CDO format database you must have the following access privileges:
  - VMS access to the VMS directory
  - S (SHOW) and U (CHANGE) access to the dictionary
- When you define a VAX DBMS database using Format 2, you cannot use the same path name for both the database and the schema.
- For Rdb/VMS and Rdb/ELN, the root file is the file specification of the database file. For VIDA, it is a VIDA database name. The VIDA database name is a set of value assignments to database name qualifiers. DATATRIEVE passes the database name to VIDA in the form of a text string literal. DATATRIEVE does not validate the text string.

  Note that for any relational database, root-file-spec can also be a logical name. If you choose to include a logical name, you must assign an appropriate value to that name before readying the database.

Examples

Define a relational database path name:

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

Define a VAX DBMS database instance. The CDD/Repository path name of the schema is SYS$COMMON:[CDDPLUS]DBMS.PARTS. The name of the subschema is PARTSS1, and the root file is DB0:[BULGAKOV]PARTSDB.ROO:

```
DTR> DEFINE DATABASE PARTS_DB
DFN>     USING PARTSS1
DFN>     OF SYS$COMMON:[CDDPLUS]DBMS.PARTS
DFN>     ON DB0:[BULGAKOV]PARTSDB.ROO;
DTR>
```

DEFINE DICTIONARY Command

Creates a dictionary directory in the CDD/Repository data dictionary system.

Format

DEFINE DICTIONARY path-name

Argument

path-name

Is the given name, the full dictionary path name, or a relative dictionary path name of the dictionary you want to create. The DEFINE DICTIONARY command accepts both DMU and CDO style path names.

Restrictions

- All ancestors of the dictionary directory you want to create must exist.
- The name you assign to the dictionary directory you want to create must not duplicate the name of any other directory or dictionary object in the parent directory.
- The dictionary path name does not have a version number.
- To define a DMU format dictionary directory you must have the following access privileges:
  - P (PASS_THRU) access to all ancestors of the dictionary directory you want to create
  - P (PASS_THRU) and X (EXTEND) access to the parent directory
- To define a CDO format dictionary directory you must have the following access privileges:
  - VMS access to the VMS directory
  - S (SHOW) and U (CHANGE) access to the dictionary
- Before you define a CDO dictionary directory, you must create the dictionary through CDO. If the dictionary does not exist, you will get an error message. For example, if you enter the following command and the dictionary DISK1:[SWANSON.DTRWORK] does not exist, the DEFINE DICTIONARY command will fail:

```
DTR> DEFINE DICTIONARY DISK1:[SWANSON.DTRWORK]NEWDICT
```

Results

- DATATRIEVE creates an empty directory in the CDD/Repository data dictionary system. In this directory, you can place other dictionary directories and the definitions of dictionary objects.
- If you do not enter the path name on the same input line as the DEFINE DICTIONARY command, DATATRIEVE prompts you for it with the DFN> prompt. After you enter the completed command, you return to the DATATRIEVE command level.
- DATATRIEVE does not make the newly created directory your default dictionary directory. Your default dictionary directory setting is unaffected by the DEFINE DICTIONARY command, regardless of its relationship to the new directory within the data dictionary.
- CDD/Repository provides the new directory with a default access control list. Two user identification criteria are supplied: a wildcarded UIC and the the user name of your process.

  If you define the dictionary directory in the DMU format dictionary, six privileges are granted: C (CONTROL), D (LOCAL_DELETE), H (HISTORY), P (PASS_THRU), S (SEE), and X (EXTEND). No privileges are denied, and none are banished. The access privileges you have to the new directory depend on the privileges that are granted, denied, and banished by the ancestors of the new directory.

  If you define the dictionary directory in the CDO format dictionary, four privileges are granted: U (CHANGE), C (CONTROL), D (DELETE), and S (SHOW).

  The chapter on ACL in the *VAX DATATRIEVE User's Guide* explains access control lists and the DATATRIEVE and CDD/Repository access privileges.

Usage Notes

- You can verify the creation of the new dictionary directory. Use the SET DICTIONARY command to make the parent of the new directory your default dictionary directory. Then enter the SHOW DICTIONARIES command. DATATRIEVE displays the name of the new directory along with those of any other dictionary directories cataloged in your default directory.
- To change your default directory to the newly created directory, use the SET DICTIONARY command. If your current default directory is the parent of the new directory, you can specify the new directory's given name in the SET DICTIONARY command.

If your current default directory is not the parent of the new directory, you can use either the new directory's full dictionary path name or the appropriate relative path name in the SET DICTIONARY command.

Once you have changed your default directory, you can use the SHOW DICTIONARY command to display the complete dictionary path name of your new default directory.

- You do not have to set your default dictionary directory to the newly created directory to define either dictionary objects or directories in that new directory. With the appropriate DEFINE command, you need only specify the full or relative dictionary path name of the directory or object you want to define.
- To enter definitions of dictionary elements in a directory you create, use the following commands: DEFINE DOMAIN, DEFINE PORT, DEFINE PROCEDURE, DEFINE RECORD, DEFINE TABLE. (See the sections on these commands in this chapter.)
- To create new versions of dictionary elements, use the following commands: REDEFINE DOMAIN, REDEFINE PORT, REDEFINE PROCEDURE, REDEFINE RECORD, REDEFINE TABLE. (See the section in this chapter on REDEFINE.)
- To change the access control list for a dictionary directory, use the DEFINEP command.
- Use the SHOWP command to see the access control list for the new dictionary directory.
- To show your access privileges to your current default dictionary directory, enter the SHOW PRIVILEGES command.

Examples

Use a given name to define a dictionary directory named TEST when CDD$TOP.RESEARCH is your default directory:

```
DTR> SHOW DICTIONARY
The default directory is CDD$TOP.RESEARCH
DTR> DEFINE DICTIONARY TEST
DTR> SHOW DICTIONARIES
Dictionaries:
        DEMO            TEST
DTR> SET DICTIONARY TEST
DTR> SHOW DICTIONARY
The default dictionary is CDD$TOP.RESEARCH.TEST
DTR> SHOW PRIVILEGES
Privileges for CDD$TOP.RESEARCH.TEST
 R (DTR_READ)     - may ready for READ, use SHOW and EXTRACT
 W (DTR_WRITE)    - may ready for READ, WRITE, MODIFY, or EXTEND
 M (DTR_MODIFY)   - may ready for READ, MODIFY
 E (DTR_EXTEND/EXECUTE) - may ready to EXTEND, or access table
                                                      or procedure
 C (CONTROL)      - may issue DEFINEP, SHOWP, DELETEP commands
 D (LOCAL_DELETE) - may delete a dictionary object
 F (FORWARD)      - may create a subdictionary
 H (HISTORY)      - may add entries to object's history list
 P (PASS_THRU)    - may use given name of directory or object
                                                       in pathname
 S (SEE)          - may see (read) dictionary
 U (UPDATE)       - may update dictionary object
 X (extend)       - may create directory or object within directory

DTR>
```

Use a relative dictionary path name to define a dictionary directory in the CDD$TOP.MANUFACTURING directory when CDD$TOP.DTR$LIB is your default directory:

```
DTR> SHOW DICTIONARY
The default directory is CDD$TOP.DTR$LIB
DTR> DEFINE DICTIONARY
DFN> -.MANUFACTURING.TEST
DTR> SET DICTIONARY -.MANUFACTURING
DTR> SHOW DICTIONARIES
        INVENTORY       TEST
DTR> SHOWP TEST
  1:    [*,*], Username: "JONES"
        Grant - CDHPSX, Deny - None, Banish - none

DTR>
```

Use a full name to define a CDO dictionary directory named TEST. DISK1:[SWANSON.DTRWORK] is your default directory:

DEFINE DICTIONARY Command

```
DTR> SHOW DICTIONARY
The default directory is DISK1:[SWANSON.DTRWORK]
DTR> DEFINE DICTIONARY DISK1:[SWANSON.DTRWORK]TEST
DTR> SHOW DICTIONARIES
Dictionaries:
        MYDICT            TEST

DTR>
```

DEFINE DOMAIN Command

Stores a domain definition in the CDD/Repository data dictionary system.

The following sections explain how to define domains for CDD$DATABASE objects, VAX DBMS databases, network domains based on domains residing at other DECnet nodes, domains based on relational databases, domains based on single RMS files, and views based on one or more domains. (Further information on VAX DBMS databases, relational databases, and network domains can be found in the *VAX DATATRIEVE User's Guide*).

Format

To define a CDD$DATABASE domain, use the following syntax:

```
DEFINE DOMAIN   domain-path-name   [USING]   database-name

    [FORM [IS] form-name [IN] file-name [USING exchange-record]]

    [WITH]   RELATIONSHIPS ;
```

To define a VAX DBMS domain, use the following syntax:

```
DEFINE DOMAIN   domain-path-name   [USING]   record-name

    [OF]   [DATABASE]   database-path-name

    [FORM [IS] form-name [IN] file-name [USING exchange-record]] ;
```

To define a network domain, use the following syntax:

```
DEFINE DOMAIN   domain-path-name [USING] remote-path-name AT node-spec

    [FORM   [IS]   form-name   [IN] file-name [USING exchange-record]]

    [[WITH] RELATIONSHIPS] ;
```

To define a relational domain, use the following syntax:

```
DEFINE DOMAIN   domain-path-name [USING] relation-name

    [OF]   [DATABASE]   database-path-name

    [FORM [IS] form-name [IN] file-name [USING exchange-record]]

    [[WITH] RELATIONSHIPS] ;
```

DEFINE DOMAIN Command

To define an RMS domain, use the following syntax:

```
DEFINE DOMAIN   path-name [USING] record-path-name ON file-spec

    [FORM [IS] form-name [IN] file-name [USING exchange-record]]

    [[WITH]   RELATIONSHIPS] ;
```

To define a view domain, use the following syntax:

```
DEFINE DOMAIN   view-path-name   OF   domain-path-name-1 [,...]  [ BY    ]
                                                                 [ USING ]

                level-number-1   field-name-1      OCCURS FOR rse-1 .

                level-number-2   field-name-2   { OCCURS FOR rse-n       }
                                                { FROM domain-path-name-n } .
                    .                .               .
                    .                .               .
                    .                .               .
                [FORM [IS] form-name [IN] file-name [USING exchange-record]]
                [[WITH]   RELATIONSHIPS]   ;
```

Arguments

domain-path-name
Is the CDD/Repository dictionary path name of the domain you are defining. The domain-path-name can be either a DMU or CDO style path name.

database-name
Is the name of a CDD$DATABASE object, defined through CDO using the DEFINE DATABASE command, that points to a CDD$RMS_DATABASE object. Refer to the *CDD/Repository Common Dictionary Operator Reference Manual* for more information on CDD$RMS_DATABASE objects.

record-name
Is the name of a record type contained in a subschema of the specified VAX DBMS database.

database-path-name
Is the DATATRIEVE definition of the database instance. The database-path-name can be both DMU and CDO style path name.

remote-path-name
Is the given name, full dictionary path name, or relative dictionary path name of a domain definition at another node in a network of VAX computers linked by DECnet. That domain, the associated record definition, and the associated data file must already exist in the data dictionary at the remote node before you can ready the network domain. The domain-path-name can be either a DMU or CDO style path name.

node-spec
Specifies the network address.

If the login procedure used by the remote process does not supply the necessary login information (user name, password, and, optionally, account name), either the person readying the network domain or the network domain definition must supply this information.

You can use any of the following formats to specify the network address and to provide the best level of access security for your installation:

- node-name"username password [account-name]")

  Examples of this format are:

  ```
  BIGVAX"WARTON KNOCKKNOCK DEPT32"
  ELEVEN"LINTE LETMEIN"
  ```

  When you specify the network address using this format, users do not have to supply login information when readying the network domain.

- node-name"*.username-prompt *.password-prompt [*.account-prompt]"

  Examples of this format are:

  ```
  WINKEN"*.USERNAME *.PASSWORD *.ACCOUNT"
  VAXTWO"*.'user name' *.'password'"
  PDPTWO"*.'user name' *.'password'"
  ```

  When you specify the network address using this format, users are prompted for login information when they ready the network domain. This method provides the best security.

- node-name

  Two examples of this format are:

  ```
  BIGVAX
  ELEVEN
  ```

When you specify the network address with this format, the account used by the remote process must provide login information automatically.

If you prefer, you can combine elements from the first two formats. For example, you can explicitly specify the user name and specify a prompting value expression for the password:

```
SNOOPY"CLARK *.PASSWORD"
```

relation-name
Is the name assigned to the relation when the database was created.

path-name
Is the given name, full dictionary path name, or relative dictionary path name of the domain being defined. The path name cannot resolve to the full dictionary path name of any other object or directory in the data dictionary system. The path-name can be either a DMU or CDO style path name.

record-path-name
Is the given name, full dictionary path name, or relative dictionary path name of the record definition to be associated with the domain. You must enter this record definition in the data dictionary (with the DEFINE RECORD command) before you can ready the domain. The dictionary path name of the record cannot resolve to the full dictionary path name of any other directory or object in the data dictionary. The record-path-name can be either a DMU or CDO style path nam

file-spec
Is the VMS file specification of the RMS file containing the data for the domain. This file must exist when you ready the domain. A complete file specification has the following format:

```
node-spec::device:[directory]file-name.type;version
```

view-path-name
Is the given name, full dictionary path name, or relative dictionary path name of the view being defined. The path name cannot resolve to the full dictionary path name of any other object or directory in the data dictionary. The view-path-name may be either a DMU or CDO style path name.

domain-path-name-1
domain-path-name-n
Is either the given name, full dictionary path name, or relative dictionary path name of a domain containing records to be included in the view. If the domain name is a domain path name, it cannot duplicate the name of the view. When specifying more than one domain path name, use a comma to separate each name

from the next. The domain-path-name may be either a DMU or CDO style path name.

level-number
Is the level number for a field in the view definition.

field-name
Is the name of a field in the view definition. If field-name is followed by an OCCURS FOR clause, field-name has no relationship to any field in the domain or domains specified in the RSE. Whether or not field-name is the same as the names of any of those fields does not matter. If field-name is followed by a FROM clause, field-name must be the name of a field in a domain specified in the OF domain-path-name-1 [,...] clause.

OCCURS FOR rse
Indicates that the associated field is to be included in the view only for those records specified by the RSE. The RSE must contain a reference to one of the domains, relations, or VAX DBMS records listed in the OF clause.

FROM domain-path-name
Indicates that the definition of the associated field is identical to that of the field of the same name in the domain, relation, or VAX DBMS record specified by domain-path-name-n. The argument domain-path-name must be the same as that used in the preceding OCCURS FOR clause.

. (period)
Ends a field definition.

form-name
Is a form name.

file-name
For TDMS and FMS forms it is the name of a form library. For DECforms forms it is the form file name, which can either be a .FORM or a .EXE file.

exchange-record
Is the CDD/Repository path name (either DMU or CDO) of a record used to send and receive data with DECforms.

RELATIONSHIPS
Causes relationships to be set up between the domain being defined and one of the following items:

- The CDD$DATABASE object

- The domain specified on the remote node
- The record definition
- The objects referenced by the view

; (semicolon)
Ends the domain definition.

Restrictions

- **General Restrictions**
  - DATATRIEVE creates a default access control list for the domain. The UIC identification matches any UIC ([*,*]), and the user name is set to your current VMS user name.

    If you define your domain with a DMU path name, the privileges granted are C (CONTROL), D (LOCAL_DELETE), E (DTR_EXTEND /EXECUTE), H (HISTORY), M (DTR_MODIFY), R (DTR_READ), S (SEE), U (UPDATE), and W (DTR_WRITE). If you define your domain with a CDO path name, the privileges granted are U (CHANGE + DEFINE), C (CONTROL), D (DELETE), E (EXTEND), M (MODIFY), R (READ), S (SHOW), and W (WRITE). In either case, the C (CONTROL) privilege allows you to change the access control list to suit your needs.
  - The RELATIONSHIPS clause requires that the record definition, or the database being referenced, exist in the CDO dictionary at the time the domain is defined.
  - The USING subclause of the FORM IS clause is supported only for domains defined in CDO format. Nevertheless, the exchange record can exist either in DMU or CDO format.
  - You can use the second optional USING clause only if preceded by a FORM IS clause.
  - To use the RELATIONSHIPS argument, you must define the domain in the CDO dictionary.
  - You cannot create a new version of a domain unless you use the REDEFINE command.
  - You cannot invoke a procedure in a domain definition.
  - Do not assign to domains a given name that duplicates a DATATRIEVE keyword.
  - The field names of an FMS or TDMS form must be the same as the corresponding DATATRIEVE field names.

- The length and the data types of the DATATRIEVE fields must match the length and the data types of the FMS or TDMS form fields.
- The FMS or TDMS form field names can contain underscores (_) but not hyphens (-).
- When you use a form with the DATATRIEVE Call Interface, all fields are passed as strings. This can create problems for COMP fields and for numeric fields with implicit decimal points.
- To initialize the DATATRIEVE interface to forms, SET FORM must be in effect when you ready a domain that contains a FORM clause or when you use the DISPLAY_FORM or WITH_FORM statement.
- You cannot use the forms interface in batch processing of DATATRIEVE commands and statements.
- For domains defined with a FORM clause, DATATRIEVE uses the forms interface for these statements:

```
PRINT, PRINT [ALL] [OF] rse

MODIFY, MODIFY ALL OF rse

STORE domain-name
```

- For domains defined with a FORM clause, DATATRIEVE does not use the forms interface for these statements:

```
PRINT print-list

MODIFY { [USING statement] [VERIFY USING statement] }

STORE domain-name { [USING statement] [VERIFY USING statement] }
```

- To use the RELATIONSHIPS argument, the record definition being referenced in the RMS and Network domains must exist in the CDO dictionary at the time the domain is defined.

- **Defining a VAX DBMS Domain**
  - Do not use the same path name for both the domain and the database instance.
  - RELATIONSHIPS are not supported for VAX DBMS domains.
- **Defining a Network Domain**
  - You can define network domains only if your VMS system is linked by DECnet to one or more VMS systems on which DATATRIEVE has been installed.

  - If you include the FORM IS clause in the network domain definition, the forms library must reside on your own VAX system.
  - Other restrictions on network domains are the same as those for RMS domains.
  - The RELATIONSHIPS argument has the following requirements:
    * The remote node referenced in the definition must be accessible.
    * CDD/Repository must be installed on the remote node.
    * The domain being referenced on the remote node must exist in the CDO dictionary.
    * You must define the network domain in the CDO directory.

- **Defining a Relational Domain**
  - RELATIONSHIPS can only be used on relational domains that reference a CDD$DATABASE object supplied by the relational database.

- **Defining a View Domain**
  - The view definition must contain an OCCURS FOR clause as the top-level field.
  - The view definition must contain at least one FROM clause for each source.
  - Statements cannot refer to the field defined in the OCCURS FOR clause to retrieve or update data. For example, in the BOAT_VIEW domain defined in the Examples section, the statement PRINT BOAT_INFO OF BOAT_VIEW will generate an error.
  - The restrictions on defining RMS domains apply to defining views.
  - To use the RELATIONSHIPS argument, the domains referenced by the view must already exist in the CDO dictionary. If a view references more than one domain and not all of the domains exist in the CDO dictionary, then DATATRIEVE will define the view domain. However, you will be warned that the domains that are not in the CDO dictionary will not be part of a relationship.

Results

- **General Results**
  - If you press the RETURN key before typing the semicolon, DATATRIEVE prompts you to continue the definition with the DFN> prompt. DATATRIEVE continues to prompt you with the DFN> prompt until you type a semicolon and press the RETURN key or until it detects a syntax error. If you make a syntax error while entering the domain definition, DATATRIEVE returns to command level (indicated by the DTR> prompt) without creating a domain definition.
  - If you omit a field in the file specification of the data file, DATATRIEVE uses the defaults listed in Table 4–3.

Table 4–3 Output File Specification Defaults

Field	Default
node-spec::	Your local node
device:	Your default device
[directory]	Your default directory
file-name	Null string
.type	.DAT, .RDB, .ROO (for VAX DBMS root files)
;version	1 or next higher version number

The minimum file specification consists of a period (.); the specification of such a file stored in your default VMS directory ends with ".;n", where n is the version number and both the file name and the extension are null strings.

- If the RELATIONSHIPS argument is used, DATATRIEVE creates a CDD$DATABASE, CDD$RMS_DATABASE, and underlying objects. The CDD$RMS_DATABASE object provides information about files created by DATATRIEVE to other products that access data through the CDD$DATABASE object.
- DATATRIEVE enters a domain definition into the directory of the data dictionary determined by the dictionary path name you specify with the DEFINE DOMAIN command. If you do not specify the dictionary path name, DATATRIEVE saves the full path name of each of the domains the view references.

DATATRIEVE stores the view domain definition in the parent directory determined by the full dictionary path name of the view domain. In the full dictionary path name, the name of the parent directory immediately precedes the given name of the view domain.

- **Defining a View Domain**
  - When you define a view with the RELATIONSHIPS argument, DATATRIEVE creates relationships between the view domain, the domains specified on the command line, and the data aggregate entity containing view field information. You can see these relationships by using the command CDO SHOW USED_BY view-path-name.
  - DATATRIEVE also attempts to create data instance relationships pointing to the fields referenced by the view. To do this, the domains you specified must have been defined with relationships. If they were not, the view definition continues, but DATATRIEVE displays the following warning message:

```
Some domains referenced by view were defined without
RELATIONSHIPS.  Therefore not all view relationships
will be formed.
```

    If the underlying domains were defined with relationships, DATATRIEVE creates the data instance relationships and searches the record owned by the underlying domain for the names of elementary fields within the view. If the field is not found, the view definition fails and DATATRIEVE will display the following error message:

```
<...> is not a field name
```

Usage Notes

- **Generic Usage Notes**
  - Relationships between objects are made to specific versions of objects; relationships are not automatically propagated to newer versions of objects. This means that when a new version of the database is created, the owner domain, since it was defined with RELATIONSHIPS, will still point to the old version.
  - The record definition and the data file associated with the domain need not be defined when you enter a DEFINE DOMAIN command.
  - You cannot modify a domain definition with the DEFINE DOMAIN command. To change an existing domain definition, use the EDIT command, which places a REDEFINE DOMAIN command with the old

definition into the main buffer of the editor. Then you can use the editor to make the desired changes.

When you exit from the editor, DATATRIEVE places the updated domain definition with a new version number into the data dictionary. If the SET EDIT_BACKUP command is in effect, the old definition is retained in an earlier version. (See the chapter on record definitions in the *VAX DATATRIEVE User's Guide* and the section on EDIT in this chapter for information on changing the definitions of domains and other dictionary objects.)

- To increase the ease of transporting your domain definitions, specify the domain name as a given name and the record name as a full dictionary path name.
- Use a more complete file specification than just the file name and type. Then you can access your data even though you may be in a different VMS directory than the one where you first defined the domain.

- **Defining a CDD$DATABASE Domain**

  Defines a domain based on an existing CDD$DATABASE object. The domain is stored in the specified or implied directory of the data dictionary and creates an access control list for the domain.

  - If you have already defined a RMS$DATABASE through CDO, this form of the DEFINE DOMAIN command provides a simple way to access it.

- **Defining a VAX DBMS Domain**

  This special form of the DEFINE DOMAIN command specifies a VAX DBMS record type within a database instance.

  - You do not need to define a VAX DBMS domain to access a VAX DBMS database.
  - You need to define a VAX DBMS domain to:
    * Create a view that includes VAX DBMS data
    * Use a VAX DBMS domain from a remote system
    * Associate a form with a VAX DBMS record

- **Defining a Network Domain**

  Stores the definition of a network domain in the specified or implied directory of the data dictionary and creates an access control list for the domain.

  - If the dictionary path name specified for the remote domain is a relative one, DATATRIEVE resolves it based on the translation of the logical name

CDD$DEFAULT, which results from starting a VMS process with the node specification indicated.

If no user name and password are specified in the node specification argument, the remote process logs in under the default DECnet account. If a user name and password are specified, the remote process logs in as that user.

If CDD$DEFAULT is not defined as a system, group, or process logical name when the remote process logs in on the remote node, a relative dictionary path name for the remote domain resolves from CDD$TOP as the default dictionary directory.

- You may find that passwords are echoed if you use the CDO utility's SHOW GENERIC command to display the definition of a remote domain entity. To avoid displaying passwords, use a prompting value expression (*.prompt) as part of the access control string in the node specification of the DEFINE DOMAIN command for network or remote domains. When you display the definition of the remote domain, you will see the *.prompt rather than an actual password.
- If the remote domain is on a PDP–11 system, you specify only the domain's given name. Otherwise, the format for referring to remote domains is the same for both the VAX and the PDP–11 systems.
- Note that the remote domain you refer to in a network domain definition is no different from any other DATATRIEVE domain. It specifies the relationship of a particular record definition and a data file. (At remote VAX/VMS systems, a VAX DBMS or relational domain can also be the access path to a VAX DBMS or relational database.) The remote domain must be defined at the remote node during a DATATRIEVE session running on that remote node. A person or program local to that system can invoke DATATRIEVE to enter the domain definition, or a person or program running on that system as a remote terminal can enter the definition.

- **Defining a Relational Domain**

  This special form of the DEFINE DOMAIN command specifies a relation in a relational database.

  – You do not need to define a domain for a relation in order to ready it. When you ready a database directly, DATATRIEVE response time is faster than when you use domains to access relational records.

  – You need to define domains for relations to:

    * Create a view that includes data from a relation

  * Use relational domains from a remote system
  * Associate a form with a relation

- **Defining an RMS Domain**

  Stores a definition of an RMS domain in the specified or implied directory of the data dictionary and creates an access control list for the domain.

- **Defining a View Domain**

  Stores the definition of a view domain in the specified or implied directory of the data dictionary and creates an access control list for the view domain.

Examples

Define the CDD$DATABASE domain NEWYACHTS. Use the already defined CDD$DATABASE YACHTS.

```
DTR> DEFINE DOMAIN NEWYACHTS USING YACHTS_DB
DFN> WITH RELATIONSHIPS;
```

Define the VAX DBMS domain EMPLOYEES. The name of the record-type is EMPLOYEE, and the name of the database instance is PARTS_DB. The domain definition includes the FORM clause.

```
DTR> DEFINE DOMAIN EMPLOYEES
DFN>     USING EMPLOYEE OF DATABASE PARTS_DB
DFN>     FORM IS EMPFOR IN FORMS:PARTS.FLB;
DTR>
```

Define a network domain called REMOTE_YACHTS:

```
DTR> DEFINE DOMAIN REMOTE_YACHTS USING
DFN>    CDD$TOP.DTR$LIB.DEMO.YACHTS AT
DFN>    VAX32"SMITH ADRIENNE" FORM IS YACHT1 IN DTRFRM;
DTR>
```

The following example defines a DATATRIEVE domain that automatically uses a form. The domain is defined for the relation EMPLOYEES.

```
DTR> DEFINE DOMAIN EMPLOYEES
DFN>     USING EMPLOYEES OF DATABASE PERSONNEL
DFN>     FORM IS EMPFOR IN FORMSLIB;
DTR>
```

Define the domain PHONES. Use the record definition PHONE_REC that is cataloged in the directory CDD$TOP.DEPARTMENT. Specify PHONE.DAT as the data file:

```
DTR> DEFINE DOMAIN PHONES USING
DFN> CDD$TOP.DEPARTMENT.PHONE_REC ON PHONE.DAT;
DTR>
```

Define a view of yacht and owner information:

```
DTR> SHOW BOAT_VIEW
DOMAIN BOAT_VIEW OF YACHTS, OWNERS USING
01 BOAT_INFO OCCURS FOR YACHTS.
   03 TYPE FROM YACHTS.
   03 SKIPPERS OCCURS FOR OWNERS WITH TYPE EQ BOAT.TYPE.
      05 NAME FROM OWNERS.
      05 BOAT_NAME FROM OWNERS.
;

DTR> READY BOAT_VIEW
DTR> PRINT FIRST 4 BOAT_VIEW

                                        BOAT
MANUFACTURER   MODEL       NAME         NAME

 ALBERG      37 MK II
 ALBIN       79
 ALBIN       BALLAD
 ALBIN       VEGA       STEVE      DELIVERANCE
                        HUGH       IMPULSE

DTR>
```

You can use a view domain such as BOAT_VIEW as a source for modifying data in an RMS domain. See the *VAX DATATRIEVE User's Guide* for more examples of view definitions.

DEFINE FILE Command

Creates an RMS sequential or indexed sequential data file for the DATATRIEVE RMS domain specified by the dictionary path name.

Format for Defining a Sequential File

```
DEFINE FILE  [FOR]  path-name

[ ALLOCATION = n ]
[ SUPERSEDE      ]  [,...]
[ MAX            ]
```

Format for Defining an Indexed File

```
DEFINE FILE  [FOR]  path-name

[ ALLOCATION = n ]
[ SUPERSEDE      ]  [,...]
[ MAX            ]

{ KEY = field-name-1  [ ( [NO]CHANGE[,] [NO]DUP) ] }  [,...]
```

Format for Defining an RMS File Using a FDL File

```
DEFINE FILE  [FOR]  domain-name  USING  fdl-file-spec
```

Arguments

path-name

Is the dictionary path name of the DATATRIEVE RMS domain for which you want to create a data file.

domain-name

Is the name of the DATATRIEVE domain for which you want to create a data file.

ALLOCATION = n

Specifies an unsigned nonzero integer that determines the number of disk blocks initially allocated for the data file. If you omit this argument, zero blocks are allocated for the file.

SUPERSEDE

Causes DATATRIEVE to delete any existing data file that exactly matches the complete file specification, including the version number, in your RMS domain definition. The new file you are defining replaces the existing data file. If your

RMS domain definition does not include a file version number, the old file is not deleted, and the new file is assigned the next higher version number.

MAX
Causes DATATRIEVE to create a fixed-length RMS file for a domain whose record definition contains an OCCURS . . . DEPENDING clause. The length of every record in the data file has the maximum possible size, as determined by the value of the MAX argument in the OCCURS . . . DEPENDING clause:

```
OCCURS min TO max TIMES DEPENDING ON field-name
```

Each record can then store the maximum number of items in the list defined by the OCCURS . . . DEPENDING clause.

If you omit this argument, DATATRIEVE does not create a file with fixed-length records of the maximum possible size. The size of each record is determined when you store the record. If the file is defined as a sequential file, a record size cannot be increased to include more list items after it is initially stored.

KEY = field-name
Causes DATATRIEVE to create an RMS indexed file and specifies a field in the domain's record definition to be used as an index key for the domain's data file. The first key field specified in the DEFINE FILE command is the primary key, and all subsequent ones are alternate keys. If you specify more than one KEY clause, use a comma (,) to separate each clause from the next. If you are defining a file for a hierarchical record, do not make a list field the primary key.

If you omit this clause, DATATRIEVE creates an RMS sequential file.

CHANGE
NO CHANGE
Determines whether or not you can modify the content of the associated key field. The default is NOCHANGE for the primary key field and CHANGE for alternate key fields. See Table 4–4 for the allowed combinations of key field attributes.

DUP
NO DUP
Determines whether or not you can assign the same value to the specified key fields of two or more records. The default is NO DUP for the primary key field and DUP for alternate key fields. See Table 4–4 for the allowed combinations of key field attributes.

Table 4–4 Allowed Combinations of Key Field Attributes

	Key Field Attributes			
Key Type	**CHANGE + DUP**	**CHANGE + NO DUP**	**NO CHANGE + DUP**	**NO CHANGE + NO DUP**
Primary	Not Allowed	Not Allowed	Allowed	Allowed
Alternate	Allowed	Allowed	Allowed	Allowed

USING fdl-file-spec
Specifies the FDL file to be used in creating the RMS file.

Restrictions

- To define a data file for a DMU RMS domain, you must have the following privileges:
  - P (PASS_THRU) access to the parent directories cataloging the domain definition and the record definition
  - P (PASS_THRU), R (DTR_READ), S (SEE), and W (DTR_WRITE) access privileges to the domain definition
  - E (DTR_EXTEND/EXECUTE), P (PASS_THRU), S (SEE), R (DTR_READ), and W (DTR_WRITE) access to the associated record definition
- To define a data file for a CDO RMS domain, you must have the following privileges:
  - S (SHOW) access to the dictionary cataloging the domain definition and the record definition
  - S (SHOW) and W (WRITE) access to the domain and record definitions
- You cannot assign the CHANGE attribute to a primary key. See Table 4–4 for the allowed combinations of key field attributes.
- If you are defining a file for a hierarchical record, you cannot designate a list field as the primary key.
- If you define a group field key with DATATRIEVE or a segmented key with RMS, DATATRIEVE uses only the first elementary item for indexed access. However, DATATRIEVE does not use any part of a multiple field key when one of the subordinate items is numeric. Therefore, all elementary fields in a multiple field key must be nonnumeric for DATATRIEVE to use the first elementary field for indexed access.

- DATATRIEVE does not use keyed access when the record source is a collection.
- The domain specified by the dictionary path name must be an RMS domain, not a view domain, VAX DBMS domain, relational domain, remote domain, or port.

Results

- DATATRIEVE checks each DEFINE FILE command for special arguments that affect the characteristics of your data file. When you use the DEFINE FILE command to create a data file for a domain defined with relationships, DATATRIEVE creates a special CDD/Repository entity called a CDD$FILE_DEFINITION file. This entity contains information on the characteristics of your data file.
- If you include no KEY clauses, DATATRIEVE creates an RMS sequential data file for the domain specified by the dictionary path name.
- If you include one or more KEY clauses, DATATRIEVE creates an RMS indexed sequential data file for the domain specified by the dictionary path name.
- The file specification of the RMS file created by this command is the same as that of the data file in the definition of the domain specified by the dictionary path name. (Note that if you are using the USING fdl-file-spec format of the DEFINE FILE command, the name of the RMS file is also taken from the domain definition, not from any file name specified in the FDL file.)

  If the domain definition omits a field in the file specification, DATATRIEVE uses the following defaults:

Table 4–5 Output File Specification Defaults

Field	Default
node-spec::	Your local node
device:	Your default device
[directory]	Your default directory
file-name	Null string
.type	.DAT
;version	1 or next higher version number

- If you omit the ALLOCATION = n clause, DATATRIEVE sets the initial disk space allocation for the data file to zero blocks. When you store records into

the data file, RMS automatically extends the data file according to the cluster size established by your VMS system manager.

- If the record definition associated with the specified domain contains no OCCURS . . . DEPENDING clauses, DATATRIEVE creates a data file with a fixed-length record format.
- If you do not include the MAX argument and the record definition associated with the specified domain contains an OCCURS . . . DEPENDING clause, DATATRIEVE creates a data file with a variable-length record format.
- If you include the MAX argument and the record definition associated with the specified domain contains an OCCURS . . . DEPENDING clause, DATATRIEVE creates a data file with a fixed-length record format.
- If you include the SUPERSEDE argument and the file specification in the domain definition specifies a version number, DATATRIEVE deletes any existing data file having that file specification (including the version number) and replaces it with the new data file created by the DEFINE FILE command. The new file has the same file specification (including the version number) as that of the deleted file.

Usage Notes

- If you define a sequential file, you cannot erase records with the DATATRIEVE ERASE statement. You can, however, change the value of any field in a record.
- If you define an indexed file, you can erase records with the DATATRIEVE ERASE statement. You cannot, however, change the value of the primary key field of a record, or the value of any secondary key field with the NO CHANGE attribute.
- If you change the size of a record, you also need to define a new file to agree with the new record definition. Otherwise, you receive an error message indicating "bad record size" when you try to ready the domain.

  To avoid defining a new file, you can define filler fields in your record definition to allow space for future additions.
- If you define a sequential file for a record with an OCCURS . . . DEPENDING clause and do not use MAX, then you cannot extend the length of the list without defining a new file.
- See the *VAX DATATRIEVE User's Guide* for more information on defining data files.

- If you are using the USING fdl-file-spec format of the DEFINE FILE command, you should also note the following:
  - DATATRIEVE does not verify the validity of the FDL file specifications. For example, DATATRIEVE does not check to see that offsets of any specified keys match fields defined for the record; nor does it check for valid record formats, file organization, segmented keys, and so forth.

    DATATRIEVE does check the record size, however, and generates an error message if the size of the record differs from the record size specified in the FDL file.
  - DATATRIEVE ignores any CONNECT-related options included in the FDL file because the file is being created, not opened.
  - DATATRIEVE dynamically activates the FDLSHR image the first time you request a DEFINE FILE with the USING fdl-file-spec clause. Subsequent requests do not incur the overhead of activating that image.
  - If DATATRIEVE receives any kind of error or warning messages from the FDL parse of the file, DATATRIEVE does not call FDL to create the RMS file. (Note that this includes syntax warnings. If you used FDL outside of DATATRIEVE such warnings might be ignored and the file would be created.) DATATRIEVE instead returns an appropriate error message. The error message may contain the statement number of the line in the FDL file that caused the error if the statement number is returned from FDL. The message will not include any additional information about the error.

    You should therefore debug your FDL file at the DCL level before attempting to use it from within DATATRIEVE.

  For more information on using FDL files to improve the performance of your application, see the *VAX DATATRIEVE User's Guide* and the VMS documentation on File Definition Language and on file applications.
- If you are defining data files for domains defined using the WITH RELATIONSHIPS clause, you should also note the following:
  - If the DEFINE FILE command line contains no arguments other than the domain name, DATATRIEVE checks to see if there is a CDD$FILE_DEFINITION entity already associated with the domain. If DATATRIEVE finds a CDD$FILE_DEFINITION entity, DATATRIEVE uses the RMS file parameters stored in that entity when it defines the file. If DATATRIEVE does not find a CDD$FILE_DEFINITION entity, it creates the data file using the default RMS file parameters and then creates a CDD$FILE_DEFINITION entity to reflect those parameters.

- If the DEFINE FILE command line contains one or more of the arguments listed in the DEFINE FILE syntax (ALLOCATION, SUPERSEDE, MAX, KEY, USING fdl-file-spec) and a CDD$FILE_DEFINITION is already associated with the specified domain, DATATRIEVE compares the arguments with the contents of the CDD$FILE_DEFINITION entity. If the argument you listed was an FDL file specification, DATATRIEVE compares the RMS parameters of the FDL file with the contents of the existing CDD$FILE_DEFINITION entity.

  If the new arguments and the CDD$FILE_DEFINITION entity do not match or if the domain does not own a CDD$FILE_DEFINITION entity, DATATRIEVE creates a CDD$FILE_DEFINITION that reflects the new DEFINE FILE (or FDL) arguments.

  DATATRIEVE also creates a new version of the domain that points to the new CDD$FILE_DEFINITION.

- If the domain is a CDD$DATABASE domain, DATATRIEVE uses the file parameters you specified when defining the RMS_DATABASE through CDO. DATATRIEVE does not update the CDD$FILE_DEFINITION entity of a CDD$DATABASE domain, and ignores any arguments on the DEFINE FILE command line for a CDD$DATABASE domain.

Examples

The following example shows how to define an indexed file for the domain PAYABLES using the field NAME as the primary key and TYPE as the alternate key, and allowing no changes to the alternate key:

```
DTR> DEFINE FILE FOR PHONES KEY=NAME(DUP), KEY=TYPE(NO CHANGE)
DTR>
```

The following example defines a sequential file for the domain FAMILIES:

```
DTR> DEFINE FILE FOR FAMILIES
DTR>
```

The following example shows how to define a new indexed file for the domain YACHTS using the group field TYPE as the primary key, allowing duplicate values for this key. This command replaces the previous data file for YACHT.

```
DTR> DEFINE FILE FOR YACHTS SUPERSEDE, KEY=TYPE (DUP)
DTR>
```

The following example defines a new file for the YACHTS domain using the specifications included in YACHTS.FDL, a file created outside of DATATRIEVE:

```
DTR> DEFINE FILE FOR YACHTS USING YACHTS.FDL
DTR>
```

DEFINE PORT Command

Enters the definition of a DATATRIEVE port in the specified or implied directory of the CDD/Repository data dictionary system and creates an access control list (ACL) for the port.

Format

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

Arguments

path-name
Is the given name, full dictionary path name, or relative dictionary path name of the port being defined. The path name cannot resolve to the full dictionary path name of any other object or directory in the data dictionary system. DEFINE PORT will accept both DMU and CDO style path names.

record-path-name
Is the given name, full dictionary path name, or relative dictionary path name of the record definition to be associated with the port. The dictionary path name of the record cannot resolve to the full dictionary path name of any other object or directory in the data dictionary system. DEFINE PORT will accept both DMU and CDO style path names.

; (semicolon)
Ends the port definition.

Restrictions

- Do not use a DATATRIEVE keyword as the given name of a port or as the given name of the record definition associated with the port.
- You cannot use a DEFINE PORT command as part of a compound statement.
- You cannot invoke a procedure in a port definition.
- You must enter the port definition in the data dictionary before using it to transfer data between DATATRIEVE and your application program.

Results

- If you press the RETURN key before typing the semicolon, DATATRIEVE prompts you with the DFN> prompt to continue the definition. DATATRIEVE continues to prompt you until you type a semicolon and press the RETURN key or until it detects a syntax error. If you make a syntax error while entering the port definition, DATATRIEVE returns to command level (indicated by the DTR> prompt) without entering the port definition in the data dictionary.
- DATATRIEVE enters the port definition into the directory of the data dictionary determined by the dictionary path name you specify with the DEFINE PORT command. DATATRIEVE stores the port definition in the parent directory determined by the full dictionary path name of the port. In the full dictionary path name, the name of the parent directory immediately precedes the given name of the port.
- DATATRIEVE creates a default access control list for the port. The UIC identification matches any UIC ([*,*]), and the user name is set to your current VMS user name.

  If you define your port with a DMU path name, the privileges granted are C (CONTROL), D (LOCAL_DELETE), E (DTR_EXTEND/EXECUTE), H (HISTORY), M (DTR_MODIFY), R (DTR_READ), S (SEE), U (UPDATE), and W (DTR_WRITE). If you define your port with a CDO path name, the privileges granted are U (CHANGE + DEFINE), C (CONTROL), D (DELETE), E (EXTEND), M (MODIFY), R (READ), S (SHOW), and W (WRITE). In either case, the C (CONTROL) privilege allows you to change the access control list to suit your needs.

Usage Notes

- Before you can use the port, you must enter in the data dictionary the record definition associated with the port. Use the DEFINE RECORD command.
- Use the SHOW path-name command to display the definition of the port.
- Use the SHOW DOMAINS command to display the names of all the ports in your default dictionary directory.
- See the *VAX DATATRIEVE Guide to Programming and Customizing* for information about using a port to transfer data between DATATRIEVE and your application program.

Example

The following example defines a port for transferring records between the YACHTS domain and an application program:

```
DTR> DEFINE PORT YPORT USING CDD$TOP.DTR$LIB.DEMO.YACHT;
DTR>
```

DEFINE PROCEDURE Command

Enters a procedure definition into the CDD/Repository data dictionary system and creates an access control list (ACL) for the procedure.

Format

```
DEFINE PROCEDURE   procedure-name

        .
        .
        .

END_PROCEDURE
```

Arguments

procedure-name
Is the given name, full dictionary path name, or relative dictionary path name of the procedure you want to define. The path name cannot resolve to the full dictionary path name of any other object or directory in the data dictionary. Procedures can be defined in either the DMU or CDO dictionary.

END_PROCEDURE
Ends the procedure definition.

Restrictions

- You must enter the DEFINE PROCEDURE command at DATATRIEVE command level (indicated by the DTR> prompt); it cannot be part of a DATATRIEVE statement.
- To define a procedure in the DMU format dictionary, you must have the following access privileges:
  - P (PASS_THRU) access to the ancestors of the procedure
  - P (PASS_THRU) and X (EXTEND) access to the parent directory
- To define a procedure in the CDO format dictionary, you must have the following access privileges:
  - VMS access to the VMS directory
  - S (SHOW) and U (CHANGE) access to the dictionary.

Results

- After you type DEFINE PROCEDURE procedure-name and press the RETURN key, DATATRIEVE displays the DFN> prompt. This prompt indicates that DATATRIEVE expects a procedure definition and that the commands and statements you enter before the END_PROCEDURE clause are included in the procedure.

  Until you end the procedure definition with END_PROCEDURE, DATATRIEVE treats your input as input to the procedure and continues to prompt you with the DFN> prompt.

- DATATRIEVE enters the procedure definition in the dictionary directory determined by the full dictionary path name of the procedure. If you use only the given name of the procedure in the DEFINE command, DATATRIEVE stores the definition in your default dictionary directory.

- DATATRIEVE creates a default access control list entry for the procedure. The UIC matches any UIC ([*,*]), and the user name is set to your current VMS user name.

  If you define your domain with a DMU path name, the privileges granted are C (CONTROL), D (LOCAL_DELETE), E (DTR_EXTEND/EXECUTE), H (HISTORY), M (DTR_MODIFY), R (DTR_READ), S (SEE), U (UPDATE), and W (DTR_WRITE). If you define your domain with a CDO path name, the privileges granted are U (CHANGE + DEFINE), C (CONTROL), D (DELETE), E (EXTEND), M (MODIFY), R (READ), S (SHOW), and W (WRITE). In either case, the C (CONTROL) privilege allows you to change the access control list to suit your needs.

- If you include comments in a procedure (by preceding your input line with an exclamation point), DATATRIEVE stores the comments in the CDD/Repository dictionary as part of the procedure definition. If you include comments in the procedure, DATATRIEVE does not display them when you invoke the procedure and DATATRIEVE executes it.

Usage Notes

- To invoke a procedure, enter a colon (or its synonym, EXECUTE) followed by the given name, full dictionary path name, or relative path name of the procedure. You can invoke a procedure in response to any DATATRIEVE prompts, except those of ADT, Guide Mode, and the editor. You can also invoke procedures in the midst of input lines.

- To invoke a procedure with a VMS command line, you must do the following:

  – Use the keyword EXECUTE instead of the colon.

- Include the EXECUTE command together with the procedure name between double quotation marks.

```
$ DATATRIEVE/INTERFACE=CHARACTER_CELL "EXECUTE TEST_PROCEDURE"
```

- You cannot modify a procedure definition with the DEFINE PROCEDURE command. To change an existing procedure definition, use the EDIT command. The EDIT command places a REDEFINE PROCEDURE command with the old procedure definition into the main buffer of the editor. You can then use the editor to make the desired changes.

  When you exit from the editor, DATATRIEVE places the updated procedure definition with a new version number into the DMU dictionary. If SET EDIT_BACKUP is in effect, the old definition is retained in an earlier version.

  See the section on the EDIT command for more information on changing the definitions of procedures and other dictionary objects.

- You must take care when using a procedure in a loop formed by a REPEAT or FOR statement. To ensure that DATATRIEVE executes all the statements in the procedure each time through the loop, enclose the procedure in a BEGIN-END block. Remember, though, that if you use a procedure in this way, it cannot include any commands or FIND, SELECT, or DROP statements.

- For more information on procedures, see the *VAX DATATRIEVE User's Guide*.

Examples

The following example shows how to define a procedure to set your default directory to the DEMO directory, which contains the sample data for the YACHTS, OWNERS, and FAMILIES domains:

```
DTR> DEFINE PROCEDURE DEMO
DFN> SET DICTIONARY CDD$TOP.DTR$LIB.DEMO
DFN>     SHOW DICTIONARY
DFN> END_PROCEDURE
DTR> :DEMO
The default directory is CDD$TOP.DTR$LIB.DEMO
DTR>
```

The following example shows how to define a procedure that displays a group of boats with a price less than a figure you supply when the procedure runs:

```
DTR> DEFINE PROCEDURE PRICE_LIST
DFN>     READY YACHTS
DFN>     PRINT SKIP, COL 20,
DFN>           '*** Price List of YACHTS ***', SKIP
DFN>     FOR YACHTS WITH PRICE NE 0 AND
DFN>         PRICE LE *.'the ceiling price'
DFN>           PRINT BOAT
DFN>     PRINT SKIP, COL 10, 'See anything interesting?'
DFN> END_PROCEDURE
DTR> :PRICE_LIST

                  *** Price List of YACHTS ***

Enter the ceiling price: 5,000

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   DISPLACEMENT BEAM  PRICE

 CAPE DORY   TYPHOON    SLOOP   19        1,900     06   $4,295
 VENTURE     21         SLOOP   21        1,500     07   $2,823
 VENTURE     222        SLOOP   22        2,000     07   $3,564
 WINDPOWER   IMPULSE    SLOOP   16          650     07   $3,500

         See anything interesting?

DTR>
```

The following example shows how to use a VMS command line to invoke the procedure created in the first example:

```
$ DTR32 EXECUTE DEMO
The default directory is CDD$TOP.DTR$LIB.DEMO

$
```

DEFINE RECORD Command

Enters a record definition in the CDD/Repository data dictionary and creates an access control list (ACL) for the record.

Format

DEFINE RECORD record-path-name [USING] [OPTIMIZE]

```
[ ALLOCATION IS { MAJOR-MINOR           } ]
                { ALIGNED-MAJOR-MINOR   }
                { LEFT-RIGHT            }
```

definition[,...]

;

Arguments

record-path-name

Is the given name, full dictionary path name, or relative dictionary path name of the record being defined. The record path name cannot resolve to the full dictionary path name of any other object or directory in the data dictionary system. DEFINE RECORD will accept both DMU and CDO style path names.

[USING] OPTIMIZE

Allows you to optimize record definitions, reducing the central processing unit (CPU) time needed to ready a domain that refers to the record. See the Usage Notes section for special considerations.

ALLOCATION IS { **MAJOR-MINOR** | **ALIGNED-MAJOR-MINOR** | **LEFT-RIGHT** }

Specifies the type of word-boundary alignment DATATRIEVE uses when storing records in the data file. It also controls the way DATATRIEVE retrieves data from data files created by user programs or other application software. The default allocation is no alignment. See the VAX COBOL documentation set for more information on word-boundary alignment and allocation of fill bytes.

definition

Is the description of the fields in the record. Each definition has one of the following formats:

DEFINE RECORD Command

```
level-number-1   field-name-1.
level-number-2   field-name-2   field-definition-2.
[level-number-n   field-name-n   field-definition-n.]
        .          .          .
        .          .          .
        .          .          .
```

or

```
level-number-n   FROM   { FIELD }   path-name.
                       { GROUP }
```

level-number
Is the level number for the field in the record definition. It indicates the relationship of the field to the other fields in the record definition.

field-name
Is the name of the field. Every field must have a name. The keyword FILLER is a special field name that can be repeated at the same level in the record definition.

field-definition
Is a field definition. A record definition must contain at least one field definition. Elementary fields must have at least one field definition clause, but group fields are not required to have any field definition clauses.

Each field definition must end with a period (.).

FROM
Allows you to create a definition using fields imported from CDO field/record definitions.

FIELD
Specifies that you are referencing an existing CDO field.

GROUP
Specifies that you are referencing an existing CDO group field.

path-name
Specifies the path name of the field or record referenced by a FROM field. The record or field specified by this path name must reside in a CDO format dictionary.

; (semicolon)
Ends the record definition.

Restrictions

- You cannot invoke a procedure in a record definition.
- The level number must be an integer between 1 and 65.
- A record definition must contain the field definition of at least one elementary field.
- The field definition of an elementary field must contain at least one field definition clause.
- The maximum number of fields that VAX DATATRIEVE is able to handle in a CDO format record is 1024.
- No field name can duplicate the domain name.
- Use of the FROM clause is restricted to records defined in the CDO dictionary.
- A FROM field cannot have subordinate fields. A FROM field is treated as an elementary field in the record definition, because any subordinate fields are predefined in the CDD/Repository dictionary.
- A FROM clause definition must end with both a period and a carriage return.
- If you want to use records defined in the CDD/Repository system, make sure that variant clauses containing more than one field are declared as structures.

Results

- If you press the RETURN key before typing the semicolon, DATATRIEVE prompts you with the DFN> prompt to continue the definition. DATATRIEVE continues to prompt you until you type a semicolon and press RETURN or until it detects a syntax error. If you make a syntax error while entering the record definition, DATATRIEVE returns to command level (indicated by the DTR> prompt) without creating the record definition.
- When you end the record definition, DATATRIEVE displays the following message to indicate the length of the new record in bytes:

  ```
  [Record is n bytes long]
  ```

- DATATRIEVE enters the record definition in the dictionary directory determined by the full dictionary path name of the record. If you use only the given name of the record definition in the DEFINE RECORD command, DATATRIEVE stores the definition in your default dictionary directory.

- Directory entries are not made for fields that are part of the record.
- When you specify the OPTIMIZE qualifier, DATATRIEVE stores its internal representation of the record (called the field tree) in the dictionary. This means DATATRIEVE does not have to reconstruct the field tree each time you ready a domain that refers to the record.

  DATATRIEVE constructs a new field tree only when the record is redefined using the OPTIMIZE qualifier. (Note that the DEFINE FILE command also uses record definitions. Its performance will also improve by optimizing records.)

  DATATRIEVE does not perform optimization by default. When defining a new record, you must specify the OPTIMIZE qualifier to optimize a record. To optimize existing record definitions, you must redefine the records (using the EDIT or EXTRACT command) and include the OPTIMIZE qualifier.

  When a new version of DATATRIEVE is installed, you may find that you have to redefine optimized records if you want to continue the benefits of optimization. If the new version of DATATRIEVE requires that the field tree be stored in a different format, then DATATRIEVE may no longer be able to use the field tree stored by previous versions. If this happens, DATATRIEVE ignores the older version field tree when you enter a READY command and displays the following message:

  ```
  Record <"record-name"> uses old record format.  Processing will
  continue, but for optimization you must redefine record.
  ```

  You can restore optimization to your record by redefining the record with the EDIT or EXTRACT command.
- DATATRIEVE creates a default access control list entry for the record definition. The UIC matches any UIC ([*,*]), and the user name is set to your current VMS user name.

  If you define your record with a DMU path name, the privileges granted are C (CONTROL), D (LOCAL_DELETE), E (DTR_EXTEND/EXECUTE), H (HISTORY), M (DTR_MODIFY), R (DTR_READ), S (SEE), U (UPDATE), and W (DTR_WRITE). If you define your record with a CDO path name, the privileges granted are U (CHANGE + DEFINE), C (CONTROL), D (DELETE), E (EXTEND), M (MODIFY), R (READ), S (SHOW), and W (WRITE). In either case, the C (CONTROL) privilege allows you to change the access control list to suit your needs.

Usage Notes

- This command creates a record definition but cannot modify or replace one. To modify a record definition, use the EDIT command.
- The EDIT command places a REDEFINE RECORD command with the old definition in the main buffer of the editor. You can then use the editor to make the desired changes. When you exit from the editor, DATATRIEVE places the updated record definition with a new version number in the data dictionary.

  If SET EDIT_BACKUP is in effect, the old definition is retained in an earlier version. See the section on the EDIT command in this chapter for information on changing the definitions of records, tables, and other dictionary objects.
- If you change a record definition, you may not be able to use old data files. If the new record definition is not the same length or the new field definitions change to data types incompatible with the previous definition, you have to define a new data file.

  The safest method for changing record definitions is to define a new domain and a new file to accompany the new record definition. Then use the STORE statement or the Restructure statement to transfer the values from the data file of the old domain to the data file of the new one.
- The FROM clause forms a relationship between the record being defined and the specific version of the field or record referenced by the FROM clause. If a newer version of the field or record is defined, your record still points to the older version. You can use the EDIT command to change the version of the field being referenced by the FROM clause.
- Consider the following performance and storage tradeoffs before using the OPTIMIZE qualifier:
  - The major benefit of the OPTIMIZE qualifier is the decrease in CPU time when readying a domain with an optimized record. On sample records, CPU times decreased anywhere from 50 to 95 percent. Larger records showed the greater improvement.
  - Using the OPTIMIZE qualifier increases the CPU time necessary to define a record. The elapsed CPU time for a DEFINE RECORD command increases anywhere from a few percentage points to nearly double the time. The smaller percentage increases occur for small records. Larger records cause the larger percentage increases in CPU time.

You can avoid increased record definition time by not using the OPTIMIZE qualifier while designing a record. Instead, edit the final version of the record and add the OPTIMIZE qualifier. This way you still benefit from the READY performance improvements.

Note also that the increase in definition time is essentially a one-time occurrence. Once you define your record, you experience the improved performance each time you ready a domain that uses that record.

– When the record is optimized, the space used by the record definition in the dictionary may increase.

The sample record definition ACCOUNT_BALANCES_REC (located in CDD$TOP.DTR$LIB.DEMO) uses the new OPTIMIZE qualifier, allowing you to see the performance improvements and tradeoffs involved in using OPTIMIZE.

Field Definition Clauses

When you define a field, you specify its characteristics with one or more field definition clauses. A field definition clause consists of a keyword (such as PICTURE or QUERY_NAME) followed by a character string or value expression. Field definition clauses specify the following characteristics of fields:

- The class and length of the data and the format in which the data is stored (PICTURE, USAGE, COMPUTED BY, OCCURS)
- The format used when DATATRIEVE writes the data to a file or output device (EDIT_STRING, QUERY_HEADER, SIGN)
- The values accepted when you store data in the field (VALID IF, DEFAULT VALUE, MISSING VALUE)
- The way DATATRIEVE computes numeric values when you refer to the field (USAGE, SCALE, COMPUTED BY, MISSING)
- An alternate and equivalent name for the field (QUERY_NAME)
- An alternate way to define another field in the record (REDEFINES)

You can also use a FROM clause to include previously defined CDO field level definitions in a DATATRIEVE record definition.

When you write DATATRIEVE record definitions, you can choose from the field definition clauses summarized in Table 4–6.

Table 4–6 Summary of Field Definition Clauses

Clause	Valid For	Purpose
COMPUTED BY	Elementary field	Describes a COMPUTED BY field.
DEFAULT VALUE	Elementary field	Specifies a value stored in the field if you do not enter a value when creating the record.
EDIT_STRING	Elementary field	Specifies the format of a value when DATATRIEVE writes a field value to a file or output device.
MISSING VALUE	Elementary field	Specifies a numeric or character-string literal denoting that no value is stored in the field. This causes DATATRIEVE to ignore a record with a MISSING value in a field when calculating statistical functions using values in that field.
OCCURS	Elementary or group field	Defines multiple occurrences of a field or group of fields.
PICTURE	Elementary field	Specifies the data type, length, and format of values stored in the field.
QUERY_HEADER	Elementary field	Specifies the column header for a field when DATATRIEVE writes one or more field values to a file or output device.
QUERY_NAME	Elementary or group field	Specifies an alternate name for a field.
REDEFINES	Elementary or group field	Creates an alternate definition for a field.
SCALE	Numeric elementary field	Specifies a scaling factor, a positive or negative integer indicating the power of 10 that determines the number of significant digits of an input value stored as a value in the field.

(continued on next page)

Table 4–6 (Cont.) Summary of Field Definition Clauses

Clause	Valid For	Purpose
SIGN	Numeric elementary field	Specifies the location and representation of the sign in a numeric field.
SYNCHRONIZED	Elementary field	Forces word boundary SYNC alignment according to data types when MAJOR_MINOR alignment is in effect.
USAGE	Numeric or date elementary field	Specifies the length and format of a numeric field or specifies a date field.
VALID IF	Elementary field	Tests a value against conditions specified in the Boolean expression before storing or modifying the value in the field.

Guide for Using Field Definition Clauses

When you write field definitions, observe these rules and guidelines for using field definition clauses:

- The definition of a group field must contain at least a level number and a field name. It can also contain one or more field definition clauses. DATATRIEVE ignores any PICTURE or USAGE clauses you include in a group field definition.
- A definition of an elementary field must contain a PICTURE, COMPUTED BY, or USAGE clause.
- If you use a PICTURE clause in a field definition, DATATRIEVE treats it as an edit string if no EDIT_STRING clause is present.
- If you use the USAGE clauses BYTE, WORD, LONG, QUAD, COMP (or INTEGER), or COMP-2 (or DOUBLE), then you do not need a PICTURE clause. The other USAGE clauses, COMP-3 (or PACKED), COMP-5 (or ZONED), and DISPLAY, require a PICTURE clause.
- If you use a COMPUTED BY clause to define a field, DATATRIEVE ignores any USAGE clause in your definition and treats a PICTURE clause as an edit string if no EDIT_STRING clause is present.

- You must end each field definition with a period. If the field is a group field with no field definition clause, place the period immediately after the field name.
- If the field definition contains one or more clauses, place the period after the last clause.
- You can put one or more field definition clauses on the same input line as the level number and field name.
- You can also put each field definition clause on one or more input lines.
- You can enter the clauses of a field definition in any order.
- Separate each field definition clause from the next by entering a space, a tab, or a carriage return.
- Indenting field definition clauses with spaces or tabs, or entering them on separate lines, does not change the characteristics of the field, but this practice makes your record definition easy to read.

Examples

The following example defines the record PHONE_REC:

```
DTR> DEFINE RECORD PHONE_REC USING
DFN> 01 PHONE.
DFN>    02 NAME PIC X(20).
DFN>    02 NUMBER PIC 9(7) EDIT_STRING IS XXX-XXXX.
DFN>    02 LOCATION PIC X(9).
DFN>    02 DEPARTMENT PIC XX.
DFN> ;
[Record is 38 bytes long.]
DTR>
```

The following example defines the record FAMILY:

```
DTR> DEFINE RECORD FAMILY USING
DFN> 01 FAMILY.
DFN>    03 PARENTS.
DFN>        06 FATHER PIC X(10).
DFN>        06 MOTHER PIC X(10).
DFN>    03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
DFN>    03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
DFN>        06 EACH_KID.
DFN>            09 KID_NAME PIC X(10) QUERY_NAME IS KID.
DFN>            09 AGE PIC 99 EDIT_STRING IS Z9.
DFN> ;
[Record is 142 bytes long.]
DTR>
```

The following example defines the record ACCOUNT_BALANCE_REC using the OPTIMIZE qualifier. Note that the USING clause is optional.

```
DTR> DEFINE RECORD ACCOUNT_BALANCE_REC USING OPTIMIZE
     .
     .
     .
DFN> ;
DTR>
```

The following example defines the record YACHT_REC in the CDO format dictionary using fields from a CDO dictionary. Note that you can mix fields from an existing CDO dictionary and fields private to DATATRIEVE in the same record definition. Field BEAM uses CDD$DEFAULT.BEAM as a path. Fields LOA and PRICE are defined only for this record.

```
DTR> DEFINE RECORD YACHT_REC USING
DFN> 01 BOAT.
DFN>    03 FROM GROUP SYS$COMMON:[CDDPLUS]DTR32.TYPE.
DFN>    03 SPECIFICATIONS.
DFN>        06 LOA PIC 99.
DFN>        06 FROM FIELD BEAM.
DFN>        06 PRICE PIC 999999.
DFN> ;
DTR>
```

DEFINE TABLE Command

Enters the definition of a dictionary or domain table in the CDD/Repository data dictionary and creates an access control list (ACL) for the table. The following sections explain how to define dictionary and domain tables.

Format

To define a dictionary table use the following syntax:

```
DEFINE TABLE path-name

  [QUERY_HEADER [IS] "header-segment"[/...]]

  [EDIT_STRING [IS] edit-string]

  [USING] code-field :  translation-field [,]
  { "code-1" }: { "translation-1" }[,]
  { code-1   }  { translation-1   }

  [ { "code-2" }: { "translation-2" } ][,]
  [ { code-2   }  { translation-2   } ]

   .                    .
   .                    .
   .                    .

  [ ELSE { "translation-n" } ]
  [      { translation-n   } ]

END_TABLE
```

To define a domain table use the following syntax:

```
DEFINE TABLE    path-name FROM [DOMAIN] domain-path-name

  [QUERY_HEADER [IS] "header-segment"[/...]]

  [EDIT_STRING [IS] edit-string]

  [USING] code-field :  translation-field [,]
  [ ELSE { "translation-string" } ]
  [      { translation-string   } ]
```

END_TABLE

Arguments

path-name
Is the given name, full dictionary path name, or relative dictionary path name of the dictionary table being defined. The full dictionary path name of the table cannot resolve to the full dictionary path name of any other object or directory in the data dictionary system. The DEFINE TABLE command accepts both DMU or CDO style path names.

"code" : "translation"
code : translation
Is a code-and-translation pair. You must separate each pair with a colon. The comma after each pair is optional. If the code or translation conforms to the rules for DATATRIEVE names given in the *VAX DATATRIEVE User's Guide*, you do not have to enclose it in quotation marks. However, DATATRIEVE converts to uppercase any lowercase letters in an unquoted code or translation.

If the code or translation does not conform to the rules for DATATRIEVE names (especially if it contains any spaces), or if you want to preserve lowercase letters, you must enclose the code or translation in quotation marks (" ") and follow the rules for character string literals. (See Chapter 1.)

ELSE "translation"
translation
Is the translation to be used if you specify a code not defined in the dictionary table. The rules for specifying this translation string are the same as those for codes and translations. (See Chapter 1.)

END_TABLE
Ends the dictionary table definition.

Restrictions

- You cannot include the invocation of a procedure (:procedure-name) in a table definition.
- The given name of a table cannot duplicate a DATATRIEVE keyword.
- If, in different directories, the data dictionary contains two tables with the same given name, you cannot have both tables in your workspace at the same time. After you refer to one of them with an IN clause or a VIA clause, you cannot use the second one until you remove the first from your workspace with a RELEASE command.

- The total length of all the code-and-translation pairs in a single dictionary table cannot exceed 63,000 bytes.
- To define a domain table stored in the DMU dictionary, you need the following privileges:
  - P (PASS_THRU) to the parent directory of the domain table definition
  - P (PASS_THRU), S (SEE), and E (DTR_EXTEND/EXECUTE) privileges to the domain table definition
  - P (PASS_THRU), S (SEE), and R (DTR_READ) privileges to the domain containing the code and translation fields
  - P (PASS_THRU), S (SEE), and E (DTR_EXTEND/EXECUTE) privileges to the record definition associated with the domain containing the code and translation fields
- To access a domain table stored in the CDO dictionary, you need the following privileges:
  - VMS access to the VMS directory of the anchor
  - S (SHOW) and U (CHANGE) access to the parent directory
  - S (SHOW) and R (READ) access to the domain and record definitions
- Do not bring a domain table and a domain with the same given names into your workspace at the same time.
- Do not ready a domain with an alias that is the same as the given name of a domain table in your workspace.

Results

- If you press the RETURN key before entering the END_TABLE argument, DATATRIEVE prompts with the DFN> prompt to continue. DATATRIEVE continues to prompt until you end the definition with the END_TABLE argument.
- DATATRIEVE enters the definition of the table in the dictionary directory determined by the full dictionary path name of the table. If you use only the given name of the table with the DEFINE command, DATATRIEVE enters the definition in your default directory.
- DATATRIEVE creates a default access control list entry for the table. The UIC matches any UIC ([*,*]), and the user name is set to your current VMS user name.

If you define your domain with a DMU path name, the privileges granted are C (CONTROL), D (LOCAL_DELETE), E (DTR_EXTEND/EXECUTE), H (HISTORY), M (DTR_MODIFY), R (DTR_READ), S (SEE), U (UPDATE), and W (DTR_WRITE). If you define your domain with a CDO path name, the privileges granted are U (CHANGE + DEFINE), C (CONTROL), D (DELETE), E (EXTEND), M (MODIFY), R (READ), S (SHOW), and W (WRITE). In either case, the C (CONTROL) privilege allows you to change the access control list to suit your needs.

- Values coming from a table are always converted to the string computational class. In most cases this causes no problems because the value may be converted to another class. But a problem may occur when you try to interpret as a date class a value computed via table that looks like a date but, since it comes from a table, is a string (i.e. "01-JAN-1991"). The result of this action will be random output. To obviate the problem, use the function FN$DATE to convert the string coming from a table to a date class. For instance, in a computation, replace the string constant "01-JAN-1991" with the date expression FN$DATE("01-JAN-1991").

Usage Notes

- When you invoke a dictionary table with an IN or VIA clause, the table is loaded into your workspace and remains available to you until you remove it from your workspace.
- The IN and VIA operators are case sensitive. DATATRIEVE finds the values on the table only if they agree with the case of the characters.
- Use the SHOW READY command to display the names of the tables loaded in your work space.
- A table loaded in your workspace remains available to you even if you change default dictionary directories.
- To remove a dictionary table from your workspace, use the RELEASE table-name command. To remove a domain table from your workspace, use either the RELEASE table-name command or the FINISH table-name command. A FINISH ALL command also removes all domain tables from your workspace.
- DATATRIEVE uses a default edit string of 10 characters when it displays the translation string or writes the value to another output device. You can specify an edit string each time you use a VIA value expression. With the EDIT_STRING clause in the table definition, you can also give one edit string to apply to all the translations.

If you do not use the EDIT_STRING clause in the table definition, DATATRIEVE uses the edit string of the code field to format its output of the translation field only if there is no ELSE clause in the domain table definition. If the domain table definition contains an ELSE clause, DATATRIEVE assigns an edit string of X(n), where n is the number of characters in either the edit string of the translation field or the ELSE translation string, whichever is longer.

- You can specify a column header each time you use a VIA value expression. You can also specify, with the QUERY_HEADER clause in the table definition, one query header to apply to all the translations.
- When you specify a query header for a VIA value expression, you must enclose the entire expression in parentheses for the query header to take effect.
- DATATRIEVE does not use the query header in the field definition of the translation field. To specify a query header for the translation field, use the QUERY_HEADER clause in the command when defining the domain table.
- The definition of a dictionary table differs from the definitions of domains and records because it contains values, not just a data description.
- To modify an existing definition of a table use the EDIT command. DATATRIEVE loads the table definition into the main buffer of the editor. Then you edit the definition, modifying any codes and translations. When you exit from the editor, DATATRIEVE stores the updated definition with a new version number in the data dictionary.If SET EDIT_BACKUP is in effect, the old definition is retained in an earlier version.
- If the data file associated with a domain table contains duplicate values for a code field specified in a VIA value expression, DATATRIEVE outputs the value of the translation field corresponding to the first code field it encounters as it searches the data file. The order of records for the search is determined by the physical order or records in sequential data files and by the sequence of values in the key fields of indexed records.
- You can replace the definition of a domain table by deleting it from the data dictionary with the DELETE command and entering another DEFINE TABLE command.

  To change individual codes or translations associated with a domain table, you can use the DATATRIEVE MODIFY and ERASE statements on the records in the domain containing the values for the code and translation fields. Use the READY command to get access to the domain containing those fields and change the field values the way you would those of any other domain.

- You cannot change the value of a code or translation field if the field is the primary key of an indexed file.
- When you define a domain table using the given name of the associated domain, DATATRIEVE stores the full dictionary path name of the domain as an attribute of the table. If you copy the table to another part of the CDD/Repository data dictionary, the definition still points to the old data dictionary location for the domain. Therefore, you must redefine the table if you move it.
- For more information on using tables, see the *VAX DATATRIEVE User's Guide*.

Examples

The following example defines a table of department codes and specify a query header for the translations of the table:

```
DTR> DEFINE TABLE DEPT_TABLE
DFN> QUERY_HEADER IS "Responsible"/"Department"
DFN> CE : "Commercial Engineering"
DFN> PE : "Plant Engineering"
DFN> CS : "Customer Support"
DFN> RD : "Research and Development"
DFN> SD : "Sales Department"
DFN> ELSE "UNKNOWN DEPARTMENT"
DFN> END_TABLE
DTR>
```

The following example defines a table with a translation for each possible rig and include an edit string in the definition that displays the translation in a 10 character wide column:

```
DTR> DEFINE TABLE RIGGING
DFN> EDIT_STRING IS T(10)
DFN> QUERY_HEADER "TYPE OF"/"RIGGING"
DFN> SLOOP : "ONE MAST"
DFN> KETCH : "TWO MASTS, BIG ONE IN FRONT"
DFN> YAWL  : "SIMILAR TO KETCH"
DFN> MS    : "SAILS AND A BIG MOTOR"
DFN> ELSE    "SOMETHING ELSE"
DFN> END_TABLE
DTR> PRINT "KETCH" VIA RIGGING

 TYPE OF
 RIGGING

TWO MASTS,
BIG ONE IN
FRONT
```

```
DTR>
```

The following example shows how to define a domain table that returns the price of a yacht when you enter a value for LENGTH_OVER_ALL. The example specifies a query header and an edit string for the translation field:

```
DTR> DEFINE TABLE LOA_PRICE_TABLE
DFN>     FROM YACHTS
DFN>     QUERY_HEADER IS "SAMPLE"/"PRICE"
DFN>     EDIT_STRING IS $$$,$$$
DFN>     USING LOA : PRICE
DFN>     ELSE "NO BOATS IN STOCK WITH THAT LOA."
DFN> END_TABLE
DTR> PRINT 26 VIA LOA_PRICE_TABLE

SAMPLE
PRICE

$17,900

DTR>
```

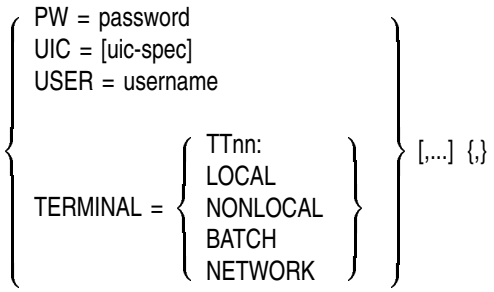
See the *VAX DATATRIEVE User's Guide* for further examples of the definition and use of domain tables.

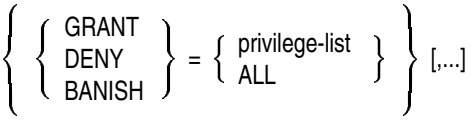
DEFINEP Command

Adds an entry to the access control list (ACL) for a dictionary object or dictionary directory.

Format

DEFINEP [FOR] path-name sequence-number [,]





Arguments

path-name

Is the given name, full dictionary path name, or relative dictionary path name of the dictionary object or dictionary directory whose ACL list you want to change. DEFINEP accepts both DMU and CDO style path names.

sequence-number

Is the sequence number of the entry to be added to the ACL. This number must be an unsigned, nonzero integer.

PW = password

Specifies a password to be appended to the given name of the dictionary object or dictionary directory when used alone in a command or statement or as part of a full or relative dictionary path name. You can specify a password in an ACL entry on a directory or object either in the DMU or in the CDO format dictionary.

UIC = [uic-spec]
Specifies the UIC or group of UICs to which the added ACL entry applies. The UIC specification must be enclosed in square brackets and must conform to the VMS rules for specifying UICs (see the VMS documentation set). You can specify numeric and alphanumeric UICs and rights identifiers. (A rights identifier is a single text string enclosed in brackets. The system manager defines a rights identifier in the system rights database. The identifier indicates all members of a particular group.)

USER = username
Specifies the VMS user name to which the added ACL entry applies. Do not put the user name in parentheses or brackets.

TERMINAL = **TTnn:** | **LOCAL** | **NONLOCAL** | **BATCH** | **NETWORK**

Specifies a particular terminal or a type of terminal to which the added ACL entry applies.

- TTnn: is the number of a specific terminal line to which the added ACL entry applies. You can specify a particular terminal only in ACL entries in the DMU format dictionary.
- LOCAL specifies that the added ACL entry applies to all terminals hard-wired to your local system.
- NONLOCAL specifies that the added ACL entry applies to the local system's dial-up terminal lines, to batch jobs on the local system, to remote terminals logged in to the system by DECnet, and to processes initiated by a DATATRIEVE Distributed Data Manipulation Facility on a remote node in a network of VAX computers.
- BATCH specifies that the added ACL entry applies to all batch jobs run on the local system.
- NETWORK specifies that the added ACL entry applies to all processes initiated by a DATATRIEVE Distributed Data Manipulation Facility on a remote node in a network of VAX computers.

, (comma)
Separates user identification criteria and privilege specifications.

GRANT
Specifies the privileges granted by the added ACL entry.

DENY
Specifies the privileges denied by the added ACL entry.

BANISH
Specifies, for a dictionary directory and all its descendants, the access privileges that the entry denies and the privileges that no ACL of any of the descendants can grant. The BANISH clause is valid in ACL entries either in the DMU or in the CDO format dictionary.

privilege-list
Is a letter or string of letters, each one of which is the abbreviation for the access privilege granted, denied, or banished by the added ACL entry.

See the chapter on ACL in the *VAX DATATRIEVE User's Guide* for more information on access privileges.

Restrictions

- To define an entry in an ACL on an object in the DMU format dictionary, you must have P (PASS_THRU) access to the parent of the dictionary object or directory to which the ACL applies. You must also have P (PASS_THRU) and C (CONTROL) access privileges to the dictionary object.
- To define an entry in an ACL on an object in the CDO format dictionary, you must have S (SHOW) and U (CHANGE) access to the dictionary. You must also have S (SHOW), U (CHANGE + DEFINE), and C (CONTROL) access to the object.
- With a DEFINEP command, you must specify at least one user identification criterion (PW=, USER=, UIC=, or TERMINAL=). You can specify one of each in an ACL entry, but you cannot specify two criteria of the same type. For example, in a DMU ACL entry you can specify a password, a user name, a UIC, and a terminal type, but you cannot specify two passwords. Likewise, in a CDO ACL entry you can specify a user name, a UIC, and a terminal type, but you cannot specify two UICs.
- With a DEFINEP command, you must enter at least one privilege specification (GRANT=, DENY=, or BANISH=). You can enter one of each in an ACL entry, but you cannot enter two criteria of the same type. For example, in a DMU ACL entry you can enter a GRANT=, a DENY=, and BANISH=, but you cannot enter more than one GRANT=. Likewise, in a CDO ACL entry you can enter a GRANT= and a DENY=, but you cannot enter more than one GRANT=.

Results

- DATATRIEVE creates an entry in the ACL. Depending on the sequence number you supply, DATATRIEVE may change the sequence number of other ACL entries:
  - If the sequence number already exists in the ACL, DATATRIEVE adds the entry immediately before the existing entry with the same number. DATATRIEVE then increments by 1 the sequence number of all entries after the new entry.
  - If the sequence number you supply is greater than the last sequence number plus 1, DATATRIEVE ignores your sequence number and adds the entry to the end of the list. Its sequence number becomes the next sequential number in the list.
- The following results apply only to ACLs in the DMU format dictionary:
  - Privileges are cumulative. A privilege granted by the ACL of a dictionary directory is also granted for all its descendants unless that privilege is explicitly denied or banished in the ACL of a particular descendant. If a GRANT clause includes a privilege the user had to the parent directory, the data dictionary takes no action regarding that privilege.
  - A privilege denied by the ACL of a dictionary directory is also denied for all the descendants of that directory unless the ACL of a descendant explicitly grants the privilege. If a DENY clause includes a privilege the user did not have to the parent directory, the data dictionary takes no action regarding that privilege.
  - A privilege banished by the ACL of a dictionary directory can never be granted by the ACL of any descendant of that directory, even if the descendant's ACL explicitly grants the privilege. If a GRANT clause or a DENY clause includes a privilege banished by the ACL of an ancestor, the data dictionary takes no action regarding that privilege.
- You can specify more than one privilege by including a string of letters (such as RWM for READ, WRITE, and MODIFY access). Do not put spaces between letters in a string of letters. You can enter the letters in any order.
- You can specify all of the DATATRIEVE and CDD/Repository privileges by using the keyword ALL in place of a string of letters. If you grant some privileges with the GRANT clause but specify ALL in the DENY or BANISH clause, no privileges are granted by the ACL entry. The DENY and BANISH clauses take precedence over the GRANT clause when you include the same privilege abbreviation in more than one clause in the same ACL entry.

- If the ACL of a DMU format dictionary object or directory is empty, you have the same privileges to that object or directory as you have to its parent directory. If the ACL of a CDO format dictionary object or directory is empty, you have all privileges to the dictionary or object.
- If the ACLs of all the ancestors of a DMU format dictionary object or directory are empty, then all users have all privileges to the object.

Usage Notes

- When designing an ACL, put the entries with the most specific user identification criteria at the top of the list. Put the entries with the most general user identification criteria at the bottom of the list. When you access a dictionary object or directory, CDD/Repository begins at the top of the list and applies the first entry in which all the user identification criteria apply to you.

  For example, the first ACL entry specifies a terminal line number as the only user identification criterion in the entry, and you are using that terminal line. You match all the user identification criteria for that entry. Neither your UIC or user name nor any password you supply matters.

  If the first ACL entry with the terminal number is the first one in the ACL that matches you, the privilege specifications in that entry apply to you. That you also match other entries in the ACL is of no consequence; your other user identification characteristics would not get checked by other ACL entries until you used another terminal to access the dictionary object or directory in question.
- When you specify a rights identifier with the DEFINEP command, you can use only identifiers that are currently in the system rights database. Your system manager can check valid identifiers using the Authorize Utility:

  ```
  $ RUN AUTHORIZE
  UAF> SHOW/ID INVENTORY
  ```
- You can enter user identification criteria and privilege specifications in any order. You need not put the user identification criteria before the privilege specifications.
- To avoid errors when making an addition, display a copy of the current ACL with the SHOWP command before issuing the DEFINEP command. See the section in this chapter on the SHOWP command for more information. Because DATATRIEVE can change sequence numbers, a new entry can affect the numbering of other table entries.
- To remove an entry from an ACL, use the DELETEP command.

- To display the privileges you have for a dictionary object or directory, use the SHOW PRIVILEGES command.
- The chapter on ACL in the *VAX DATATRIEVE User's Guide* discusses the use of ACLs.

Example

The following example defines an ACL entry for a DMU format dictionary directory that uses all the user identification criteria and all the privilege specifications:

```
DTR> DEFINEP FOR MONTHLY_DATA 1 PW = "SECRET", USER = JONES,
[Looking for define privilege option]
CON>    UIC = [240,240], TERMINAL = NETWORK, GRANT = PSRWME,
[Looking for define privilege option]
CON>    DENY = CDUXH, BANISH = FG
DTR>
```

DELETE Command

Deletes one or more dictionary objects and their access control lists from the CDD/Repository data dictionary system.

Format

```
DELETE  path-name-1  [,...] ;
```

Arguments

path-name
Is the given name, full dictionary path name, or relative path name of the dictionary object you want to remove from the data dictionary system. DELETE accepts both DMU and CDO style path names. If you specify more than one dictionary path name, separate each path name from the next with a comma.

; (semicolon)
Ends the DELETE command.

Restrictions

- To delete a DMU dictionary object, you must have the following access privileges:
  - P (PASS_THRU) access to the ancestors of the dictionary object
  - P (PASS_THRU) and X (EXTEND) access to the parent directory of the object
  - P (PASS_THRU) and either D (LOCAL_DELETE) *or* G (GLOBAL_ DELETE) access to the object
- To delete a CDO dictionary object, you must have the following access privileges:
  - S (SHOW) and U (CHANGE) access to the dictionary containing the object
  - S (SHOW) and D (DELETE) access to the object
- You can delete dictionary objects in directories other than your default directory, but you must have the appropriate access privileges to the object, the directory containing the object, and, for objects in the DMU format dictionary, to the ancestors of the object.
- If you do not explicitly include a version number in the path name, DATATRIEVE deletes the highest version of the object.

- You cannot delete a dictionary directory with the DELETE command, even if the directory is empty. To delete a directory from the data dictionary, you must use the DELETE command of the Dictionary Management Utility (DMU) for DMU directories or the DELETE command of the Common Dictionary Operator Utility (CDO) for CDO directories. See the VAX CDD/Repository documentation for an explanation of how to delete dictionary directories.

Results

- If you specify the dictionary path name of a domain definition or a record definition with the DELETE command, DATATRIEVE deletes the highest or the specified version of the *definition* of the domain or record; the associated data file and its contents are unaffected.

  A readied domain in your workspace is not affected by the deletion of the domain definition or the definition of its associated record. However, after you finish a domain whose definition or record definition has been deleted from the data dictionary, you cannot ready the domain again.
- If the dictionary object you specify with a DELETE command is a procedure, DATATRIEVE deletes the procedure itself from the data dictionary. You cannot invoke a procedure after it has been deleted from the DMU data dictionary.
- If the dictionary object you specify with a DELETE command is a dictionary table, DATATRIEVE deletes the dictionary table itself from the data dictionary.
- If the dictionary object you specify in a DELETE command is the definition of a domain table, DATATRIEVE deletes the definition of the domain table from the data dictionary; the definitions of the associated domain and record are unaffected, and the associated data file and its contents are unaffected.
- A dictionary or domain table loaded in your workspace remains there until you release it, even if you delete its definition while the table is loaded in your workspace.
- When DATATRIEVE deletes a dictionary object, it also deletes from the data dictionary the ACL associated with the object.
- If you enter the names of more than one dictionary object with a DELETE command, DATATRIEVE deletes the objects in the order you specify in the command.

Usage Notes

- Be sure to specify the version number of the object you want to delete. If you do not specify a version number in the path name, DATATRIEVE deletes the highest version number of the specified object.
- You must include a semicolon at the end of the DELETE command line, regardless of whether you specify a version number:
  - If you specify a version number, the command will have two semicolons. For example:

    ```
    DTR> DELETE PERSONNEL;1;
    ```

  - If you do not specify a version number, the command will have one semicolon and DATATRIEVE will delete the version with the highest version number. For example:

    ```
    DTR> DELETE PERSONNEL;
    ```

- You cannot delete an object if it is a member of a relationship with another dictionary object. The owner of a relationship must be deleted before the member of a relationship. In the case of a domain, the domain must be deleted before an object owned by the domain (a record, database, or another domain) is deleted.

Examples

The following example shows how to delete two domain versions from your default dictionary:

```
DTR> SHOW DOMAINS
Domains:
  * YACHTS;4  * YACHTS;3  * YACHTS;2  * YACHTS;1
```

The following example does not specify a version number, so DATATRIEVE deletes the highest version, YACHTS;4.

```
DTR> DELETE YACHTS;
DTR> SHOW DOMAINS
Domains:
  * YACHTS;3  * YACHTS;2  * YACHTS;1

DTR> DELETE YACHTS;2;
DTR> SHOW DOMAINS
Domains:
  * YACHTS;3  * YACHTS;1
```

DELETEP Command

Deletes an entry from the access control list (ACL) of an object or directory in the data dictionary system.

Format

```
DELETEP  path-name  sequence-number
```

Arguments

path-name

Is the dictionary path name of the object or directory whose ACL you want to change. DELETEP accepts both DMU and CDO style path names.

sequence-number

Is a nonzero integer indicating the entry's position in the ACL.

Restrictions

- To delete an entry from a DMU format ACL you must have the following access privileges:
  - P (PASS_THRU) access to the parent directory of the object or directory whose ACL you want to change
  - P (PASS_THRU) and C (CONTROL) access to the dictionary object or directory whose ACL you want to change
- To delete an entry from a CDO format ACL you must have the following access privileges:
  - S (SHOW) and U (CHANGE) access to the dictionary containing the object whose ACL you want to change
  - S (SHOW), U (CHANGE + DEFINE), and C (CONTROL) access to the dictionary object or directory whose ACL you want to change
- You must enter one DELETEP command for each ACL entry you want to delete. You cannot delete more than one ACL entry with one DELETEP command.
- There must be an entry with the sequence number you specify.

Results

- DATATRIEVE deletes the entry with the specified sequence number from the ACL of the object or directory.
- If the sequence number you specify is greater than the number of entries in the ACL, DATATRIEVE displays the following message from CDD/Repository .

```
DTR> DELETEP YACHTS 26
%CDD-E-ACLNOTFND, Access control list entry not found
DTR>
```

- When you remove an entry from any position in the ACL, except the last one, DATATRIEVE renumbers the remaining entries so that the sequence numbers begin at 1 and increase in steps of 1.

Usage Notes

- To ensure that you delete the correct entry, display the ACL with the SHOWP command before entering the DELETEP command.
- If you need to remove many entries from a long ACL, begin with the entries at the bottom of the list and work your way toward the top. This method preserves the original sequence numbers of the entries you want to remove. If you start removing entries at the top of the list, every time you remove an entry, the numbers of all the other ones you want to remove change.
- Do not delete the one ACL entry whose user identification criteria identify all users. Users not identified by an entry in a DMU ACL have all the privileges to the object or directory that they have to the parent directory. Users not identified by an entry in a CDO ACL have all privileges to the object or directory.

Example

The following example shows the ACL of the DMU YACHTS domain and deletes an entry from it:

```
DTR> SHOWP YACHTS
  1:    [*,*], Username: "STARKEY"
        Grant - CDEFGHMPRSUWX, Deny - none, Banish - none
  2:    [*,*], Username: "DUNCAN"
        Grant - EHMPRSUW, Deny - CDFGX, Banish - none
  3:    [*,*], Username: "HARRISON"
        Grant - CDEFGHMPRSUWX, Deny - none, Banish - none
  4:    [*,*]
        Grant - none, Deny - CDEFGHMPRSUWX,  Banish - none
```

```
DTR> DELETEP YACHTS 2
DTR> SHOWP YACHTS
  1:    [*,*], Username: "STARKEY"
        Grant - CDEFGHMPRSUWX, Deny - none, Banish - none
  2:    [*,*], Username: "HARRISON"
        Grant - CDEFGHMPRSUWX, Deny - none, Banish - none
  3:    [*,*]
        Grant - none, Deny - CDEFGHMPRSUWX,  Banish - none

DTR>
```

See the chapter on ACL in the *VAX DATATRIEVE User's Guide* for other examples of working with ACLs.

DISCONNECT Statement

Removes records from the sets you specify in the TO list of the CONNECT statement. The DISCONNECT statement can be used only for sets in which VAX DBMS retention is optional.

Format

DISCONNECT context-name-1 [FROM] set-name-1 [,...]

Arguments

context-name-1

Is the name of a valid context variable or the name of a collection with a selected record. The target record must be a member of the specified sets.

set-name

Is the name of a VAX DBMS set.

Result

The record is removed from the VAX DBMS set you specify.

Examples

The following example removes a part with a specified PART_ID from its membership in the set ALL_PARTS_ACTIVE:

```
DTR> FOR P IN PART WITH PART_ID = "TU4722AS"
CON>     DISCONNECT P FROM ALL_PARTS_ACTIVE
DTR>
```

The following example removes the group named PLANT ENGINEERING from the set MANAGES:

```
DTR> FIND GROUPS WITH GROUP_NAME = "PLANT ENGINEERING"
DTR> SELECT
DTR> DISCONNECT CURRENT FROM MANAGES
DTR>
```

DISPLAY Statement

Displays the value of a single DATATRIEVE value expression. The value displayed is not formatted by any edit string associated with the value expression.

Format

DISPLAY value-expression

Argument

value-expression
Is a DATATRIEVE value expression.

Restrictions

- To display the value of a field in a record, the domain containing that record must be readied for READ, WRITE, or MODIFY access. If the domain is readied for EXTEND access, DATATRIEVE displays an error message when you try to specify a field name from that domain in the DISPLAY statement.
- If you specify a field name as the value expression in a DISPLAY statement, you must establish a valid context for the record or records containing the values you want to display. For a discussion of DATATRIEVE context, see the *VAX DATATRIEVE User's Guide*.

Results

- DATATRIEVE displays the current value of the specified value expression.
- If the value expression has an edit string associated with it, the DISPLAY command ignores the edit string when displaying the value.
- If the value expression you specify is a group field, the DISPLAY command concatenates the values of the elementary fields and leaves no spaces between fields except the blanks that pad character fields.
- The value DATATRIEVE displays is not affected by the COLUMNS_PAGE setting. See the section in this chapter on the SET command for more information.

Examples

The following example shows how to declare a numeric variable with a money edit string, give it a value, and use both the PRINT statement and the DISPLAY statement to display that value:

```
DTR> DECLARE SALARY PIC Z(5)9V99 EDIT_STRING $$$$$$.99.
DTR> SALARY = 15753.67
DTR> PRINT SALARY

 SALARY

$15753.67

DTR> DISPLAY SALARY
DISPLAY: 15753.67
DTR>
```

The following example redeclares the above variable as a character variable, assigns a new value, and displays that value:

```
DTR> DECLARE SALARY PIC X(15).
DTR> SALARY = "MUCH TOO LOW"
DTR> PRINT SALARY

   SALARY

MUCH TOO LOW

DTR> DISPLAY SALARY
DISPLAY: MUCH TOO LOW
```

The following example displays the group fields TYPE and SPECS from the domain YACHTS:

```
DTR> READY YACHTS
DTR> FIND FIRST 1 YACHTS
[1 Record found]
DTR> SELECT; PRINT

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

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

DTR> DISPLAY TYPE
DISPLAY: ALBERG    37 MK II
DTR> DISPLAY SPECS
DISPLAY: KETCH 37 200001236951
DTR>
```

DISPLAY_FORM Statement

Lets you display data on a form and collect data from a TDMS or FMS form.

Format

DISPLAY_FORM form-name IN file-name

[USING statement-1]

[RETRIEVE [USING] statement-2]

Arguments

form-name

Is the name of the TDMS or FMS form to be used with the domain.

file-name

Is the file specification of the form library file containing the form. 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.

statement-1

Is a DATATRIEVE statement or a series of statements within a BEGIN-END block. Statement-1 can include one or more PUT_FORM assignment statements for assigning values to fields on a form.

The PUT_FORM statement has the following format:

```
PUT_FORM form-field = value-expression
```

form-field

Is the name of a field in a form.

value-expression

Is any DATATRIEVE value expression.

statement-2

Is a DATATRIEVE statement or a series of statements within a BEGIN-END block. Statement-2 can include GET_FORM value expressions for assigning values on a form to DATATRIEVE fields or variables.

The GET_FORM value expression has the following format:

```
GET_FORM form-field
```

form-field
Is the name of a field in a form.

Restriction

If SET NO FORM is in effect, DATATRIEVE does not use its form interface and does not attempt to open a form library. You can not use the SEND or RECEIVE clauses of the WITH_FORM statement with a TDMS or FMS form. DATATRIEVE returns an error message.

Result

DATATRIEVE displays the form specified. If you have included the USING clause, DATATRIEVE displays only the fields specified in the PUT_FORM assignment statements.

Usage Notes

- The DISPLAY_FORM statement lets you use forms to modify and store data for specified fields. See the *VAX DATATRIEVE Guide to Interfaces* for more information on using forms with DATATRIEVE.
- Note that, because you specify both record and form field names in the DISPLAY_FORM format, form and record field names need not match.

Examples

The following example shows that you can display a form for a domain even if the form was not specified in the domain definition:

```
DTR> DISPLAY_FORM YACHTF IN FORMSLIB;
```

The following example displays the MANUFACTURER and MODEL fields on a form for the first five records of YACHTS:

```
DTR> FOR FIRST 5 YACHTS
CON>      DISPLAY_FORM YACHTF IN FORMSLIB USING
CON>          BEGIN
CON>             PUT_FORM MANUFA = MANUFACTURER
CON>             PUT_FORM MODEL = MODEL
CON>          END;
```

The following example displays the MANUFACTURER and MODEL fields on a form for the first record of YACHTS and assigns the values to two variables, BUILT and MODELLER:

```
DTR> DECLARE BUILT PIC X(10).
DTR> DECLARE MODELLER PIC X(10).
DTR> FOR FIRST 1 YACHTS
CON> DISPLAY_FORM BOATS IN [MORRIS]DTR32.FLB USING
CON>     BEGIN
CON>       PUT_FORM MANUFA = MANUFACTURER
CON>       PUT_FORM MODEL = MODEL
CON>     END  RETRIEVE USING
CON>        BEGIN
CON>          BUILT = GET_FORM MANUFA
CON>          MODELLER = GET_FORM MODEL
CON>        END
DTR> PRINT BUILT

  BUILT

ALBERG

DTR> PRINT MODELLER

 MODELLER

37 MK II

DTR>
```

You can use a form to store and modify values for selected fields. You can also associate more than one form with a single domain. See the *VAX DATATRIEVE Guide to Interfaces* for more examples.

DROP Statement

Removes the selected record from a collection, but does not remove that record from the data file in which it resides.

Format

DROP [collection-name]

Argument

collection-name

Is the name of a collection. If the DROP statement does not contain this argument, it affects the CURRENT collection.

Restrictions

- You must use the SELECT statement to establish a selected record in a collection before entering a DROP statement.
- You cannot drop a record you have already dropped. If you have dropped the selected record of a collection, DATATRIEVE responds with the following message when you enter a DROP statement:

  ```
  Target record has already been dropped.
  ```

- You cannot drop a record that has been erased. If you have erased the selected record of a collection, DATATRIEVE responds with the following message when you enter a DROP statement:

  ```
  No collection with selected record for DROP.
  ```

- If you have no established collections, or if no collection has a selected record, DATATRIEVE responds with the following message when you enter a DROP statement:

  ```
  No collection with selected record for DROP.
  ```

Results

- When you drop a record from a collection, the record is no longer available to you for retrieval or update. The dropped record is not erased from the data file, and it can be retrieved again by forming a record stream or another collection that contains it.
- When you enter a DROP statement without specifying a collection name, DATATRIEVE drops the selected record in the nearest single record context.

If the CURRENT collection has a selected record, DATATRIEVE drops it from the CURRENT collection. If the CURRENT collection has no selected record but other named collections do, DATATRIEVE drops the selected record from the most recently formed of those collections.

If the selected record in the nearest single record context has been dropped, DATATRIEVE responds to a DROP statement with the following message:

```
Target record has already been dropped.
```

- When you specify a collection name in the DROP statement, DATATRIEVE drops the selected record in the specified collection. If the named collection has no selected record, or if the selected record has been erased, DATATRIEVE responds to the DROP statement with the appropriate message as described above.

Usage Notes

- By using the DROP statement, you can refine a collection until it contains exactly the records you want.
- To see if a selected record has been dropped or erased, use the SHOW collection-name command.
- Before dropping a selected record in the nearest single record context, display the record by typing PRINT and pressing the RETURN key.

Example

In the following example, a record is stored in YACHTS and a series of collections is formed. The DROP statement is illustrated using the SELECT, DROP, ERASE, and PRINT statements and the SHOW collection-name command:

```
DTR> READY YACHTS WRITE
DTR> STORE YACHTS USING BUILDER = "HINKLEY",
DTR> FIND YACHTS WITH BUILDER = "HINKLEY"
[1 record found]
DTR> FIND A IN CURRENT
[1 record found]
DTR> FIND B IN YACHTS
[114 records found]
DTR> SELECT B; PRINT B.BOAT

                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

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

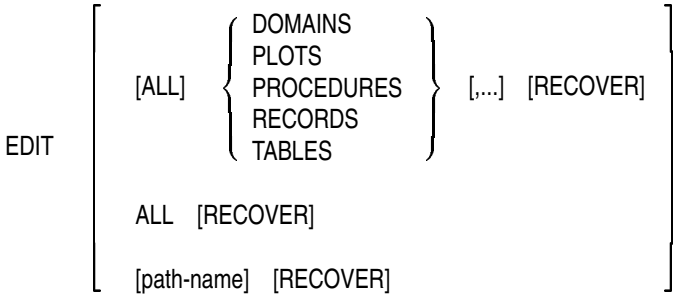
```
DTR> FIND C IN YACHTS
[114 records found]
DTR> SELECT LAST; PRINT
                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 WRIGHT      SEAWIND II SLOOP   32    14,900  00  $34,480

DTR> SHOW C
Collection C
    Domain: YACHTS
    Number of Records: 114
    Selected Record: 114
DTR> DROP
DTR> SHOW C
Collection C
    Domain: YACHTS
    Number of Records: 114
    Selected Record: 114 (Dropped)
DTR> PRINT
                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

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

DTR> DROP

Target record has already been dropped.
DTR> SHOW B
Collection B
    Domain: YACHTS
    Number of Records: 114
    Selected Record: 1
DTR> DROP B
DTR> SHOW B
Collection B
    Domain: YACHTS
    Number of Records: 114
    Selected Record: 1 (Dropped)
DTR> RELEASE C
DTR> DROP
Target record has already been dropped.
DTR> ERASE
No target record for ERASE.
DTR> RELEASE B
DTR> PRINT
No record selected, printing whole collection
                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE
```

```
 HINKLEY                                  0  00

DTR> DROP
No collection with selected record for DROP.
DTR> SHOW CURRENT
Collection A
    Domain: YACHTS
    Number of Records: 1
    No Selected Record
DTR> SELECT; ERASE; SHOW CURRENT
Collection A
    Domain: YACHTS
    Number of Records: 1
    Selected Record: 1 (Erased)
DTR> DROP
No collection with selected record for DROP.
DTR>
```

EDIT Command

Invokes an editor to edit the previous command or statement, one or more types of object definitions, or the dictionary object specified by the dictionary path name.

Format



Arguments

ALL
Places all the objects in your CDD/Repository default directory in an editing buffer. The keyword ALL is optional when used with the object types, but required when used alone or with only RECOVER.

path-name
Is the given name, full dictionary path name, or relative path name of a DATATRIEVE domain, record, procedure, or table definition you want to edit. EDIT accepts both DMU and CDO style path names.

DOMAINS
PLOTS
PROCEDURES
RECORDS
TABLES
You can specify one or more types of object definitions with the EDIT command. This allows you to edit all the domains, plots, procedures, records, or tables from your current default CDD/Repository directory.

RECOVER
Allows recovery for edited dictionary objects.

Restrictions

- To edit a DATATRIEVE domain, record, procedure, or table definition defined in the DMU format dictionary, you must have the following access privileges:
  - P (PASS_THRU) and X (EXTEND) access to the parent directory of the definition
  - P (PASS_THRU), S (SEE), and R (DTR_READ) access to the object to get the definition into the main buffer of the editor
  - U (UPDATE) if you are editing in EDIT_BACKUP mode
  - Either D (LOCAL_DELETE) or G (GLOBAL_DELETE) if you are editing in NO EDIT_BACKUP mode
- To edit a DATATRIEVE domain, record, procedure, or table definition defined in the CDO format dictionary, you must have the following access privileges:
  - S (SHOW) and U (CHANGE) access to the directory containing the definition
  - S (SHOW) and R (READ) access to the object to get the definition into the main buffer of the editor
  - U (CHANGE + DEFINE) if you are editing in EDIT_BACKUP mode
  - D (DELETE) and U (CHANGE + DEFINE) if you are editing in NO EDIT_BACKUP mode
- When you use the EDIT command, the definition you are editing must have access privileges that allow creation of later versions. Typically, you need not worry about these privileges. The data dictionary is usually set up by the system manager to include the ACL access privileges you need. See the chapter on ACL in the *VAX DATATRIEVE User's Guide* for a description of privileges necessary to edit definitions.
- When you specify multiple CDD/Repository objects for the EDIT command, the length of the names of the dictionary objects together cannot exceed 227 characters.
- You cannot specify both object types and object path names in the same argument list. The following combination of PLOTS and object path name would generate an error message:

```
DTR> EDIT PLOTS, CDD$TOP.DTR$LIB.DEMO.YACHTS
Argument list cannot contain both the object path name and
the object type.
```

- If you specify the RECOVER option to restore changes made to CDD/Repository objects during an aborted edit session, use exactly the same syntax you used for the original editing session (with the addition of the keyword RECOVER). For example:

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

- If you use the EDIT command in a procedure, the statements and commands affected by the edit do not execute until the rest of the procedure has finished.
- If you are running the DATATRIEVE DECwindows interface, you cannot set your DTR$EDIT logical to EDT. The EDT editor does not work in the DATATRIEVE DECwindows environment.

Results

- When you specify the dictionary path name with the EDIT command, DATATRIEVE invokes the editor and places the text of the object in the edit buffer. The main buffer of the editor contains a REDEFINE command for the object, followed by the contents of the dictionary object. If SET EDIT_BACKUP is in effect, DATATRIEVE saves the original definition in the data dictionary when you exit the editor.
- When you omit the dictionary path name from the EDIT command line, DATATRIEVE invokes the editor and loads the previous command or statement into the main text buffer of the editor.
- While you are editing CDD/Repository objects, DATATRIEVE places a journal file for the editing session in your default VMS directory. The journal file is automatically deleted upon successful completion of the editing session. Table 4–7 shows the default file types for journal files.

Table 4–7 Default Journal File Types

Editor	Default Journal File Type
EDT	.JOU
LSE	.TJL
VAXTPU	.TJL

If you exit an editing session abnormally (if you enter a CTRL/Y or the operating system fails), you can recover the editing session with the RECOVER qualifier. For example:

```
DTR> EDIT YACHTS RECOVER;
```

- DATATRIEVE automatically executes the content of the main text buffer if you end the editing session with an EXIT command. This lets you use the editor to correct typographical, syntax, and logical errors in less time than you could retype most commands and statements.

  The content of the main text buffer when you exit need not have any relationship to the content it had when you invoked the editor.

- To end an editing session, use either the EXIT or QUIT command.

  QUIT causes DATATRIEVE to ignore the contents of the editor's main buffer and to return you to the DATATRIEVE command level (indicated by the DTR> prompt).

  EXIT causes DATATRIEVE to take one of two actions, depending on how you invoked it:

  – If you specified a dictionary path name and DATATRIEVE is in EDIT_ BACKUP mode, EXIT causes DATATRIEVE to create a new definition of the object, with the next highest version number. The original version of that object remains in the data dictionary. The new version of the object contains the ACL and the history list entries from the previous version.

  – If you invoked the editor with only the keyword EDIT, the EXIT command causes DATATRIEVE to execute the commands and statements in the editor's main buffer. If the commands and statements are incomplete and SET PROMPT is in effect, DATATRIEVE prompts you for the next syntax element in the command or statement.

- The editor does not check the syntax of any DATATRIEVE commands or statements in the main buffer. If you make a syntax error when correcting your definition or your previous command or statement, DATATRIEVE responds to the error only when it executes the commands or statements after you exit from the editor or when you invoke the edited procedure or ready the domain.

Usage Notes

- If SET VERIFY is in effect, DATATRIEVE displays the contents of the buffer when you exit from the editor. Then DATATRIEVE executes any commands and statements. If SET NOVERIFY or SET NO VERIFY is in effect, DATATRIEVE does not provide this display.

- Object types are placed in the edit buffer in the order you specify. In the following example, the record object definitions are placed in the edit buffer before the domain object definitions:

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

- You can use EDIT in two modes, EDIT_BACKUP and NO EDIT_BACKUP:
  - When you edit a definition in EDIT_BACKUP mode, DATATRIEVE creates a new definition of the object, with the next highest version number.
  - When you edit a definition in NO EDIT_BACKUP mode, DATATRIEVE places a DELETE command, a REDEFINE command, and the text of the object in the edit buffer.

    When you exit the edit buffer, DATATRIEVE deletes the highest existing version, or the version you specified, and replaces it with the definition in the edit buffer. You receive the informational message "object to be redefined not found" during this process. You can ignore this message.

    Because DATATRIEVE deletes the highest version before it redefines the new version, you will lose the existing definition, the definition's ACL, and its history list if you exit the edit buffer and the definition fails. When you exit the edit buffer and the definition succeeds, DATATRIEVE gives the new definition the ACL and the history list of the highest existing version. If there is no existing version of the definition, DATATRIEVE gives the new definition the default ACL and history list.
- To delete previous versions of an object in the data dictionary, use the DELETE command with an explicit version number or use the PURGE command.
- If you invoke a procedure that contains one or more errors, DATATRIEVE stops executing the procedure and displays an error message. If you invoke the editor with the keyword EDIT, DATATRIEVE puts the faulty command or statement and the remainder of the commands and statements in the main buffer. When you have corrected the error and leave the editor with the EXIT command, DATATRIEVE executes the remaining commands and statements in the procedure.

  The changes you make in these circumstances are not lasting changes to the procedure's definition that is stored in the data dictionary. To make the correction permanent, you must include the procedure name when you invoke the editor and make changes to the CDD/Repository definition of the procedure.
- Be careful when editing record definitions. If you change the length of the record definition, you have to create a new data file and transfer the old information from the old file to the new one.

  If you change the name of any fields, any procedures and reports that refer to those fields have to be edited to reflect the changes.

If you change the definition of a domain or its associated record definition while you have that domain readied in your workspace, the changes do not take effect until you finish the domain with the FINISH command and ready it again. Simply rereadying the domain without finishing it does not use any new definitions you may have entered into the data dictionary since you first readied the domain.

- To perform the editing function, DTR makes use of the VMS mailbox mechanism. DATATRIEVE uses temporary mailboxes to transfer the edit information to and from the editor. This has several implications for users:
  - To use the DATATRIEVE EDIT command, you must have the VMS process privilege TMPMBX. This is the privilege that allows the creation of a temporary mailbox. If your account does not have TMPMBX privilege, VMS generates a privilege violation error message when you use the EDIT command.
  - VAX DATATRIEVE uses a logical name for the temporary mailbox. The logical name is entered into the logical name table LNM$TEMPORARY_MAILBOX. If you redefine LNM$TEMPORARY_MAILBOX, you must be careful to follow the guidelines provided in the section on SYS$CREMBX in the *VMS System Services Reference Manual* so that DATATRIEVE can continue to access LNM$TEMPORARY_MAILBOX using the mailbox system services.
- For more information on the DTR$EDIT logical and on how to use the EDIT command, see the *VAX DATATRIEVE User's Guide*.

Example

The following example shows how to edit the definition for the record YACHT:

```
DTR> EDIT YACHT
REDEFINE RECORD YACHT USING
01 BOAT.
  03 TYPE.
     06 MANUFACTURER PIC X(10)
        QUERY_NAME IS BUILDER.
     06 MODEL PIC X(10).
  03 SPECIFICATIONS
     QUERY_NAME SPECS.
     06 RIG PIC X(6)
        VALID IF RIG EQ "SLOOP","KETCH","MS","YAWL".
     06 LENGTH_OVER_ALL PIC XXX
        VALID IF LOA BETWEEN 15 AND 50
        QUERY_NAME IS LOA.
     06 DISPLACEMENT PIC 99999
        QUERY_HEADER IS "WEIGHT"
        EDIT_STRING IS ZZ,ZZ9
        QUERY_NAME IS DISP.
     06 BEAM PIC 99 MISSING VALUE IS 0.
     06 PRICE PIC 99999
        MISSING VALUE IS 0
        VALID IF PRICE>DISP*1.3 OR PRICE EQ 0
        EDIT_STRING IS $$$,$$$.
;
[Record is 41 bytes long.]
```

You can now edit the record definition. If SET EDIT_BACKUP is in effect, the old version of the YACHT record is retained in the data dictionary when you exit the editor.

See the *VAX DATATRIEVE User's Guide* for examples of using the editor to edit previous commands and statements in an interactive DATATRIEVE session.

EDIT_STRING Clause

Specifies the output format of a field value.

Format

EDIT_STRING [IS] edit-string

Argument

edit-string

Is one or more edit string characters describing the output format of the field value.

Restrictions

- The EDIT_STRING clause is valid only for elementary fields.
- Edit strings must consist of valid edit string characters.

Restriction on Certain Date Edit Strings

The use of the D edit string character without also using one of the other date edit string characters (M, Y, N, J, or W) is restricted. The D character by itself is not sufficient to uniquely identify it as a specific character.

This stems from the fact that DATATRIEVE is flexible in interpreting data, regardless of type. The ambiguity lies with the characters D and B. Used together they can be interpreted either as a request to print the letters DB to indicate a negative value or to print the corresponding day of the month followed by a blank.

For the proper interpretation, you must use the edit string character D in the context of the other date characters.

However, if your application requires the use of just the D (day) portion of the date you may consider using the following to generate that information:

```
DTR>DECLARE abc USAGE DATE.
DTR>abc = "20-Feb-1900"
DTR>DECLARE xyz PIC X(2) COMPUTED BY (FORMAT abc USING DDM).
DTR>PRINT xyz

XYZ

20

DTR>
```

Result

DATATRIEVE uses the edit string as the default format when writing a field value to a file or output device.

Usage Notes

- If you do not include an EDIT_STRING clause in a field definition, the PICTURE clause determines the default output format. However, DATATRIEVE does not display a sign specified in a PICTURE clause unless you specify those characters in an edit string.
- You can override the default output format specified by either a PICTURE or an EDIT_STRING clause. With the print list modifier USING edit-string, you can specify an output format for a value within the PRINT statement.
- You specify the format of the field value with a string of one or more edit characters. Specify the edit characters as a string without embedded spaces. In general, each edit character corresponds to one character position in the printed output. For example, 999999 specifies that the output will be six digits in six character positions.
- To enter more than one of the same edit character, you can shorten the edit string by placing a repeat count in parentheses following the edit character. For example, the edit string 9(6) is equal to 999999.
- The TOTAL function is the only statistical function that ignores edit strings. This is the expected behavior. The field's edit string is not used because the value generated by TOTAL is likely to be quite a bit larger than the individual field values. This larger value could overflow the edit string associated with the field. You can provide an edit string for the totaled value by specifying an edit string on the print statement as follows:

```
DTR> PRINT TOTAL PRICE OF CURRENT USING $$$$$.99
```

The edit string characters you can specify for a field depend on the class of the field: alphabetic, alphanumeric, numeric, or date. Do not use editing characters designated only alphabetic or alphanumeric on numeric fields (or vice versa).

Remember that field type is determined by the PIC or USAGE clause, not the value in the field. A field defined as PIC X(10) might contain only numbers, for example, but you should use only alphanumeric editing characters to format the way you want to display those numbers.

Fields in general

?	If the field has a MISSING VALUE clause, the question mark separates two edit strings. The first edit string formats the output of the field if the value is not the missing value. If the content of the field is the missing value, the second edit string controls the output of the field.
(n)	a repeat count (0 < n). The previous character is repeated a total of "n" times. The edit string XXXX is equivalent to X(4).

The MISSING VALUE clause designates a value for a field that DATATRIEVE recognizes, not as the literal value, but as a marker that no value is stored in the field. DATATRIEVE ignores fields containing the "missing value" marker when calculating statistical functions. If you do not have a default value defined for a field, DATATRIEVE uses the missing value for the default value.

Format

MISSING##[VALUE]##[IS] literal

For more information on missing values, see the MISSING VALUE clause.

Alphanumeric Fields

For a text edit string:

T	indicates text. Each T reserves a column on a line for the associated print list element. For example, PRINT "1234567890" USING T(5) displays 12345 in the first five columns of one line and 67890 in the first five columns of the next line. Edit strings containing a T cannot contain other characters.

For alphanumeric edit strings:

A	is replaced by an alphabetic character.
X	is replaced by one character.
9	is replaced by one digit.
B	is replaced by one space.
"literal"	the literal enclosed in quotation marks is inserted. The outer quotation marks are not inserted in the output.
0, $, %, *, +, -, ., /	are simply inserted.

The edit string for an alphanumeric field specifies the number and type of characters to be printed, except for T edit strings, which print all the characters. The edit character X is replaced by one character from the field's content. The characters are transferred from the field to the output in left-to-right order.

DATATRIEVE always left justifies alphanumeric fields. If the edit string contains X and the field's content has more characters than the edit string, only the leftmost characters are printed. If it has fewer characters than its edit string, DATATRIEVE pads the output with blanks on the right.

If the field contains only digits, you may use a numeric edit string. The restrictions on edit strings for numeric fields are explained in the following section.

If you use a field in any arithmetic computations, you should define it as a numeric field. You can perform computations with alphanumeric fields that contain only digits, but because alphanumeric fields are padded with spaces on the right, the results may not be valid.

DATATRIEVE drops leading zeros from alphanumeric fields when it converts from a numeric to an alphanumeric field.

Table 4–8 contains sample edit strings and the format of the DATATRIEVE output for two field values: CHALLENGER and 123. The picture-string X(10) is used in the PICTURE clause of both fields. This defines the internal format of the field value. The format of the output is determined by the edit strings. In the table, a number sign (#) represents a space.

Table 4–8 EDIT_STRING Output for Two Field Values

Picture String	Edit String	Output for CHALLENGER	Output for 123
X(10)	X(10)	CHALLENGE	123#######
X(10)	X(3)	CHA	123
X(10)	XX/X(8)	CH/ALLENGER	12/3#######
X(10)	X(5)/X(5)	CHALL/ENGER	123##/#####
X(10)	X(5)-XX	CHALL-EN	123##-##

The edit character T allows you to print alphanumeric field values on one or more lines. The primary use of T is to print fields containing large amounts of text. The number of Ts in the edit string indicates the maximum number of characters to be printed on one line. For example, the edit string T(20) indicates that a line of output will contain no more than 20 characters (unless a single word contains more than 20 characters).

If the field contains more characters than specified in the edit string, DATATRIEVE prints as many full words on the line as possible. (A word, in this sense, is a string of characters delimited by a space.) It does not divide words unless a word is longer than the edit string. DATATRIEVE then prints the remaining characters on the next line and following lines, if necessary. DATATRIEVE does not print out trailing spaces when you use a T edit string.

The following example uses data from EMP_REVIEW, a domain set up to keep information about employee reviews. Here is the record definition:

```
DTR> SHOW EMP_REVIEW_REC
RECORD EMP_REVIEW_REC USING
     01 EMP_REC.
        05 BADGE      PIC IS 9(5).
        05 NAME       PIC IS X(10).
        05 JOB               PIC IS X(15).
        05 EVALUATION PIC IS X(60).
        05 EVAL_DATE  USAGE IS DATE
              EDIT_STRING IS  DD-MMM-YY.
;
```

In the following DATATRIEVE session, a PRINT statement specifies a T(20) edit string for the EVALUATION field:

```
DTR> FOR EMP_REVIEW
[Looking for statement]
CON> PRINT NAME, EVALUATION USING T(20), EVAL_DATE, SKIP

                                 EVAL
  NAME          EVALUATION       DATE

BRAD H.     Brad did a fine job  29-Apr-83
            in implementing
            staged output.

DAVID D.    David is a master of 12-Mar-83
            developing report
            specifications.

TERRY C.    Terry is an          21-Mar-83
            exceptional system
            manager and
            developer.

DTR>
```

Note that when you print a long field value, the field name can be in any position within the print list. EVAL_DATE followed EVALUATION in the print list, but its value was printed on the same line as the first line of text.

EDIT_STRING Clause

Numeric Fields

For numeric and floating point edit strings:

9	is replaced by one digit. The digits are right justified in the output, and leading character positions are filled with zeros.
Z	is replaced by a space if it matches a leading zero; otherwise it is replaced by a digit.
*	(asterisk) is inserted if it matches a leading zero; otherwise, it is replaced by a digit.
+	(plus sign) is replaced by a plus or minus sign depending on the field's value. If more than one plus sign is specified, all leading zeros are suppressed, the sign (plus or minus) is displayed to the immediate left of the leftmost character position determined by the other edit string characters.
-	(minus sign) is replaced by a space or minus sign depending on the field's value. If more than one minus sign is specified, all leading zeros are suppressed. If the value of the field is negative, a minus sign is inserted to the immediate left of the leftmost character position determined by the other edit string characters.
.	(decimal point) specifies the position of the decimal point, and is inserted in that character position.
,	(comma) If all the digits to the left of the comma are suppressed zeros, the comma is replaced by a blank. If not, a comma is inserted in that character position.
( )	(parentheses) If the field's content is negative, the value is enclosed in parentheses. Otherwise, the field value is printed without parentheses.
"literal"	the literal enclosed in quotation marks is inserted. The outer quotation marks are not inserted in the output.
CR	If the field's content is negative, the letters CR are inserted. If the field's content is positive, CR is replaced by 2 blanks.
DB	If the field's content is negative, the letters DB are inserted. If the field's content is positive, DB is replaced by 2 blanks.
B	a space is inserted.

$ If only one dollar sign is specified, it is replaced by a $ in that character position. If more than one dollar sign is specified, any leading zeros are suppressed, and one dollar sign is displayed to the immediate left of the leftmost digit.

For a floating point edit string:

E The E divides the edit string into two parts for floating_point or scientific notation. The first part is the mantissa edit string and the second part is the exponent edit string.

0, %, / are simply inserted.

An EDIT_STRING clause prints numeric field values in a format that is easy to read. With edit strings, you can suppress leading zeros and print dollar signs, percent signs, commas, decimal points, and plus or minus signs. For example, the EDIT STRING clause $$$,$$$ causes DATATRIEVE to print the value 09870 as $9,870.

You can use three types of edit characters in edit strings for numeric fields: replacement characters, insertion characters, and floating characters.

Replacement Characters

Table 4–9 shows the differences between the 9, Z and * (asterisk) replacement characters. In the table, a number sign (#) represents a space.

Table 4–9 Replacement Characters in Numeric Fields

Picture String	Edit String	Field Content	Output
99999	9(5)	04092	04092
99999	Z(5)	04092	#4092
99999	*(5)	04092	*4092
99V99	99.99	0001	00.01
99V99	ZZ.99	0001	##.01

Before printing a field, DATATRIEVE computes the number of digits to the left of the decimal point. (If there is no V in the picture string, all digits are to the left of the decimal.) If there are more digits to the left of the decimal than the edit string specifies (either with replacement characters or floating characters), DATATRIEVE prints an asterisk in each character position specified by the edit string. If the field has fewer digits than the edit string specifies, DATATRIEVE pads the output with leading zeros. (These are suppressed by the Zs or floating characters you use. If there are more leading zeros than Zs or floating characters,

the remaining zeros are displayed at the left of the field value but to the right of any floating character.)

For example, a PICTURE clause could specify four digits for a field and its edit string only two digits:

```
03 MODEL_NUMBER
PICTURE IS 9999
EDIT_STRING IS 99.
```

If the field value is 1234, DATATRIEVE prints two asterisks (**) for the field value. Therefore, be careful to specify edit strings that are long enough for numeric fields.

If you define a numeric field with the clauses PIC 9(4) and USAGE IS COMP, then the field can contain numbers as high as 32,768. To print the field when it contains a 5-digit number, you must use an edit string that specifies five digits.

Insertion Characters

With insertion characters, you can print the sign of a field, a decimal point, a dollar sign, DB or CR or parentheses for a negative value, a percent sign, commas, zeros, slashes, and character string literals.

Printing a Sign To print a sign (+ or –) in a field value (indicated by an S in its PICTURE clause), you must specify a plus (+) or minus (–) sign in the edit string for that field. The sign must be the first or last character in the edit string.

If you specify only one sign in the edit string, DATATRIEVE prints the sign in the position you indicate. If you specify more than one sign in the leftmost part of the edit string, the sign is a floating character.

You can use only one DB or CR in an edit string, and it must be the leftmost or rightmost element of the edit string.

If you specify double parentheses around the edit string, DATATRIEVE prints single left and right parentheses around a field value if it is negative.

Table 4–10 shows the use of the edit characters +, –, DB, CR, and parentheses. In the table, a number sign (#) represents a space.

Table 4–10 Sign Insertion Characters

Picture String	Edit String	Field Content	Output
S9999	None	–1234	1234
S9999	–9999	–1234	–1234
S9999	–9999	+1234	#1234
S9999	9999+	–1234	1234–
S9999	+9999	+1234	+1234
S9999	9999DB	–1234	1234DB
S9999	9999CR	–1234	1234CR
S9999	CR9999	+1234	##1234
S9999	((9999))	–1234	(1234)

Printing a Decimal Point By default, DATATRIEVE prints the implied decimal point in a field value (indicated by a V in its PICTURE clause). You can, however, control the placement of the decimal point (.) by specifying one in the edit string for that field. When DATATRIEVE prints the field content, it aligns all output on the decimal point.

DATATRIEVE matches the decimal point in the edit string with the implied decimal place in the field. If the edit string contains fewer digits to the right of the decimal, the extra digits are not printed. If the edit string contains fewer digits to the left of the decimal, DATATRIEVE prints asterisks. Thus it is best to place the period in the edit string in the same position as the V in the picture string.

Table 4–11 shows printing decimal points in several different numeric fields. In the table, a number sign (#) represents a space.

Table 4–11 Decimal Point Insertion

Picture String	Edit String	Field Content	Output
99V99	(None)	1234	12.34
99V99	Z9.99	1234	12.34
99V99	999.9	1234	012.3

(continued on next page)

Table 4–11 (Cont.) Decimal Point Insertion

Picture String	Edit String	Field Content	Output
99V99	9.999	1234	*****
99V99	9.999	0123	1.230
99V99	Z(4)	1234	##12

If the last character of the edit string is a period and is not followed by other input on the same line, DATATRIEVE treats the period as the termination of the field definition and not as part of the edit string. If the EDIT STRING clause is the last part of the field definition, specify two periods; the first period is part of the edit string and the second period ends the field definition. If the EDIT STRING clause is not the last part of the field definition, place the next clause on the same line or place a hyphen at the end of the line.

Printing Other Characters To print a comma, slash, percent sign, dollar sign, or zero, specify that character in the edit string. If there are only spaces to the left of a comma, DATATRIEVE prints a space instead of a comma. If you include more than one dollar sign, it is a floating character. Table 4–12 shows how these characters are used. In the table, a number sign (#) represents a space.

Table 4–12 Special Insertion Characters

Picture String	Edit String	Field Content	Output
99	99%	45	45%
9(6)	$999,999	100000	$100,000
9(6)	$$$$,$$$	100000	$100,000
9(6)	ZZZ,ZZZ	000040	#####40
9(6)	999/999	123456	123/456

Printing Literals You can include character string literals in edit strings. Do not leave any spaces between the elements of the edit string. You can have spaces embedded in the quoted character strings but do not leave spaces between the quotation marks and the other edit string characters.

Here are two examples of using character strings in edit strings. Field definition clauses in DECLARE statements perform just as they do when you use them in field definitions.

```
DTR> DECLARE NUM PIC 9(5).
DTR> NUM = 12345
DTR> PRINT NUM USING "THIS NUMBER, "ZZ,Z99", WILL SURPRISE YOU."

                  NUM

THIS NUMBER, 12,345, WILL SURPRISE YOU.

DTR> DECLARE NOTE USAGE DATE EDIT_STRING IS
[Looking for picture or edit string]
CON> "Today is "W(9)", the "DD"th day of "M(9)", in the year "Y(4).
DTR> NOTE = "TODAY"
DTR> PRINT NOTE

                              NOTE

Today is Monday, the 14th day of August, in the year 1989

DTR>
```

Floating Characters

You can specify three edit string characters ($, –, and +) as floating characters. A floating character replaces all but the last leading zero with spaces. The last leading zero is replaced by the edit string character ($, –, or +). Floating characters that correspond to digits are replaced by those digits. Because one character is replaced by a +, –, or $, you must specify one more character than the number of digits in the field.

To use a floating character, you must specify two or more of the same character. You can specify only one floating character in an edit string. For example, you cannot have both a floating minus sign and a floating dollar sign in the same edit string. The floating characters must be the leftmost characters in an edit string.

Table 4–13 shows the use of floating characters in edit strings. In the table, a number sign (#) represents a space.

Table 4–13 Floating Characters in Edit Strings

Picture String	Edit String	Field String	Output
S9(4)	++++9	+0187	#+187
S9(4)	++++9	–5764	–5764

(continued on next page)

Table 4–13 (Cont.) Floating Characters in Edit Strings

Picture String	Edit String	Field String	Output
S9(4)	—-9	+0187	##187
S9(4)	—-9	–0001	###–1
9(5)V99	$9(5).99	0015786	$00157.86
9(5)V99	$$$,$$$.99	0015786	###$157.86
9(5)V99	$$$,$$$.00	0015786	###$158.00
S9(5)	$$$,$$$CR	+54362	$54,362CR

Changing the Defaults for Currency Symbols

You can change the default displays for the currency symbol, for the decimal point, and for the digit separator. To make your output conform to other conventions for numeric and monetary notation, you can override the system defaults for these symbols by redefining the following logical names:

SYS$CURRENCY
SYS$RADIX_POINT
SYS$DIGIT_SEP

You can use the necessary DCL DEFINE commands in your LOGIN.COM file at the DCL command level. You can also have your system manager set up system logical names for these symbols.

Date Fields

For date edit strings:

D	is replaced by a digit of the day of the month.
M	is replaced by a letter of the name of the month.
N	is replaced by a digit of the number of the month.
Y	is replaced by a digit of the numeric year.
J	is replaced by a digit of the Julian date.
W	is replaced by a letter from the name of the day of the week.

0, $, %, *, +, -, ., / are simply inserted.

A field defined as USAGE IS DATE is stored internally as a binary value. To print this field, DATATRIEVE must convert it to another format. If you do not include an EDIT_STRING clause in the field definition for a date field, DATATRIEVE prints the field value in the following format:

```
DD-MMM-YYYY
```

To print the date in any other format, you must include an EDIT_STRING clause in the field's definition. (You can also specify the output format with the print list modifier USING edit-string in your output statement.) The edit string for a date field gives you several formatting choices for printing a date: For example, the date can include the following:

- The name of the day of the week (such as Monday, Tuesday) or just the first characters of the name (such as Mon, Tue, Wed)
- The day of the month (such as 1, 2, 3)
- The name of the month (such as January, February) or just the first characters of the name (such as Jan, Feb, Mar)
- The number of the month (for example, 1 for January and 12 for December)
- The year (such as 1988) or just the last two digits of the year (88)
- The Julian date (for example, 001 for January 1 and 366 for December 31 in a leap year)
- Delimiters to separate the parts of the date (such as a slash or period after the month and day)

To print just the day part of a date field, extract the day using the FN$DAY function. For example:

```
DTR> PRINT ("TODAY") USING DD-MMM-YYYY
15-Nov-1990

DTR> PRINT FN$DAY("TODAY")

  FN$DAY

         15
```

You specify the characters to be printed (letters, digits, spaces, slashes, hyphens, or periods) and the order in which the parts of the date are to appear (such as month, day, year).

DATATRIEVE does not always output the total number of characters you specify with an alphabetic edit string such as M(9) or W(9). If the content of the field is shorter than its edit string specifies, DATATRIEVE output equals the length of the field value, not the length of the edit string. For example, an edit string can specify a 9-letter month: M(9). If the field contains a month with a name shorter than nine letters (such as June), DATATRIEVE prints only four characters, "June." DATATRIEVE does not pad the output with blanks.

EDIT_STRING Clause

Table 4–14 contains edit strings and the format of the output for two field values: June 4, 1990 and November 27, 1989. In the table, a number sign (#) represents a space.

Table 4–14 EDIT_STRING Output for Date Values

Edit String	Output if Field Value Is June 4, 1990	Output if Field Value Is November 27, 1989
DD-MMM-YY	#4-Jun-90	27-Nov-89
MMMBDDBY(4)	Jun##4#1990	Nov#27#1989
M(9)BDDBY(4)	June##4#1990	November#27#1989
NN/DD/YY	#6/04/90	11/27/89
W(9)	Monday	Monday
YYYY/JJJ	1990/156	1989/331
DDBMMMBYY/WWW	#4#Jun#90/Mon	27#Nov#89/Mon
DD.NN.YY	#4.06.90	27.11.89

END_REPORT Statement (Report Writer)

Ends the report specification.

Format

END_REPORT

Restriction

The END_REPORT statement must be the last statement in the report specification.

Results

After you enter the END_REPORT statement, the Report Writer takes one of three courses of action:

- Prompts you for the values you specified with a *.prompt in the record specification and then produces the report.
- Sends you a message indicating a syntax error in the report specification. When you see the DTR> prompt, type EDIT and press RETURN so that you can make the needed changes. When you leave the editor with the EXIT command, DATATRIEVE executes the new report specification. You can also edit a report specification by enclosing it in a procedure by using the DEFINE PROCEDURE command.
- Produces the report and sends it to the device or file you have specified in the REPORT command.

Example

For examples of report specifications, see the chapters on writing reports in the *VAX DATATRIEVE User's Guide*.

ERASE Statement

Permanently removes one or more data records from an indexed or relative data file, a VAX DBMS database, or a relational database.

Format

```
ERASE  [ALL  [OF rse] ]
```

Arguments

ALL

Causes DATATRIEVE to permanently remove from the data file every record in the current collection.

ALL OF rse

Causes DATATRIEVE to permanently remove from the data file every record identified by the record selection expression.

Restrictions

- The domain containing the targeted records must be readied for WRITE access. (See the section on the READY command.)
- The data file containing the targeted records must be an indexed sequential file or a relative file. You cannot delete records from a sequential file.
- You cannot delete a record from a view domain or a port.
- You cannot use the ERASE statement to change or remove fields from a list in a hierarchical record.
- When you erase a VAX DBMS record from DATATRIEVE, you erase not only that record, but all records in all sets owned by the erased record, all records in all sets owned by those records, and so forth. This effect is more far-reaching than that of the DML ERASE statement, so use caution when erasing VAX DBMS records from DATATRIEVE.

Results

- DATATRIEVE deletes the targeted records from the domain. If the domain is a VAX DBMS domain or a relational domain or relation, the changes are not permanent until you enter a COMMIT statement, a FINISH statement, or you exit from DATATRIEVE.
- If you use the argument ALL, DATATRIEVE deletes from the domain every record in the current collection.

- If you use the argument ALL OF rse, DATATRIEVE permanently deletes from the domain every record identified by the record selection expression.
- If you put the keyword ERASE by itself in a FOR statement, DATATRIEVE permanently deletes from the domain each record specified by the FOR statement.

  If you erase a collection or record stream that contains a selected record, the values of the fields of that record are still available in your workspace, even though the record has been removed from the domain by the ERASE statement. You can display the fields of that selected record, and you can use those field values in value expressions. Those values remain in your workspace until you change the single record context with another SELECT statement or with a DROP statement or a RELEASE or FINISH command.
- The ERASE statement permanently removes the selected record from the data file if you do not establish a target record stream with a FOR statement, with the OF rse clause, or with the keyword ALL. If you have no selected record in any collection, DATATRIEVE displays this message:

```
DTR> ERASE
No target record for ERASE.
DTR>
```

Usage Note

Before using the ERASE statement, you can check the current collection and selected record with the SHOW CURRENT command or the PRINT statement.

Examples

The following example shows how to erase all the yachts built by Albin:

```
DTR> FIND YACHTS WITH BUILDER EQ "ALBIN"
[3 records found]
DTR> ERASE ALL
```

In the following example, a procedure is defined that erases selected yachts:

```
DTR> DEFINE PROCEDURE SELL_BOAT
DFN>     FIND YACHTS WITH BUILDER EQ *.BUILDER AND
DFN>          MODEL = *.MODEL
DFN>     PRINT ALL
DFN>     IF *."Y IF BOAT SOLD" CONT "Y" THEN ERASE ALL
DFN> END_PROCEDURE
DTR>
```

EXIT Command

Ends a DATATRIEVE session.

Format

```
{ EXIT   }
{ CTRL/Z }
```

Parameters

None.

Restriction

You must issue the EXIT command at the DTR> prompt or, if you are using the DATATRIEVE DECwindows interface, choose the Exit item of the File menu.

Results

- EXIT stops a DATATRIEVE session and returns you to the DCL command level (indicated by the dollar sign prompt).
- Entering CTRL/Z from the DTR> prompt acts as an EXIT command and stops a DATATRIEVE session.
- When you stop your DATATRIEVE session by typing either EXIT or CTRL/Z, DATATRIEVE automatically finishes all readied domains and releases all collections, global variables, and tables.
- Entering CTRL/Z from the DFN>, CON>, and RW> prompts brings you back to DATATRIEVE command level.
- Using CTRL/Z does not stop your DATATRIEVE session if you are in HELP, ADT, Guide Mode, or the editor. Three consecutive CTRL/Zs in response to the line mode prompt of the editor act like a QUIT command and return you to DATATRIEVE command level.
- Entering CTRL/Z does not stop your DATATRIEVE session when you enter it in response to an Enter prompt during the execution of a STORE or MODIFY statement. Entering CTRL/Z in response to an Enter prompt returns you to DATATRIEVE command level and aborts the STORE or MODIFY statement.
- When changes have been made to a VAX DBMS or relational database, entering EXIT or CTRL/Z from the DTR> prompt is equivalent to issuing a VAX DBMS or relational database COMMIT command.

Examples

The following example shows how to end a DATATRIEVE session:

```
DTR> EXIT
$
```

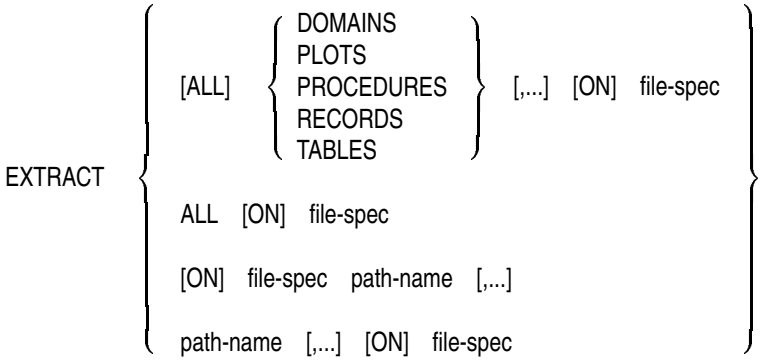
The following example shows the result of entering CTRL/Z to the Enter prompt of a STORE statement:

```
DTR> READY YACHTS WRITE
DTR> STORE YACHTS
Enter MANUFACTURER: CTRL/Z
Execution terminated by operator
DTR>
```

EXTRACT Command

Copies the CDD/Repository data dictionary definition of one or more dictionary objects or types of object definitions to a command file.

Format



Arguments

ALL
Causes DATATRIEVE to copy into the specified command file the definitions of one or more dictionary objects in your default dictionary directory.

The keyword ALL is optional when used with the types of object definitions such as PLOTS or DOMAINS.

The keyword ALL is required, however, when used with the EXTRACT ALL [ON] file-spec syntax.

DOMAINS
PLOTS
PROCEDURES
RECORDS
TABLES
Allows you to extract all the domains, plots, procedures, records, or tables from your current default CDD/Repository directory.

file-spec
Is the VMS specification of the RMS file to contain the definitions. A complete file specification has the following format:

```
node-spec::device:[directory]file-name.type;version
```

path-name
Is the given name, full dictionary path name, or relative path name of the dictionary object whose definition you want to copy. If you specify more than one dictionary path name, use a comma to separate each one from the next. EXTRACT accepts both DMU and CDO style path names.

Restrictions

- To extract the definition of a dictionary object from the DMU format dictionary, you must have the following access privileges to it:
  - P (PASS_THRU) and X (EXTEND) access to the parent directory of the dictionary object
  - P (PASS_THRU), S (SEE), and R (DTR_READ) access to the object you want to extract
- To extract the definition of a dictionary object from the CDO format dictionary, you must have the following access privileges to it:
  - S (SHOW) access to the directory containing the object
  - S (SHOW) and R (READ) access to the object you want to extract
- To check what privileges you have to a dictionary object, use the SHOW PRIVILEGES command (see the section on the SHOW command.)
- You cannot specify both object types and object path names in the same argument list. The following combination of PLOTS and object path name would generate an error message:

```
DTR> EXTRACT PLOTS, CDD$TOP.DTR$LIB.DEMO.YACHTS
Argument list cannot contain both the object path name and
the object type.
```

- An ON file-spec clause may precede or follow the list of dictionary path names, but it must occur exactly once.
- You must specify at least one field of the file specification.
- Do not extract a procedure containing comment lines that end with a hyphen. DATATRIEVE interprets the hyphen as a continuation mark and appends the next line to the comment line.
- If you do not specify an explicit version number, DATATRIEVE copies the highest version of the definition to a command file preceded by the DELETE and REDEFINE commands.

Results

- DATATRIEVE creates a command file with the file specification you provide. For each dictionary object you name, DATATRIEVE enters a DELETE command and a REDEFINE command in the command file. The REDEFINE command operates on the highest or a specified version of the object.
- For all dictionary objects entered in the data dictionary by DATATRIEVE, the EXTRACT command copies the text of the highest or a specified version of the definition, exactly as it was entered. For example, when you extract a domain definition, the dictionary path names of the domain and the record are extracted just as they were entered. DATATRIEVE does not adjust or generalize the path names. If the path name was stored as a relative path name, the relative path name is copied into the command file.
- When DATATRIEVE constructs the DELETE command, it puts only the given name of the dictionary object in the DELETE command.
- When the EXTRACT command copies the definitions into the command file, DATATRIEVE does not delete the definitions from the data dictionary
- If a record definition has been stored in the data dictionary by something other than DATATRIEVE, you can extract a definition of that record. When DATATRIEVE extracts such a record, however, it translates the field definition clauses into DATATRIEVE terminology and puts those clauses into a DATATRIEVE DEFINE RECORD command.

  You cannot extract and edit a record definition defined using the CDD/Repository Common Data Dictionary Data Definition Language's (CDDL) VARIANT field unless each VARIANT field has a STRUCTURE statement. If the CDDL record definition includes a STRUCTURE field description statement for each VARIANT field, you can extract and edit the record definition.
- When you invoke the command file, each DELETE command in that file causes DATATRIEVE to delete from the data dictionary the definition of any dictionary object with the same dictionary path name as that of the extracted dictionary object.
- When you invoke the command file, each DEFINE command in it enters its definition in the directory determined by the dictionary path name of the object specified in the command.

  If the path name in the command is a full dictionary path name, DATATRIEVE enters the definition in the specified directory, whether or not it happens to be your default dictionary directory.

If the path name in the command is a relative dictionary path name, DATATRIEVE first checks whether the path name is valid relative to your default dictionary directory. If it is valid, DATATRIEVE enters the definition in the appropriate directory. If it is not valid, DATATRIEVE displays an error message and the definition is not entered into any CDD/Repository directory.

- If you omit a field in the file specification, DATATRIEVE uses the defaults listed in Table 4–15:

Table 4–15 Output File Specification Defaults

Field	Default
node-spec::	Your local node
device:	Your default device
[directory]	Your default directory
file-name	Null string
.type	.COM
;version	1 or next higher version number

The minimum file specification consists of a period (.). The specification of such a file stored in your default VMS directory ends with .;n, where n is the version number and both the file name and the type are null strings.

Usage Notes

- You can use the EDIT command in place of EXTRACT to edit dictionary objects. If SET NO EDIT_BACKUP is in effect, EDIT enters a DELETE command to delete the old definition from the data dictionary and places the old definition into the main buffer of the editor in the form of a REDEFINE command. This allows you to modify the old definition. Essentially, EDIT combines the EXTRACT, DELETE, and REDEFINE commands. If SET EDIT_BACKUP is in effect, the old definition is retained in an earlier version.
- DATATRIEVE extracts object types in the order you specify. In the following example, the record object definitions are extracted before the domain object definitions:

```
DTR> EXTRACT ALL RECORDS, DOMAINS ON SAMPLE.DTR
```

- The EXTRACT command lets you modify dictionary objects and redistribute data definitions in the data dictionary without ending your DATATRIEVE session and without having to work directly on the data dictionary with DMU or CDO.

Without ending your DATATRIEVE session, you can use this method to manipulate your data descriptions and minimize the risk of corrupting the data dictionary:

- Extract the definitions you want to change.
- Edit the definitions in the command file with the editor.
- Invoke the command file to replace the old definitions.

See the section on the EDIT command in this chapter and the *VAX DATATRIEVE User's Guide* for details about using the editor to change command files.

- The EXTRACT command enables you to create backup copies of your CDD/Repository objects in a file that you specify.
- The EXTRACT command enables you to transport the definitions to other CDD/Repository directories.
- The EXTRACT command enables you to convert domains and records from DMU format to CDO format.
- To invoke the command file, type an at sign (@) and the command file name in response to the DTR> prompt. If the file type is not .COM, you must give the file type when entering the file specification.
- When you are moving a definition from one dictionary directory to another, use a SHOW path-name command to see if an object exists in the new directory with the same given name as that in the DELETE command or with the same path name as the one specified in the REDEFINE command.

  If one exists, you must decide whether or not you want it deleted when you invoke the command file. If you do not want that object deleted, you must edit the command file to change one or both of the names in the DELETE and REDEFINE commands.

  Be sure to anticipate the problems that might arise when you change default dictionary directories. No two objects in the same dictionary directory can have the same name, but if you shift to a new default dictionary directory, that new directory may legitimately contain an object with the same given name or the same relative path name (especially if relative path name in the command file contains a minus sign to indicate "the next level up" in the CDD/Repository hierarchy).

  For example, you extract the definition of a procedure ABC from the directory CDD$TOP.WIDGETS. You then set your default directory to CDD$TOP.THINGS, which contains a domain table called ABC. If you invoke the command file, the DELETE ABC command in the command file deletes

the domain table definition from CDD$TOP.THINGS, and the REDEFINE DOMAIN ABC command enters the procedure definition.

- When you extract a domain definition defined with the RELATIONSHIPS clause, you might have to edit the definition before you move it to another dictionary directory. You will need to edit the definition if you want the new domain definition to refer to a different record path name than the original definition. This is because the domain has the full record path name, including the version number, stored in it.

  For example, if you move a domain and all the associated records from one dictionary directory to another, you will need to edit the domain and change the record definitions to match the new ones.

- When you extract a record that contains one or more FROM clauses, the FROM clause will be extracted exactly as it was defined in the record. Therefore, a FROM clause that contains a full directory specification is more transportable than a FROM clause that relies on a specific default dictionary directory. In either case, you may need to edit the record definition if you move it from one dictionary directory to another.

- To invoke a command file produced by using the EXTRACT command on an object in the DMU format dictionary, you must have the following privileges:

  – P (PASS_THRU) and X (EXTEND) to the parent directory

  – P (PASS_THRU) and either D (LOCAL_DELETE) *or* G (GLOBAL_ DELETE) access to the object named in the DELETE command

- To invoke a command file produced by using the EXTRACT command on an object in the CDO format dictionary, you must have the following privileges:

  – S (SHOW) and U (CHANGE) to the parent directory

  – S (SHOW), U (CHANGE + DEFINE), and D (DELETE) access to the object named in the DELETE command

- To invoke the command file, you must have VMS R (read) access privilege to it. If you create the file, you are the owner of it and have R (read), W (write), E (execute), and D (delete) access to the command file, or whatever you or your LOGIN.COM file establish as the default file protection setting for your process.

- The EXTRACT command does not copy the access control list (ACL) of the dictionary object. After you invoke the command file, the ACL for the dictionary object is the default ACL assigned by CDD/Repository . Use the DEFINEP command to add other entries to the ACL.

- When you execute the command file, the existing object and its associated ACL may be deleted before the REDEFINES command can execute and propagate a new definition. Therefore, you may want to remove the DELETE command from the command file.

Example

Extract all the definitions in a dictionary directory to create a backup file:

```
DTR> EXTRACT ALL ON BAKUP1
DTR>
```

FIND Statement

Establishes a collection of records from a domain, view, collection, or list. The collection formed with the FIND statement becomes the current collection.

Format

FIND rse

Arguments

rse
Is a record selection expression specifying the records to be included in the collection.

Restrictions

- The domains containing the records specified in the RSE must be readied for READ, WRITE, or MODIFY access. The source domain cannot be readied for EXTEND access (see the section on the READY command.)
- The source domain cannot be a port.
- Do not use a FIND statement in a compound statement, such as a BEGIN-END, FOR, REPEAT, or THEN statement. (To establish single record context in compound statements, use the RSE clauses of FOR, PRINT, or MODIFY statements. These RSEs can establish the record streams needed to control the name recognition and single record context inside compound statements.)
- The following restrictions apply to the number of records you can include in a collection:
  - The maximum number of RMS records that can be included in a collection is 1,301,264. Only half this many are allowed when you create collections by crossing two sources. This is because each record created by a CROSS operation is considered two records. A collection created by crossing three sources can contain only a third of 1,301,264 records as a maximum (each record in the collection is made up of three records, one from each source), and so on.

    Because a collection is formed when a SORT is performed, 1,301,264 is also the maximum number of RMS records that can be sorted from within DATATRIEVE.

  - The maximum number of VAX DBMS records that can be included in a collection is approximately 979,776 records. The maximum is half this number in a collection created by crossing two sources, one-third this number in a collection involving three VAX DBMS sources, and so on.
  - The maximum number of relational database records that can be included in a collection from a single relation is 979,776 records. The maximum is half this number in a collection from an Rdb/VMS or Rdb/ELN view relation that involves two relations or a collection created by crossing two relational sources. The maximum is one-third this number in collections created by crossing three relations, and so on.

- You cannot use the FIND statement to find a record by its position. For example, FIND nth YACHTS does not work. You can use FIND FIRST n YACHTS and then use the SELECT statement with the LAST argument.

Results

- When you issue a FIND statement, DATATRIEVE forms a current collection. If you specify a context variable in the RSE, the collection has two names: CURRENT and that of the context variable.
- When the collection has been formed, DATATRIEVE displays the following message (n is the number of records in the collection):

  ```
  [n records found]
  ```

  If you put the FIND statement inside a procedure, DATATRIEVE does not display this message unless the FIND statement is the last one in the procedure.

  If you put the FIND statement on the same input line with other DATATRIEVE statements or commands, using semicolons (;) to separate them, DATATRIEVE does not display this message unless the FIND statement is the last one on the line.
- DATATRIEVE collects the records in the order it finds them in the data file. Do not assume there is any order to the collection unless you include the SORTED BY clause in the record selection expression or unless the domain uses an indexed file.

Examples

The following example forms a collection of yachts longer than 30 feet and gives the collection the name BIG-ONES:

```
DTR> FIND BIG-ONES IN YACHTS WITH LOA GT 30
[57 records found]
DTR>
```

The following example forms a collection of the 10 most expensive yachts:

```
DTR> FIND FIRST 10 YACHTS SORTED BY DESC PRICE
[10 records found]
DTR>
```

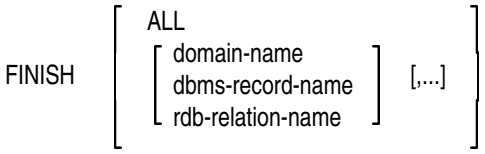
FINISH Command

Ends your access to domains, domain tables, relations, and VAX DBMS records. The FINISH command also releases any collections associated with the domains.

For VAX DBMS, when the last VAX DBMS domain or record is finished, collections are purged, a commit with no retention is performed, and databases are unbound. The last FINISH commits all VAX DBMS databases. A commit is not done until you finish the last VAX DBMS record or domain.

For relational sources, when the last domain or relation is finished, a commit is performed and no collections are retained.

Format



Arguments

ALL

Ends your control over all readied domains, relations, VAX DBMS records, and all domain or dictionary tables loaded in your workspace.

domain-name

Is the given name of a readied domain or domain table you want to finish. If you specify the names of more than one domain or domain table, separate each from the next with a comma (,).

rdb-relation-name

Is the given name of a readied relation you want to finish. If you specify the names of more than one relation, separate each from the next with a comma (,).

dbms-record-name

Is the given name of a VAX DBMS record readied with the READY database command. If you specify the names of more than one record, separate each from the next with a comma (,).

Restrictions

- You must enter the FINISH command at DATATRIEVE command level.
- You cannot use a full or relative dictionary path name when you specify the name of a domain or domain table.
- You cannot specify a database name.

Results

- If you use the keyword ALL or do not specify any domains, relations, VAX DBMS records, or domain tables in the FINISH command, DATATRIEVE ends your control over all readied domains, relations, VAX DBMS records, and all domain or dictionary tables loaded in your workspace.
- If you specify one or more domains, relations, records, or domain tables with the FINISH command, DATATRIEVE ends your control over only those specified.
- DATATRIEVE releases all collections associated with the domains, relations, and records you finish.
- The FINISH command can commit changes made to a VAX DBMS or relational database. A COMMIT statement executes when you finish the last readied domain, relation, or record, or when you finish all of them at once.

Usage Notes

- With the FINISH command, you can clear your workspace of unneeded domains, relations, records, and tables.
- With the FINISH command, you can end your access to the data associated with a readied domain, relation, or VAX DBMS record. For example, if you ready a domain for exclusive use, no one else can get access to the data file associated with that domain until you use the FINISH command or ready the domain again with a less restrictive access control option.
- You do not need to issue a FINISH command before an EXIT command. EXIT automatically finishes all readied domains and all domain tables, releases all collections and dictionary tables, and commits VAX DBMS and relational sources if necessary.

- If you redefine the format of the record associated with a readied RMS domain, the change in the record definition does not take effect until you use the FINISH command to finish the domain and the READY command to ready it again. Simply readying the domain again does not activate the new record definition.
- If you delete from the data dictionary the domain or record definition associated with a readied domain, you can continue to work with the domain, performing any operations consistent with the mode of your access to the domain. You can also ready the domain again to change your access mode to it, as long as you avoid any conflicts of access control options (see the section on the READY command.) Under these circumstances, however, when you use the FINISH command to end your control over the domain, the domain definition is no longer in the data dictionary, and you can no longer get any access to the domain.
- If you delete a domain table definition from the data dictionary, you can continue to use the domain table. However, when you use the FINISH or RELEASE command to end your control over the domain table, the definition of the domain table is no longer in the data dictionary, and you can no longer get any access to the domain.
- If you have more than one domain linked to a DECforms form, finishing a single domain will not disable the form session. You disable the session only when you issue the FINISH command on the last domain linked to the form.

Example

The following example releases control of the domain YACHTS:

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

DTR> FINISH YACHTS
DTR> SHOW READY
No ready sources.
No loaded tables.
DTR>
```

FOR Statement

Causes DATATRIEVE to execute a statement or group of statements once for each record in the record stream formed by a record selection expression (RSE). The FOR statement provides repeating loops for DATATRIEVE operations.

Format

FOR rse statement

Arguments

rse

Is a record selection expression that forms the record stream that controls the number of times DATATRIEVE executes the statement and controls the single-record context for the statement. See the *VAX DATATRIEVE User's Guide* for a discussion of DATATRIEVE context.

statement

Is either a simple or a compound statement you want DATATRIEVE to execute once for each record in the record stream formed by the RSE. You can form compound DATATRIEVE statements with the BEGIN-END, IF-THEN-ELSE, and THEN statements, which are described in this chapter.

Restrictions

- Each domain associated with the record selection expression must be readied for READ, WRITE, or MODIFY access.
- Do not include FIND, SELECT, SORT, DROP, or RELEASE statements in a FOR statement.
- You cannot include a DATATRIEVE command in a FOR statement.

Results

- For each record in the record stream, DATATRIEVE executes the statement once, unless an ABORT statement, a CTRL/C, or a CTRL/Z to an "Enter...:" prompt forces an early end to the repetitions of the statement.
- For each record in the record stream, DATATRIEVE executes the statements of a BEGIN-END block in the order you entered them.
- Each time DATATRIEVE executes the statement in the FOR loop, it establishes a single record context for one record from the record stream.

However, a statement nested in a BEGIN-END block in a FOR loop can create its own single record context. The new context lasts until DATATRIEVE completes the execution of the nested statement. The single record context then returns to its state before the execution of the statement.

Similarly, when DATATRIEVE completes the execution of the FOR loop, the single record context returns to its state prior to the execution of the FOR loop.

Usage Notes

- Use the FOR statement to repeat sequences of DATATRIEVE statements. You do not have to know how many records the record stream contains to control the number of times DATATRIEVE loops through the statement. You can use an ABORT statement in the statement to end the loop when certain conditions are met, regardless of the number of records remaining in the record stream.
- You can nest FOR statements. You can use nested FOR loops to work with lists (the repeating fields contained in hierarchical records).

  You establish the single record context in the outer loop, and the single list-item context in the inner loop. The statement acts on all list items in one target record before acting on any in the next record in the record stream.

  You can, however, use the CROSS clause in record selection expressions to simplify work with lists (see the chapter on RSE in the *VAX DATATRIEVE User's Guide*).
- Inside a FOR loop, you can force an exit from the loop by specifying the exit conditions in the IF clause of an IF-THEN-ELSE statement and putting an ABORT statement in the THEN clause.

  You can also stop the execution of a FOR loop by entering a CTRL/C during the execution of a statement, or you can enter a CTRL/Z in response to an "Enter . . . :" prompt.
- Inside the FOR loop, you can use a variable as a counter to force an exit from the loop before all records in the record stream have been acted upon:

  1. Declare the variable outside the FOR loop.
  2. Inside the loop, increase the value of the variable by 1 during each repetition of the loop.

3. Put an ABORT statement in the THEN clause or the ELSE clause of an IF-THEN-ELSE statement, and specify the conditions for the exit, based on the value of the variable, in the conditional expression of the IF clause.

Examples

The following example assigns a value to the field PRICE for three yachts with prices equal to zero:

```
DTR> READY YACHTS MODIFY
DTR> SET NO PROMPT
DTR> FIND FIRST 3 A IN YACHTS WITH PRICE = 0
[3 records found]
DTR> PRINT A

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 BLOCK I.    40         SLOOP   39    18,500  12
 BUCCANEER   270        SLOOP   27     5,000  08
 BUCCANEER   320        SLOOP   32    12,500  10

DTR> FOR A
CON>MODIFY USING PRICE = DISP * 1.3 + 5000
DTR> PRINT A

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 BLOCK I.    40         SLOOP   39    18,500  12  $29,050
 BUCCANEER   270        SLOOP   27     5,000  08  $11,500
 BUCCANEER   320        SLOOP   32    12,500  10  $21,250

DTR>
```

The following example uses a variable to force an end to a FOR loop before all records in the record stream have been acted upon:

```
DTR> READY YACHTS
DTR> DECLARE A PIC 9.
DTR> PRINT A

A

0
```

FOR Statement

```
DTR> SET NO PROMPT
DTR> FOR YACHTS
CON> BEGIN
CON>   A = A + 1
CON>   PRINT A, BOAT
CON>   IF A = 5 THEN ABORT "END OF LOOP"
CON> END

                                  LENGTH
                                   OVER
A MANUFACTURER   MODEL     RIG     ALL   WEIGHT BEAM  PRICE

1  ALBERG      37 MK II   KETCH   37    20,000  12  $36,951
2  ALBIN       79         SLOOP   26     4,200  10  $17,900
3  ALBIN       BALLAD     SLOOP   30     7,276  10  $27,500
4  ALBIN       VEGA       SLOOP   27     5,070  08  $18,600
5  AMERICAN    26         SLOOP   26     4,000  08   $9,895
ABORT: END OF LOOP

DTR>
```

The following example uses nested FOR loops to increase by one year the age of each child in the first two records of the domain FAMILIES:

```
DTR> READY FAMILIES MODIFY
DTR> PRINT FIRST 2 FAMILIES

                      NUMBER    KID
  FATHER     MOTHER    KIDS     NAME    AGE

JIM         ANN           2   URSULA      7
                              RALPH       3
JIM         LOUISE        5   ANNE       31
                              JIM        29
                              ELLEN      26
                              DAVID      24
                              ROBERT     16

DTR> SET NO PROMPT
DTR> FOR FIRST 2 FAMILIES
CON>   FOR KIDS
CON>     MODIFY USING AGE = AGE + 1
DTR> PRINT FIRST 2 FAMILIES

                      NUMBER    KID
  FATHER     MOTHER    KIDS     NAME    AGE

JIM         ANN           2   URSULA      8
                              RALPH       4
JIM         LOUISE        5   ANNE       32
                              JIM        30
                              ELLEN      27
                              DAVID      25
                              ROBERT     17
```

HELP Command

Provides on-line information about the use of DATATRIEVE commands, statements, and language elements.

Format

{ HELP | ? } [ERROR] [topic][...]

Arguments

ERROR

Provides additional information on the last error message that you received or on any other error that you specify.

topic

Is a DATATRIEVE command, statement, statement element, or error. Use commas to separate each topic from the next.

. . . (ellipsis)

Indicates that all help subtopics and any of their subtopics should be displayed.

? (question mark)

Is a synonym for HELP.

Note

If you enter HELP HELP, DATATRIEVE displays an explanation of the HELP command.

Restrictions

- You must issue the HELP command at DATATRIEVE command level in response to the DTR> prompt.
- When you use DECLARE SYNONYM to customize DATATRIEVE language elements, you cannot display the appropriate help message if you specify a synonym with the HELP command. You can customize the DATATRIEVE Help library to accommodate any site-specific synonyms or shared user-defined functions.

Results

- DATATRIEVE displays the help message for each topic you list. If prompting for help is in effect, you receive the help prompt for help topics or subtopics. For more information, issue the command HELP SET HELP.
- The help messages for each command or statement contain on-line information about optional arguments associated with the command or statement.
- On VT100-type video terminals, help messages can appear in a window of the screen. For more information, issue the command HELP VIDEO or HELP SET HELP.
- In most circumstances, after DATATRIEVE has displayed an error message, you can issue the HELP ERROR command to have DATATRIEVE display the help text pertaining to the error message.

Usage Notes

- You can write your own Help messages for the use of specific domains and procedures, site-specific synonyms, user-defined functions, and any other topic you find appropriate. DATATRIEVE uses the VMS Librarian Utility to access the help messages. Refer to the chapter on the Librarian Utility in the *VAX DATATRIEVE Guide to Programming and Customizing* for information and instructions on creating and maintaining a private help library.
- If you are running DATATRIEVE in a DECwindows environment, you can get help on the objects displayed in the main application window by following these steps:

  1. Press MB1 and drag the pointer to the object on which you want help.
  2. Hold down the HELP key on the keyboard.
  3. Release MB1.

  For more information on obtaining help on DECwindows objects, see the *VAX DATATRIEVE User's Guide*.
- If you are running DATATRIEVE in a DECwindows environment, you can get the traditional DATATRIEVE help by issuing the HELP command at the DTR> prompt.
- For more information on obtaining the traditional DATATRIEVE help, see the *VAX DATATRIEVE User's Guide*.

Examples

The following example shows how to ask for a list of the help that is available:

```
DTR> HELP
```

The following example shows how to ask for help on the screen-oriented help facility:

```
DTR> HELP VIDEO
```

The following example shows how to ask for help on the last error message you received. For example, if you use the SORT statement without first forming a collection, DATATRIEVE displays an error message:

```
DTR> READY YACHTS
DTR> SORT BY LOA
No collection for sort.
```

You can issue the HELP ERROR command to find out the reason for the error message, possible actions to correct the error, and how to obtain more on-line information:

```
DTR> HELP ERROR
No collection for sort.

ERROR

  NOCOLSOR

    EXPLANATION:

    You can only use the SORT statement with a collection.

    USER ACTION:

    Form a collection with a FIND statement.   You can sort the
    collection by  including a SORTED BY  clause in  the RSE of
    the FIND statement.  Or you can  use a SORT statement after
    the collection is established.

    FOR MORE INFORMATION TYPE:

    HELP SORT
    HELP RSE

Topic?
```

In response to the "Topic?" prompt, you can type SORT or RSE for more information on these topics. Or you can press the RETURN key to return to the DATATRIEVE command level. At the DATATRIEVE command level, you can type HELP SORT or HELP RSE for more information on these topics.

HELP Command

The following example shows how to ask for help on adjusting the window for help on video terminals:

```
DTR> HELP SET HELP
```

The following example shows how to get help on the FIND command:

```
DTR> HELP FIND
```

IF THEN ELSE Statement

Causes DATATRIEVE to execute one of two statements or compound statements, depending on the evaluation of a conditional (Boolean) expression.

Format

```
IF  boolean-expression  [THEN] statement-1 [ELSE statement-2]
```

Arguments

boolean-expression
Is a Boolean expression. (See Chapter 1).

THEN
Is an optional language element you can use to clarify syntax.

statement-1
Is a simple or compound statement you want DATATRIEVE to execute if the Boolean expression evaluates to true.

ELSE statement-2
Specifies the statement you want DATATRIEVE to execute if the Boolean expression evaluates to false.

Restriction

You must observe all restrictions on the statements used in the IF-THEN-ELSE statement.

Results

- If the Boolean expression evaluates to true, DATATRIEVE executes statement-1 in the THEN clause.
- If you specify an ELSE clause and the Boolean expression evaluates to false, DATATRIEVE executes statement-2 in the ELSE clause.
- If you do not specify an ELSE clause and the Boolean expression evaluates to false, DATATRIEVE does not execute statement-1 in the THEN clause and is ready to execute the next command or statement it encounters.

Usage Notes

- You can press the RETURN key before all elements of the statement except ELSE. If you press the RETURN key before typing ELSE, DATATRIEVE considers the syntax of the statement complete. It executes or ignores the THEN clause, depending on the evaluation of the Boolean expression. It then tries to execute the ELSE clause as though it were a separate statement and displays an error message.

  When you have to break lines in an IF-THEN-ELSE statement, put ELSE at the end of a line rather than at the beginning of the next. This practice is especially important when you are writing a procedure, because DATATRIEVE does not check the syntax of the statements in the procedure until you invoke it.

- You can use an IF-THEN-ELSE statement to force an exit from a BEGIN-END block or from a FOR loop. Put an ABORT statement in either the THEN clause or the ELSE clause, and put the resulting IF-THEN-ELSE statement in an appropriate place in the BEGIN-END block or FOR loop.

- You can nest IF-THEN-ELSE statements by using an IF-THEN-ELSE statement as either statement-1 or statement-2, or both.

Examples

The following example shows how to print each yacht built by Pearson, and modify the price if you want to:

```
DTR> SET NO PROMPT
DTR> READY YACHTS WRITE
DTR> FOR YACHTS WITH BUILDER = "PEARSON"
CON>    BEGIN
CON>         PRINT
CON>         IF *."Y TO MODIFY PRICE, N TO SKIP" CONT "Y"
CON>            THEN MODIFY PRICE ELSE
CON>            PRINT "NO CHANGE"
CON>         IF *."Y TO CONTINUE" NOT CONT "Y" THEN
CON>            ABORT "END OF PRICE CHANGES"
CON>    END

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

```
 PEARSON     10M         SLOOP   33    12,441  11
Enter Y TO MODIFY PRICE, N TO SKIP: N
NO CHANGE
Enter Y TO CONTINUE, N TO ABORT: Y
 PEARSON     26          SLOOP   26     5,400  08
Enter Y TO MODIFY PRICE, N TO SKIP: N
NO CHANGE
Enter Y TO CONTINUE, N TO ABORT: N
ABORT: END OF PRICE CHANGES

DTR>
```

In the following example the IF statement is used to select families with fathers named Jim and list the children in those families:

```
DTR> READY FAMILIES
DTR> FOR FAMILIES WITH ANY KIDS
CON>      IF FATHER EQ "JIM" THEN
CON>         PRINT "The Kids of JIM and"|||MOTHER,
CON>         ALL KID_NAME ("Kids with Fathers"/
CON>             "Named Jim") OF KIDS, SKIP

                               Kids with Fathers
                                   Named Jim

The Kids of JIM and ANN           URSULA
                                  RALPH

The Kids of JIM and LOUISE        ANNE
                                  JIM
                                  ELLEN
                                  DAVID
                                  ROBERT

DTR>
```

LIST Statement

Causes DATATRIEVE to format and write to your terminal, to a file, or to a unit record device one or more values of implied or specified fields from records in one or more readied domains.

Format

For retrieving from selected records and target record streams formed by FOR loops:

```
LIST  [print-list]  [ ON { file-spec   } ]
                     { *.prompt }
```

For retrieving from the current collection:

```
LIST  ALL  [print-list]  [ ON { file-spec   } ]
                          { *.prompt }
```

For retrieving from record streams formed by the LIST statement using one RSE:

```
LIST  [print-list  OF]  rse  [ ON { file-spec   } ]
                              { *.prompt }
```

For retrieving from record streams formed by the LIST statement using two RSEs (the inner print list follows another print list):

```
LIST   print-list,  ALL  print-list  OF  rse-1  [,print-list]  OF  rse-2
       [ ON { file-spec   } ]
            { *.prompt }
```

For retrieving from record streams formed by the LIST statement using two RSEs (the inner print list precedes any other print list):

```
LIST  ALL  ALL  print-list  OF  rse-1  [,print-list]  OF  rse-2
      [ ON { file-spec   } ]
           { *.prompt }
```

Arguments

print-list
Is a list of field names; it can also include only the print list elements SKIP, NEW_PAGE, and inner print lists. See Table 4–24, in the PRINT command section, for a description of these two print-list elements. You can control the format in which DATATRIEVE displays values from the specified fields with the USING edit-string modifiers described in the Results section of the the PRINT command section.

ALL
When used alone following LIST, causes the records in the current collection to be displayed or written to the specified file or device.

When used with a print list, ALL causes the print list to be evaluated for each record in the current collection.

When used with the OF rse clause, ALL is optional. You can use it to clarify the LIST statement, but, regardless of the presence of ALL, DATATRIEVE evaluates the LIST statement once for each record in the record stream formed by the RSE.

When the print list begins with an inner print list, ALL is required to establish the proper context in which to resolve references to the items in the hierarchical list.

rse
Is a record selection expression that creates the record stream DATATRIEVE uses to evaluate the elements of the print list.

file-spec
Is the file specification to which you want to write the output of the statement. A complete file specification has the following format:

```
node-spec::device:[directory]file-name.type;version
```

If you omit a field in the file specification, DATATRIEVE uses the defaults listed in Table 4–16.

Table 4–16 Output File Specification Defaults

Field	Default
node-spec::	Your local node

(continued on next page)

Table 4–16 (Cont.) Output File Specification Defaults

Field	Default
device:	Your default device
[directory]	Your default directory
file-name	Null string
.type	.LIS
;version	1 or next higher version number

The minimum file specification consists of a period (.). The specification of such a file stored in your default VMS directory ends with ".;n", where n is the version number and both the file name and the type are null strings.

***.prompt-name**
Is a prompting value expression that prompts you for a device name or file specification to which you want to write the output of the statement.

Restrictions

- To list data from records in a domain, you must ready the domain for READ, WRITE, or MODIFY access. You cannot list data from domains readied for EXTEND access because you cannot establish collections or record streams from domains readied for EXTEND access. See the section in this chapter on the READY command for more information.
- If you specify a device name in a LIST statement, the device must be one to which you have access, such as a line printer, a tape drive, your own terminal, or another terminal. You cannot cause DATATRIEVE to display the output of the LIST statement on another terminal that is logged in.
- To send your output to a tape drive, you must mount your tape and assign the tape drive to your process at DCL command level before you run DATATRIEVE. Only then can you specify a tape drive as the output device for the LIST statement.
- When you use ON LP: to send LIST statement output directly to a line printer, DATATRIEVE assumes your system has a line printer. If your system does not have a device defined as LPA0:, using the clause ON LP: does not work.

  Although this restriction applies to any system without a line printer, you may encounter it unexpectedly if your system is part of a VAXcluster with a common line printer. The ON LP: clause does not work in a VAXcluster that uses a common printer not directly connected to your system.

If the nodes in the cluster are connected with DECnet, you can work around this restriction. To send output from a node without a line printer, you must include the node name of the system with the line printer in the LP: specification. For example, if the cluster's line printer is on a node named BIGVAX, the following list statement sends output to it:

```
DTR> LIST YACHTS ON BIGVAX::LP:
```

Note that you cannot directly specify the line printer on BIGVAX by using the cluster device name BIGVAX$LPA0: in the ON clause.

Results

- The LIST statement causes DATATRIEVE to display each field and its value on one line. The value displayed is the one in the field of the record in the single record context. DATATRIEVE evaluates the LIST statement once for the selected record, or once for each record in the CURRENT collection or in the record stream formed by a FOR statement or the OF rse clause of the LIST statement.

  For record streams or collections containing more than one record, DATATRIEVE inserts a blank line in the display between the blocks of fields that derive from the evaluation of the records in the record stream.

- If you do not include ALL or an RSE in a LIST statement, DATATRIEVE evaluates the print list once in the context of the nearest selected record and creates one or more lines of output, depending on the number of fields and the type and number of formatting options you specify.

- If you specify the argument ALL and do not include an RSE, DATATRIEVE uses the data in the records of the CURRENT collection to evaluate the value expressions in the print list. DATATRIEVE evaluates the print list once for each record in the CURRENT collection and creates one or more lines of output for each record depending on the number of fields and the type and number of formatting options you specify.

- If you include an RSE in a LIST statement, DATATRIEVE uses the data in each record in the record stream to evaluate the print list. DATATRIEVE evaluates the print list once for each record in the record stream and creates one or more lines of output for each record depending on the number of fields and the type and number of formatting options you specify.

- If you put a LIST statement in a FOR loop, DATATRIEVE uses the data in each record in the record stream created by the RSE in the FOR statement. DATATRIEVE evaluates the print list once for each record and creates one or more lines of output for each record, depending on the number of fields and the type and number of formatting options you specify.

- If you do not put the LIST statement in a FOR loop and do not include an RSE or the argument ALL, DATATRIEVE uses the data from the selected record in the nearest single record context to evaluate the print list. DATATRIEVE evaluates the print list once and creates one or more lines of output, depending on the number of fields and the type and number of formatting options you specify.
- If you end your file specification with the name of a line printer or another terminal that is not a spooled device, the output of a LIST statement can be immediately displayed on the device. Consult the VMS documentation set for details regarding spooled devices.

Usage Notes

- You can use inner print lists to output hierarchical displays of data contained in the lists of variable-length records.
- With SET SEARCH in effect, you can print the data in lists by letting DATATRIEVE generate implicit inner print lists.

Examples

The following example lists three records from YACHTS:

```
DTR> READY YACHTS
DTR> LIST FIRST 3 YACHTS

MANUFACTURER    : ALBERG
MODEL           : 37 MK II
RIG             : KETCH
LENGTH_OVER_ALL : 37
DISPLACEMENT    : 20,000
BEAM            : 12
PRICE           : $36,951

MANUFACTURER    : ALBIN
MODEL           : 79
RIG             : SLOOP
LENGTH_OVER_ALL : 26
DISPLACEMENT    :  4,200
BEAM            : 10
PRICE           : $17,900

MANUFACTURER    : ALBIN
MODEL           : BALLAD
RIG             : SLOOP
LENGTH_OVER_ALL : 30
DISPLACEMENT    :  7,276
BEAM            : 10
PRICE           : $27,500
```

The following example lists the first two records in FAMILIES:

```
DTR> READY FAMILIES
DTR> LIST FIRST 2 FAMILIES

FATHER      : JIM
MOTHER      : ANN
NUMBER_KIDS :  2
    KID_NAME    : URSULA
    AGE         :  7
    KID_NAME    : RALPH
    AGE         :  3

FATHER      : JIM
MOTHER      : LOUISE
NUMBER_KIDS :  5
    KID_NAME    : ANNE
    AGE         : 31
    KID_NAME    : JIM
    AGE         : 29
    KID_NAME    : ELLEN
    AGE         : 26
    KID_NAME    : DAVID
    AGE         : 24
    KID_NAME    : ROBERT
    AGE         : 16

DTR>
```

MATCH Statement

Relates two list names with their subordinate elementary fields, so that data from the second list can be stored in the first list.

Format

MATCH list-rse-1, list-rse-2 statement

Arguments

list-rse-1
Is an RSE containing a list name from the record definition of the domain that is to receive the data.

list-rse-2
Is an RSE containing a list name from the record definition of the source domain for the data.

statement
Is a DATATRIEVE Assignment statement or series of Assignment statements enclosed by a BEGIN-END block.

Restrictions

- The MATCH statement should be used only when you need to reorganize the structure of lists — for example, when you want to restructure two list fields into one list field.
- In each Assignment statement, the expression to the left of the equal sign (=) must be an elementary field subordinate to the list name in list-rse-1. The expression to the right of the equal sign must be an elementary field subordinate to the list name in list-rse-2.

Result

DATATRIEVE associates the left field name in each Assignment statement with the list name list-rse-1. DATATRIEVE associates the right field name in each Assignment statement with the list name in list-rse-2. When the MATCH statement is included within a STORE statement, DATATRIEVE stores data from the second list into the first list.

Usage Note

Use the MATCH statement to transfer data from one list to another. The MATCH statement can be part of a BEGIN-END block in the USING clause of a STORE statement, associating list names and elementary field names. The BEGIN-END block can also include Assignment statements for data transfer between fields that are not part of lists.

Use the following compound statement to reorganize data from one hierarchical domain to another:

```
FOR domain-name-1
  STORE domain-name-2 USING statement
```

The records of the first domain are the source of the data for the second domain. The first domain can be a view domain.

Example

Consider the following record definition for the domain FAM:

```
DTR> SHOW FAM_REC
RECORD FAM_REC USING
  01 FAMILY.
    03 PARENTS.
       06 FATHER PIC X(10).
       06 MOTHER PIC X(10).
    03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
    03 KIDS_N OCCURS 10 TIMES.
       06 EACH_KID.
           09 KID_NAME PIC X(10) QUERY_NAME IS KID.
    03 KIDS_A OCCURS 10 TIMES.
       06 EACH_KID.
           09 AGE PIC 99 EDIT_STRING IS Z9.
;
```

The fixed-length list format means that values are displayed for 10 KIDS_N and 10 KIDS_A, no matter what the value for NUMBER_KIDS. In displays and reports, this means that most FAM records would be separated by strings of blanks (for “empty” occurrences of KIDS_N) and zeros (for “empty” occurrences of KIDS_A).

```
DTR> PRINT FAM

                      NUMBER    KID
  FATHER     MOTHER    KIDS     NAME    AGE
```

```
JIM        ANN           2   URSULA      7
                             RALPH       3
                                         0
                                         0
                                         0
                                         0
                                         0
                                         0
                                         0
JIM        LOUISE        5   ANNE       31
                             JIM        29
                             ELLEN      26
                             DAVID      24
                             ROBERT     16
                                         0
                                         0
                                         0
                                         0
                                         0
          .          .          .
          .          .          .
          .          .          .
```

To improve display and report formats for this domain, you could restructure it using the MATCH statement so that it contains one variable-length list field. In other words, the modified record definition would look like the current one for the FAMILIES domain.

```
DTR> SHOW FAMILY_REC
RECORD FAMILY_REC
01 FAMILY.
   03 PARENTS.
      06 FATHER PIC X(10).
      06 MOTHER PIC X(10).
   03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
   03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
      06 EACH_KID.
         09 KID_NAME PIC X(10) QUERY_NAME IS KID.
         09 AGE PIC 99 EDIT_STRING IS Z9.
;
```

If you attempted to restructure FAM without using the MATCH statement, DATATRIEVE would not store the list elements. This problem occurs because the modified record definition combines two list fields into one. Therefore, to restructure the FAM domain, you must use the MATCH statement within a STORE statement that is controlled by a FOR loop. The following series of statements stores the data from the records in FAM into the records in FAMILIES and then prints the results:

```
DTR> READY FAM
DTR> DEFINE FILE FOR FAMILIES
DTR> READY FAMILIES WRITE
DTR> FOR FAM
CON> STORE FAMILIES USING
CON>   BEGIN
CON>     PARENTS = PARENTS
CON>     NUMBER_KIDS = NUMBER_KIDS
CON>     MATCH KIDS, KIDS_N
CON>       KID_NAME = KID_NAME
CON>     MATCH KIDS, KIDS_A
CON>       AGE = AGE
CON>   END
DTR> PRINT FAMILIES

                       NUMBER    KID
  FATHER     MOTHER     KIDS     NAME    AGE

JIM         ANN           2    URSULA      7
                               RALPH       3
JIM         LOUISE        5    ANNE       31
                               JIM        29
                               ELLEN      26
                               DAVID      24
                               ROBERT     16
           .          .          .
           .          .          .
           .          .          .
```

MISSING VALUE Clause

Designates a value for a field that DATATRIEVE recognizes, not as the literal value, but as a marker that no value is stored in the field. DATATRIEVE ignores fields containing the "missing value" marker when evaluating statistical expressions (AVERAGE, MAX, MIN, TOTAL, and STD_DEV).

When you store a record and do not directly assign a value to a field with a MISSING VALUE defined, DATATRIEVE uses the missing value to initialize the field if it contains no DEFAULT VALUE clause.

Format

```
MISSING   [VALUE [IS]] literal
```

Arguments

VALUE IS

Are optional keywords you can use to clarify the syntax of the clause.

literal

Is either a numeric or character string literal. (See Chapter 1 for a discussion of these two types of literals).

Restrictions

- This clause is valid only for elementary fields that are not OCCURS fields.
- The missing value for a field must be consistent with the data type for that field.

Results

- If part of an expression is missing, then the expression is missing. It will still have a value that is the result of the calculation, but it will be considered missing for assignments and edit string processing.
- The MISSING VALUE clause is ignored on fields or variables with the COMPUTED BY clause. The COMPUTED BY value is missing if the computed expression is missing. The EDIT_STRING clause is applied and may have a missing component.
- If the source value of an assignment is missing and the target field has a missing value clause, then the value specified in the missing value clause of the target field is stored in the target field.

Usage Notes

- When you create a record with Assignment statements in the USING clause of a STORE statement but make no assignment to a field that has a MISSING VALUE clause and no DEFAULT VALUE clause in its field definition, DATATRIEVE initializes the field with the missing value.
- When you create a record by responding to prompts from a STORE statement without a USING clause or from prompting value expressions in the USING clause of a STORE statement, DATATRIEVE does the following: If you respond to a prompt by pressing the TAB key once and pressing the RETURN key, DATATRIEVE initializes a field with the value specified in the MISSING VALUE clause.
- In the YACHT record definition, the definition of the field PRICE includes a MISSING VALUE clause designating that the missing value is zero. Having zero in the PRICE field means that no price was available when the record was stored, not that the boat is free.
- If you include a MISSING VALUE clause in a field definition, you can also include a MISSING VALUE edit string to control the output format when you retrieve the missing value from the field. The question mark (?) divides an edit string into two parts. The first part applies to the output of the field value if that value is not the missing value. The second part of the edit string applies if the field contains the designated missing value. Because you can include character string literals in edit strings, you do not have to display the actual value stored in the field. You can display a text message or a string of repeated characters for easy recognition of records with missing values.
- DATATRIEVE also has a relational operator you can use on fields with missing values to establish selected records and to form record streams and collections. The Boolean expression has this form:

  ```
  field-name MISSING.
  ```

  A record selection expression using this form of Boolean looks like this:

  ```
  YACHTS WITH PRICE MISSING.
  ```

  When DATATRIEVE does a statistical calculation on fields that contain missing values, it displays a message telling you the total number of records in the record stream or collection and the number of records in the collection that were used in the calculation.

Examples

The following example defines a record that contains MISSING VALUE clauses:

```
DTR> DEFINE DOMAIN THINGS USING THINGREC ON THINGS;
DTR> DEFINE RECORD THINGREC USING
DFN> 01 THINGS.
DFN>    03 NUM PIC 9(5)
DFN>       MISSING VALUE IS 11111
DFN>       EDIT_STRING IS ZZ,Z99?"***MISSING***".
DFN>    03 STR PIC X(10) MISSING VALUE IS "EMPTY"
DFN>       EDIT_STRING IS X(10)?"***MISSING***".
DFN> ;

DTR>
```

The following example defines a new domain based on YACHTS that uses a new missing value and a MISSING VALUE edit string:

```
DTR> DEFINE DOMAIN YACHTS_PRICE_LIST USING YPL_REC ON YPL.DAT;
DTR> DEFINE RECORD YPL_REC USING
DFN> 01 BOAT.
DFN>    03 TYPE.
DFN>       05 BUILDER PIC X(10).
DFN>       05 MODEL PIC X(8).
DFN>    03 PRICE PIC 9(5) MISSING VALUE IS 0
DFN>       EDIT_STRING $$$,$$$?"NOT LISTED".
DFN> ;
[Record is 23 bytes long.]
DTR> DEFINE FILE FOR YACHTS_PRICE_LIST KEY = TYPE
DTR> READY YACHTS_PRICE_LIST AS YPL WRITE
DTR> READY YACHTS
DTR> YPL = YACHTS WITH LOA GT 35
DTR> FIND YPL WITH PRICE MISSING
[12 records found]
DTR> PRINT FIRST 3 CURRENT

 BUILDER    MODEL     PRICE

BLOCK I.   40       NOT LISTED
CABOT      36       NOT LISTED
DOWN EAST  38       NOT LISTED

DTR>
```

MODIFY Statement

Changes the value of one or more fields in a selected record or in any or all records in a collection or record stream.

Format 1

```
MODIFY  [ALL]  [ field-name  [,...]
               [ USING  statement-1 ]

    [VERIFY  [USING]  statement-2]

    [OF  rse]
```

Format 2

```
MODIFY  [ALL]  rse

    USING  statement-1

    [VERIFY  [USING]  statement-2]
```

Arguments

ALL

Specifies that you want to modify either all records in the CURRENT collection or all records in the record stream specified in the record selection expression (rse).

field-name

Specifies the name of a field in the target records you want to modify. If you specify more than one field name, use a comma to separate each field name from the next. DATATRIEVE prompts you to supply a value for each field you specify.

USING statement-1

Specifies a simple or compound DATATRIEVE statement that assigns values to one or more fields in the target records you want to modify. This clause can also contain any other DATATRIEVE statements, such as PRINT, STORE, and other MODIFY statements.

VERIFY [USING] statement-2

Specifies a statement that DATATRIEVE executes before modifying the target record.

OF rse

Is a record selection expression that forms a record stream of the records you want to modify. An OF rse clause is optional in format 1 of the MODIFY statement. In format 2, an rse without OF is required.

Restrictions

- The domain containing the records you want to modify must be readied for MODIFY or WRITE access. See the section in this chapter on the READY command for more information.
- For data stored in an RMS indexed file, you cannot modify a primary key or an alternate key with the NO CHANGE attribute. (See the section on the DEFINE FILE command.)
- You cannot modify a COMPUTED BY field.
- You cannot modify list fields or their subordinates in hierarchical records in remote domains.
- When entering a complex MODIFY statement, you must use the hyphen continuation character if you want to press the RETURN key before typing the keywords USING, VERIFY, or OF rse. If you must break an input line near one of these keywords, you can also type the keyword at the end of a line and press the RETURN key.
- You cannot use a prompting value expression in a USING clause to assign a value to a group field.
- You must establish a context to specify the records on which the MODIFY statement acts. You can establish context in four ways:
  - By forming a collection with a FIND statement and using a SELECT statement to identify a selected record (see Table 4–17)
  - By forming a CURRENT collection with a FIND statement and specifying ALL (see Table 4–18)
  - By forming a record stream with an RSE within the MODIFY statement (see Table 4–19)
  - By using a FOR loop and forming a record stream with the rse in the FOR statement (see Table 4–20)
- If you use ALL without an OF rse clause in a MODIFY statement that is part of a FOR loop, all the records in the CURRENT collection are modified each time through the FOR loop. The result is a series of redundant changes to the records.

  You should also avoid similarly redundant loops when using the OF rse clause in MODIFY statements used in FOR loops. The OF rse clause should contain a Boolean expression that matches each record to be modified with one record from the record stream providing the data. (See the fourth example.)

- If you specify one or more field names in a MODIFY statement, they must be the names of group or elementary fields that DATATRIEVE can recognize in the context established for the MODIFY statement. If you specify more than one field name, use a comma to separate each field name from the next.
- If you specify a record stream with an OF rse clause in the MODIFY statement and omit the field list or the USING clause, you must include the keyword ALL: MODIFY ALL OF rse. This special case of the MODIFY statement syntax reminds you that this statement makes all the records in the record stream identical.
- If you want to use a form to display the records you are changing with the MODIFY statement, you can use one of two methods:
  - Include a FORM clause in the definition of the domain you wish to access. Then you must omit the field list and the USING clauses of the MODIFY statement. That is, you must retrieve entire records from the target collection or record stream.
  - Use the DISPLAY_FORM statement for FMS and TDMS forms, or the WITH_FORM statement for DECforms forms.

    Use the DISPLAY_FORM statement with the GET_FORM value expression to collect data from a form. With the DISPLAY_FORM statement, you can specify a field list in the USING clause of the MODIFY statement.

    Use the WITH_FORM statement with the SEND and RECEIVE clauses to send data to the form and receive it back modified.

  See the sections on DEFINE DOMAIN and DISPLAY_FORM in this chapter. SET FORM must be in effect when you ready the domain and when you enter the MODIFY statement. Check the status of the SET FORM/NO FORM setting with the SHOW SET_UP command.

Results

- When DATATRIEVE executes a MODIFY statement, it immediately changes the information in the data file.
- DATATRIEVE prompts you for field values with this message:

  ```
  Enter field-name:
  ```

- When DATATRIEVE prompts for a field value, you can enter any of the following responses:
  - To leave the value unchanged, press the TAB key and then the RETURN key.

  - To change the value of the field, type a new value, and then press the RETURN key. The new value should conform to the data description of the field, as defined in the record definition.
  - To change to spaces the value of an alphabetic or alphanumeric field, press the SPACE bar and then press the RETURN key.
  - To change to zero the value of a numeric field, enter a zero or press the SPACE bar and then the RETURN key.
  - To end the prompting cycle, press CTRL/Z. DATATRIEVE discards the changes made to the one record being modified when you enter the CTRL/Z. That record is unchanged, but if you have already modified records from the collection or record stream, those changes remain in effect. Each of those records was changed in the data file as soon as you entered your response to the last prompt for that record.

- If you press the RETURN key in response to a prompt for a field value, DATATRIEVE reprompts you for a field value.
- If, in response to a prompt, you enter a value that is longer than the length of the field to which it is being assigned, DATATRIEVE displays an error message and reprompts you for the value.
- If you omit from a MODIFY statement both a list of field names and a USING clause, DATATRIEVE prompts for each elementary field in the record.
- If you include a list of field names, DATATRIEVE prompts you for each elementary field specified or implied by the list. If you include a group field name in the list, DATATRIEVE prompts you for each elementary field contained in the group field.
- If you specify a USING clause, DATATRIEVE uses any Assignment statements in statement-1 to modify the values of the specified fields. DATATRIEVE does not then prompt you for any field values unless the USING clause contains an Assignment statement with a prompting value expression (value-expression = *.prompt-name). See Table 4–17, Table 4–18, Table 4–19, and Table 4–20 for syntax examples of the USING clause.
- If you do not include ALL or an RSE in a MODIFY statement and do not put the statement in a FOR loop, DATATRIEVE modifies the selected record. It assigns to each specified or implied field in the selected record the value you supply to the corresponding prompt or in the Assignment statements in the USING clause.

  DATATRIEVE prompts you to supply a value for each specified or implied field in the selected record, unless you specify a USING clause that contains no prompting value expressions. (See the first example.)

If, however, in the USING clause, you assign a value to the field with an arithmetic calculation that involves a prompting value expression (USING PRICE = PRICE * *.EXCHANGE_RATE), DATATRIEVE uses your response to the prompt to calculate the value of the arithmetic expression. The value of that arithmetic expression is the value put in the updated field, not the value you enter in response to the prompt. (See the third example.)

Table 4–17 lists the syntax, prompts, and results of the statements that modify selected records.

Table 4–17 Modifying Selected Records

Syntax	Result
MODIFY	Prompts once for each elementary field in the record definition. Changes each elementary field in the selected record to the value you supply in response to the corresponding prompt.
MODIFY field-list	Prompts once for each elementary field specified or implied by the list. Changes each elementary field specified or implied by the list to the value you supply in response to the corresponding prompt.
MODIFY USING statement-1	No prompts unless statement-1 contains prompting value expressions. Changes each specified or implied elementary field in the selected record to the values supplied by the Assignment statements in statement-1.

For these forms of the MODIFY statement, the context is determined by the selected record of the most recently established collection.

If no selected record exists for the CURRENT collection, then the context is determined by the most recently established collection that has a selected record.

If there is no selected record for any existing collection, DATATRIEVE issues an error message.

- If you specify the argument ALL but do not include an RSE clause in a MODIFY statement, DATATRIEVE changes all the records in the CURRENT collection. It assigns the same value to each specified and implied field of every record in the CURRENT collection. If, however, in the USING clause, you assign a value to the field with an arithmetic calculation that involves the old value of the field, DATATRIEVE uses the value of each calculation to modify the field value in the corresponding record.

DATATRIEVE prompts you only once to supply a value for each specified or implied field. If you specify a USING clause that contains no prompting value expressions, DATATRIEVE does not prompt you for the field values.

Table 4–18 lists the syntax, prompts, and results of the statements that modify all the records in the CURRENT collection.

Table 4–18 Modifying All Records in the CURRENT Collection

Syntax	Result
MODIFY ALL [CURRENT]	Prompts once for each field in the record definition. Changes each field of every record in the CURRENT collection to the value you supply in response to the corresponding prompt.
MODIFY ALL field-list	Prompts once for each elementary field specified or implied by the list. Changes each specified or implied field of every record in the CURRENT collection to the value you supply in response to the corresponding prompt.
MODIFY ALL [CURRENT] USING statement-1	No prompts unless statement-1 contains prompting value expressions. Changes each specified or implied elementary field of every record in the CURRENT collection to the value supplied by the Assignment statements in statement-1.

- If you use an RSE in a MODIFY statement, DATATRIEVE changes all the records in the record stream specified by the RSE. It assigns the same value to each specified and implied field of every record in the record stream. If, however, in the USING clause, you assign a value to the field with an arithmetic calculation that involves the old value of the field, DATATRIEVE uses the value of each calculation to modify the field value in the corresponding record.

  DATATRIEVE prompts you only once to supply a value for each specified and implied field, unless you specify a USING clause that contains no prompting value expressions.

  Table 4–19 lists the syntax, prompts, and results of the statements that modify all the records in a target record stream.

Table 4–19 Modifying All Records in a Record Stream

Syntax	Result
MODIFY [ALL] OF rse	Use with care. MODIFY ALL OF domain-name changes all the records in the domain. Prompts once for each field in the record definition. Changes each field of every record in the record stream to the value you supply in response to the corresponding prompt.
MODIFY [ALL] field-list OF rse	Prompts once for each elementary field specified or implied by the list. Changes each specified or implied field of every record in the record stream to the value you supply in response to the corresponding prompt.
MODIFY [ALL] rse USING statement-1 *MODIFY [ALL] USING statement-1 OF rse*	No prompts unless statement-1 contains prompting value expressions. Changes each specified or implied elementary field of every record in the record stream to the value supplied by the Assignment statements in statement-1.

When using a MODIFY statement that contains an RSE, you do not change the result by including the optional ALL or by excluding it.

- If you include a MODIFY statement within a FOR loop, DATATRIEVE modifies the records specified by the RSE in the FOR statement, but it handles the prompts differently from the other methods of changing records in record streams and collections. When you want DATATRIEVE to prompt for field values of each record that it modifies, include the MODIFY statement within a FOR statement.

  Table 4–20 lists the syntax, prompts, and results of the statements that modify each record of a record stream specified by the RSE in a FOR statement.

Table 4–20 Modifying Records in a Record Stream Formed by a FOR Loop

Syntax	Result
FOR rse MODIFY	Prompts once for every elementary field in the record definition for each record in the record stream. Changes each field of each record to the value you supply in response to the corresponding prompt.
FOR rse MODIFY field-list	Prompts once for every elementary field specified or implied by the list for each record in the record stream. Changes each specified or implied field of each record to the value you supply in response to the corresponding prompt.
FOR rse MODIFY USING statement-1	No prompts unless statement-1 contains prompting value expressions. A *.prompt value expression prompts once for each record in the record stream. A **.prompt value expression prompts only once, during the first execution of the FOR loop. Changes each specified or implied elementary field of each record in the record stream to the value supplied by the Assignment statements in statement-1.
FOR rse MODIFY list-rse USING statement-1	Useful for modifying list items within hierarchical records. The first RSE specifies the records to be modified, and the second RSE (list-rse) specifies the occurrences of the list to be modified. Changes each specified or implied elementary field of each list of each record in the record stream. (See the last example in the Examples section.)

- If you include a VERIFY clause in a MODIFY statement, DATATRIEVE executes statement-2 for each target record before changing that record. If statement-2 contains an ABORT statement and a target record causes the abort conditions to be met, DATATRIEVE aborts the MODIFY statement without changing that record. Any previous records changed by that MODIFY statement, however, remain changed, and DATATRIEVE returns you to command level (indicated by the DTR> prompt).

- If a MODIFY statement with an ABORT statement in the VERIFY clause is not part of a procedure or command file, neither SET ABORT nor SET NO ABORT has an effect on the result of the ABORT.
- If a MODIFY statement with an ABORT statement in a VERIFY clause is part of a procedure or command file and SET ABORT is in effect, DATATRIEVE aborts the MODIFY statement without changing the record that caused the abort. DATATRIEVE returns you to command level without executing the rest of the procedure or command file.

  On the other hand, if SET NO ABORT is in effect, DATATRIEVE aborts the MODIFY statement without changing the record that caused the abort and executes the next statement in the procedure or command file.

Usage Notes

- You should use the MODIFY statement with the utmost care because it changes the information in the data file. DATATRIEVE catches any syntax errors you might make in entering the statement. But even if the syntax is correct, the statement may still contain a logical error that causes the wrong records to be changed or the wrong values to be entered into the fields.

  If DATATRIEVE prompts you for a field you did not expect, enter CTRL/Z to prevent changing records or fields you do not wish to change. A good practice is to print each record before you modify it.
- Once you have modified the values in one or more fields, you can recover the previous information only by explicitly changing the new values back to the old ones. You may have to use several MODIFY statements to make the necessary changes.
- When modifying records in relations that are dependent on other relations because of a definition constraint, ready all the involved relations.
- The statement in the USING statement-1 clause of a MODIFY statement is usually an Assignment statement or a BEGIN-END block containing more than one Assignment statement. See the section in this chapter on the Assignment statement for more information.
- When you change all the records in the CURRENT collection or in a record stream created by an RSE, you do not have to assign the same field value to every target record. In the USING clause, you can put an Assignment statement that makes the new value of a field in the target record depend on the old value of that field. For example, you change the price of a collection of yachts with a MODIFY statement containing an assignment such as PRICE = PRICE * 1.1, or PRICE = PRICE + 500. (See the third example.)

The same calculation must apply to each target record, but the new values derived for each record are independent of one another. The operation performed on a field value, however, must be appropriate to the data type of that field. Do not, for example, try to do arithmetic with the nonnumeric data in an alphanumeric field.

- With the MODIFY statement, you can transfer information from one domain to another, from one group field to another, and from one elementary field to another. Use one of the following Assignment statements in the USING clause:

  ```
  field-name-1 = field-name-2
  ```

  ```
  group-field-name-1 = group-field-name-2
  ```

  In each case, put the target field name on the left side of the Assignment statement and the source field name on the right side. Each field name must be adequately qualified to establish the proper context for each side of the Assignment statement. (See the discussion of context in the *VAX DATATRIEVE User's Guide*.)

  This transfer of data is easiest when the field definitions are exactly the same on both sides of the Assignment statement. If the field definitions differ, make sure you understand the truncations that may result if the lengths of the fields do not match.

  You must also anticipate any conflicts between data types. For example, you can modify an alphanumeric field with the content of a numeric, but modifying a numeric field with the content of an alphanumeric gives you a warning if your alphanumeric field contains nondigit characters.

  For further information about using field names in Assignment statements, see the section in this chapter on the Assignment statement.

- To include other DATATRIEVE statements in the USING clause of a MODIFY statement, put them in a BEGIN-END block. For example, to see the target record before changing it you can precede the Assignment statements in the BEGIN-END block with a PRINT statement.

- With a BEGIN-END block in the USING clause of a MODIFY statement, you can create an audit trail of the previous values of the records you modify. You need to define and ready another domain that uses the same record definition as the one whose records you intend to modify. You then put an appropriate STORE statement in a BEGIN-END block in the USING clause of a MODIFY statement. (See the fifth example.)

- Typically, statement-2 in a VERIFY clause contains an IF-THEN-ELSE statement and an ABORT statement (see the sections on these statements in this chapter).

- The values you supply to the prompts of the MODIFY statement are first checked against any validation conditions specified for that field in the record definition. If the value conforms to the conditions specified in the appropriate VALID IF clause in the record definition, only then is it checked against the conditions in the VERIFY clause of the MODIFY statement.

  If you always use the same validation conditions for modifying and storing data, put those conditions in VALID IF clauses in the record definition. That way, DATATRIEVE prompts you for another value for the same field. When you couple the validation conditions with an ABORT statement in the VERIFY clause of a MODIFY statement, you return to DATATRIEVE command level and must reenter the MODIFY statement.

- If you try to use the first format of MODIFY and the name of a field duplicates the name of a readied domain, DATATRIEVE interprets the field name as a readied domain. In addition, if the field name duplicates the name of a DATATRIEVE keyword such as FIRST, DATATRIEVE interprets the field name as the beginning of an RSE. To avoid these outcomes, use a query name for the field that does not duplicate the name of a keyword or domain.

Examples

The following example changes one field value in a selected record:

```
DTR> READY YACHTS MODIFY
DTR> FIND FAMILIES WITH FATHER = "JOHN"
[2 records found]
DTR> PRINT
No record selected, printing whole collection.

                      NUMBER    KID
  FATHER     MOTHER    KIDS     NAME    AGE
JOHN       JULIE         2   ANN         29
                             JEAN        26
JOHN       ELLEN         1   CHRISTOPHR  0

DTR> SELECT 1
DTR> MODIFY FATHER
Enter FATHER: JON
DTR> PRINT

                      NUMBER    KID
  FATHER     MOTHER    KIDS     NAME    AGE

JON        JULIE         2   ANN         29
                             JEAN        26

DTR>
```

The following example makes a 10 percent increase in the price of the first five yachts. Each yacht begins and ends with a unique price, but the new price of each yacht is 10 percent greater than the old price of that same yacht.

```
DTR> MODIFY FIRST 5 YACHTS USING PRICE = PRICE * 1.1
DTR> PRINT FIRST 5 YACHTS

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 ALBERG       37 MK II   KETCH   37    20,000  12  $40,646
 ALBIN        79         SLOOP   26     4,200  10  $19,690
 ALBIN        BALLAD     SLOOP   30     7,276  10  $30,250
 ALBIN        VEGA       SLOOP   27     5,070  08  $20,460
 AMERICAN     26         SLOOP   26     4,000  08  $10,885

DTR>
```

The following example forms a collection of yachts and modifies all the elementary fields within the group field SPECIFICATIONS for the first record:

```
DTR> SET NO PROMPT
DTR> READY YACHTS MODIFY
DTR> FIND YACHTS WITH BEAM = 0
[5 records found]
DTR> FOR CURRENT MODIFY USING
CON> BEGIN
[Looking for statement]
CON>     PRINT SPECS
CON>     RIG = **.RIG
CON>     LOA = **.LOA
CON>     DISP = *.WEIGHT
CON>     BEAM = *.BEAM
CON>     PRICE = PRICE * 1.1
CON> END

       LENGTH
        OVER
 RIG    ALL   WEIGHT BEAM  PRICE
```

```
SLOOP   32     9,500  00
Enter RIG: TAB
Enter LOA: 33
Enter WEIGHT: 12000
Enter BEAM: 10
SLOOP   32    11,000  00  $29,500
Enter WEIGHT: TAB
Enter BEAM: 11
SLOOP   31    13,600  00  $32,500
Enter WEIGHT: 15000
Enter BEAM: 12
SLOOP   35    23,200  00
Enter WEIGHT: TAB
Enter BEAM: 13
SLOOP   32    14,900  00  $34,480
Enter WEIGHT: TAB
Enter BEAM: 9
DTR> PRINT ALL

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 METALMAST   GALAXY     SLOOP   33    12,000  10
 O'DAY       32         SLOOP   33    11,000  11  $32,450
 RYDER       S. CROSS   SLOOP   33    15,000  12  $35,750
 TA CHIAO    FANTASIA   SLOOP   33    23,200  13
 WRIGHT      SEAWIND II SLOOP   33    14,900  09  $37,928

DTR>
```

The following example uses a MODIFY statement in a FOR loop to update a master file with the data in a transaction file:

```
DTR> FOR UPDATES
DTR>     MODIFY USING
DTR>         PRICE = UPDATE.PRICE OF
DTR>     YACHTS WITH TYPE = UPDATE.TYPE
DTR>
```

The following example provides an audit trail by forming a domain AUDIT_YACHTS to store records of changes made to YACHTS. The record definition for AUDIT_YACHTS is the same as the one for YACHTS except for the addition of a field CHANGE_DATE. This field enables you to store data on the date of the change. A DEFAULT VALUE clause automatically assigns the current system date as the value for the field. The field definition is as follows:

```
06 CHANGE_DATE USAGE IS DATE
               DEFAULT VALUE IS "TODAY".
```

MODIFY Statement

The following statement allows you to modify records in YACHTS while storing the changes in AUDIT_YACHTS:

```
DTR> SHOW AUDIT_YACHTS
DOMAIN AUDIT_YACHTS USING
AUD_REC ON AUDYACHT;

DTR> FOR A IN YACHTS MODIFY USING
CON> BEGIN
CON>   BUILDER = *.BUILDER
CON>   MODEL   = *.MODEL
CON>   RIG     = *.RIG
CON>   LOA     = *.LOA
CON>   DISP    = *.DISP
CON>   BEAM    = *.BEAM
CON>   PRICE   = *.PRICE
CON>  STORE B IN AUDIT_YACHTS USING
CON>      B.BOAT = A.BOAT
CON> END
DTR>
```

The following example modifies the name of the child of TOM and ANNE from PATRICK to PATRICIA and prints the record before and after the change:

```
DTR> READY FAMILIES MODIFY
DTR> FOR FAMILIES WITH FATHER = "TOM" AND MOTHER = "ANNE"
CON>    BEGIN
CON>      PRINT
CON>      MODIFY KIDS WITH KID_NAME = "PATRICK" USING
CON>        KID_NAME = "PATRICIA"
CON>      PRINT
CON>    END

                      NUMBER    KID
  FATHER     MOTHER    KIDS     NAME    AGE

TOM         ANNE          2   PATRICK     4
                              SUZIE       6

                      NUMBER    KID
  FATHER     MOTHER    KIDS     NAME    AGE

TOM         ANNE          2   PATRICIA    4
                              SUZIE       6

DTR>
```

Common Context Errors

The previous sections contain the correct formats to modify the records you want to change from the record source you intend to use. This section describes some problems you can encounter if you inadvertently use the wrong format or combine format elements incorrectly. When you make this kind of mistake, DATATRIEVE either displays an error message or modifies records from the wrong record source, depending on the type of error you make.

Modifying All Records Rather Than Just the Selected Record

If you want to modify a selected record, do not include the keyword ALL in the MODIFY statement.

If you type MODIFY ALL, you are telling DATATRIEVE either to target the entire collection for the modify operation or to expect an RSE in the MODIFY statement. Because there may be times when this is your intention, DATATRIEVE does not display an error message. If you create this situation unintentionally, you can make all the records in the CURRENT collection identical for the field values you supply when you intended to change values in only one of those records.

Modifying the Wrong Selected Record

This error can occur if you forget to enter a SELECT statement for the CURRENT collection. There may be times when you have more than one collection in your workspace, and the collections formed before the CURRENT one have selected records.

When you enter a MODIFY statement appropriate for a selected record and your CURRENT collection does not have one, DATATRIEVE tries to apply the modify operation to the selected records you do have available. It modifies the most recently selected record to which it can apply your statement.

If your statement contains a field name that is not in any of the available selected records, DATATRIEVE does not modify any record but tells you that the field name is used out of context.

There may be times when you want to modify a selected record in a collection other than the CURRENT one. In this case, you can enter SELECT NONE statements to "unselect" records associated with any collections formed after the one containing the selected record you want to change. In effect, this process releases selected records from the current collection, and you can repeat it until you reach the individual records selected from the target collection.

The error you want to avoid in this situation is entering too many or too few SELECT NONE statements. You can use the PRINT statement to see which is the current selected record. You can also use SHOW CURRENT or SHOW COLLECTIONS to make sure that you are working with the collection you want.

Modifying Records in the Wrong RSE

This error occurs when you intend to modify records using a FOR statement RSE, but you also include an RSE or the keyword ALL in the MODIFY statement itself. If you do this, you are telling DATATRIEVE to modify records specified in the MODIFY statement for as many iterations as there are records in the FOR statement RSE. (The only time you want to do something like this is when you modify repeating fields in a hierarchical record. When you modify hierarchical records, however, the RSE in the MODIFY statement specifies an OCCURS field name as the record source, rather than a true record source, such as a domain or collection.) If you inadvertently include two RSEs in the combined statements that carry out a modify operation, the results can be unexpected.

In the following example, the user intends to change the last name in the first PERSONNEL record. The superfluous RSE in the FOR statement causes DATATRIEVE to prompt for the field as many times as there are records in the PERSONNEL domain:

```
DTR> SET NO PROMPT
DTR> FOR PERSONNEL MODIFY FIRST 1 PERSONNEL USING BEGIN
CON> PRINT
CON> LAST_NAME = *."last name"
CON> END

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

00012 EXPERIENCED CHARLOTTE  SPIVA  TOP 12-Sep-1972 $75,892 00012
Enter last name: WHITE
00012 EXPERIENCED CHARLOTTE  WHITE  TOP 12-Sep-1972 $75,892 00012
Enter last name: TAB
00012 EXPERIENCED CHARLOTTE  WHITE  TOP 12-Sep-1972 $75,892 00012
Enter last name: TAB
00012 EXPERIENCED CHARLOTTE  WHITE  TOP 12-Sep-1972 $75,892 00012
                                  .
                                  .
                                  .
```

A correct statement in the previous example would have been either:

```
MODIFY FIRST 1 PERSONNEL USING . . .
```

or

```
FOR FIRST 1 PERSONNEL MODIFY USING . . .
```

In the following example, the user intends to modify the DEPT field of only the employee record with ID number 34456. Because of the keyword ALL in the MODIFY statement, however, DATATRIEVE uses the value entered to modify the DEPT field of all the records in the CURRENT collection:

```
DTR> FIND PERSONNEL WITH DEPT = "T32"
[4 records found]
DTR> PRINT
No record selected, printing whole collection.

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

34456 TRAINEE     HANK    MORRISON  T32 1-Mar-1982 $30,000 87289
38462 EXPERIENCED BILL    SWAY      T32 5-May-1980 $54,000 00012
48573 TRAINEE     SY      KELLER    T32 2-Aug-1981 $31,546 87289
83764 EXPERIENCED JIM     MEADER    T32 4-Apr-1980 $41,029 87289

DTR> FOR PERSONNEL WITH ID = 34456
CON> MODIFY ALL DEPT
Enter DEPT: F11
DTR> PRINT
No record selected, printing whole collection.

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

34456 TRAINEE     HANK    MORRISON  F11 1-Mar-1982 $30,000 87289
38462 EXPERIENCED BILL    SWAY      F11 5-May-1980 $54,000 00012
48573 TRAINEE     SY      KELLER    F11 2-Aug-1981 $31,546 87289
83764 EXPERIENCED JIM     MEADER    F11 4-Apr-1980 $41,029 87289

DTR> ! YIPES!!!!
DTR>
```

To avoid this, the user could enter either:

```
SELECT 1; MODIFY DEPT
```

or

```
FOR PERSONNEL WITH ID = 34456 MODIFY DEPT
```

OCCURS Clause

The OCCURS clause defines multiple occurrences (or repetitions) of a field or group of fields. The multiple occurrences, called a list, create a hierarchy in the domain. (See the *VAX DATATRIEVE User's Guide* for more information on hierarchies.)

The OCCURS clause has two formats: one format for a fixed number of occurrences and one for a variable number of occurrences. Each is described in the following sections.

Fixed Number of Occurrences

Defines a fixed number of occurrences for a field or group of fields.

Format

```
OCCURS  n  TIMES
```

Argument

n TIMES

Is a positive integer specifying the number of occurrences for the field.

Restrictions

- A field definition cannot contain both an OCCURS and a COMPUTED BY clause. You cannot specify multiple occurrences of a COMPUTED BY field.
- You cannot use an OCCURS clause and a MISSING VALUE clause to describe the same field in a record definition.

Result

This format of the OCCURS clause defines a list with a fixed number of occurrences. It reserves enough space in each record to contain all the occurrences of the field (or fields), whether or not they contain data.

Usage Notes

- A record definition can contain any number of OCCURS clauses in this format.
- A field in a group field with an OCCURS clause can contain an OCCURS clause. The result is a sublist: a list nested within a list. (See the third example.)

- Retrieving and modifying data in fields defined with the OCCURS clause requires more complicated syntax than retrieving data from other types of fields. For information on alternatives to defining records using the OCCURS clause, see the chapter on using hierarchies in the *VAX DATATRIEVE User's Guide*.

Examples

The following field definition reserves enough space in every record for two occurrences of the elementary field KIDS_NAMES; each occurrence is 10 characters long:

```
03 KIDS_NAMES PIC X(10)
   OCCURS 2 TIMES.
```

You can store up to two names (containing up to 10 characters each) in any record. The following definition specifies that the group field KIDS_NAMES is repeated twice. Each group field contains two fields: FIRST_NAME (10 characters long) and MIDDLE_INIT (1 character). A total of 22 characters is reserved in every record for the group field:

```
03 KIDS_NAMES OCCURS 2 TIMES.
   05 FIRST_NAME PIC X(10).
   05 MIDDLE_INIT PIC X.
```

In the record, the fields are stored in the following order:

```
KIDS_NAMES
   FIRST_NAME
   MIDDLE_INIT
KIDS_NAMES
   FIRST_NAME
   MIDDLE_INIT
```

By nesting a sublist within a list, reserve enough space in a record to store up to three nicknames for each KIDS_NAMES:

```
03 KIDS_NAMES OCCURS 2 TIMES.
   05 FIRST_NAME PIC X(10).
   05 MIDDLE_INIT PIC X.
   05 NICKNAME PIC X(10)
   OCCURS 3 TIMES.
```

The fields are stored in the following order:

OCCURS Clause

```
KIDS_NAMES
   FIRST_NAME
   MIDDLE_INIT
   NICKNAME
   NICKNAME
   NICKNAME
KIDS_NAMES
   FIRST_NAME
   MIDDLE_INIT
   NICKNAME
   NICKNAME
   NICKNAME
```

Variable Number of Occurrences

Defines a variable number of occurrences of a group of fields.

Format

OCCURS min **TO** max **TIMES DEPENDING ON** field-name

Arguments

min

Is a nonnegative integer specifying the minimum number of occurrences of the field. This value can be zero. DATATRIEVE does not check this value.

max

Is a positive integer specifying the maximum number of occurrences of the field. This value must be greater than zero.

field-name

Is the name of a field in the same record definition. The value of the field determines the number of occurrences of this field. The field must be a numeric field containing a nonnegative integer; it cannot be defined with digits to the right of the decimal point.

Restrictions

- No other field definition can follow the last elementary field in the group field containing this clause.
- A record definition can contain only one OCCURS clause in this format.
- A field definition cannot contain both an OCCURS and a REDEFINES clause or an OCCURS and a COMPUTED BY clause.
- You cannot use a qualified field name, such as TYPE.BUILDER, as the DEPENDING ON variable in an OCCURS clause.

Result

The number of occurrences of the group field in any record is equal to the value of the field specified in the OCCURS clause. Therefore, the sizes of records in the domain can vary. If you use the MAX argument when you define the data file, all records in the file have the same length. See the section on the DEFINE FILE command.

Usage Notes

- A group field containing this format of the OCCURS clause can contain one or more fields with an OCCURS clause. The nested OCCURS clause, however, must specify a fixed number of occurrences of the field.
- Retrieving and modifying data in fields defined with the OCCURS clause requires more complicated syntax than retrieving data from other types of fields. For information on alternatives to defining records using the OCCURS clause, see the chapter on using hierarchies in the *VAX DATATRIEVE User's Guide*.

Example

The FAMILY_REC record definition shows the use of an OCCURS clause in the following format:

```
01 FAMILY.
03 PARENTS.
   06 FATHER PIC X(10).
   06 MOTHER PIC X(10).
03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
   06 EACH_KID.
      09 KID_NAME PIC X(10) QUERY_NAME IS KID.
      09 AGE PIC 99 EDIT_STRING IS Z9.
```

The number of occurrences of the KIDS field depends on the value of the NUMBER_KIDS field. If the value is 0, there are no occurrences of the field; if it is 1, there is one occurrence, and so on. Each occurrence of KIDS contains three fields: the group field EACH_KID and the elementary fields KID_NAME and AGE.

ON Statement

Sends the output of all indicated statements to the specified output file or device.

Format

ON { file-spec | *.prompt } statement

Arguments

file-spec

Specifies the file to which you want to write the output of the statement. The file specification has the following format:

```
node-spec::device:[directory]file-name.type;version
```

The minimum file specification consists of a period (.). The specification of such a file stored in your default VMS directory ends with ".;n", where n is the version number and both the file name and the type are null strings.

If you omit a field in the file specification, DATATRIEVE uses the defaults listed in Table 4–21.

Table 4–21 Output File Specification Defaults

Field	Default
node-spec::	Your local node
device:	Your default device
[directory]	Your default directory
file-name	Null string
.type	.LIS
;version	Highest version number

statement

Is a simple or compound statement you want DATATRIEVE to execute and write the output to the specified file.

Restrictions

- If you specify a compound statement, you must enclose the simple statements within a BEGIN-END block.
- When you use ON LP: to send output directly to a line printer, DATATRIEVE assumes your system has a line printer. If your system does not have a device defined as LPA0:, the clause ON LP: will not work. Although this restriction applies to any system without a line printer, you may encounter it unexpectedly if your system is part of a VAXcluster with a common line printer. The ON LP: clause does not work in a VAXcluster that uses a common printer not directly connected to your system.

  If the nodes in the cluster are connected with DECnet, you can work around this restriction. To send output from a node without a line printer, you must include the node name of the system with the line printer in the LP: specification. For example, if the cluster's line printer is on a node named BIGVAX, the following PRINT statement sends output to it:

```
DTR> PRINT YACHTS ON BIGVAX::LP:
```

  Note that you cannot directly specify the line printer on BIGVAX by using the cluster device-name BIGVAX$LPA0: in the ON clause.
- Use the ON statement, not the ON clause, to generate multiple reports within iterations of a loop.

  The following example generates a new version of the file Y for each iteration of the WHILE loop. The example uses a domain and record definition that are not in the DATATRIEVE samples. The company in this example uses invoices, each having a unique, identifying number. INVOICE_NUM represents these invoice numbers. CURRENT_NUM is the count of all invoices plus 1.

```
DTR> READY INVOICES
DTR> WHILE INVOICE_NUM LT CURRENT_NUM
CON>   BEGIN
CON>   ON Y
CON>     REPORT INVOICE WITH PRICE = *
CON>     PRINT INVOICE_HEADER
CON>     END_REPORT
CON>  !
CON>  ! further processing of invoices
CON>  !
CON>  END
```

  To generate a single version of the Y file containing all the WHILE reports, use the ON clause of the REPORT statement.

Result

DATATRIEVE writes the output of the simple or compound statement to the file specified.

Usage Notes

- Use the ON file-spec statement to file output of REPORT, PRINT, SUM, or LIST statements.
- To write the output to more than one file, nest the ON statement. Do not use a comma to separate each occurrence of ON file-spec. (See the second example.)

Examples

The following example shows how to search all yachts built by AMERICAN and print the TYPE, PRICE, and RIG for each sloop. For all yachts built by AMERICAN which are not sloops, a line is skipped and a message is printed. The output is written to an RMS file.

Since an IF-THEN-ELSE statement is used and the same file name is specified in the THEN and ELSE clauses, DATATRIEVE creates two versions of the file.

```
DTR> FOR YACHTS WITH BUILDER = "AMERICAN"
CON> IF RIG = "SLOOP" THEN
CON>     PRINT TYPE, PRICE, RIG ON BOAT.RNO ELSE
CON>     PRINT SKIP, "NOT A SLOOP" ON BOAT.RNO
Creating file DB0:[MORRISON.REF]BOAT.RNO;1 ...
Creating file DB0:[MORRISON.REF]BOAT.RNO;2 ...
```

The content of BOAT.RNO;1 is as follows:

```
MANUFACTURER   MODEL     PRICE   RIG

 AMERICAN    26          $9,895 SLOOP
```

The content of BOAT.RNO;2 is as follows:

```
NOT A SLOOP
```

However, if you use the ON statement, all of the data can be written to the same file as follows:

```
DTR> ON SHIP.RNO
CON> FOR YACHTS WITH BUILDER = "AMERICAN"
CON> IF RIG = "SLOOP" THEN PRINT TYPE, PRICE, RIG ELSE
CON> PRINT SKIP, "NOT A SLOOP"
Creating file DB0:[MORRISON.REF]SHIP.RNO;1 ...
```

The contents of SHIP.RNO is as follows:

```
MANUFACTURER   MODEL     PRICE   RIG

 AMERICAN    26          $9,895 SLOOP

NOT A SLOOP
```

The following example shows how to write the output of a LIST statement to three files and display the output. Data on every family with three children is included:

```
DTR> ON FAM1.RNO
CON> ON FAM2.RNO
CON> ON FAM3.RNO
CON> ON TT:
CON> LIST FAMILIES WITH NUMBER_KIDS = 3
Creating file DB0:[MORRIS.REF]FAM1.RNO;1 ...
Creating file DB0:[MORRIS.REF]FAM2.RNO;1 ...
Creating file DB0:[MORRIS.REF]FAM3.RNO;1 ...
Sending output to terminal TT.

FATHER      : GEORGE
MOTHER      : LOIS
NUMBER_KIDS :  3

    KID_NAME    : JEFF
    AGE         : 23

    KID_NAME    : FRED
    AGE         : 26

    KID_NAME    : LAURA
    AGE         : 21

FATHER      : HAROLD
MOTHER      : SARAH
NUMBER_KIDS :  3
    KID_NAME    : CHARLIE
    AGE         : 31

    KID_NAME    : HAROLD
    AGE         : 35

    KID_NAME    : SARAH
    AGE         : 27

DTR>
```

DATATRIEVE sends the output to the terminal and creates the three files specified. All three files contain the data displayed on the terminal.

OPEN Command

Opens an RMS file to serve as a log of your interactive dialogue with DATATRIEVE. DATATRIEVE copies your input and the DATATRIEVE output including error messages to the file exactly as displayed.

Format

OPEN file-spec

Argument

file-spec

Is the VMS file specification of the file to be opened. The file specification must be in the following format:

```
node-spec::device:[directory]file-name.type;version
```

Restrictions

- None of your input to or the output from the editor or Guide Mode is written to the log file.
- If you use the OPEN command in a procedure, no statements or commands in the procedure are written to the log file. The output of a procedure, however, is written to the file.
- If you invoke a procedure after you have used the OPEN command, none of the commands or statements in the procedure are written to the log file.
- You cannot use the OPEN command in BEGIN-END, FOR, or REPEAT statements.
- You cannot have two log files open at the same time. If you enter a second OPEN command without closing the first log file, DATATRIEVE automatically closes the first log file and opens another one, even if you give the same file specification in both OPEN commands.

Results

- Except for the dialogue with the editor, Guide Mode, and procedures, DATATRIEVE writes both your input and its own output to the VMS file you specify in the command.
- If you have a log file open when you invoke a command file, DATATRIEVE includes the various DATATRIEVE prompts (such as DTR> and CON>) in the log file, even though it does not display those prompts when it executes the commands and statements in the command file.

- DATATRIEVE closes the file under four circumstances:
  - When you enter another OPEN command
  - When you enter a CLOSE command
  - When you exit from DATATRIEVE with the EXIT command or with CTRL/Z
  - When you exit from DATATRIEVE with CTRL/Y
- If you close the file with a CLOSE or EXIT command, that command is also included in the file. If you close the file by exiting from DATATRIEVE with a CTRL/Z or CTRL/Y, neither the control character nor any of the input line is included in the file.

Usage Notes

- Keeping log files of your dialogue with DATATRIEVE can provide you with a transaction log.
- Keeping a log file of your dialogue with DATATRIEVE can aid in developing and debugging DATATRIEVE applications.
- Keeping a log file of your dialogue with DATATRIEVE is essential when submitting a Software Performance Report (SPR) to Digital.
- To include the contents of a procedure into a log file, enter a SHOW command to show the procedure before invoking the procedure.

Example

This example opens a log file, displays the contents of a procedure, invokes the procedure, closes the log file with a CTRL/Z exit from DATATRIEVE, and uses the VMS TYPE command to display the contents of the log file:

```
DTR> OPEN LOG
DTR> !THIS IS A TEST OF THE OPEN COMMAND.
DTR> READY YACHTS
DTR> SHOW SELL_BOAT
PROCEDURE SELL_BOAT
FIND YACHTS WITH BUILDER EQ *.BUILDER AND MODEL EQ *.MODEL
PRINT ALL
IF *."Y IF BOAT SOLD" EQ "Y" THEN ERASE ALL ELSE
       PRINT "SELL IT NOW!"
END_PROCEDURE

DTR> :SELL_BOAT
Enter BUILDER: ALBIN
Enter MODEL: VEGA
```

OPEN Command

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

 ALBIN       VEGA       SLOOP   27     5,070  08  $18,600

Enter Y IF BOAT SOLD: N

SELL IT NOW!
DTR> CTRL/Z
$ TYPE LOG.LIS
DTR> !THIS IS A TEST OF THE OPEN COMMAND.
DTR> READY YACHTS
DTR> SHOW SELL_BOAT
PROCEDURE SELL_BOAT
FIND YACHTS WITH BUILDER EQ *.BUILDER AND MODEL EQ *.MODEL
PRINT ALL
IF *."Y IF BOAT SOLD" EQ "Y" THEN ERASE ALL
PRINT "SELL IT NOW !"
END_PROCEDURE

DTR> :SELL_BOAT
Enter BUILDER: ALBIN
Enter MODEL: VEGA
                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 ALBIN       VEGA       SLOOP   27     5,070  08  $18,600

Enter Y IF BOAT SOLD: N

SELL IT NOW!

$
```

PICTURE Clause

Specifies the format of the field value as it is stored.

Format

PIC[TURE] [IS] picture-string

Argument

picture-string
Is one or more picture-string characters describing the format in which the field value is stored.

Restrictions

- This clause is valid for elementary fields only.
- Specify the picture-string characters as a string. Do not embed a space in the picture string.

Result

The storage format for the field's content is determined by the specified picture string. In general, each character in the string corresponds to one character position in the field value.

Usage Notes

- For numeric fields, you can also include a USAGE clause to specify the internal format of the digits. For example, for a numeric field without a USAGE clause, 999999 specifies six digits in six character positions, occupying six bytes of storage.
- To enter a series of identical picture characters, you can shorten the string by placing a repeat count in parentheses following the picture character. For example, the picture string 9(6) is equal to 999999.

Table 4–22 contains a list of the picture-string characters. The picture-string characters you specify for a field depend on the class of the field: alphabetic, alphanumeric, or numeric.

Table 4–22 Picture-String Characters

Field Class	Picture Character	Meaning
Alphabetic	A	Each A represents one alphabetic character in the field.
Alphanumeric	X	Each X represents one character in the field.
Numeric	9	Each 9 represents one digit in the field. You can specify from 1 to 31 digits for a numeric field.
	S	An S indicates that a sign (+ or -) is stored in the field. A picture string can have only one S and it must be the leftmost character. If there is no SIGN clause for the field, the sign shares the rightmost character position with the lowest-valued digit.
	V	A V indicates an implied decimal point. The decimal point does not occupy a character position in the field, although DATATRIEVE uses its location to align data in the field. A picture string can contain only one V.
	P	Each P specifies a decimal scaling position. Each P represents a "distance" in digits from an implied decimal point. (A P does not count toward the limit of 31 digits per field.) A P can appear at the right or left of the picture string. A V is unnecessary for any picture string containing a P.

Alphabetic Fields

The picture string for an alphabetic field specifies the number of characters in the field. Only the picture character A is valid in the picture string for an alphabetic field. Each A corresponds to a single character position in the field. For example, the following field definition specifies an alphabetic field of six characters:

```
06 LETTERS_ONLY PIC A(6).
```

Alphanumeric Fields

The picture string for an alphanumeric field specifies the number of characters in the field. Only the picture character X is valid in the picture string for an alphanumeric field. Each X corresponds to a single character position in the field. For example, the following field definition specifies that the MODEL field contains 10 alphanumeric characters:

```
06 MODEL PIC X(10).
```

Numeric Fields

A numeric field can contain the characters 9, S, V, and P in its picture string to specify: the number of digits in the field, a sign, an implied decimal point, and a decimal scaling factor.

Specifying the Number of Digits The picture character 9 represents one digit in the field value. For example, the picture string 9(4) indicates four digits; the field value can range from 0 to 9999. There is one exception to this; if you specify PIC 9999 and USAGE IS COMP, then it is possible to store numbers as large as 32,768. You can specify from 1 to 31 digits for a numeric field.

The following field definition specifies that the BEAM field contains two digits:

```
03 BEAM PIC 99.
```

Specifying a Sign To specify that a numeric field can contain a sign (+ or -), you must include an S in its picture string. The S must be the leftmost character in the picture string; a picture string can contain only one S. For example, the picture string S9(4) indicates a signed field, four digits in length; the field value can range from -9999 to +9999.

The sign specified with the PICTURE clause is not printed unless you include an EDIT_STRING clause with the field definition. For example, the following field definition specifies a sign in the picture string, which is printed following the last digit of the field value:

```
03 CURRENT_BALANCE
   PICTURE IS S9999V99
   EDIT_STRING IS $$$$9.99-.
```

Specifying a Decimal Place The picture character V specifies the position of an "implied" decimal point. For example, the picture string 9(5)V99 specifies a 7-digit field; the last two digits of the field value follow the decimal point.

The decimal point does not occupy a character position in the record; DATATRIEVE uses the implied decimal point in computations, Boolean expressions, and other arithmetic operations.

DATATRIEVE does, however, display the implied decimal point when you retrieve the value from a field with a V in the picture string. You do not have to use an edit string solely for the purpose of expressing the decimal point in the output format.

If there is no V in the picture string, DATATRIEVE treats the field value as an integer (that is, as if a V were specified to the right of the rightmost digit). Thus, the picture strings 999 and 999V are equal.

Scaling Factor The picture character P specifies a decimal scaling position. Each P represents one decimal position between the value stored in the field and the implied decimal point of the value. A P does not occupy a character position in the field.

The P (or multiple Ps) must be the leftmost or rightmost characters in the picture string. If Ps are the leftmost characters, the decimal point is assumed to be to the left of the leftmost P. For example, the picture strings PPP99 and VPPP99 are equal. (If you specify a P in a picture string, the V character is optional.) If Ps are the rightmost characters, the decimal point is assumed to be to the right of the rightmost P. Thus, 999PP and 999PPV are equal.

DATATRIEVE treats each P in a picture string as a zero. The string PPP99 specifies that the field can contain values in the range 0 to .00099; the three leftmost digits are assumed to be zeros. The range of values for a field with a picture string of 999PP is 0 and 100 to 99,900, but the two rightmost digits are assumed to be zeros.

You do not need to use an edit string to display the decimal point in fields with P picture string characters in them.

If the picture string of a field contains one or more Ps at the left of its picture string, DATATRIEVE includes the implied decimal in the default formats when you retrieve a value from the field. The scaling positions are displayed as zeros between the decimal point and the value stored in the field.

If the picture string of a field contains one or more Ps at the right of its picture string, DATATRIEVE includes the implied decimal in the default formats when you retrieve a value from the field. The scaling positions are displayed as zeros between the decimal point and the value stored in the field.

PLOT Statement

Each DATATRIEVE plot statement uses the following general syntax.

Format

```
PLOT plotname    [USING] [ALL]
                     [arg [,arg]...] [OF rse]
                     [ ON { file-spec  } ]
                          { *.prompt   }
```

Arguments

plotname

Is the name of the plot, for example:

```
DTR>  PLOT BAR
```

BAR is the name of the plot. Refer to the *VAX DATATRIEVE User's Guide* for plotname descriptions.

USING

Is an optional keyword to make the syntax more like English. The keyword USING does not affect the plot statement.

ALL

Is a keyword that may be optional or required, depending on the structure of the plot statement you use. It is optional if you use the OF rse clause; it is required if you use the current collection without any qualification.

For example, the following two plot statements are the same:

```
DTR>  PLOT BAR BUILDER, PRICE OF YACHTS WITH
CON>  PRICE NE 0

DTR>  FIND YACHTS WITH PRICE NE 0
DTR>  PLOT BAR ALL BUILDER, PRICE
```

The first example uses an OF rse clause in which the keyword ALL is not required. In the second example, ALL is required because it refers to the collection created in the FIND statement.

In the syntax diagrams of the sections that follow, the keyword ALL is shown in brackets as an optional element. Note, however, that if you use the keyword ALL to refer to the contents of the current collection, ALL is required.

arg
Is a field name or other value expression. The argument can also contain an optional label string. To specify a label string, put a quoted string inside parentheses after the value expression. For example:

```
DTR>  PLOT X_Y LOA,
CON>  DISP / 2000 ("Weight in tons") OF YACHTS
```

OF rse
Specifies the record stream to be used in the plot. (See the chapter on record selection expressions in the *VAX DATATRIEVE User's Guide* for more information on RSEs.)

If you do not specify an RSE, the plot statement will use data from the current collection.

ON { **file-spec** / ***.prompt** }

Specifies a device or file for output. If you do not specify a device or file specification, DATATRIEVE displays the plot on your terminal screen.

You can use a complete VMS file specification or simply a file name. DATATRIEVE uses .LIS as the default file type. You can also use a prompting value expression that prompts you for a file or device specification. (See the section on prompting value expressions in Chapter 1 for more information.) For example:

```
DTR>  PLOT PIE ALL RIG ON DRA0:[IACOBONE]RIGPLOT.LIS
DTR>  PLOT PIE ALL RIG ON *."File Name or Device Specification"
```

You can use the ON clause with any of the plot statements to create a file containing the ReGIS graphics commands. Note the following rules and restrictions:

Restrictions

- You can use any terminal when you issue the plot statement that creates such a file. However, you need a Digital-supported ReGIS terminal to convert the code in that file into a graphic representation of data.
- You can display the plot file you create at your Digital-supported ReGIS terminal with the DCL TYPE command. If you examine the file with a text editor, it is meaningless unless you are familiar with the ReGIS graphics language.
- You can print the plot file on any printer supporting ReGIS graphics (i.e. LPS20).

- Sometimes the ReGIS code is not sufficient to draw the picture. For example, some plot statements (such as PLOT CROSS_HATCH after PLOT MULTI_ SHADE) take the output of another plot, change the plot in some way, and send the output to a file. The ReGIS code stored in that file produces a crosshatched plot only if the multishaded plot is still in the graphics memory of your Digital-supported ReGIS terminal.
- You cannot plot data that includes null date values. To exclude records with null date values from a plot, put the expression WITH DATE NE " " in the RSE of your PLOT statement.

Usage Note

The *VAX DATATRIEVE User's Guide* illustrates the results of all the PLOT statements and presents many other examples.

Examples

The following example produces a simple bar chart showing the salary for each employee in PERSONNEL:

```
DTR> FIND PERSONNEL SORTED BY DEPT
DTR> PLOT BAR ALL LAST_NAME, SALARY
DTR>
```

The following example plots a marsupial:

```
DTR> PLOT WOMBAT
DTR>
```

PRINT Statement

Causes DATATRIEVE to format and write one or more values of specified or implied DATATRIEVE value expressions to your terminal or workstation, to a file, or to a unit record device.

Format

For retrieving from selected records and target record streams formed by FOR loops:

PRINT [print-list] [ON { file-spec / *.prompt }]

For retrieving from the current collection:

PRINT ALL [print-list] [ON { file-spec / *.prompt }]

For retrieving from record streams formed by the PRINT statement using one RSE:

PRINT [print-list OF] rse [ON { file-spec / *.prompt }]

For retrieving from record streams formed by the PRINT statement using two RSEs (the inner print list follows another print list):

PRINT print-list, ALL print-list OF rse-1 [,print-list] OF rse-2

[ON { file-spec / *.prompt }]

For retrieving from record streams formed by the PRINT statement using two RSEs (the inner print list precedes any other print list):

PRINT ALL ALL print-list OF rse-1 [,print-list] OF rse-2

[ON { file-spec / *.prompt }]

Arguments

print-list
Is a list of value expressions and formatting specifications. Table 4–24 describes the print list elements. See the Results section for a description of the modifiers you can use to control the column header and format for each data field in the output of the PRINT statement.

ALL
When used alone following PRINT, ALL causes the records in the current collection to be displayed or written to the specified file or device.

When used with a print list, ALL causes the print list to be evaluated for each record in the current collection.

When the print list begins with an inner print list, ALL is required to establish the proper context in which to resolve references to the items in the hierarchical list.

rse
Is a record selection expression that creates the record stream DATATRIEVE uses to evaluate the elements of the print list.

file-spec
Is the file specification of the file to which you want to write the output of the statement.

A complete file specification has the following format:

```
node-spec::device:[directory]file-name.type;version
```

If you omit a field in the file specification, DATATRIEVE uses the defaults listed in Table 4–23.

Table 4–23 Output File Specification Defaults

Field	Default
node-spec::	Your local node
device:	Your default device
[directory]	Your default directory
file-name	Null string

(continued on next page)

Table 4–23 (Cont.) Output File Specification Defaults

Field	Default
.type	.LIS
;version	1 or next higher version number

The minimum file specification consists of a period (.); the specification of such a file stored in your default VMS directory ends with .;n, where n is the version number and both the file name and the type are null strings.

***.prompt-name**
Is a prompting value expression that prompts you for a file specification to which you want to write the output of the statement.

Restrictions

- To print data from records in a domain, you must ready the domain for READ, WRITE, or MODIFY access. You cannot print data from domains readied for EXTEND access because you cannot establish collections or record streams from domains readied for EXTEND access. See the section in this chapter on the READY command for more information.
- If you specify a device name in a PRINT statement, the device must be one to which you have access, such as a line printer, a tape drive, your own terminal, or another terminal. You cannot cause the output of the PRINT statement to be displayed on another terminal that is logged in.
- When you use ON LP: to send PRINT statement output directly to a line printer, DATATRIEVE assumes your system has a line printer. If your system does not have a device defined as LPA0:, the clause ON LP: will not work.

  If you do not specify an ON statement or ON clause, DATATRIEVE sends the output of the PRINT statement to the device assigned to the logical name SYS$OUTPUT. Usually, this is your terminal (TT:).

  Although this restriction applies to any system without a line printer, you may encounter it unexpectedly if your system is part of a VAXcluster with a common line printer. The ON LP: clause does not work in a VAXcluster that uses a common printer not directly connected to your system.

If the nodes in the cluster are connected with DECnet, you can work around this restriction. To send output from a node without a line printer, you must include the node name of the system with the line printer in the LP: specification. For example, if the cluster's line printer is on a node named BIGVAX, the following print statement sends output to it:

```
DTR> PRINT YACHTS ON BIGVAX::LP:
```

Note that you cannot directly specify the line printer on BIGVAX by using the cluster device-name BIGVAX$LPA0: in the ON clause.

- To print character string literals, you must enclose the string with quotation marks. Single and double quotation marks may be used, but you must use like types of quotation marks in pairs. That is, you cannot begin a string with a single quotation mark and end with a double quotation mark.

  Either kind of quotation mark may be contained in a character string literal defined by the other kind. Here are two valid examples:

```
DTR> PRINT "THEY'RE GONE."
THEY'RE GONE.

DTR> PRINT 'THEY SAID, "GOOD-BYE!"'
THEY SAID, "GOOD-BYE!"

DTR>
```

  To include a quotation mark of the same type as the ones that define the literal, you must type two quotation marks for every one you want to include in the output of the statement. The following examples illustrate the type of alternatives you can choose for complex character string literals.

```
DTR> PRINT "THEY SAID, ""WE'RE GOING."""
THEY SAID, "WE'RE GOING."

DTR> PRINT 'THEY SAID, "WE''RE GOING."'
THEY SAID, "WE'RE GOING."

DTR>
```

- If you want to use a form to display the records with the PRINT statement, the definition of the domain with which you get access to the records must include a FORM clause. The domain and form definitions must adhere to the restrictions listed in the section on the DEFINE FILE command.

  SET FORM must be in effect when you ready the domain and when you enter the PRINT statement. Check the status of the SET FORM/NO FORM setting with the SHOW SET_UP command.

  DATATRIEVE uses a form to display records only if you omit the print list from the PRINT statement. That is, you must retrieve entire records from the target collection or record stream.

- To suppress or specify a column header on a print list element that includes a CDD/Repository object, you must enclose the entire value expression in parentheses. Examples of such print list elements include the following:
  - A field from a domain table or dictionary table using a VIA clause. This following example shows such a print list element:

    ```
    DTR> PRINT (RIG VIA RIG_TABLE) (-)
    ```

  - A value that is determined by a procedure. For example:

    ```
    DTR> PRINT (:DURATION) ("DURATION")
    ```

  You must use parentheses around the CDD/Repository object expression in these cases to avoid confusing the column header specification with a password for the CDD/Repository object.

Results

- DATATRIEVE evaluates the print list and writes the resulting output to the specified or implied file or device. The format and content of the output depends on the print list elements you include in the statement. Unless a COL n, SPACE [n], or TAB [n] element changes the position of the cursor, all output begins at column 1.
- If you do not put the PRINT statement in a FOR loop and do not include an RSE or the argument ALL, DATATRIEVE uses the data from the selected record of the most recently formed collection with a selected record to evaluate the print list. DATATRIEVE evaluates the print list once and creates one or more lines of output, depending on the formatting options you specify.
- If you specify the argument ALL and do not include an RSE, DATATRIEVE uses the data in the records of the CURRENT collection to evaluate the value expressions in the print list. DATATRIEVE evaluates the print list once for each record in the CURRENT collection and creates one or more lines of output for each record, depending on the formatting options you specify.
- If you include a record selection expression in a PRINT statement, DATATRIEVE uses the data in each record in the record stream to evaluate the value expressions in the print list. DATATRIEVE evaluates the print list once for each record in the record stream and creates one or more lines of output for each record, depending on the formatting options you specify.
- If you put a PRINT statement in a FOR loop, DATATRIEVE uses the data in each record in the record stream created by the RSE in the FOR statement. DATATRIEVE evaluates the value expressions in the print list once for each record and creates one or more lines of output for each record, depending on the formatting options you specify.

- Table 4–24 describes the print list elements you can include in the PRINT statement.

Table 4–24 Print List Elements

Print List Element	Function and Results
`field-name [modifier]` `list-field-name [modifier]`	Specifies the field whose contents are to be output. The optional modifier describes the column header for the field, or the format of the output, or both (see Results section). If the field is a group field, DATATRIEVE displays all elementary fields contained in that group field. If the field is a list and SET SEARCH is in effect, DATATRIEVE outputs one value of the list field per line per record. If you omit this element, all elementary fields are output.

`literal [modifier]`
`*.prompt-name [modifier]`
`**.prompt-name [modifier]`
`arithmetic-exp [modifier]`
`statistical-exp [modifier]`
	Specifies a value expression to be evaluated and output. The optional modifier describes the column header for the value expression, or the format of the output, or both (see Results section). Chapter 1 discusses these value expressions.

`SPACE [n]`	Inserts n horizontal spaces before the next print list element. If you omit n, DATATRIEVE inserts one space before the next print list element.

(continued on next page)

Table 4–24 (Cont.) Print List Elements

Print List Element	Function and Results
`TAB [n]`	Inserts the space of n tab characters before the next print list element. If you omit n, DATATRIEVE inserts the space of one tab before the next print list element. DATATRIEVE assumes that tabs are set every eight spaces and inserts enough spaces (not actually tab characters) in the print line to start the next print list element in appropriate column.
`COL n`	Determines that the following print list element begins in column n of the detail line. If n is less than the current column number, DATATRIEVE skips a line and begins the next print list element in column n. The first column in the line is column 1.
`SKIP [n]`	Begins the output of the next print list element at the beginning of the nth line from the current line. If n is greater than 1, the intervening lines are blank. If you omit n, DATATRIEVE moves the cursor to the beginning of the next line. If you omit this print list element, DATATRIEVE displays multilined output on consecutive lines.
`NEW_PAGE`	Moves the cursor to the top of a new print page. Column headers are suppressed, and output begins at column 1 unless another print list element changes the position of the cursor.

(continued on next page)

Table 4–24 (Cont.) Print List Elements

Print List Element	Function and Results
`ALL print-list OF rse`	Specifies an inner print list. DATATRIEVE evaluates the inner print list once for each record specified by the outer RSE. This print list element is generally used with a list in a hierarchical record to display all the values in the list for each of the records. When you work with lists, the list name is the source of the RSE in the print list element. If an inner print list is the first element in a print list, you must add a required ALL before the inner print list (ALL ALL print-list OF rse OF rse). This additional ALL is not required if another print list element precedes the inner print list.

- You can use print list modifiers to control the column header displayed above a value expression. The modifiers also allow you to specify an edit string that determines the format of each individual value expression in the print list. DATATRIEVE allows the following modifiers:

  ("header-segment"[/...])

  Specifies one or more character string literals to be displayed as a column header above the first line of output from the PRINT statement. The entire modifier must be enclosed in parentheses and must immediately follow its associated field name or value expression.

  If you specify one header segment, the literal is printed on one line above the first line of output from the PRINT statement. If the header is shorter than the field reserved for that value, the header is centered above the field. If the header is longer than the field reserved for its associated value by the edit string, the field is centered under the header. In this case, however, DATATRIEVE determines the placement of the other output fields relative to the length of the header, not the length of the field.

  If you specify more than one header segment (`"header-1"/"header-2"[/...]`), the segments are printed on successive lines, centered above the associated field. The width of the field is determined by the edit string for the field or the longest header segment, whichever is longer.

(–)

If you specify a hyphen in parentheses (-) following a field name or a dictionary table value expression, DATATRIEVE does not display any query header associated with the field in its record definition or with the dictionary table in its CDD/Repository definition.

If you omit the column header modifier after a field name, variable, or dictionary table value expression, DATATRIEVE uses the query header for the element if one has been defined. If no query header has been defined for a dictionary table, no header is displayed. If no query header has been defined for the elementary field, the field name is used. If the field name has underscores in it, DATATRIEVE suppresses the underscores and converts the field name to a multiline header (for example, see LENGTH_OVER_ALL). If no query header has been defined for the variable, the variable name is used.

For the header of value expressions formed with the statistical functions, MAX, MIN, AVERAGE, STD_DEV, and TOTAL (for example, MAX PRICE), DATATRIEVE combines the name of the function and the field name to form a multiline query header. For functions (beginning with FN$), DATATRIEVE uses the function name to form a column header. For all other value expressions, DATATRIEVE does not output a column header.

USING edit-string

Imposes the characteristics of the specified edit string on the preceding field or value expression. The edit string must conform to the rules that govern the EDIT_STRING clause of a DATATRIEVE record definition. See the section in this chapter on the EDIT_STRING clause for more information.

If you follow an edit string with other print list items, be sure to put a space between the last character of the edit string and the comma that separates the edit string from the next print list element.

If you omit this modifier for a field name or statistical expression, DATATRIEVE uses the edit string specified for the field in the record definition or the PICTURE (or PIC) clause if no edit string is given.

If you omit this modifier for a variable, DATATRIEVE uses either the edit string specified in the DECLARE statement that created the variable or the PICTURE (or PIC) clause if no edit string was given.

If you omit this modifier for a prompting value expression, DATATRIEVE uses a default alphanumeric edit string 10 characters long, that is, X(10).

- If you use a prompting value expression to specify the output file or device, DATATRIEVE prompts you for the name when it executes the PRINT statement. If you omit this argument, DATATRIEVE displays the output on your terminal or workstation.

- If you end your file specification with the name of a line printer or another terminal that is not a spooled device, the output of a PRINT statement can be immediately displayed on the device. Consult the VMS documentation set for details regarding spooled devices.

Usage Notes

- The print list argument allows you to specify the following types of information:
  - The data to be included in the output. Data can be the contents of a field, the value of a variable, or any other value expression.
  - The format of the data in the output. You can specify an edit string to override any edit string in the field or variable definition or to format a value expression.
  - The spacing (both horizontal and vertical) for the output. You can insert tabs or spaces between columns or skip lines between lines of output.
  - Column headers for each column of data in the output. You can also indicate that no header is to be printed above a column.
- When you enter a PRINT statement that has no print list (PRINT ALL or PRINT rse), DATATRIEVE automatically formats the output for you. DATATRIEVE uses the following defaults when you omit the print list:
  - The data included in the output is the contents of all fields in the selected record (PRINT), the records in the CURRENT collection (PRINT ALL), or the records in the record stream formed by the RSE in the PRINT statement (PRINT rse).
  - The format of the field contents is determined by the record definition.
  - The horizontal spacing is based on the longest of three items: the edit string if one is specified, the longest segment of the header, or the length of the value of the print list element.

    The output begins in column 1. The output is single spaced with a single blank line following the header line.
  - Column headers for fields are the query headers or the field names if there is no query header. If the field name contains an underscore, DATATRIEVE suppresses the underscore and places each part of the field name on a separate line. The header is centered above the column of data. If the query header contains only a hyphen, DATATRIEVE does not print a header.

- You can use inner print lists to display data in the lists of hierarchical records.
- You can use the ON clause of the PRINT statement to specify only one output file at a time. Use nested ON statements to write the output of the PRINT, LIST, REPORT, or SUM statements to multiple files.
- With SET SEARCH in effect, you can print the data in lists by letting DATATRIEVE generate implicit inner print lists. You must, however, establish the appropriate context for the target record containing the list and for the list itself. SET SEARCH supplies the ALL and OF rse elements of the inner print list; you supply the print list.

  For example, display the father and children of the first two families:

```
DTR> SET SEARCH
DTR> READY FAMILIES
DTR> PRINT FATHER, EACH_KID OF FIRST 2 FAMILIES

                KID
  FATHER        NAME     AGE

JIM           URSULA       7
              RALPH        3
JIM           ANNE        31
              JIM         29
              ELLEN       26
              DAVID       24
              ROBERT      16

DTR>
```

Examples

The following example retrieves data from selected records and prints the data:

```
DTR> FIND FIRST 2 YACHTS
DTR> SELECT; PRINT TYPE, LOA, PRICE

                         LENGTH
                          OVER
MANUFACTURER    MODEL      ALL     PRICE

 ALBERG       37 MK II     37    $36,951
```

The following example shows how to retrieve data from target streams formed by FOR loops:

```
DTR> FOR FIRST 2 YACHTS
CON> PRINT TYPE, LOA, PRICE

                        LENGTH
                         OVER
MANUFACTURER   MODEL      ALL    PRICE

 ALBERG      37 MK II     37    $36,951
 ALBIN       79           26    $17,900

DTR>
```

The following example shows how to retrieve data from hierarchical records using nested FOR loops:

```
DTR> FOR FIRST 2 FAMILIES
CON> FOR KIDS
CON> PRINT MOTHER, FATHER, KID_NAME

                        KID
  MOTHER     FATHER     NAME

ANN         JIM        URSULA
ANN         JIM        RALPH
LOUISE      JIM        ANNE
LOUISE      JIM        JIM
LOUISE      JIM        ELLEN
LOUISE      JIM        DAVID
LOUISE      JIM        ROBERT

DTR>
```

The following example shows how to retrieve data from the CURRENT collection:

```
DTR> FIND FIRST 2 YACHTS
DTR> PRINT ALL TYPE, LOA, PRICE

                        LENGTH
                         OVER
MANUFACTURER   MODEL      ALL    PRICE

 ALBERG      37 MK II     37    $36,951
 ALBIN       79           26    $17,900

DTR>
```

The following example shows how to retrieve data from record streams formed by the PRINT statement using One RSE:

```
DTR> PRINT TYPE, LOA, PRICE OF FIRST 2 YACHTS

                        LENGTH
                         OVER
MANUFACTURER   MODEL      ALL    PRICE

 ALBERG      37 MK II     37    $36,951
 ALBIN       79           26    $17,900
```

```
DTR>
```

The following example shows how to retrieve data from record streams formed by the PRINT statement using two RSEs with the inner print list following another list:

```
DTR> PRINT FATHER, MOTHER,
CON>    ALL KID-NAME OF KIDS OF FIRST 1 FAMILIES

                          KID
  FATHER      MOTHER      NAME

JIM         ANN         URSULA
                        RALPH

DTR>
```

The following example shows how to retrieve data from record streams formed by the PRINT statement using two RSEs with the inner print list preceding all other print lists:

```
DTR> PRINT ALL ALL KID-NAME OF FIRST 1 KIDS,
CON>    MOTHER OF FIRST 2 FAMILIES

   KID
   NAME       MOTHER

URSULA      ANN
ANNE        LOUISE

DTR>
```

PRINT Statement (Report Writer)

Specifies the following characteristics of the detail lines in a report:

- The content, such as field values, other desired values, and text strings
- The format of fields, including issues such as the order, column position, print attributes, and edit string format of print objects
- Column headers for print objects

You can include only one PRINT statement in a report specification. If your report specification contains an AT statement, then it does not have to contain a PRINT statement.

Format

PRINT print-list-element [, . . .]

Argument

print-list-element

Specifies the values, position, print attributes, and format of the print objects in the detail line.

Table 4–25 indicates the parameters of the report controlled by various print list elements.

Table 4–25 Report Parameters Controlled by Print List Elements

Parameter	Print List Element	Usage Notes
Content of detail line	Field-name [modifier]	Can include elementary, group, list, REDEFINES or COMPUTED BY fields; to print all fields, specify the top-level field name.
	Related value-expression [modifier]	Derived from field values using arithmetic operators or RUNNING TOTAL.
	Other value-expression [modifier]	Can include literals, variables, or RUNNING COUNT.

(continued on next page)

Table 4–25 (Cont.) Report Parameters Controlled by Print List Elements

Parameter	Print List Element	Usage Notes
Format of detail line	ATT { RESET / name }	Resets the attributes, or uses the named attribute list. (See the DECLARE_ATT statement.)
	SKIP [n]	Begins printing the next print list element n lines from the current line. Ignored in DTIF format.
	SPACE [n]	Leaves spaces between the output of the preceding and following print list elements. See also Usage Notes.
	COL n	Specifies where the output of the next print list element begins. See also Usage Notes.
	TAB [n]	Inserts the space of n tab characters before the output of the next print list element. See also Usage Notes.
Beginning of new page	NEW_PAGE	Causes the Report Writer to start a new report page. Ignored in DTIF

Table 4–26 indicates the parameters of the report controlled by modifiers of print list elements.

Table 4–26 Report Parameters Affected by Print List Modifiers

Parameter	Print List Modifier	Usage Notes
Column headers for print items	("header-segment"[/ . . .])	Specifies one or two line headers for the preceding field or value expression, overriding the field name or query header from the record definition.

(continued on next page)

Table 4–26 (Cont.) Report Parameters Affected by Print List Modifiers

Parameter	Print List Modifier	Usage Notes
	(-)	Suppresses the header indicated for the field in the record definition.
Format of the detail line item	USING edit-string	Imposes the characteristics of the edit string on the preceding field name or value expression.

Usage Notes

- Unlike the PRINT statement used at the DATATRIEVE command level, the Report Writer PRINT statement must be followed by at least one print list element. If you enter PRINT without a print list element, the Report Writer prompts you for one.
- If the value of the COLUMNS_PAGE attribute (of the Report Writer SET statement) is too small to accommodate all the fields, the Report Writer carries the overflow fields onto the next line. No field is split between lines, but the column headers of the overflow fields may be lost.
- When specifying the detail line, you are not restricted to the field order of the record definition. In the PRINT statement, you can list the fields (and their column headers and edit strings) in any order you choose.
- When the data you are reporting includes a list, use an inner print list element (ALL [print-list] OF rse) in the PRINT statement to specify the value, position, and format for fields in the list. Each item of the list takes at least one physical line of printing.
- For relational sources, DATATRIEVE uses the relation's name as the top-level group field. This top-level field cannot be used as a print list element. For example, the following statements using the COLLEGES relational domain produce an error message:

```
DTR> REPORT COLLEGES
RW>  PRINT COLLEGES
RW>  END_REPORT
"COLLEGES" is undefined or used out of context.
```

- To suppress or specify a column header with a print list element that includes a dictionary object, you must enclose the entire value expression in parentheses. Examples of such print list elements include the following:
  - A field from a domain table or dictionary table using a VIA clause. An example of such a print list element follows:

```
                    .
                    .
                    .
    RW> PRINT (RIG VIA RIG_TABLE) (-)
```

  - A value that is determined by a procedure, for example:

```
                    .
                    .
                    .
    RW> PRINT (:DURATION) ("DURATION")
```

    The use of parentheses around the dictionary object expression is necessary in these cases to avoid confusing the column header specification with a password for the dictionary object.

    For more information on column headers, see the chapter in the *VAX DATATRIEVE User's Guide* on designing a report with the Report Writer.

```
RW> PRINT MANUFACTURER ("VENDOR"), -
CON> MODEL (-), LOA, RIG, PRICE (-)
```

    This produces the following report:

```
                                                    5-May-1987
                                                    Page 1

                              LENGTH
                               OVER
    VENDOR                     ALL        RIG

  ALBERG        37 MK II        37       KETCH     $36,951

DTR>
```

- For fields with one or two character values, you can compress the headers. The following PRINT statement specifies a three-line header for the LOA field:

```
RW>  PRINT MANUFACTURER, LOA ("L"/"O"/"A"), RIG, PRICE
```

This produces the following report:

```
                                          28-May-1987
                                          Page 1

                      L
                      O
MANUFACTURER          A           RIG          PRICE

 ALBERG               37        KETCH         $36,951
```

- If you do not specify positions or edit strings for any of the fields in a detail line, the Report Writer determines the format for those fields using these criteria:
  - If the field definition contains an edit string, that edit string determines the format for the field.
  - If the field definition has no edit string, the PICTURE clause determines the format for the field.
  - If the field definition has neither an edit string nor a PICTURE clause, the Report Writer invents a picture clause to accommodate the data in the field.

  To gain full control over the formats of the fields of your detail lines, explicitly define edit strings with a USING clause.
- If the print list includes a variable or a field name and a statistical function (TOTAL, RUNNING TOTAL, AVERAGE, MAX, MIN, or STD_DEV) based on that variable or field, the Report Writer puts both values in the same column, even if you specify a separate column header with the statistical function:

```
DTR> FIND FIRST 3 PAYABLES
DTR> REPORT CURRENT
RW> PRINT WHSLE_PRICE,
RW>  TOTAL WHSLE_PRICE ("GRAND TOTAL"/"OWED") USING $$$,$$$
RW> END_REPORT
                                                    13-Aug-1987
                                                    Page 1

                                    WHSLE
                                    PRICE

                                  $40,000
                                  $82,000
                                     .
                                     .
                                     .
```

You can print the statistical function in a separate column by specifying a column assignment:

```
DTR> FIND FIRST 3 PAYABLES
DTR> REPORT CURRENT
RW> PRINT WHSLE_PRICE,
RW>  COL 15, TOTAL WHSLE_PRICE ("GRAND TOTAL"/"OWED") USING $$$,$$$
RW> END_REPORT
                                                    13-Aug-1987
                                                    Page 1

    WHSLE      GRAND TOTAL
    PRICE         OWED

   $40,000       $82,000
             .
             .
             .
```

Note that you may also wish to specify an explicit column for the statistical function in cases where the calculated size of the statistical function value is significantly larger than the size of the field or variable it is based on. This can happen, for example, if the definition of the field or variable includes a COMPUTED BY clause.

- COL, TAB, SKIP, and SPACE clauses are ignored in DTIF format.
- In DDIF and PS formats, the physical page is divided into a grid of either 80 columns (if PAPER_ORIENTATION is set to PORTRAIT) or 132 columns (if PAPER_ORIENTATION is set to LANDSCAPE). One tab stop refers to a group of 8 of these columns. The clauses COL, TAB, and SPACE are used with reference to this grid.
- In ASCII formats (TEXT and default), the COL, TAB, and SPACE clauses refer to actual character spaces. Space characters (blank) are inserted to produce the desired spacing.
- When you insert an ATT clause in a PRINT statement, the relevant attributes will be applied for all subsequent fields until the end of the print list. To reset the attributes to their default values, use the ATT RESET clause. Note that an ATT clause changes only the specified attributes (the effect of several ATT clauses is therefore cumulative) and that attributes are reset at the end of the print list.
- Some or all of the default text attributes for the PRINT statement can be modified using the logical name DTR$RW_BODY_ATTRIBUTES. The equivalence string will be an attribute list, using the same syntax as the DECLARE_ATT statement. Attributes which are not defined in such a list remain set to the default value (see the DECLARE_ATT statement).

For example, if a 10-point Italic Courier font is desired as default for the detail line, the following logical name definition should be applied before invoking DATATRIEVE:

```
$ define DTR$RW_BODY_ATTRIBUTES
_Equ name: "FAMILY=COURIER,ITALIC"
```

- DTIF format reports have some special characteristics:
  - all the values in a column have the same datatype, edit string and alignment
  - every value expression, field, or value derived from a field is put in a separate column
  - print items are drawn in columns, not necessarily in the same order as they would have been in the corresponding printed format. For example, given the statement

    ```
    AT TOP OF field 1
     PRINT field 1
    PRINT field2,field3,field4
    AT BOTTOM OF field1
     PRINT "Total: ",TOTAL field4
    ```

    the result of "TOTAL field4" is put in the same column as field4, while "Total: " is put in a separate column after those occupied by field1, field2, field3, and field4.
  - Group fields are expanded so that columns contain elementary fields only.
- When using proportionally-spaced fonts, (which are accessible from DDIF and PostScript) column data is aligned according to the following conventions:

Data Type	Alignment
Integer	Right
Decimal	Decimal Point
Floating Point	Decimal Point
Character String	Left
Date	Right

- The scope of each attribute is the Report Writer PRINT statement. Therefore a text attribute is effective until the end of the PRINT statement, unless it is superseded by other attributes.

Example

For examples of the PRINT statement in the Report Writer, see the *VAX DATATRIEVE User's Guide*.

PURGE Command

Deletes all but the highest version of specified dictionary objects.

Format

PURGE [path-name [,...] | ALL] [KEEP [=] n]

Arguments

path-name

Specifies the object you want to purge. The path name must include the name of a domain, record, procedure, or table. The path name cannot contain a version number or a semicolon. If you do not specify a path name, DATATRIEVE purges all objects in your default dictionary directory. PURGE accepts both DMU and CDO style path names.

ALL

Purges all the definitions in the default dictionary directory. ALL is the default.

KEEP [=] n

Specifies the number of versions of each object you want to keep. The number must be greater than zero. The default is KEEP=1.

Restriction

You cannot purge out a version of an object if that version is a member of a relationship with another dictionary object. The owner of the relationship must be deleted before you purge out the member of a relationship. In the case of a domain, the domain must be deleted before an object owned by the domain (a record, database, or another domain) is deleted.

Usage Notes

- By default, PURGE with no arguments is the same as PURGE ALL KEEP = 1.
- Because dictionary directories cannot have multiple versions, PURGE does not delete them. PURGE deletes only the objects in the directory. Specifying a dictionary as the final object in a path name generates an error message.

- If you are purging objects in a DMU format dictionary, you need P (PASS_THRU), S (SEE), R (DTR_READ), and either D (LOCAL_DELETE) or G (GLOBAL_DELETE) access privileges for each object to be deleted. If you do not have S (SEE) and R (DTR_READ) access to every object in a directory, PURGE and PURGE ALL do not delete any versions of any object in that directory.
- If you are purging objects in a CDO format dictionary, you need S (SHOW) and U (CHANGE) access privileges to the dictionary and S (SHOW) and D (DELETE) access privileges to each object to be deleted.

Examples

The following example shows how PURGE deletes all but the highest versions of objects in a user's DMU format directory:

```
DTR> SHOW ALL
Domains:
  * FAMILIES;3  * FAMILIES;2   * FAMILIES;1   * OWNERS;2
  * OWNERS;1    * PETS;2       * PETS;1       * PROJECTS;3
  * PROJECTS;2  * PROJECTS;1   * YACHTS;5     * YACHTS;4

Records:
  * FAMILY_REC;2 * FAMILY_REC;1 * OWNER_RECORD;2 * OWNER_RECORD;1
  * PET_REC;1    * PROJECT_REC;1 * YACHT;2       * YACHT;1

The default directory is CDD$TOP.DTR$USERS.BELL
No established collections.
No ready sources.
No loaded tables.
DTR> PURGE
DTR> SHOW DOMAINS, RECORDS
Domains:
  * FAMILIES;3   * OWNERS;2     * PETS;2      * PROJECTS;3
  * YACHTS;5
Records:
  * FAMILY_REC;2 * OWNER_RECORD;2 * PET_REC;1 * PROJECT_REC;1
  * YACHT;2

DTR>
```

The following example shows the error message generated if you specify a dictionary directory as the final object in the path name. It then shows how PURGE works with the path name and KEEP arguments.

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

DTR> PURGE CDD$TOP.DTR$USERS.BELL
Element "CDD$TOP.DTR$USERS.BELL" is not a Domain, Record, Procedure,
 or Table.
No objects purged for dictionary element "CDD$TOP.DTR$USERS.BELL".

DTR> SHOW RECORDS

Records:
  * FAMILY_REC;4   * FAMILY_REC;3   * FAMILY_REC;2 * FAMILY_REC;1
  * OWNER_RECORD;2 * OWNER_RECORD;1 * PET_REC;1    * PROJECT_REC;1
  * YACHT;2        * YACHT;1

DTR> PURGE CDD$TOP.DTR$USERS.BELL.FAMILY_REC KEEP=2
DTR> SHOW RECORDS
Records:
  * FAMILY_REC;4 * FAMILY_REC;3  * OWNER_RECORD;2 * OWNER_RECORD;1
  * PET_REC;1    * PROJECT_REC;1 * YACHT;2        * YACHT;1

DTR>
```

QUERY_HEADER Clause

Specifies the column header DATATRIEVE uses when it formats the display of a field value for the PRINT statement or for the Report Writer AT and PRINT statements.

Format

QUERY_HEADER [IS] {"header-segment"} [/...]

Argument

"header-segment"

Is the column header displayed above a column of data. If you specify only one character string literal, that string is printed on one line above the column. You can specify more than one character string literal by separating each from the next with a slash (/). The literals are printed on successive lines, centered above the column.

Restriction

This clause is valid for elementary fields only.

Results

- If you include this clause, DATATRIEVE uses the specified query header as the default column header when printing this field.
- If you omit this clause, DATATRIEVE uses the field name as the default column header when printing this field.

Usage Notes

- The column header can include any character except a carriage return, a line feed, or a control character. To include a quotation mark in a column header, precede it with another quotation mark. (See the third example in the Examples section.)
- You can perform the functions of the QUERY_HEADER clause with the column-header modifier for print list elements in the PRINT statement and the Report Writer AT and PRINT statements. The column-header modifier overrides the default query header specified in the field definition. (See the Results section of the PRINT Statement section for more information on the print list modifiers.)

Examples

The following example shows how to set up a record so that when the DISPLACEMENT field is displayed, a column header of WEIGHT is used:

```
06 DISPLACEMENT PIC 99999
            QUERY_HEADER IS "WEIGHT"
            EDIT_STRING IS ZZ,ZZ9
            QUERY_NAME IS DISP.
```

DATATRIEVE prints the column header as follows:

```
WEIGHT
```

The following example shows how to set up a record so that when the LENGTH_OVER_ALL field is printed, a column header of LENGTH (IN FEET) is printed on two separate lines:

```
06 LENGTH_OVER_ALL PIC XXX
   QUERY_HEADER IS "LENGTH" /"(IN FEET)".
```

DATATRIEVE prints the column header as follows:

```
LENGTH (IN FEET)
```

The following example shows how to set up a record so that when the LENGTH_OVER_ALL field is printed, a column header of Length Over All ("LOA") is printed on two separate lines:

```
06 LENGTH_OVER_ALL PIC XXX
   QUERY_HEADER IS "Length Over All"/"(""LOA"")".
```

DATATRIEVE prints the column header as follows:

```
Length Over All
   ("LOA")
```

The following example shows how to set up a record so that when the LENGTH_OVER_ALL field is printed, a 3-line column header is printed with one letter per line:

```
06 LENGTH_OVER_ALL PIC XXX
   QUERY_HEADER IS "L"/"O"/"A".
```

DATATRIEVE prints the column header as follows:

```
L
O
A
```

QUERY_NAME Clause

Specifies an alternate name for the field.

Format

QUERY_NAME [IS] query-name

Argument

query-name
Is the query name. The rules for forming and using a query name are the same as those for a field name.

Restriction

The query name must conform to the rules for names (see the *VAX DATATRIEVE User's Guide*).

Result

You can specify the query name as an alternate name for the field name.

Usage Notes

- This clause is valid for both group and elementary fields.
- Use the QUERY_NAME clause when a field name is too long to use easily.
- Like a field name, a query name can duplicate another query name (or a field name) in the record. A query name can also be qualified by other query names or field names.

Examples

The following example shows how to set up a record so that DISP is defined as an alternate name for the DISPLACEMENT field:

```
06 DISPLACEMENT PIC 99999
   QUERY_NAME IS DISP.
```

The following example shows how to set up a record so that SPECS is defined as an alternate name for the group field SPECIFICATIONS:

```
03 SPECIFICATIONS
   QUERY_NAME SPECS.
```

The following example shows how to set up a record so that a query name of SPECIAL_HANDLING is defined for the field DELINQUENT_ACCOUNT_STATUS:

```
09 DELINQUENT_ACCOUNT_STATUS
   PIC X
   QUERY_NAME IS SPECIAL_HANDLING.
```

READY Command

Gives you access to one or more domains, relations, databases, or record types and controls the access of other users to those domains or databases. You can also use the READY command to ready a domain or database again in order to change your access mode or access option.

Format 1

READY domain-path-name [AT node-spec] [AS alias-1]

```
[ PROTECTED ]  [ READ    ]  [ CONSISTENCY ]
[ SHARED    ]  [ WRITE   ]  [ CONCURRENCY ] [,...]
[ EXCLUSIVE ]  [ MODIFY  ]
               [ EXTEND  ]
```

[SNAPSHOT]

Format 2

READY database-path-name

```
[SNAPSHOT]
[ PROTECTED ]  [ READ    ]  [ CONSISTENCY ]
[ SHARED    ]  [ WRITE   ]  [ CONCURRENCY ]
[ EXCLUSIVE ]  [ MODIFY  ]
               [ EXTEND  ]

[ USING { rdb-relation-name } [ AS alias ]                                  ]
        { dbms-record-name }
        [SNAPSHOT]                                                          [,...]
        [ PROTECTED ]  [ READ    ]  [ CONSISTENCY ]                         ]
        [ SHARED    ]  [ WRITE   ]  [ CONCURRENCY ]                         ]
        [ EXCLUSIVE ]  [ MODIFY  ]                                          ]
                       [ EXTEND  ]                                          ]
```

Arguments

domain-path-name

Is the dictionary path name of a domain to which you want access or the domain table whose access option you want to change.

node-spec
Is the name of a node and an optional access control string.

A node name is a 1- to 6-character name that identifies the location on the network. Examples are BIGSYS and LILSYS.

An access control string indicates a particular account on the remote node. It consists of a user name, followed by one or more blanks or tabs, a password, and an optional account name.

The following three node specifications are valid:

```
BIGSYS
BIGSYS"MORRISON RYLE"
BIGSYS"MORRISON RYLE KANT"
```

On DECnet links to some non-VMS systems, you can use the UIC number in place of the user name. For example:

```
BIGSYS"[240,240] RYLE"
BIGSYS"[240,240] RYLE KANT"
```

In the examples, the remote node is BIGSYS the user name is MORRISON, the UIC is [240,240], the password is RYLE, and the account name is KANT.

You can also use a prompting value expression to prompt the user for the user name, password, account name, or UIC. Use single quotation marks for the prompt string. For example:

```
DTR> READY CDD$TOP.DTR32.MORRISON.YACHTS AT
CON>     BIGSYS"*.'username' *.'password'"
Enter username: MORRISON
Enter password:
DTR>
```

alias
Is a name you use if you include the AS clause in the READY command to refer to the domain, relation, or VAX DBMS record specified. You use the alias in place of the given name of the domain, relation, or VAX DBMS record where the syntax of a statement calls for that domain, relation, or VAX DBMS record name. If you are using an alias for a domain name, do not use the alias in full or relative dictionary path names.

database-path-name
Is the CDD/Repository path name defined for the DATATRIEVE definition of the VAX DBMS, Rdb/VMS, Rdb/ELN, or VIDA facility database, or for the relational database definition created by Rdb/VMS. If you ready a VAX DBMS or relational database without specifying any relation or record names, DATATRIEVE readies

all relations or records in the database. READY accepts both DMU and CDO style path names.

relation-name
Is the name assigned to the relation when the relational database was created. The relation name can be the name of a view relation.

record-name
Is the name assigned to the VAX DBMS record when the database was created.

SNAPSHOT
PROTECTED
SHARED
EXCLUSIVE
Are the options you can select to control the access of other users to a domain, relation, or VAX DBMS record you ready. The specific constraints are explained in Table 4–27:

Table 4–27 Access Options

Option	Access Constraints
SNAPSHOT	SNAPSHOT is a READ access for databases that takes a "picture" of the database when it is readied. In order to have SNAPSHOT access, all relations or records pertaining to the database must be readied with SNAPSHOT access. Any other user can access the same database with any access mode and option. In general, you do not see other users' changes until the end of the transaction. If you use SNAPSHOT with another option, such as CONCURRENCY, you may see other users' changes, depending on the database system. SNAPSHOT is the default for relational databases, relations, and domains based on relational sources.
PROTECTED	Any other user can have only READ access to records in the domain or relation. No other user can have WRITE, MODIFY, or EXTEND access to the records in the domain or relation. This option is the default for domains containing records from VAX RMS files and for all view domains.

(continued on next page)

Table 4–27 (Cont.) Access Options

Option	Access Constraints
SHARED	Any other user can have access to the domain, database, relation, or VAX DBMS record at the same time, in any access mode. This option is the default for VAX DBMS domains, VAX DBMS databases, and VAX DBMS records.
EXCLUSIVE	No other user can have access to the domain, database, relation, or VAX DBMS record at the same time, in any access mode. If the domain is an RMS domain, the file containing the data is locked by RMS.

READ
MODIFY
WRITE
EXTEND

Are the options you can select to request a mode of access to a domain, database, relation, or VAX DBMS record. When using Format 1, whether you get that mode of access is determined by privileges assigned to you in the access control list of the domain. (See the chapter on ACL in the *VAX DATATRIEVE User's Guide*.) When using Format 2, whether you get that mode of access is determined by privileges assigned to you in the access control list of the database definition.

When you are using either Format 1 or Format 2 and you are accessing a VAX DBMS or relational database, you must also have appropriate privileges in the VAX DBMS, Rdb/VMS or Rdb/ELN access control lists to request the access mode you select. For VAX DBMS, you need access to the schema, subschema, record and DATATRIEVE database definitions in the data dictionary. Appendix B explains the access modes and lists the privileges (in the DATATRIEVE access control list) that give you each mode of access to a domain.

CONSISTENCY
CONCURRENCY

Are the options you can select to determine whether you can see changes made by other users to the data you are accessing. CONSISTENCY guarantees that while you are accessing data, you do not see updates 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 from the most recent ready still in effect for a relational source.

CONCURRENCY allows you to see other users' updates to the data you are accessing.

READY Command

Table 4–28 summarizes the effects of various combinations of access options and access modes for RMS domains.

Table 4–28 Multi-User Access to RMS Domains

You Ready a Domain	Another User Can Then Ready the Domain	Your Effect on Other Users	Other Users'Effect on You
EXCLUSIVE READ EXCLUSIVE WRITE	No access.	No one else can read the file.	No effect.
PROTECTED READ	PROTECTED READ SHARED READ	No one else can write to the file.	No effect.
PROTECTED WRITE	SHARED READ	No one else can write to the file.	No effect.
SHARED READ	PROTECTED READ PROTECTED WRITE SHARED READ SHARED WRITE	No one with WRITE access can select your selected record.	Users with WRITE access may change records you are reading or have read.

(continued on next page)

Table 4–28 (Cont.) Multi-User Access to RMS Domains

You Ready a Domain	Another User Can Then Ready the Domain	Your Effect on Other Users	Other Users'Effect on You
SHARED WRITE	SHARED READ SHARED WRITE	No one else can modify your selected record or the target record of your MODIFY or ERASE statement. You can modify a record another user has just modified.	You cannot write to the selected record of any other user. You cannot write to the target record of a MODIFY or ERASE statement entered by a SHARED WRITE user. A SHARED WRITE user can also write to a record you have just modified.

When two applications try to access the same RMS domain, RMS may lock a record that DATATRIEVE needs to access. DATATRIEVE then tries for 12 seconds to access the record. At the end of this period, DATATRIEVE takes one of two actions, depending on whether SET LOCK_WAIT is in effect.

- If SET NO LOCK_WAIT is in effect, you receive an RMS$_RLK message: "Target record currently locked by another stream." Then DATATRIEVE aborts the statement. SET NO LOCK_WAIT is the default.
- If SET LOCK_WAIT is in effect, DATATRIEVE turns control over to RMS to wait for the record. You cannot use CTRL/C to cancel the wait. RMS waits until the record is released, or, in case of deadlock, you receive the RMS deadlock error message.

The LOCK_WAIT setting applies to all sources, including VAX DBMS and relational databases which are readied after the SET LOCK_WAIT command is issued. NO LOCK_WAIT was selected as the default for DATATRIEVE because it is the RMS default. However, NO LOCK_WAIT is not always the recommended mode for relational access. LOCK_WAIT is required for VIDA.

Table 4–29 summarizes the effects of various combinations of access options and access modes for VAX DBMS domains and Rdb/VMS or Rdb/ELN domains and relations.

Table 4–29 Multi-User Access to VAX DBMS, Rdb/VMS, and Rdb/ELN Sources

You Ready a Domain, Database, Relation, VAX DBMS Record	Another User Can Then Ready the Domain, Database, VAX DBMS Record	Your Effect on Other Users	Other Users' Effect on You
EXCLUSIVE READ EXCLUSIVE WRITE	No access.	No one else can read the realm or relation.	No effect.
PROTECTED READ	PROTECTED READ SHARED READ	No one else can write to the realm or relation.	No effect.
PROTECTED WRITE	SHARED READ	No one else can write to the realm or relation. Other users may encounter write locks during your transaction.	You may encounter read locks other users have put on a record when you try to modify it.
SHARED READ	PROTECTED READ PROTECTED WRITE SHARED READ SHARED WRITE	A SHARED WRITE user may have to wait until you release your read locks.	You may encounter write locks during another user's transaction.

(continued on next page)

Table 4–29 (Cont.) Multi-User Access to VAX DBMS, Rdb/VMS, and Rdb/ELN Sources

You Ready a Domain, Database, Relation, VAX DBMS Record	Another User Can Then Ready the Domain, Database, VAX DBMS Record	Your Effect on Other Users	Other Users' Effect on You
SNAPSHOT for Rdb/VMS or Rdb/ELN domains and VAX DBMS records	With any access and mode.	No effect on users.	You do not see changes other users make to the database until a COMMIT or ROLLBACK is performed.
SHARED WRITE	SHARED READ SHARED WRITE	Other users may encounter read and write locks during your transaction.	You may encounter read and write locks during another user's transaction.

Restrictions

Readying many sources at one time may produce the following severe error message:

```
Internal error (all available dynamic pools have been exhausted).
```

This error occurs when DATATRIEVE can no longer allocate an internal memory pool. To avoid this error, keep in mind the following:

- Ready only the sources you need at any one time.
- When dealing with very large relational or VAX DBMS databases, use the USING clause to ready only those relations or records that you need.
- Use the FINISH command to end access to any sources you are no longer using.

The following restrictions apply when you use Format 1:

- You can ready only one version of a domain at any one time. To achieve the effect of readying multiple versions of the same domain, ready the most recent version by the domain name and then ready other versions using aliases.

- When you move a domain definition from one part of the data dictionary to another using the CDD/Repository Dictionary Management Utility (DMU) COPY command, you may not be able to ready the domain in its new location.

  Certain DATATRIEVE objects in the data dictionary, such as domains, refer to other objects. For example, domain definitions refer to record definitions and domain table definitions refer to domain definitions. DATATRIEVE stores pointers to referred objects by full path name, even if you used relative path names for those objects in your definitions.

  Whenever DATATRIEVE processes these definitions, it translates references into full CDD/Repository path names. Because of this, you must redefine any object that you copy elsewhere in the CDD/Repository hierarchy using the DMU COPY command. For more information, see the VAX CDD/Repository documentation.
- PROTECTED READ is the default access for RMS domains.
- SNAPSHOT mode is the default access for relational domains. All readied domains pertaining to a relational database must be readied with SNAPSHOT to have SNAPSHOT access.
- SHARED READ is the default access for VAX DBMS domains.
- To ready a domain that is defined in the DMU format dictionary, you must have E (DTR_EXTEND/EXECUTE), S (SEE), and P (PASS_THRU) access privileges to the record definition associated with that domain. (See the chapter on ACL in the *VAX DATATRIEVE User's Guide*.)

  To ready a domain associated with a relation in a relational database, you must also have E (DTR_EXTEND/EXECUTE), S (SEE), and P (PASS_THRU) access privileges to the database definition associated with that domain.
- To ready a domain that is defined in the CDO format dictionary, you must have S (SHOW) access to the dictionary containing the domain, and S (SHOW) access to the record definition associated with that domain. (See the chapter on ACL in the *VAX DATATRIEVE User's Guide*.)

  To ready a domain associated with a relation in a relational database, you must also have E (EXTEND) access to the database definition associated with that domain.
- You must have the privilege to the domain that gives you access to the domain in the mode you specify. See Table 4–29 for a list of the privileges that you need for each access mode.
- The dictionary path name of a DMU domain to be readied must include any passwords associated with the domain and the dictionary directories in its full dictionary path name.

- To ready a domain whose definition is stored in a DMU dictionary directory other than the default one, you must have at least P (PASS_THRU) access to all the ancestors in the domain's path name.
- To ready a domain whose associated record definition is stored in a DMU dictionary directory other than the default one, you must have at least P (PASS_THRU) access to all the ancestors in the record's path name.
- You cannot ready a domain unless the data file specified in the domain definition exists.
- You cannot ready a domain unless the data file associated with it is stored in a VMS directory that matches the file specification included in the domain definition (including any defaults for devices, directories, file name, and file type).
- To ready a domain for READ access, you must have at least VMS READ access to the data file associated with it.
- To ready a domain for WRITE, MODIFY, or EXTEND access, you must have VMS WRITE access to the data file.
- If another user has readied the domain for EXCLUSIVE use, you cannot ready the domain.
- If another user has readied the domain for PROTECTED use, you can ready the domain only for READ access.
- You cannot specify the SHARED access option for a domain using an RMS sequential data file.
- If a conflict occurs between the access mode and the access option you specify (such as trying to ready an RMS sequential file for PROTECTED WRITE), DATATRIEVE automatically readies the domain with the EXCLUSIVE access option.
- RMS does not enforce the EXCLUSIVE access option when you combine it with the READ access mode.
- A restriction exists on the ready of a CDD$DATABASE domain that refers to a CDD$RMS_DATABASE object when the READY command is issued from a DMU format dictionary or with a DMU format path name. Such a ready may cause unpredictable results.

  A CDD$DATABASE domain is a domain based on an existing CDD$DATABASE object. This type of object can be stored only in a CDO format dictionary.

However, if the CDD$DATABASE object is stored in the compatibility dictionary, you may encounter a problem under the following conditions:

- Your default dictionary directory is set to a DMU format dictionary and you use the given name of the CDD$DATABASE domain, as in the command READY CDD_DB_DOM.
- The argument of the READY command uses a DMU format path name, as in the command READY CDD$TOP.DTR32.CDD_DB_DOM

For example, readying the CDD$DATABASE with the following command may cause an access violation:

```
DTR>READY CDD$TOP.DTR32.CDD_DB_DOM
```

You can avoid this problem in either of the following ways:

- Explicitly set your default dictionary to the compatibility dictionary using a path name that begins with either the logical CDD$COMPATIBILITY or with the CDO anchor SYS$COMMON:[CDDPLUS].

  ```
  DTR>SET DICTIONARY CDD$COMPATIBILITY.DTR32
  DTR>READY CDD_DB_DOM
  DTR>
  ```

- Use a full CDO format path name to identify the CDD$DATABASE object, as in the following example:

  ```
  DTR>READY CDD$COMPATIBILITY.DTR32.CDD_DB_DOM
  ```

  Note that you can use either the anchor SYS$COMMON:[CDDPLUS] or the logical CDD$COMPATIBILITY that represents that anchor.

The following restrictions apply when you use Format 2:

- For information about the restrictions on VIDA databases, see the *VAX DATATRIEVE Guide to Interfaces* or the VIDA documentation.
- SNAPSHOT is the default access for Rdb/VMS and Rdb/ELN databases and domains and gives users read-only access to the database. Any other user can access the database with any access option and mode. You do not see changes other users make to the database until you perform a COMMIT, ROLLBACK, or a READY that starts a new transaction.
- SNAPSHOT is the default access for VIDA databases. But, you do see changes that other users make to the data you are accessing when you use SNAPSHOT access for VIDA databases or when you use SNAPSHOT with CONCURRENCY for either VIDA or Rdb/ELN databases. When you ready a VIDA or Rdb/ELN database with both SNAPSHOT and CONCURRENCY, DATATRIEVE behaves as though the access were SHARED READ.

- In the DMU format dictionary, when you ready a database, relation, or a domain pointing to a VAX DBMS record, you must have E (DTR_EXTEND /EXECUTE), S (SEE), and P (PASS_THRU) access privileges to the database definition associated with the relation or the VAX DBMS schema and subschema, the DATATRIEVE/VAX DBMS database definition that points to the VAX DBMS schema and subschema, and the DATATRIEVE definition that points to the Rdb/VMS relation.

  You must also have the privilege for the database definition that gives you access to the relations or VAX DBMS records in the mode you specify. See Table 4–29 for a list of the privileges that you need for each access mode.

- In the CDO format dictionary, when you ready a database or relation you must have E (EXTEND) and S (SHOW) access privileges to the database definition associated with the relation and the DATATRIEVE definition that points to the Rdb/VMS relation.

  You must also have the privilege for the database definition that gives you access to the relations in the mode you specify. See Table 4–29 for a list of the privileges that you need for each access mode.

- The dictionary path name of a database to be readied must include any passwords associated with the database and the dictionary directories in its full dictionary path name.

- To ready a database whose definition is stored in a DMU dictionary directory other than the default one, you must have at least P (PASS_THRU) access to all the ancestors in the database path name.

- You cannot ready a database unless the database root file specified in the database definition exists.

- You cannot ready a database unless the root file specification in the database CDD/Repository definition matches the VMS file specification (including any defaults for devices, directories, file name, and file type).

- To ready a database, one or more relations, or one or more VAX DBMS records for READ access, you must have at least VMS READ access to the database root file associated with it.

- To ready a database, a relation, or a VAX DBMS record for WRITE, MODIFY, or EXTEND access, you must have VMS WRITE access to the database root file.

- If another user has readied a database, relation, or VAX DBMS record for EXCLUSIVE use, you cannot ready the relation or record.

- If another user has readied the database, relation, or VAX DBMS record for PROTECTED use, you can ready the relation or record only for PROTECTED READ access, for SNAPSHOT access, or for SHARED READ access.
- You cannot mix access options when you use SNAPSHOT access. In order to have SNAPSHOT access for any VAX DBMS record, you must ready all VAX DBMS records with SNAPSHOT access.
- DATATRIEVE ignores commit operations while any VAX DBMS, Rdb/VMS, and Rdb/ELN sources are readied in SNAPSHOT mode. If you ready two databases and use SNAPSHOT mode for one of them, you must finish the database readied in SNAPSHOT mode or ready it again in a mode other than SNAPSHOT before you can commit changes to the database that is not in SNAPSHOT mode.
- You cannot ready any part of a remote Rdb/VMS or VAX DBMS database directly. You can, however, create Rdb/VMS and VAX DBMS domain definitions on the remote node and ready the remote databases using those domains. In the case of Rdb/VMS, you must define a DATATRIEVE domain on the remote node for each relation you wish to access. To ready a record type in a remote VAX DBMS database, you must define a domain on the remote node for each record type you wish to access in the remote database.
- You cannot ready a relational database that contains no relations.
- DATATRIEVE requires unique names for all readied sources. When you ready different databases with relations or domains that have the same names, you must use an alias to prevent one name from duplicating another. If you attempt to ready two relations of the same name from different relational databases, DATATRIEVE issues a warning message and does not ready the second source.

Results

- DATATRIEVE gives you access to one or more databases, domains, relations, or VAX DBMS records with the access options and access modes you specify.
- The default access option for RMS domains is PROTECTED, and the default access mode is READ. If you accept the defaults, other users can have READ access to the domain, but not WRITE, MODIFY, or EXTEND. You can retrieve records, but you cannot store, modify, or erase records.
- The default access option for relational sources is SNAPSHOT. Other users can have READ, WRITE, MODIFY, or EXTEND access to the database, domain, or relation. You can retrieve records but cannot store, modify, or erase records.

- The default access option for a VAX DBMS database, domain, or VAX DBMS record, is SHARED. The default access mode is READ. With these defaults, other users can have READ, WRITE, MODIFY, or EXTEND access to the domain, database, or VAX DBMS record. There is read locking, and you do see changes other users make to the data.
- If you use Format 2 and do not specify any relations or VAX DBMS record names, DATATRIEVE gives you access to all the relations or VAX DBMS records in the database with the access options and access mode you specify.
- If you use the USING clause, DATATRIEVE readies only those relations and VAX DBMS records you specify.
- If you want to ready a relation or VAX DBMS record with an access mode and option different from the default, you must specify the access mode and option for each relation or VAX DBMS record.
- If you ready a VAX DBMS database and specify a record with more restrictive access than the access specified in the ready, the following occurs:
  - SHOW READY displays the specified access mode and option for the domains or records.
  - You limit the access of other users to those records or domains until you rollback, commit, and then ready any VAX DBMS domain or record, or until you finish the database.
- If you ready an Rdb/VMS or an Rdb/ELN database with the SNAPSHOT access option and specify a relation with a more restrictive access, the following occurs:
  - The domains or relations are shown as readied with the specified access mode and option.
  - You can access the database and relations that were readied with SNAPSHOT access with only the more restrictive SHARED READ mode until all relations are readied with SNAPSHOT access.
- If you ready a VAX DBMS source with SNAPSHOT access and then ready a second VAX DBMS source with a mode other than SNAPSHOT, DATATRIEVE gives you SHARED READ access to the first source. SHOW READY indicates SNAPSHOT access, but the access mode is SHARED READ. DATATRIEVE returns the access mode on the first source to SNAPSHOT when you finish the second source or ready the second source again in SNAPSHOT mode.

- The CONSISTENCY and CONCURRENCY options relate to relational databases only. They do not have any effect on the readying of a VAX DBMS database. Note, also, that you can ready selected VAX DBMS records in the database. You need not ready them all. You can specify access options for each record, as well as options for the entire database. The following example illustrates this:

```
DTR> READY PARTS_DB USING EMPLOYEE READ, DIVISION WRITE
DTR>
DTR> SHOW READY
Ready sources:
   DIVISION:   Record, DBMS, shared write
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   EMPLOYEE:  Record, DBMS, shared read
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
No loaded tables.
```

  This READY command 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)

- CONSISTENCY is the default access option for the first ready of a relational source. CONSISTENCY access means that while you are accessing data, you cannot see updates made by other users. CONCURRENCY access to relational sources allows you to see other users' updates to the data that you are accessing. Currently, Rdb/ELN, Rdb/VMS, and VIDA implement CONSISTENCY and CONCURRENCY in different ways. See the chapter on accessing relational data in the *VAX DATATRIEVE User's Guide* for more information and for examples of how to access databases using CONSISTENCY and CONCURRENCY.
- Once you specify CONSISTENCY or CONCURRENCY, that option becomes the default until you change the option in a subsequent ready or you finish the database.
- If you supply no alias for a domain, relation, or VAX DBMS record in the READY command, DATATRIEVE uses the given name of the domain, relation, or VAX DBMS record to respond to your commands and statements dealing with that domain, relation, or VAX DBMS record. From then until you release control over the domain, relation, or VAX DBMS record with the FINISH command, you can use the given name in commands, and you must use the given name in all statements requiring a domain name.

- If you assign an alias for a domain, relation, or VAX DBMS record in the READY command, DATATRIEVE assigns that alias as the name of the domain, relation, or VAX DBMS record and you cannot perform operations using its given name.
- If you assign an alias as the name of a database, you receive an error message, and DATATRIEVE ignores the alias.
- If you ready a domain, relation, or VAX DBMS record as SHARED READ and then establish a selected record, DATATRIEVE retrieves the record from storage every time you enter a statement referring to that record.
- If you issue a READY command for a relational or VAX DBMS database, a relational or VAX DBMS domain, a relation, or VAX DBMS record that is ready, the following conditions apply:
  - You can use only the given name or the alias of the domain, relation, or record.
  - The access option and access mode specified in the new READY command replace those previously in effect.
  - You must end access to the database, domain, relation, or VAX DBMS record with the COMMIT, ROLLBACK, or FINISH statement before you can ready any part of the database again. DATATRIEVE can then start a new transaction if the READY command changes an access mode or option.
  - DATATRIEVE preserves collections from databases, domains, records, or relations even if you issue a READY statement that changes access mode.
- DATATRIEVE displays any messages attached to a domain when the domain is readied. Message reporting is done only on domains defined in the CDO dictionary with RELATIONSHIPS.

Usage Notes

- You can optimize record definitions using the OPTIMIZE qualifier for the DEFINE RECORD and REDEFINE RECORD commands. This greatly reduces the CPU time needed to ready a domain that refers to the record.

  DATATRIEVE does not perform this optimization by default; you must specify the OPTIMIZE qualifier in the record definition to optimize a record.

  There are performance and storage tradeoffs you should carefully consider when using the OPTIMIZE qualifier. See the DEFINE RECORD section in this chapter for more information.

- If you have defined a new data file for an RMS domain, you must finish the domain and ready it again for the new file to take effect.
- When you call a domain table based on a relational source, DATATRIEVE performs an implicit ready of the relational source mentioned in the domain table. If a relational source has already been readied, DATATRIEVE cannot perform this implicit ready because of the previous pending transaction.
- If you change the database definition associated with a relational database or VAX DBMS database, you must finish all ready sources associated with the database and then ready it again for the modified database definition to take effect.
- To show the domains, relations, and VAX DBMS records that are ready, use the SHOW READY command. SHOW READY displays the name, file type (for RMS domains), access option, access mode, and full dictionary path name of all readied RMS, VAX DBMS, and relational domains.

  For readied relations and VAX DBMS records, SHOW READY displays the name, type, access option, access mode, and dictionary path name of the database. For relational sources, SHOW READY also displays the consistency option (either CONCURRENCY or CONSISTENCY) that you specified.

  The SHOW READY command displays the most recently readied domain, relation, or VAX DBMS record first.
- The access mode you specify determines the commands and statements you can use on this domain, relation, or VAX DBMS record. The access modes needed for the commands and statements are listed in Table 4–30. Before issuing any other command or statement, you can enter a SHOW READY command to check that you have readied the target domain, relation, or VAX DBMS record with the appropriate access mode.

Table 4–30 Access Modes Required by DATATRIEVE Statements

Statement	Access Mode Required
DISPLAY	MODIFY, READ, or WRITE
ERASE	WRITE
FIND	MODIFY, READ, or WRITE
LIST	MODIFY, READ, or WRITE
MODIFY	MODIFY or WRITE

(continued on next page)

Table 4–30 (Cont.) Access Modes Required by DATATRIEVE Statements

Statement	Access Mode Required
PRINT	MODIFY, READ, or WRITE
REPORT	MODIFY, READ, or WRITE
Restructure	WRITE or EXTEND
SORT	MODIFY, READ, or WRITE
STORE	WRITE or EXTEND
SUM	MODIFY, READ, or WRITE

- You can define your own default access option using the logical name DTR$READY_MODE. DATATRIEVE checks the definition of DTR$READY_MODE only when an access option is not found on the READY command line. DTR$READY_MODE will be applied when you ready RMS, relational, and VAX DBMS sources.

  If you do not specify a default using DTR$READY_MODE, DATATRIEVE uses the default access options listed in Table 4–31.

Table 4–31 DATATRIEVE Default Access Modes

Source	Default Access Mode
RMS sources	PROTECTED
Relational sources	SNAPSHOT
VAX DBMS sources	SHARED

You can assign a default to DTR$READY_MODE in the following ways:

– Use the DATATRIEVE function FN$CREATE_LOG. For example:

```
DTR> FN$CREATELOG ("DTR$READY_MODE", "SHARED")
DTR> FINISH
DTR> READY YACHTS
DTR> SHOW READY
Ready sources:
   YACHTS:  Domain, RMS indexed, shared read
            <CDD$TOP.DTR32.DAB.YACHTS;3>
```

– Use the DIGITAL Command Language (DCL) command DEFINE. For example:

```
$ DEFINE DTR$READY_MODE "SHARED"
```

If you define DTR$READY_MODE using FN$CREATE_LOG, the definition lasts only until the end of the DATATRIEVE session. DATATRIEVE checks the value of DTR$READY_MODE every time a READY command is executed. Therefore, you can use FN$CREATE_LOG to change the definition numerous times during a DATATRIEVE session.

You can also override a system-wide definition of DTR$READY_MODE by making a user or process level definition.

The logical name translation of DTR$READY_MODE is not iterative. If there are multiple equivalence names for DTR$READY_MODE, the first is used as the translation. The translation of DTR$READY_MODE is not case sensitive.

DATATRIEVE performs the following error handling only if you have not included an access option on the READY command line and DTR$READY_MODE is specified but cannot be used:

- DATATRIEVE verifies if the definition of DTR$READY_MODE is a valid access option or synonym for an access option. If it is not valid, DATATRIEVE displays the following message:

  ```
  The value of DTR$READY_MODE, "NOGOOD", is not
  a valid access option.
  ```

  The preceding error occurred while translating DTR$READY_MODE. Therefore, DATATRIEVE will use the default access option.

  The readying will continue using the DATATRIEVE default access option.

- You might specify an access option for DTR$READY_MODE that is not valid for the type of source you are readying. In this instance, DATATRIEVE treats the error as if the invalid access option had been included on the command line.

- When you create a VAX DBMS database, you can specify whether SNAPSHOT access is available. If you allow SNAPSHOT access, you can use the DBO/MODIFY/SNAPSHOTS=NOENABLE command later to disallow it.

  When you ready a VAX DBMS source using SNAPSHOT, and SNAPSHOT is not allowed for that source, DATATRIEVE attempts to ready the source using SHARED READ and displays an informational message. If the source cannot be readied using SHARED READ, DATATRIEVE does not ready it and displays a message that indicates the source has not been readied.

- When you specify access options for a database, domain, relation, or VAX DBMS record, impose as few restrictions on other people as the needs of your task allow. Remember that if you specify EXCLUSIVE access, no other person can get access to the database, domain, relation, or VAX DBMS record.

However, EXCLUSIVE access gives better performance with VAX DBMS because no locking is done.

- If you have a database, domain, relation, or VAX DBMS record readied for SHARED WRITE, you are permitting another user to modify records from the file or database. No other user can modify your selected record or the current target record of your MODIFY statement. Other users cannot modify VAX DBMS records held in collections.

  However, when you modify a record, make sure that you see the current record before you change any values. For example, in the following case, you may not be aware that another user has modified your selected record:

```
DTR> PRINT FIRST 1 YACHTS

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

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

DTR> FIND FIRST 1 YACHTS
[1 record found]
DTR> SELECT
DTR> MODIFY USING PRICE = PRICE * 1.1
```

  Because you displayed the record before you selected it, another user could have modified the record in the interval between entering your PRINT and your SELECT statements.

  To guarantee that the record you are modifying is the same as the record you see, enter the statements in this order: select the record, print it, and then modify it. DATATRIEVE locks the record from the time you select it until you have finished your update. For example:

```
DTR> FIND YACHTS
[113 records found]
DTR> SELECT 1
DTR> PRINT

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

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

DTR> MODIFY USING PRICE = PRICE * 1.1
```

  Another safe method is to set up a FOR loop with an RSE controlling a BEGIN-END block. You can include the PRINT and MODIFY statements in the BEGIN-END block. DATATRIEVE locks the record that is current in the loop, displays it, and then modifies it according to your MODIFY statement. For example:

```
DTR> FOR YACHTS
CON> BEGIN
CON> PRINT
CON> MODIFY PRICE
CON> PRINT
CON> END

                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG     ALL   WEIGHT BEAM  PRICE

 ALBERG      37 MK II   KETCH   37    20,000  12  $36,951
Enter PRICE:
```

- If you include a password in the domain, relation name, or VAX DBMS database to get access to records in the mode you need, you can prevent the display of the password. Enter an asterisk (*) in place of the password in the parentheses following the segment of the dictionary path name with which the password is associated.

  After you enter the READY command, DATATRIEVE prompts you for the password. As you enter the password, DATATRIEVE does not echo the characters.

  You can also include a password prompt in a domain or database definition. Put the asterisk (*) in parentheses after the path name of the record in the domain definition. When you ready an RMS domain, DATATRIEVE prompts you for the password and uses your response to search the entries of the access control list of the record definition to determine which privileges you have to the record definition.

- A database, domain, relation, or VAX DBMS record stays ready until you release it with the FINISH command or until you end your DATATRIEVE session with EXIT or CTRL/Z; the FINISH command releases the collections of records, allows access to other users, and frees computer resources. See the FINISH command in this chapter for more information.

  If you are working with VAX DBMS sources or relational sources and you end your session with EXIT or CTRL/Z, or you finish the last readied source, DATATRIEVE commits any changes you made to the data.

- If you redefine the record associated with a readied RMS domain, the change in the record definition does not take effect until you use the FINISH command to finish the domain and the READY command to ready it again. Simply readying the domain again does not activate the new record definition.

You can make use of this fact if you want to change a record definition or change the type of file organization of a data file. Follow these steps to change the record definition or file organization without redefining the domain; in both cases, you define a new data file and transfer the data with the Restructure statement:

1. Ready the domain as an alias. For example:

```
DTR> READY YACHTS AS OLD_YACHTS
DTR> SHOW READY
Ready domains:
   OLD_YACHTS:  Domain, RMS sequential, protected read
           <CDD$TOP.INVENTORY.YACHTS;1>
No loaded tables.

DTR>
```

2. Change the record definition with the REDEFINE RECORD command, if you wish.

3. Define a new data file for the domain. Do not use the SUPERSEDE option of the DEFINE FILE command. This creates a new version of the file associated with the readied domain but does not interfere with the link between that readied domain and the original version of the data file. For example:

```
DTR> DEFINE FILE FOR YACHTS KEY = TYPE (NODUP)
DTR>
```

4. Ready the domain using a different alias and specify WRITE access. The READY command uses the new record definition, if you created one, and opens the new data file created by the DEFINE FILE command. For example:

```
DTR> READY YACHTS AS NEW_YACHTS WRITE
DTR> SHOW READY
Ready domains:
   NEW_YACHTS:  Domain, RMS indexed, protected write
   OLD_YACHTS:  Domain, RMS sequential, protected read
           <CDD$TOP.INVENTORY.YACHTS;1>
No loaded tables.

DTR>
```

5. Use the Restructure statement to move the data from the original data file to the new one. DATATRIEVE transfers data from fields in the original data file into fields with the same names in the new data file. For example:

```
DTR> NEW_YACHTS = OLD_YACHTS
DTR>
```

Note

You cannot reorganize relational databases using DATATRIEVE. You can, however, use the DATATRIEVE Restructure statement to store records from an RMS data file into an Rdb/VMS or Rdb/ELN database or relation, or a VAX DBMS database. This can be useful if you are converting RMS files to relations.

You can also store records from a relational database or relation into an RMS data file. Similarly, you can store records from one Rdb/VMS or Rdb/ELN database or relation into another Rdb/VMS or Rdb/ELN database or relation.

Because VIDA databases are read-only databases, you can use the Restructure statement to store VIDA data in an RMS data file, but you cannot store RMS data in a VIDA source.

Examples

The following example readies the domain YACHTS for WRITE access:

```
DTR> READY YACHTS WRITE
```

The following example readies the domain PHONES for EXTEND access:

```
DTR> READY PHONES (*) EXTEND
Enter password for PHONES:

DTR>
```

The following example defines a domain with the prompt built into the domain definition. DATATRIEVE does not display the password:

```
DTR> DEFINE DOMAIN PROMPT_YACHTS USING YACHT(*) ON YACHT;
DTR> READY PROMPT_YACHTS AS PYTS
Enter password for YACHT:

DTR>
```

The following example readies the relations EMPLOYEES and SALARY_HISTORY in the Rdb/VMS database PERSONNEL for SNAPSHOT access:

```
DTR> READY PERSONNEL USING EMPLOYEES, SALARY_HISTORY
DTR>
```

The following example readies the SALARY_HISTORY relation in the Rdb/VMS database PERSONNEL for SHARED WRITE access with the CONCURRENCY option:

```
DTR> READY PERSONNEL SHARED WRITE USING SALARY_HISTORY CONCURRENCY
```

The following example readies the VAX DBMS domain VENDORS for the default access mode SHARED READ:

```
DTR> READY VENDORS
DTR>
```

RECONNECT Statement

Removes a record from the set occurrence in which it participates and connects it to the set occurrence specified by the TO list. Before the RECONNECT is performed, DATATRIEVE sets up a currency indicator for each set specified in the TO list.

Format

RECONNECT context-name-1

[TO] [context-name-2.] set-name-1 [,...]

Arguments

context-name-1

Is the name of a valid context variable or the name of a collection with a selected record. The target record must be a member of the sets specified by the TO list.

context-name-2

Is the name of a valid context variable or the name of a collection with a selected record. It must identify a record that participates in the specified set. If the SYSTEM owns the set, you do not need to establish a context for the set. If the set is not owned by the SYSTEM and the context name is not present, DATATRIEVE uses the most recent single record context of a domain with a record type that participates in the specified set type.

set-name

Is the name of a set type.

Example

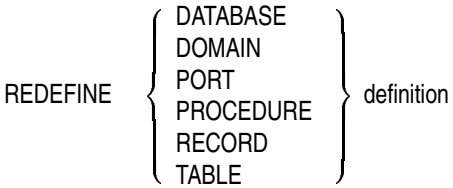
The following example uses nested FOR loops to create the necessary contexts. The procedure uses prompting value expressions to get information from the user.

```
DTR> DEFINE PROCEDURE NEW_MANAGER
DFN> FOR G IN GROUPS WITH GROUP_NAME = *."the name of the group"
DFN> FOR E IN EMPLOYEES WITH EMP_ID =
DFN> *."the id of the new manager"
DFN> RECONNECT G TO E.MANAGES
DFN> END_PROCEDURE
DTR>
```

REDEFINE Command

Creates a new version of an object.

Format



Argument

definition

Is the path name of the definition and any other keywords, path names, or arguments that an object takes. (For an explanation of these arguments, see the DEFINE command for each object. For example, to understand REDEFINE DOMAIN, see the DEFINE DOMAIN command in this chapter.)

Restrictions

- To redefine a DATATRIEVE definition in the DMU format dictionary, you must have P (PASS_THRU) and X (EXTEND) access privileges to the parent directory of the definition and S (SEE), P (PASS_THRU), and U (UPDATE) privileges to the highest existing version.
- To redefine a DATATRIEVE definition in the CDO format dictionary, you must have S (SHOW) and U (CHANGE) access privileges to the directory and S (SHOW) and U (CHANGE + DEFINE) privileges to the highest existing version.
- You cannot specify a relative version number with the REDEFINE command.
- You cannot specify a version number with the REDEFINE command if an object of the same name and version number already exists in the data dictionary
- You cannot redefine a dictionary.
- When you use the REDEFINE command, the definition you are redefining must have access privileges that allow creation of later versions. Typically, you need not worry about these privileges. The data dictionary is usually set up by the system manager to include the ACL access privileges you need. See

the chapter on ACL in the *VAX DATATRIEVE User's Guide* for a description of privileges necessary to redefine definitions.

Results

- The REDEFINE command creates a new version of the object in the data dictionary. The previous version remains in the data dictionary.
- When you specify an explicit version number as part of the REDEFINE command, DATATRIEVE creates the object with that version number.
- When you issue a REDEFINE command for an object that does not exist, DATATRIEVE creates a version 1 of that object after issuing a warning.
- When you specify a version number as part of the REDEFINE command and the object already exists in the data dictionary with that version number, DATATRIEVE issues an error message.
- The new object always has the ACL and the history list from the previous highest version. If a previous version did not exist, DATATRIEVE gives the new object the default ACL.

Usage Notes

- The REDEFINE command is less useful than the EDIT command if you are revising existing data dictionary objects, because DATATRIEVE does not place the text of the earlier version of the object in the buffer as it does with the EDIT command.
- You can use the REDEFINE command everywhere you can use the DEFINE command, except for DEFINE DICTIONARY and DEFINE FILE.

Examples

The following example redefines the domain YACHTS using the record definition YACHT and storing the data in the file DB2:[SMYTHE]YACHT.DAT:

```
DTR> SHOW DOMAINS
Domains:
     * YACHTS;1

DTR> REDEFINE DOMAIN YACHTS
DFN> USING YACHT ON DB2:[SMYTHE]YACHT.DAT;

DTR> SHOW DOMAINS
Domains:
     * YACHTS;2     * YACHTS;1
```

The following example redefines the domain YACHTS and specifies 4 as the explicit version number:

```
DTR> SHOW DOMAINS
Domains:
     * YACHTS;2     * YACHTS;1

DTR> REDEFINE DOMAIN YACHTS;4
DFN> USING YACHT ON DB2:[SMYTHE]YACHT.DAT;

DTR> SHOW DOMAINS
Domains:
     * YACHTS;4   * YACHTS;2   * YACHTS;1
```

REDEFINES Clause

Provides an alternate way to define a field.

Format

level-no field-name-1 REDEFINES field-name-2

Arguments

level-no
Is the level number of field-name-1. Although not a part of the REDEFINES clause, the level number is shown in the format to clarify its position relative to the clause.

field-name-1
Is the name of the REDEFINES field. You use this name when you want to refer to this field. Although not a part of the REDEFINES clause, the field name is shown in the format to clarify its function and its position relative to the clause.

field-name-2
Is the name of the field being redefined.

Restrictions

- The field to be redefined (field-name-2) must appear in the record definition before its REDEFINES field (field-name-1). Both fields must have the same level number.
- The definition of field-name-2 cannot contain a REDEFINES clause. However, it can be subordinate to a group field with a REDEFINES clause.
- Neither field-name-1 nor field-name-2 can be defined with or contain a field defined with an OCCURS . . . DEPENDING clause.
- Neither field-name-1 nor field-name-2 can contain a COMPUTED BY clause. (You cannot redefine a COMPUTED BY field.)
- In the definition of field-1, the REDEFINES clause must immediately follow the field name. No other clause can be used between the field name and the keyword REDEFINES.
- The REDEFINES field cannot describe an area larger than the area of field-name-2. However, the area can be smaller than that of field-name-2.
- You cannot use a qualified field name, such as TYPE.BUILDER, as the field to be redefined in a REDEFINES field.

Result

The REDEFINES clause redefines an elementary or group field. The redefinition refers to the same area of the record as the original definition, but it uses the content of the field in a different way.

Usage Note

If you need to refer to parts of a numeric field as well as to the field itself, you can redefine the field as a group field. The subordinate fields of the group fields would contain the parts of the numeric field value that you refer to. Thus, the REDEFINES clause allows you to redefine a numeric field as a group field. (A group field cannot be numeric; a group field is always alphanumeric.)

Example

The following record definition shows a redefinition of the field PART_NUMBER. PART_NUMBER is a numeric field containing 10 digits. Two group fields redefine PART_NUMBER: PART_NUMBERS_PARTS and PART_NUMBER_GROUPS. Each redefinition specifies a group field containing a total of 10 digits (the total number of digits in all subordinate fields):

```
05 PART_NUMBER PIC 9(10)
05 PART_NUMBER_PARTS REDEFINES PART_NUMBER.
   07 PRODUCT_GROUP PIC 99.
   07 PRODUCT_YEAR PIC 99.
   07 ASSEMBLY_CODE PIC 9.
   07 SUB_ASSEMBLY PIC 99.
   07 PART_DETAIL PIC 999.
05 PART_NUMBER_GROUPS REDEFINES PART_NUMBER.
   07 PRODUCT_GROUP_ID PIC 9(4).
   07 PART_DETAIL_ID PIC 9(6).
```

In this example, the field PRODUCT_GROUP refers to the lowest-valued digits of PART_NUMBER; PRODUCT_YEAR refers to the next two lowest-valued digits, and so on.

REDUCE Statement

Retains only the unique field values or combinations of field values in a DATATRIEVE collection, dropping all other values, depending on the reduce key or keys specified.

Format

REDUCE [collection-name] TO reduce-key-1 [,...]

Arguments

collection-name

Is the name of a collection from which you want to retrieve unique occurrences of values.

reduce-key

Is a field whose values form the basis for the reduction. You can also use a value expression as a reduce key, if the value expression refers to at least one field of the records forming the collection.

Use a comma to separate multiple reduce keys.

Restrictions

- You can use the REDUCE statement only on a collection you have already formed with a FIND statement.
- You cannot use the REDUCE statement in a compound statement.
- You must specify at least one reduce key.
- You cannot specify more than 255 reduce keys in an RSE or in a REDUCE statement.

Results

- If you use one reduce key and it is a field name, DATATRIEVE retains in the collection each unique value of the field. All other field values are dropped from the collection.

  DATATRIEVE does not retain the following values in the collection:

  - Duplicate occurrences of each value
  - Values of other fields in the record

For example:

```
DTR> READY YACHTS
DTR> FIND YACHTS
[113 records found]
DTR> REDUCE CURRENT TO BEAM
DTR> PRINT CURRENT

BEAM

 00
 06
 07
 08
 09
 10
 11
 12
 13

DTR>
```

The field name BEAM is a reduce key. DATATRIEVE retains only the unique values for BEAM in the collection.

- If you use a value expression as a reduce key, DATATRIEVE searches the collection for each unique occurrence of the value expression. DATATRIEVE retains the field value that is a component of the value expression. All other field values are dropped from the collection. For example:

```
DTR> FIND YACHTS
[113 records found]
DTR> REDUCE CURRENT TO (BEAM * 2)
DTR> PRINT CURRENT

BEAM

 00
 06
 07
 08
 09
 10
 11
 12
 13

DTR>
```

The value expression (BEAM * 2) is a reduce key. DATATRIEVE retains the unique values for BEAM, not for double the value of BEAM.

- If you use a virtual field as a reduce key you must include the name of the field on which the virtual field depends as an additional reduce key. In the following example, the reduce key is a COMPUTED BY field that is based on another field in the record. You can include such a field as a reduce key only if you also name the dependent field or expression as an additional reduce key.

```
DTR> DECLARE BEAM_PLUS_TWO COMPUTED BY (BEAM + 2).
DTR> PRINT YACHTS REDUCED TO BEAM_PLUS_TWO
"BEAM" is undefined or used out of context.

DTR> PRINT YACHTS REDUCED TO BEAM, BEAM_PLUS_TWO

     BEAM
     PLUS
BEAM TWO

 00     2
 06     8
 07     9
 08    10
 09    11
 10    12
 11    13
 12    14
 13    15

DTR>
```

- If you use two or more reduce keys, DATATRIEVE retains in the collection all the unique combinations of values, based on the reduce keys specified. DATATRIEVE does not retain any other field values in the collection. For example:

```
DTR> FIND FIRST 12 YACHTS
[12 records found]
DTR> REDUCE CURRENT TO BUILDER, RIG
DTR> PRINT CURRENT

MANUFACTURER  RIG

 ALBERG      KETCH
 ALBIN       SLOOP
 AMERICAN    MS
 AMERICAN    SLOOP
 BAYFIELD    SLOOP
 BLOCK I.    SLOOP
 BOMBAY      SLOOP
 BUCCANEER   SLOOP
 CABOT       SLOOP

DTR>
```

BUILDER and RIG are reduce keys. The unique combination of values for the two fields is retained. AMERICAN appears twice because it is the only manufacturer in the collection that makes yachts with more than one type of rig.

Usage Notes

- If you omit the collection name, DATATRIEVE reduces the CURRENT collection to unique field values.
- To reduce record streams to unique field values, use the REDUCED TO clause of the record selection expression (RSE) that creates the record stream. To reduce collections when you first form them, use the REDUCED TO clause of the RSE in the FIND statement. (See the chapter on RSE in the *VAX DATATRIEVE User's Guide*.)

Examples

The following example searches through the YACHTS domain and, for each type of RIG, lists the prices of all boats that cost over $35,000:

```
DTR> DEFINE PROCEDURE RIG_QUERY
DFN> FIND YACHTS
DFN> REDUCE CURRENT TO RIG
DFN> PRINT SKIP, RIG, ALL PRICE OF YACHTS WITH
DFN>      PRICE GT 35000 AND RIG = Y.RIG OF Y IN CURRENT
DFN> END_PROCEDURE
DTR> :RIG_QUERY

 RIG    PRICE

KETCH  $36,951
       $51,228
       $41,350
       $39,500
       $36,950
       $54,970
       $50,000
       $80,500

MS     $35,900

SLOOP  $37,850
       $39,215
       $37,850
       $48,490

DTR>
```

Note the format of this PRINT statement:

```
PRINT print-list, ALL print-list OF rse-1 OF rse-2
```

RSE-1 is: ALL PRICE OF YACHTS WITH PRICE GT 35000 AND RIG = Y.RIG
RSE-2 is: Y IN CURRENT

RSE-2 controls the printing of the first print list (SKIP, RIG) and RSE-1 controls the printing of the second print-list (PRICE). For more information on inner print lists, see the section in this chapter on the PRINT statement.

You can use the REDUCED TO statement to match values of a date field for the month and year.

The following example uses the domain PAYABLES described in the *VAX DATATRIEVE User's Guide*. Records in PAYABLES have the same TYPE field as in YACHTS, a date field (INVOICE_DUE), and a field for wholesale price (WHSLE_PRICE). The example shows how to display the records in PAYABLES where INVOICE_DUE is later than January 1, 1983. The records are separated according to the month they are due. The REDUCED TO statement is used to identify the unique values of month and year for PAYABLES. Then the records are searched for matches to these values.

```
DTR> SHOW MONTHLY_RPT
PROCEDURE MONTHLY_RPT
READY PAYABLES
FIND PAYABLES WITH INVOICE_DUE AFTER "JAN 1, 1983"
REDUCE CURRENT TO FORMAT INVOICE_DUE USING YYNN
FOR A IN CURRENT
   BEGIN
     PRINT SKIP, "Invoices Due for the Month of"|||
       FORMAT A.INVOICE_DUE USING M(9), SKIP
     FOR PAYABLES WITH FORMAT INVOICE_DUE USING YYNN =
       FORMAT A.INVOICE_DUE USING YYNN SORTED BY INVOICE_DUE
     PRINT INVOICE_DUE, TYPE, WHSLE_PRICE
   END
END_PROCEDURE

DTR> :MONTHLY_RPT

Invoices Due for the Month of January

INVOICE                          WHSLE
  DUE      VENDOR   ITEM_TYPE    PRICE

 1/02/83 ALBERG     37 MK II    $28,500
 1/25/83 SALT       19           $4,850
 1/31/83 AMERICAN   26-MS       $15,150

Invoices Due for the Month of February
```

```
 2/12/83 WINDPOWER  IMPULSE     $1,500
   .        .        .            .
   .        .        .            .
   .        .        .            .

Invoices Due for the Month of April

 4/01/83 BAYFIELD   30/32      $13,000
 4/01/83 IRWIN      37 MARK II $29,999
 4/15/83 ALBIN      VEGA       $14,250

DTR>
```

RELEASE Command

Ends your control over one or more collections, forms, tables, or global variables and frees the workspace occupied by them.

Format

```
RELEASE [ ALL           ] [,...]
        | collection-name |
        | form-name       |
        | table-name      |
        | variable-name   |
        [                 ]
```

Arguments

ALL
Causes DATATRIEVE to release all collections, tables, or variables.

form-name
collection-name
table-name
variable-name
Is the name of a form, a collection, a dictionary or domain table, or a variable you want to release. If you specify more than one item, use a comma to separate each from the next.

Restrictions

- You must issue this command at the DATATRIEVE command level (indicated by the DTR> prompt).
- You cannot use the RELEASE command to release a form that was loaded when a domain was readied. The form must have been loaded with the DISPLAY_FORM statement or with the WITH_FORM statement.

Results

- If RELEASE or RELEASE ALL is specified, DATATRIEVE releases all collections, dictionary tables, domain tables, and variables, freeing the workspace they occupied. The effect is very much like the FINISH command.
- DATATRIEVE releases the collection, dictionary table, domain table, or global variable and frees the workspace it occupied.
- The records and domains associated with the collection you want to release are not changed by this command.

- When you release a DECforms form, you must specify the form name, not the form file name. If more than one form has that name, only one is removed: the first in the list displayed by the SHOW FORMS command.
- If you specify more than one item in the command, DATATRIEVE releases the items in left-to-right order. If the command fails before all items are released, DATATRIEVE prints a message indicating which item could not be released. In such a case of failure, DATATRIEVE releases all items in the command preceding the one that failed and does not release the ones that follow it.

Usage Notes

- To learn which collections you have in the workspace and in which order you created them, enter a SHOW COLLECTIONS command.
- To learn which dictionary tables and domain tables you have in your workspace, enter a SHOW READY command.
- To learn which global variables you can release, enter a SHOW VARIABLES command or a SHOW FIELDS command.
- When you have two or more collections in your workspace and you release the CURRENT one, the remaining collection you formed most recently becomes the new CURRENT collection.
- The RELEASE command is implicit in the following cases:
  - A FIND statement that successfully forms a collection releases an existing collection with the same name. If the CURRENT collection has no other name, a new collection formed by a FIND statement releases and replaces the previous CURRENT collection. DATATRIEVE releases the previous collection, even if the new collection contains no records.
  - Ending a session (with EXIT or CTRL/Z) releases all collections, all dictionary tables, all domain tables, and all global variables in your workspace.
  - When you use a FINISH command, DATATRIEVE releases all collections associated with the specified domains.
  - When you declare a global variable that has the same name as one that exists, DATATRIEVE releases the old global variable.
- You cannot assign a value to or retrieve a value from a global variable once you release it. You can redefine the variable with the DECLARE statement, but, if you do, the previous value is lost and the variable is initialized to the default value: zero, the null string, or the default value established by the

DEFAULT VALUE clause or the MISSING VALUE clause in the DECLARE statement that created the variable.

Examples

The following example releases first one of two named collections, then the other:

```
DTR> SHOW COLLECTIONS
Collections:
       BIG-ONES          (CURRENT)
       A

DTR> RELEASE BIG-ONES
DTR> SHOW COLLECTIONS
Collections:
       A                 (CURRENT)

DTR> RELEASE A
DTR> SHOW COLLECTIONS
No established collections.
DTR>
```

The following example releases the dictionary table DEPT-TABLE and the global variables X and Y:

```
DTR> SHOW READY
No ready sources.

Loaded tables:
   DEPT_TBL:  Dictionary table
           <CDD$TOP.WORK.DEPT_TBL>

DTR> SHOW VARIABLES
Global variables
   X                    <Character string>
   Y                    <Number>

DTR> RELEASE DEPT_TBL, X, Y

DTR> SHOW READY; SHOW VARIABLES
No ready sources.
No loaded tables.
No global variables are declared.
DTR>
```

The following example uses the RELEASE ALL command to release the collections LITTLE_ONES and B, and also the global variables T and TERRY:

```
DTR> SHOW COLLECTIONS
Collections:
       B                (CURRENT)
       LITTLEONES

DTR> SHOW VARIABLES
Global variables
   T                    <Date>
   TERRY                <Character string>

DTR> RELEASE ALL
DTR> SHOW COLLECTIONS
No established collections.

DTR> SHOW VARIABLES
No global variables are declared.

DTR>
```

RELEASE SYNONYM Command

Releases the definition of a synonym for a DATATRIEVE keyword.

Format

RELEASE SYNONYM synonym-name-1 [,...]

Argument

synonym-name
Is a synonym already defined for a DATATRIEVE keyword.

Restriction

You must issue this command at the DTR> prompt.

Result

DATATRIEVE releases the synonym or synonyms specified. You can no longer use them in place of DATATRIEVE keywords.

Usage Notes

- To see defined synonyms, enter a SHOW SYNONYMS command.
- To release more than one synonym, separate the synonym names by commas.

Example

The following example defines synonyms for PRINT and READY and then releases the synonym definitions:

```
DTR> DECLARE SYNONYM P FOR PRINT, R FOR READY
DTR> R OWNERS; P FIRST 1 OWNERS

                BOAT
   NAME         NAME         BUILDER     MODEL

SHERM      MILLENNIUM FALCON ALBERG     35

DTR> RELEASE SYNONYM R, P
DTR> R YACHTS
R YACHTS

Expected statement, encountered "R".
DTR> P OWNERS
P OWNERS
```

```
Expected statement, encountered "P".
DTR>
```

REPEAT Statement

Causes DATATRIEVE to execute a simple or compound DATATRIEVE statement a specified number of times.

Format

REPEAT value-expression statement

Arguments

value-expression

Is a value expression indicating the number of times to execute the statement. This argument must evaluate to a positive integer less than or equal to 2,147,483,647.

statement

Is any simple or compound DATATRIEVE statement (except a FIND, SELECT, DROP, or SORT statement).

Restrictions

- Do not use a FIND, SELECT, DROP, or SORT statement in a REPEAT statement.
- You must observe all restrictions on the simple or compound statements you use in a REPEAT statement.
- If the statement in the REPEAT statement is the invocation of a procedure (for example, REPEAT n :procedure-name), the procedure cannot contain a command or a FIND, SELECT, DROP, or SORT statement as its first element.
- If you invoke a procedure in a compound statement in a REPEAT statement (for example, REPEAT n BEGIN :procedure-name; END), that procedure cannot contain a DATATRIEVE command, or a FIND, SELECT, DROP, or SORT statement as its first element.

Results

- DATATRIEVE executes the statement the number of times specified by the value expression. Then DATATRIEVE executes the command or statement following the REPEAT statement.

- If you invoke a procedure in a REPEAT statement (for example, REPEAT n :procedure-name), only the first statement (whether simple or compound) in the procedure is executed the number of times specified in the value expression. Each succeeding statement in the procedure is executed only once.

Usage Notes

- Use the REPEAT statement to repeat a simple or compound statement a fixed number of times.
- You can force an exit from a loop created by a REPEAT statement in the following ways:
  - Type CTRL/Z in response to any prompt within the loop.
  - Use an IF-THEN-ELSE statement with an ABORT statement in the THEN or ELSE clauses to exit from the loop according to the conditions specified in the IF clause. See the sections in this chapter on ABORT and IF-THEN-ELSE for more information.
  - Type CTRL/C at any time during the execution of the statement (but not in response to a prompt).

    Having SET ABORT or SET NO ABORT in effect does not change the DATATRIEVE response to an ABORT statement in a REPEAT loop. If the conditions for the ABORT are met in either case, DATATRIEVE executes no statement following the ABORT statement in the REPEAT loop. When the ABORT occurs in a REPEAT loop, DATATRIEVE returns you to command level (indicated by the DTR> prompt).
- You can nest REPEAT statements. DATATRIEVE executes each inner REPEAT statement the specified number of times each time it loops through the outer REPEAT statement.

Examples

The following example prints TEST REPEAT three times:

```
DTR> REPEAT 3 PRINT "TEST REPEAT"
TEST REPEAT
TEST REPEAT
TEST REPEAT

DTR>
```

The following example aborts a REPEAT statement by responding to a prompt with a CTRL/Z:

```
DTR> READY YACHTS WRITE
DTR> REPEAT 5 STORE YACHTS
Enter MANUFACTURER: HOBIE
Enter MODEL: CAT
Enter RIG: SLOOP
Enter LENGTH-OVER-ALL: 22
Enter DISPLACEMENT:  4000
Enter BEAM: 8
Enter PRICE: 6500
Enter MANUFACTURER: CTRL/Z
Execution terminated by operator
DTR> FIND YACHTS WITH BUILDER = "HOBIE"
[1 record found]
DTR>
```

The following example shows the effect of nesting REPEAT statements in procedures. The procedure NUM1 contains two PRINT statements. The procedure NUM2 contains two REPEAT statements, one nested in the other. The inner REPEAT statement causes DATATRIEVE to execute the first PRINT statement in NUM1 twice each time DATATRIEVE loops through the outer REPEAT statement:

```
DTR> SET NO PROMPT
DTR> SHOW NUM1
PRINT SKIP, "ONE, TWO, THREE"
PRINT "ONE, TWO, THREE, FOUR, FIVE"

DTR> :NUM1

ONE, TWO, THREE

ONE, TWO, THREE, FOUR, FIVE

DTR> SHOW NUM2
REPEAT 2
     BEGIN
        REPEAT 2 :NUM1
     END
:NUM1

DTR> :NUM2

ONE, TWO, THREE

ONE, TWO, THREE
ONE, TWO, THREE, FOUR, FIVE

ONE, TWO, THREE

ONE, TWO, THREE
ONE, TWO, THREE, FOUR, FIVE
```

```
ONE, TWO, THREE
ONE, TWO, THREE, FOUR, FIVE

DTR>
```

REPORT Statement (Report Writer)

Invokes the Report Writer and is the first entry in a report specification. In the REPORT statement you can specify the following:

- The data you want to report
- The output device for the report
- The output format for the report

The other statements in the report specification are AT BOTTOM, AT TOP, DECLARE_ATT, END_REPORT, PRINT, and SET. These statements are discussed in separate sections of this chapter.

Format

REPORT [rse] [ON { file-spec / *.prompt }] [FORMAT format-spec]

Arguments

rse

Specifies the data for your report. To create a record stream for your report, enter the appropriate RSE in the REPORT statement. You can make reports using data from:

- Readied domains
- Collections
- Lists

When you omit the RSE, the Report Writer uses the data in your current collection for the report. If there is no current collection, DATATRIEVE displays the following error message:

```
A current collection has not been established.
```

file-spec

Is the file to which you want to write the report. A complete file specification has the following format:

```
node-spec::device:[directory]file-name.type;version
```

The minimum file specification consists of a period (.); the specification of such a file stored in your default VMS directory ends with ".;n", where n is the version number and both the file name and the type are null strings.

If you omit a field in the file specification, DATATRIEVE uses the defaults as listed in Table 4–32.

Table 4–32 Output File Specification Defaults

Field	Default
node-spec::	Your local node
device:	Your default device
[directory]	Your default directory
file-name	Null string
.type	.LIS
;version	Highest version number

When you omit the ON clause in a REPORT statement, DATATRIEVE displays the report on your terminal.

format_spec
Is the format in which you want the report presented. The following formats are supported:

Format	Explanation	Output type
DDIF[1]	The CDA format for page-based documents. DDIF allows files produced by the Report Writer to be processed directly, for example, by DECwrite, DECpresent, or conversion to other formats.	page
PS	PostScript™, produced by conversion from DDIF to obtain high quality printout.	page
null	The default ASCII format produced by the Report Writer. Format encoded as ASCII characters.	page
TEXT	Format encoded as ASCII characters, with ANSI escape sequences that produce certain attributes on terminals and printers.	page
DTIF[1]	The CDA format for tables. DTIF allows files produced by the Report Writer to be processed directly, for example, by DECdecision, DECchart, or conversion to other formats.	table

[1]DDIF and DTIF output can be converted to a multitude of different formats using the CDA converter. See the CDA documentation for further details.

***.prompt**
Is a prompting value expression that allows you to specify a file specification when DATATRIEVE processes the report specification.

Restrictions

- For spreadsheet outputs (DTIF format), relationships between columns are lost: for example when COMPUTED BY fields are printed, the result of the computation is passed to the output file, rather than the formula used to calculate it.

Usage Notes

- You can report on data only in domains readied for READ, WRITE, or MODIFY access. Because you cannot establish collections or record streams in domains readied for EXTEND access, you cannot report on data in domains readied for EXTEND access.
- The data you report must be contained in the current collection or in the record stream established by the RSE in the REPORT statement.
- The default format is ASCII text. This takes two forms:
  - if the FORMAT clause is not specified, the RW output is in ASCII format, and the ATT clauses are completely ignored. No attributes are applied. (See the DECLARE_ATT Statement.)
  - if the FORMAT TEXT clause is explicitly specified, attributes are applied wherever possible (BOLD, UNDERLINE, and REVERSE attributes) through the use of ANSI escape sequences.
- All formats, except for the ASCII formats, require the ON clause in the REPORT statement. In these cases, if the REPORT statement containing the ON clause is embedded in a ON compound statement, the output will be directed to the file specified in the ON clause, while the file specified in the ON compound statement will contain no output.
- If you are using an ASCII format, and want the report written to more than two output media, you can include multiple ON statements before you invoke the Report Writer, for example:

```
DTR> READY PAYABLES
DTR> ON [MORRISON]PAY1.LIS
CON> ON [MORRISON.RPTS]PAY1.LIS
CON> REPORT PAYABLES ON TT:
RW>  PRINT INVOICE_DUE
RW>  END_REPORT
```

- Every execution of a REPORT statement, for example within a loop, creates a new file.
- When producing DDIF or DTIF reports, note that these may be converted to a range of other formats by means of the CDA Converter Library. For further information, refer to the CDA documentation.

Restructure Statement

Transfers data from the fields of records in a record stream to fields with the corresponding names in a domain.

Format

```
domain-name = rse
```

Arguments

domain-name
Is the given name or alias of a domain readied for EXTEND or WRITE access. That domain receives data from the records identified by the RSE.

rse
Is an RSE that identifies records containing one or more fields corresponding to fields of the same name in the domain that receives the data.

Restrictions

- You must ready the receiving domain for WRITE or EXTEND access.

  The domains referred to in the RSE must be readied for READ, MODIFY, or WRITE access.

- To update a receiving domain defined in the DMU format dictionary, you must have P (PASS_THRU), S (SEE), and W (DTR_WRITE) or E (DTR_EXTEND /EXECUTE) access to the domain definition and P (PASS_THRU), S (SEE), and E (DTR_EXTEND/EXECUTE) access to the record definition.

  To form the record stream with the RSE, you must have P (PASS_THRU), S (SEE), and R (DTR_READ) access to the definitions of the domains from which the record stream derives and P (PASS_THRU), S (SEE), and E (DTR_ EXTEND/EXECUTE) access to the record definition associated with the domains from which the record stream derives.

- To update a receiving domain defined in the CDO format dictionary, you must have S (SHOW) access to the dictionary containing the definition, and S (SHOW) and W (WRITE) access to the domain and record definitions.

  To form the record stream with the RSE, you must have S (SHOW), and R (READ) access to the definitions of the domains from which the record stream derives and S (SHOW), and E (EXTEND) access to the record definition associated with the domains from which the record stream derives.

- The receiving domain and the source in the RSE must have the name of at least one field in common.

- The source in the RSE cannot have two elementary field names with the same name associated with different group fields. The same rule holds for the receiving domain. DATATRIEVE does not always associate the elementary fields with the appropriate group fields. In these cases, a STORE domain-name USING statement is the best approach. See the section in this chapter on the STORE statement.
- If the record is hierarchical, use the MATCH statement with a STORE domain-name USING statement to restructure the record. See the section in this chapter on the MATCH statement.

Results

- DATATRIEVE copies values from the record stream to new records in the data file of the receiving domain. DATATRIEVE also makes any data type conversions that might be necessary.
- If the record stream produced by the RSE contains no records, then no records are stored in the receiving domain.

Usage Notes

- The Restructure statement provides a simple way of copying data from one domain to another. This statement is especially useful when you modify a record definition or want to change the file organization of a domain's data file. After entering the appropriate READY and DEFINE commands, you use the Restructure statement to transfer data from the old domain to the new.
- The Restructure statement matches fields from the new domain with fields of the same name in the old one and then transfers data from the fields in the old data file to those in the new one.
- If you define a new record, you can add new fields, omit old ones, change the data type of fields, and change the length of fields. DATATRIEVE does the data conversions automatically. The lengths and data types of the corresponding fields do not have to be the same. If they are different, you need to anticipate any problems (such as truncations or overflows) that might arise. Only the field names or query names of the corresponding fields must be the same.
- You can also use the Restructure statement when you want to change the type of file organization of a domain's data file. You can change from a sequential file to an indexed file without defining a new domain. This process uses the AS alias clause of the READY command and is explained in the section on the READY command.

Note

You cannot reorganize VAX DBMS or relational databases using DATATRIEVE. You can, however, use the DATATRIEVE Restructure statement to store records from an RMS data file in a relational database or relation, and in a VAX DBMS domain or record. You can also use the Restructure statement to store records from a relational database or relation in an RMS data file, and to store records from a VAX DBMS domain or record in an RMS data file.

Examples

The following example defines a new domain using FAMILY_REC and stores only those records of families that have no children younger than 15:

```
DTR> DEFINE DOMAIN NEWFAMS USING FAMILY_REC ON FAMS;
DTR> DEFINE FILE FOR NEWFAMS MAX, KEY = MOTHER (DUP)
DTR> NEWFAMS = FAMILIES WITH NOT ANY KIDS WITH AGE LE 15
DTR> FIND NEWFAMS
[8 records found]
DTR> FIND FAMILIES
[16 records found]
DTR>
```

The following example defines a new domain called YACHTS_PRICE_LIST, which contains only the fields TYPE and PRICE from the old YACHT record definition. The number of records transferred is checked with the FIND statement and the accuracy of the transfer is checked with the CROSS statement. The example displays some of the records from the new domain to check the presence of the MISSING VALUE edit string.

```
DTR> DEFINE DOMAIN YACHTS_PRICE_LIST USING YPL_REC ON YPL.DAT;
DTR> DEFINE RECORD YPL_REC USING
DFN> 01 BOAT.
DFN>    03 TYPE.
DFN>       05 BUILDER PIC X(10).
DFN>       05 MODEL PIC X(8).
DFN>    03 PRICE PIC 9(5) MISSING VALUE IS 0
DFN>       EDIT_STRING $$$,$$$?"NOT LISTED".
DFN> ;
[Record is 23 bytes long.]
DTR> DEFINE FILE FOR YACHTS_PRICE_LIST KEY = TYPE
DTR> READY YACHTS_PRICE_LIST AS YPL WRITE
DTR> READY YACHTS
DTR> SHOW READY
Ready domains:
   YACHTS:  Domain, RMS indexed, protected read
            <CDD$TOP.DTR32.WAJ.YACHTS;1>
   YPL:  Domain, RMS indexed, protected write
            <CDD$TOP.DTR32.WAJ.YACHTS_PRICE_LIST;1>
No loaded tables.

DTR> YPL = YACHTS WITH LOA GT 35
DTR> FIND YACHTS WITH LOA GT 35
[23 records found]
DTR> FIND YPL
[23 records found]
DTR> FIND A IN YPL CROSS B IN YACHTS OVER
[Looking for field name]
CON> TYPE WITH A.PRICE NE B.PRICE
[0 records found]
DTR> FIND YPL WITH PRICE MISSING
[12 records found]
DTR> PRINT FIRST 3 CURRENT

 BUILDER    MODEL     PRICE

BLOCK I.   40       NOT LISTED
CABOT      36       NOT LISTED
DOWN EAST  38       NOT LISTED

DTR>
```

The following example uses the Restructure statement to transfer data from an indexed file to a sequential file:

```
DTR> SET NO PROMPT
DTR> READY YACHTS AS OLD
DTR> DEFINE FILE FOR YACHTS
DTR> READY YACHTS AS NEW WRITE
DTR> NEW = OLD
DTR> FIND NEW
[113 records found]
DTR>
```

If the field has a DEFAULT VALUE clause, DATATRIEVE initializes the field with the default value. If the field has a MISSING VALUE clause and no DEFAULT VALUE clause, DATATRIEVE initializes the field with the missing value. If the field has neither a DEFAULT VALUE clause nor a MISSING VALUE clause, DATATRIEVE initializes a numeric field as 0 and an alphabetic or alphanumeric field as spaces.

ROLLBACK Statement

Undoes all the changes you made to the database since the last COMMIT or ROLLBACK statement, or since your first READY if you have not done a ROLLBACK or a COMMIT. The ROLLBACK statement performs a VAX DBMS or relational rollback and releases all collections associated with VAX DBMS domains and records and with relational domains and relations. ROLLBACK then readies VAX DBMS realms again or finishes and starts a new relational source transaction.

The ROLLBACK statement also acts as an ABORT in procedures, nested statements, and command files.

The ROLLBACK statement affects all readied parts of VAX DBMS and relational databases, whether or not you made any changes to the data they contain. RMS domains are not affected by the ROLLBACK statement.

Format

ROLLBACK

Usage Notes

- If you do a FINISH that did not cause a COMMIT, the ROLLBACK undoes changes made since the previous COMMIT, ROLLBACK, or the first READY.
- If you have done a final FINISH on a database, ROLLBACK undoes changes to that FINISH.

Example

The following example for the VAX DBMS database PARTS_DB connects an employee named Hill to a part LA36 in the RESPONSIBLE_FOR set. The ROLLBACK statement undoes the change:

```
DTR> FIND E IN EMPLOYEES WITH EMP_LAST_NAME = "HILL"
DTR> SELECT 1
DTR> FOR P IN PART WITH PART_DESC = "LA36"
CON>     CONNECT P TO E.RESPONSIBLE_FOR
DTR>  ROLLBACK
DTR>
```

SCALE Clause

Establishes explicitly the scale factor to be applied to the value stored in the field.

Format

SCALE [IS] [-]integer

Arguments

– (minus sign)
Is an optional minus sign to indicate a negative scale factor.

integer
Indicates the number of decimal places the implied decimal point is from the right or left end of the value stored in the field.

Restriction

Do not use a SCALE clause in the same definition with a V or one or more Ps in the picture string.

Results

Scale factors help determine the position of the implied decimal point associated with the value stored in the field. A positive scale determines how many places the implied decimal point is to the right of the digits stored in the field. If the scale factor is 2 and the field contains the digits 12, the value of the field is 1200.

A negative scale factor determines how many places the implied decimal point is to the left of the digits stored in the field. If the scale factor is –2 and the field contains the digits 12, the value of the field is .12.

Examples

The following example uses a positive scale factor to store a large number in a small field:

```
03 BARRELS_PER_DAY WORD
   SCALE 6
   EDIT_STRING Z(5)" Million".
```

The following example uses a negative scale factor to store a minute number in a small field:

```
03 PROJECT.
   05 PROBABILITY_OF_FINISHING.
      07 EVER DEFAULT VALUE IS 1
         PIC 99 SCALE -2.
      07 ON_TIME 99 SCALE -6
         MISSING VALUE 0
         EDIT_STRING 0.9(6)?"Better late than never".
```

SELECT Statement

Establishes a target record in a collection.

Format

```
SELECT [ FIRST | NEXT | PRIOR | LAST | value-expression | NONE ] [collection-name] [WITH boolean]
```

Arguments

FIRST

Selects the first record in the target collection.

NEXT

Selects the next record in the target collection. (See the second and third items in the Results section.) When you omit a position specification, NEXT is the default.

PRIOR

Selects the previous record in the target collection. (See the fifth and sixth items in the Results section.)

LAST

Selects the last record in the target collection.

value-expression

Evaluates to a positive number. DATATRIEVE uses the integer part of the number to select the record with that position number in the collection.

NONE

Releases the selected record so that no selected record exists for the current collection. If the collection was formed from a file-structured database, SELECT NONE also releases the RMS lock on the selected record.

collection-name

Is the name of the target collection containing the record to be selected. If you omit the collection name, the target collection is the current collection.

WITH boolean
Causes DATATRIEVE to select the record that satisfies both the Boolean expression and the collection position references (FIRST, LAST, NEXT, PRIOR, and value-expression).

Restrictions

- You must establish the target collection with a FIND statement before entering this statement.
- The target collection cannot be empty.
- With the SELECT statement, you can move the collection cursor (which points to the selected record in a collection) to the position of a record that has been dropped, but you cannot retrieve any data from the record that occupied that position before you dropped it. You must form a new collection or record stream containing that record to retrieve its data. See the section in this chapter on the DROP command.
- Do not use a SELECT statement in a compound statement.
- Do not use a SELECT statement in FOR, REPEAT, or WHILE statements.
- If you specify a value expression, the expression must evaluate to a positive number between 1 and 1,301,265, which is the limit on collection size for records in RMS data files. The integer part of the number must not exceed the number of records in the collection. If the value expression exceeds the number of records in the collection, DATATRIEVE displays the following message:

  ```
  Record number out of range for collection.
  ```

- You cannot use a SELECT statement with a Boolean expression on a remote collection. For example:

  ```
  DTR>  READY REMOTE-YACHTS
  DTR>  FIND REMOTE-YACHTS
  [113 records found]
  DTR>  SELECT FIRST WITH PRICE = 0

  SELECT with a boolean is not supported for remote collections.

  DTR>
  ```

Results

- The SELECT statement establishes a selected record in the target collection and, thus, establishes a single record context for one record in that collection.

  When you enter this statement, the record DATATRIEVE selects depends on the arguments you supply, the content of the target collection, and the existence and position of a previous selected record in the collection.

- If you have established a selected record for a collection and that record is not the last one, entering SELECT NEXT causes the next record in the collection to become the selected record.
- If you have not established a selected record in a collection and you enter SELECT NEXT, the first record in the collection becomes the selected record.
- If the collection cursor points to the last record in the collection and you specify SELECT NEXT, DATATRIEVE displays an error message, and the last record in the collection is still the selected record.
- If you have not established a selected record in a collection and you enter SELECT PRIOR, DATATRIEVE displays an error message and does not select any record in the collection.
- If the first record in the collection is the selected record and you specify SELECT PRIOR, DATATRIEVE displays an error message, and the first record in the collection is still the selected record.
- If the record identified by the SELECT statement has been dropped from the collection with a DROP statement, DATATRIEVE displays a message, and moves the collection cursor (which points to the selected record in a collection) to the position of the dropped record.
- If the target collection is not the CURRENT collection, selecting a record from it does not make it the CURRENT collection.
- When you omit the argument that determines the position of the selected record, DATATRIEVE responds as though you had entered SELECT NEXT. (See the second, third, and fourth items in the Results section.)
  - If no selected record exists for the target collection, DATATRIEVE selects the first record.
  - If a selected record exists for the target collection and that selected record is not the last one in the collection, DATATRIEVE selects the next record in the collection.

  - If the last record in the target collection is the selected record, DATATRIEVE displays an error message, and the last record in the collection is still the selected record.

- When you specify a value expression in a SELECT statement, remember that the positive integer to which the expression evaluates is the position number of the selected record in the collection.

  The integer does not designate the number of records that are selected. For example, if you type SELECT 5, you select the fifth record in the target collection, not five records from the collection. You can never select more than one record at a time per collection.

- If you select a record from the CURRENT collection and then select a record from another collection, the CURRENT collection and its selected record remain unchanged.

- If you include a Boolean expression in the SELECT statement, DATATRIEVE forms a temporary record stream of records that match the conditions specified by the Boolean expression. DATATRIEVE then uses the position references FIRST, NEXT, LAST, and PRIOR to select a record from that temporary record stream.

Usage Notes

- Use the SELECT statement to establish a single record context for a record in a collection. Having a selected record allows you to retrieve and compare values in the fields of a selected record without specifying a target record stream.

  When referring to fields of records in a single record context, you can frequently use the field names without field name qualifiers, almost as though they were variables.

- When you use a field name by itself in a value expression, its value is retrieved from the nearest selected record with a field of that name. That field name is said to resolve to the nearest single record context for that name.

  DATATRIEVE establishes the nearness of selected records based on the order in which you created the collections with the FIND statement. For example, suppose that you have three established collections, created in the order A, then B, then C, and that each collection has a selected record. If you use a field name by itself, DATATRIEVE first tries to retrieve the field name value from C's selected record, then B's, then A's.

- Use a SELECT statement to establish a target record for the ERASE and MODIFY statements.

SELECT Statement

- Use a SELECT statement to establish a target record for PRINT and LIST statements in which you include only the print list or in which you use only the statement name. (See the third and fourth examples.)
- To show the position of the selected record in the target collection, use the SHOW command:

  ```
  SHOW collection-name
  ```

  This command prints the name of the collection, the name of the domain or domains from which the collection is derived, the number of records in the collection, the names of any fields used to establish the sort order of the collection, and the position number of the selected record in the target collection. It also tells you if the selected record has been dropped from the collection by a previous DROP statement.

  To show the name and attributes of the CURRENT collection, use the SHOW CURRENT command. See the section in this chapter on the SHOW command.
- If you use the SELECT statement to establish a selected record for the CURRENT collection, you need type only PRINT and press the RETURN key to display that selected record.

  If the CURRENT collection has no selected record, DATATRIEVE displays the selected record from the most recently established named collection that has a selected record.

  If you enter a PRINT statement without a print list and no existing collection has a selected record, DATATRIEVE displays a message and displays the entire CURRENT collection.
- To display all fields of the selected record of a named collection that is not the CURRENT collection, you must provide a properly qualified top-level field name in the print list of a PRINT statement. Use the collection name as the qualifier.

  For example, a collection of YACHTS called BIGGIES is not the CURRENT collection, and the CURRENT collection has a selected record. To display the selected record in BIGGIES, type PRINT BIGGIES.BOAT and press RETURN.

  If you created the collections in question from different domains, you do not have to qualify the top-level field names if they are not the same.

  If the top level-field names of the different domains are the same, you can use the domain name to qualify the top-level field names.

- You can refer to the fields of the selected record as though they were variables. DATATRIEVE resolves an unqualified field name to the most recently formed collection with a selected record containing a field with that name.

  To refer to a field name beyond the most recently formed collection with a selected record containing that same name, you must provide a suitable qualifier to establish the appropriate context. Use the name of the collection containing the target record as qualifier.

- To distinguish between two or more selected records referred to in one complex DATATRIEVE statement, use context variables. Context variables can be particularly useful if you derived the collections from the same domain and the field names of the collections in question are identical.

- If you want to perform the same set of statements on each record in the CURRENT collection, do not use a SELECT statement in a BEGIN-END block inside a REPEAT statement.

  The following example illustrates the proper method of looping through the CURRENT collection:

```
FOR CURRENT
   BEGIN
      PRINT
      MODIFY ...
      PRINT
   END
```

- See the *VAX DATATRIEVE User's Guide* for a discussion of name recognition and single record context in DATATRIEVE.

Examples

The following example selects the last record in the CURRENT collection:

```
DTR>  SELECT LAST
DTR>
```

The following example selects the fifth record in the collection BIG_ONES:

```
DTR>  SELECT 5 BIG_ONES
DTR>
```

The following example selects the last 30-foot boat in the YACHTS inventory from the CURRENT collection:

```
DTR>  FIND YACHTS
[113 records found]
DTR>  SELECT LAST WITH LOA = 30
DTR>  PRINT

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 SOLNA CORP  SCAMPI     SLOOP   30     6,600  10

DTR>
```

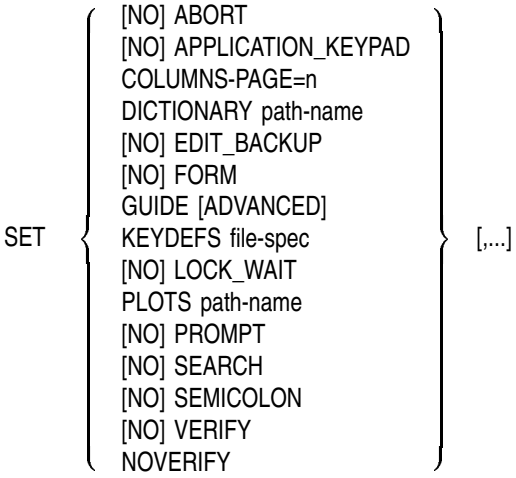
The following example selects a record from the CURRENT collection, modifies a field value, and releases the selected record and, in this case, the RMS lock on the record:

```
DTR>  READY YACHTS SHARED MODIFY
DTR>  FIND YACHTS WITH BUILDER = "SOLNA CORP"
[1 record found]
DTR>  SELECT
DTR>  PRINT

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 SOLNA CORP  SCAMPI     SLOOP   30     6,600  10

DTR>  MODIFY PRICE
Enter PRICE:  50000
DTR>  PRINT

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 SOLNA CORP  SCAMPI     SLOOP   30     6,600  10  $50,000

DTR>  SELECT NONE
DTR>
```

SET Command

- Controls the DATATRIEVE response to ABORT statements.
- Sets the keypad mode (application or numeric) within DATATRIEVE.
- Sets the maximum number of columns per page for DATATRIEVE output.
- Establishes your default directory in the CDD/Repository data dictionary.
- Determines whether DATATRIEVE creates a DTREDIT.DTR backup file when you edit dictionary objects.
- Determines whether DATATRIEVE uses its forms interface to control the video display of your terminal.
- Starts DATATRIEVE Guide Mode.
- Permits or inhibits automatic syntax prompting for continued commands and statements.
- Permits or inhibits creation of an implicit inner print list in a PRINT statement and an implicit ANY in a Boolean expression.
- Determines whether a semicolon is required at the end of each command or statement.
- Controls whether commands and statements in command files are displayed when the command file is invoked

Format



To change settings for HELP:

```
SET  { [NO] HELP_PROMPT   }
     { [NO] HELP_WINDOW   }  [,...]
     { HELP_LINES n TO m  }
```

Arguments

ABORT

Causes DATATRIEVE to abort the remainder of a procedure or command file when DATATRIEVE executes an ABORT statement, when you enter a CTRL/Z to a prompt, or when a syntax or logical error occurs during the execution of a command or statement.

(The exception to the logical error condition is the DELETE command. If you list two or more objects as arguments for the DELETE command, DATATRIEVE does not abort if it fails to find an object of a specified name. Instead, DATATRIEVE continues to delete the remaining objects in the list.)

NO ABORT

Causes DATATRIEVE, when processing a procedure or a command file, to abort only the one statement containing an ABORT statement. SET NO ABORT also causes DATATRIEVE to take the same action when you respond with a CTRL/Z to a prompt in a procedure or command file. DATATRIEVE then executes the next command or statement in the procedure or command file. SET NO ABORT is in effect when you start a DATATRIEVE session.

APPLICATION_KEYPAD

Sets the keypad mode as application keypad within DATATRIEVE.

The default keypad mode for DATATRIEVE is numeric keypad mode.

(You can also set the keypad mode using the function FN$KEYPAD_MODE.)

This command is not available in a DECwindows environment.

NO APPLICATION_KEYPAD

Sets the keypad mode as numeric keypad within DATATRIEVE.

The default keypad mode for DATATRIEVE is numeric keypad mode.

(You can also set the keypad mode using the function FN$KEYPAD_MODE.)

This command is not available in a DECwindows environment.

COLUMNS_PAGE = n
Establishes the number of columns per page for DATATRIEVE output and the default page width for the Report Writer. When you start your session, the default COLUMNS_PAGE setting is 80.

(To set the terminal's width, use the DATATRIEVE function FN$WIDTH. See the chapter on functions for more information about FN$WIDTH.)

DICTIONARY path-name
Causes DATATRIEVE to set your default directory node of the CDD/Repository data dictionary to the node specified by the dictionary path name. When you start your DATATRIEVE session, your default CDD/Repository directory is either CDD$TOP or the directory designated by the logical name CDD$DEFAULT.

SET DICTIONARY accepts both DMU and CDO style path names.

EDIT_BACKUP
Causes DATATRIEVE to save the original definition in the CDD/Repository data dictionary when you use the EDIT command to edit a dictionary object. When you start your DATATRIEVE session, SET EDIT_BACKUP is in effect. Use the SHOW EDIT command to see whether or not SET EDIT_BACKUP is currently in effect.

NO EDIT_BACKUP
Causes DATATRIEVE to delete the highest version of the object in the CDD/Repository data dictionary, or the version you specify, and replace it with the definition in the edit buffer when you use the EDIT command to edit a dictionary object.

FORM
Determines whether DATATRIEVE uses its forms interface when you use the PRINT, MODIFY, and STORE statements. For terminals supported by the forms product you are using, SET FORM causes DATATRIEVE to display and use forms when you enter the PRINT, MODIFY, or STORE statement. If SET NO FORM was in effect when you accessed the domain with the READY command, you can use the associated form by entering a SET FORM command.

To use forms with a domain, you can use one of the following methods:

- You can define a domain with a FORM clause in it to specify the name of the form and the name of the form file in which the form definition is stored.
- You can associate a form with a domain using either the DISPLAY_FORM statement (for FMS and TDMS forms), or the WITH_FORM statement (for DECforms forms).

When you start your DATATRIEVE session, SET FORM is in effect.

NO FORM
Prevents DATATRIEVE from using its forms interface. If SET NO FORM is in effect when you use the PRINT, MODIFY, or STORE statements, DATATRIEVE does not use the forms interface.

GUIDE
Starts Guide Mode, the tutorial mode of DATATRIEVE. Refer to the *VAX DATATRIEVE User's Guide* for a description of Guide Mode.

CAITIFFS file-spec
Lets you define multiple keypad keys from a file containing DCL DEFINE/KEY commands (See the VMS documentation on the DIGITAL Command Language for more information on the DEFINE/KEY command.) By using SET CAITIFFS, you do not have to make multiple calls to the FN$DEFINE_KEY function.

The file specification is the full DCL file specification for the file containing the DCL DEFINE/KEY commands.

This command is not available in a DECwindows environment.

LOCK_WAIT
When two applications try to access the same file, RMS may lock a record that DATATRIEVE needs to access. DATATRIEVE tries for 12 seconds to access a locked record. SET LOCK_WAIT causes DATATRIEVE to turn control over to RMS after this period. RMS then waits for the locked record until it is released, or until RMS sends you a deadlock message. The default is SET NO LOCK_ WAIT.

NO LOCK_WAIT
Instructs DATATRIEVE not to try to access a locked record after 12 seconds. At the end of this period, you receive an RMS message informing you that the record is locked. When you begin your DATATRIEVE session, SET NO LOCK_WAIT is in effect.

PLOTS
Establishes the default CDD/Repository DMU data dictionary directory for your DATATRIEVE plot definitions. For more information, see the *VAX DATATRIEVE User's Guide*.

PROMPT
Causes DATATRIEVE to prompt for elements needed to complete the syntax of the current command or statement. When you press the RETURN key before completing a command or statement, DATATRIEVE prompts you for the next syntactic element of that statement or command. The prompt takes the following form:

```
[Looking for element]
```

At the start of a DATATRIEVE session, SET PROMPT is in effect.

NO PROMPT
Prevents DATATRIEVE from prompting for elements needed to complete the syntax of the current command or statement.

SEARCH
Causes the DATATRIEVE Context Searcher to create implicit inner print lists in PRINT statements and implicit ANYs in Boolean expressions. When you work with VAX DBMS domains, the SET SEARCH command causes the DATATRIEVE Context Searcher to walk sets to look for a context to resolve references to field names.

NO SEARCH
Prevents the DATATRIEVE Context Searcher from creating implicit inner print lists in PRINT statements and implicit ANYs in Boolean expressions. At the start of a DATATRIEVE session, SET NO SEARCH is in effect.

SEMICOLON
Causes DATATRIEVE to require a semicolon at the end of commands or statements.

NO SEMICOLON
Causes DATATRIEVE to make semicolons at the end of commands or statements optional. At the start of a DATATRIEVE session, SET NO SEMICOLON is in effect.

VERIFY
Causes lines from command files to be displayed when a command file is invoked.

NO VERIFY
Suppresses the display of lines from command files when a command file is invoked. At the start of your DATATRIEVE session, the current setting for SET VERIFY/NOVERIFY at the DCL level is in effect.

HELP_PROMPT
Causes DATATRIEVE to prompt for the topic or subtopic when help text is displayed.

NO HELP_PROMPT
Suppresses the prompting for the topic or subtopic when help text is displayed.

HELP_WINDOW
Causes help text to be displayed in a scrolling region of a video terminal.

NO HELP_WINDOW
Causes help text to be displayed in a nonscrolling region of a video terminal.

HELP_LINES n TO m
Sets the lines for scrolling help text between lines n and m, where n represents the beginning of the scrolling region and m represents the end.

Restrictions

- You must enter most SET commands at the DATATRIEVE command level, indicated by the DTR> prompt.

  If you are using the DATATRIEVE DECwindows interface, you can issue some SET commands by choosing items from the Setup menu. The items you can choose are Abort, Lock_Wait, Prompt, Search, Semicolon, Verify, and Columns_Page.

- You cannot use SET commands in compound statements: neither in THEN, IF-THEN-ELSE, or BEGIN-END statements.
- You cannot use SET commands in FOR, REPEAT, or WHILE statements.
- In the SET COLUMNS-PAGE command, the argument n must be an unsigned, nonzero integer less than or equal to 255.
- You must have at least P (PASS_THRU) access to all nodes of the dictionary path name of the directory node you specify in the SET DICTIONARY command.
- The value for n in SET HELP HELP_LINES must be at least 1. The value for m must be no greater than 24. In addition, m must be at least 4 greater than n.
- You should not have two users operating out of the same VMS directory with different settings for EDIT_BACKUP. DATATRIEVE could delete a backup file of the user who specified SET EDIT_BACKUP, because the other user had specified SET NO EDIT_BACKUP. This result is possible because SET NO EDIT_BACKUP instructs DATATRIEVE to delete the backup file created last.

Results

- When SET ABORT is in effect and DATATRIEVE is executing a procedure or command file, DATATRIEVE aborts the entire procedure or command file if it executes an ABORT statement or you enter a CTRL/Z in response to a prompt.
- When SET NO ABORT is in effect and DATATRIEVE is executing a procedure or command file, DATATRIEVE aborts only the statement containing the ABORT statement or prompt and executes the next statement in the procedure or command file.
- With the SET COLUMNS_PAGE command, you can affect the output of a PRINT statement that contains no implicit line feeds, that is, no SKIP, no NEW_PAGE, or no COL n with n less than the column reserved for previous print list elements.

  If an element in a print list would extend beyond the right column limit determined by the value of the COLUMNS_PAGE setting, DATATRIEVE shifts that element to the second line. By adjusting the COLUMNS_PAGE setting, you can control the way DATATRIEVE breaks the detail lines of its output.
- If you use a relative dictionary path name with the SET DICTIONARY command, DATATRIEVE uses your default CDD/Repository data dictionary directory to supply the missing part of the relative path name.
- If a SET DICTIONARY command fails because the specified CDD/Repository data dictionary directory does not exist or because you do not have adequate privileges, your default directory does not change. That is, you remain at the directory node where you were before you issued the SET DICTIONARY command.
- When SET HELP HELP_WINDOW is in effect, the location and size of the scrolling region is determined by the values for n and m with the SET HELP HELP_LINES command. The default location for the scrolling region is the upper part of the video terminal screen.

Usage Notes

- To display the settings of ABORT/NO ABORT, APPLICATION_KEYPAD/NO APPLICATIONKEYPAD, PROMPT/NO PROMPT, SEARCH/NO SEARCH, COLUMNS_PAGE, FORM/NO FORM, and VERIFY/NO VERIFY use the SHOW SET_UP command. See the section in this chapter on the SHOW command for more information.

- To display your default CDD/Repository data dictionary directory, use the SHOW DICTIONARY command.
- To display the names of the CDD/Repository data dictionary directories listed in your default directory, use the SHOW DICTIONARIES command. To set your dictionary to any descendant of your default directory, you do not have to specify the entire dictionary path name beginning with CDD$TOP. You can use a relative dictionary path name. For more information about dictionary path names and the CDD/Repository data dictionary, see the *VAX DATATRIEVE User's Guide*.
- You can assign a logical name to the full DCL file specification, then use the logical name with the SET CAITIFFS command. This is useful if you want to switch between different sets of key definitions for different applications:

```
DTR>  FN$CREATELOG("KYPD1",
CON>  "YOUR$DISK:[YOURDIR]DTRKEYS1.COM")
DTR>  FN$CREATELOG("KYPD2",
CON>  "YOUR$DISK:[YOURDIR]DTRKEYS2.COM")
DTR>  SET CAITIFFS KYPD1
DTR>  SET CAITIFFS KYPD2
```

- You can assign two file specifications in succession with the SET CAITIFFS command. DATATRIEVE implements the key definitions in both files:

```
DTR>  SET CAITIFFS YOUR$DISK:[YOURDIR]SHOWKEYS.COM,
CON>  CAITIFFS YOUR$DISK:[YOURDIR]SHOWKEYS2.COM
```

  If a conflict exists between key definitions (if a particular key is defined in both files), DATATRIEVE uses the definition from the file you specify last. In this example, DATATRIEVE would resolve key definition conflicts by using the definition from the file SHOWKEYS2.COM.

Examples

The following example displays the default settings and change the default settings with one SET statement:

```
DTR>  SHOW SET_UP
Set-up:
    Columns-page: 80
    No abort
    Prompt
    No search
    Form
    No verify
    No semicolon
    No lock_wait
    Application keypad mode
```

```
DTR>  SET COLUMNS_PAGE = 132, ABORT, NO PROMPT, SEARCH,
[Looking for SET option]
CON> 9 NO FORM, VERIFY, SEMICOLON, LOCK_WAIT, NO APPLICATION_KEYPAD

DTR> SHOW SET_UP
Set-up:
    Columns-page: 132
    Abort
    No prompt
    Search
    No form
    Verify
    Semicolon
    Lock_wait
    Numeric keypad mode
DTR>
```

The following example sets your default dictionary directory at CDD$TOP and uses a variety of path names to change your default directory:

```
DTR>  SET DICTIONARY CDD$TOP
DTR>  SET DICTIONARY DTR$LIB.DEMO
DTR>  SHOW DICTIONARY
The default directory is CDD$TOP.DTR$LIB.DEMO
DTR>  SHOW DICTIONARIES
Dictionaries:
DTR>  SET DICTIONARY -.-;
DTR>  SHOW DICTIONARY
The default directory is CDD$TOP
DTR>  SET DICTIONARY CDD$TOP.DTR32
Element "CDD$TOP.DTR32.JONES" not found in dictionary.
DTR>  SET DICTIONARY CDD$TOP.DTR32.TEST
DTR>  SHOW DICTIONARIES
Dictionaries:
DTR>  SET DEF -; SHOW DICTIONARY
The default directory is CDD$TOP.DTR32
DTR>  SHOW DICTIONARIES
Dictionaries:
        AWS             BRADS           BRENT           DBF
        DDD             DEMO            DENN            DETRIC
        DUNCAN          JAS             KELLERMAN       LANDAU
        MARISON         PLOTS           STRONG          TEST
        WAYNE

DTR>  SET DICTIONARY WAYNE
DTR>  SET DICTIONARY DISK1:[SWANSON.DTRWORK]TEST.NEWDICT
DTR>
```

The following example uses the SET SEARCH command to walk sets in a VAX DBMS database:

```
DTR>  READY SUPPLIES, VENDORS, PART_S
DTR>  SET SEARCH
DTR>  PRINT VEND_NAME, PART_DESC OF
[Looking for name of domain, collection, or list]
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
```

When SET SEARCH is not in effect, you need to provide explicitly the inner print lists to give DATATRIEVE the necessary context, as follows:

```
DTR>  SET NO SEARCH
DTR>  PRINT VEND_NAME, ALL ALL PART_DESC OF PART_S OWNER OF
[Looking for set name]
CON>     PART_INFO OF SUPPLIES MEMBER OF VENDOR_SUPPLY OF
[Looking for name of domain, collection, or list]
CON>     VENDORS WITH VEND_NAME = "QUALITY COMPS"
--------------Vendor Name---------------

QUALITY COMPS
VT100 KEYBOARD ASSY
NUMERIC KEYPAD FRAME
VT52 HOUSING

DTR>
```

SET Statement (Report Writer)

Controls the report header and defines the size of report pages and the length of the report. With Report Writer SET statements, you can specify the following:

- The report header—the report name (if any), the date, page numbering, and their print attributes
- The size of report pages—the number of columns and the number of lines per page, or the physical paper size
- The orientation of the paper to be printed on.
- The length of the report—the maximum number of lines and the maximum number of pages
- Whether or not column headers are printed

Format

For naming the report:

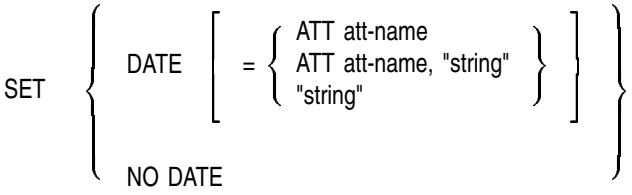
```
                                         { "string"[/ . . . ] }
SET   REPORT_NAME =   [ ATT   att-name, ] {                    }
                                         {  *.prompt          }
```

For controlling the printing of default page numbers and for specifying the beginning page number at the upper right of a page:

```
      {                 [    { ATT att-name                      } ] }
      {                 [    {                  {  n        }    } ] }
      {                 [    { ATT att-name,    { *.prompt  }    } ] }
      {   NUMBER        [ =  {                                   } ] }
SET   {                 [    { {  n        }                     } ] }
      {                 [    { { *.prompt  }                     } ] }
      {                                                              }
      {   NO NUMBER                                                  }
```

For specifying a date or string at the upper right of each page or for disabling the printing of a date:

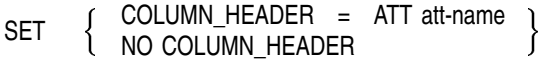
SET Statement (Report Writer)



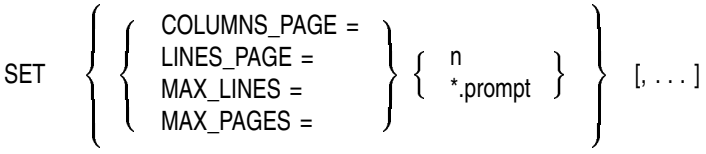
For disabling the printing of the entire report header:

```
SET NO REPORT_HEADER
```

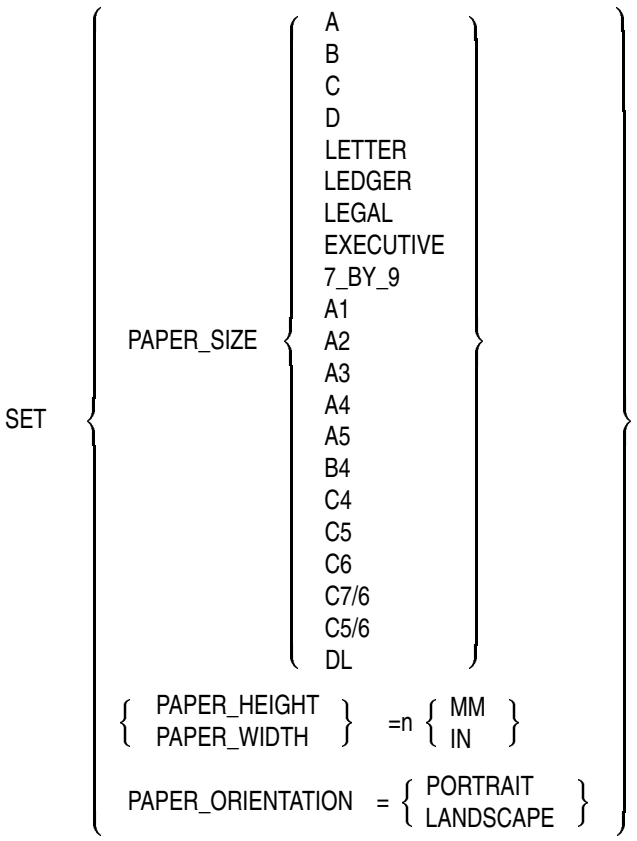
For specifying the printing of column headers:



For specifying page width or length, or overall report length:



For specifying paper size:



Arguments Table 4–33 summarizes information about each type of SET statement.

Table 4–33 SET Statement Arguments

Argument	Function	Default	Prompt Option	Maximum Value
Report Header				
REPORT_NAME	Centers a report name on the first line of each page.	No report name	[1] Yes	-
DATE	Specifies a date or string that is printed on the upper right of each page.	Current system date	No	-
NO DATE	Suppresses the printing of a date.	Current system date	No	-
NUMBER	Causes the printing of a page number below the date.	Current page number	Yes	99,999
NO NUMBER	Suppresses the printing of a page number.	Current page number	No	-
[2] NO REPORT_ HEADER	Suppresses the printing of report name, date, column headers, and page number on each page.	Header printed on each page	No	-
Column Headers				
COLUMN_HEADER	Only used to specify the attributes for the column headers	Headers printed on each page	No	-
NO COLUMN_ HEADER	Suppresses the printing of column headers.	Headers printed on each page	No	-
Page Size				
COLUMNS_PAGE	Specifies the page width in columns.	Value at DCL or 80 columns	Yes	255 columns
LINES_PAGE	Specifies the page length in number of lines.	60 lines	Yes	About 2 billion lines

[1]Response to prompt must be enclosed in quotation marks.

[2]SET NO REPORT_HEADER overrides all other SET commands that control or specify elements in the report header.

(continued on next page)

Table 4–33 (Cont.) SET Statement Arguments

Argument	Function	Default	Prompt Option	Maximum Value
Page Size				
MAX_LINES	Specifies the maximum lines for the report.	No line limit	Yes	About 2 billion lines
Report Size				
[3]MAX_PAGES	Specifies the maximum pages for the report.	No page limit	Yes	About 2 billion pages
Paper				
PAPER_SIZE	Specifies the size of paper in standard formats	Letter(A)	No	-
PAPER_HEIGHT	Specifies the height of paper in millimeters or inches	As defined in PAPER_SIZE	No	-
PAPER_WIDTH	Specifies the width of paper in millimeters (MM) or inches (IN)	As defined in PAPER_SIZE	No	-
PAPER_ORIENTATION	Specifies Landscape or Portrait orientation	PORTRAIT	No	-

[3]Although the maximum number of pages is about 2 billion, the maximum page number printed at the upper right of a page is 99,999.

Usage Notes

- If you embed a SET REPORT_NAME statement with a prompting value expression in a compound statement that also has other prompts, DATATRIEVE may execute the prompts in an unexpected order. Consider the following example:

```
WHILE ORDER_NUM NOT STARTING WITH "1"
  BEGIN
  ORDER_NUM = *."Order Number"
  REPORT JOBITM ON *."FILE,LP: OR TT"
  SET REPORT_NAME= *."Report Name"
  PRINT JOBNBR
  END_REPORT
  END
```

In this case, the prompt for "Report Name" occurs before either of the other two prompts.

This is because DATATRIEVE processes the entire compound statement (beginning with the WHILE statement) at one time. However, in order to process a REPORT statement, DATATRIEVE must know the length of the report name first so that it can format the report. This means that DATATRIEVE must prompt for the report header out of sequence with the other prompts. DATATRIEVE then prompts for the other values in the same order as the prompting value expressions appear.

- If your report contains a list, each item in the list counts as a separate detail line.
- The Report Writer counts all the lines or pages of the report. If you have specified a limit and it is reached, the Report Writer stops producing the report and prints one of the following error messages:

  ```
  Maximum line count exceeded - report terminated.
  ```

  ```
  Maximum report pages exceeded - report terminated.
  ```

- DATATRIEVE calculates the maximum lines set for a report at the end of a page, not in the middle, even if the MAX_LINES setting is reached in the middle of a page. If DATATRIEVE reaches the MAX_LINES setting mid-page, it prints out the full page before stopping the report.
- The options PAPER_HEIGHT and PAPER_WIDTH are not necessary when PAPER_SIZE is defined. If present, they supersede the corresponding values for the standard size specified through the PAPER_SIZE option.
- Placing an ATT clause before the value of a report header element sets the attributes for that element. The name of the ATT clause must have previously been declared by a DECLARE_ATT statement. Only the attributes specifically set by the DECLARE_ATT statement are affected. All the other attributes remain at their default value.

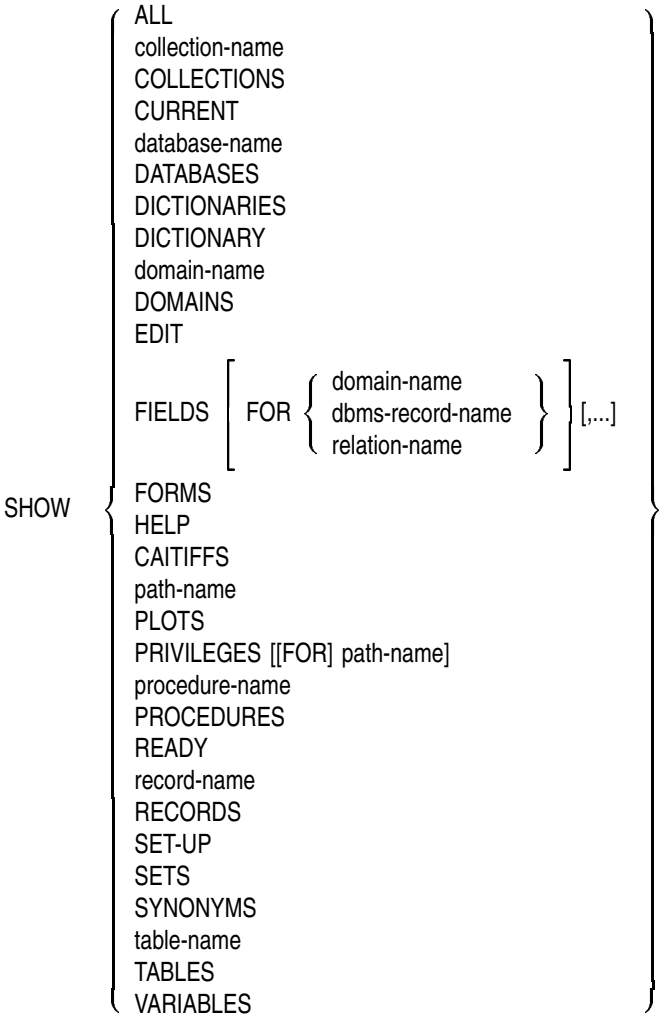
- The user can override the default attributes for the report header elements by defining the logical names DTR$RW_BODY_ATTRIBUTES (for report name, date, and page number) and DTR$RW_HEADER_ATTRIBUTES (for report column header). The equivalence string will be an attribute list, using the same syntax as the DECLARE_ATT statement. Attributes which are not defined in such a list remain set to the default value (see the DECLARE_ATT statement).
- The following table summarizes the applicability of the SET statements to various formats:

SET statement	ASCII (TEXT)	DDIF, PS	DTIF
REPORT_NAME	Applicable	Applicable	Ignored
[NO] DATE	Applicable	Applicable	Ignored
[NO] NUMBER	Applicable	Applicable	Ignored
NO REPORT_HEADER	Applicable	Applicable	Ignored
[NO] COLUMN_HEADER	Applicable	Applicable	Applicable
COLUMNS_PAGE	Defines page width (in characters)	Ignored	Ignored
LINES_PAGES	Defines page height (in lines)	Ignored	Ignored
PAPER_SIZE	Ignored	Defines page size (standard sizes)	Ignored
PAPER_WIDTH	Ignored	Defines page width (in IN or MM)	Ignored
PAPER_HEIGHT	Ignored	Defines page height (in IN or MM)	Ignored
MAX_LINES	Applicable	Applicable	Applicable
MAX_PAGES	Applicable	Applicable	Ignored

SHOW Command

Displays information about the CDD/Repository data dictionary and its contents.

Format



Arguments

SHOW ALL
Displays the names of all the objects and directories listed in your default CDD/Repository directory, the name of your default directory, and the names of the collections, the other readied RSE sources (domains, relations, VAX DBMS records), and the loaded tables in your workspace. DMU format record and domain definitions are indicated with an asterisk (*).

SHOW collection-name
Displays the collection name, the name of the domain, relation, or VAX DBMS record within which the collection has been established, the number of records in the collection, the status of the selected record within the collection, and the names of the keys on which the collection has been sorted.

SHOW COLLECTIONS
Displays the names of the collections you are using.

SHOW CURRENT
Displays the name of the domain, relation, or VAX DBMS record within which the CURRENT collection has been formed, the number of records in the CURRENT collection, the status of the selected record in the CURRENT collection, and the names of the keys on which the collection has been sorted.

SHOW database-name
For relational databases, displays the name and the file specification of the database. For VAX DBMS, displays the name, the subschema name, the schema path name, and the root file specification of the database.

SHOW DATABASES
Displays the names of the relational and VAX DBMS databases listed in your default directory.

SHOW DICTIONARIES
Displays the names of the dictionary directories appended to your default directory. (This option tells you if the dictionary branch continues lower than your current location.)

SHOW DICTIONARY
Displays the full dictionary path name of your default directory.

SHOW Command

SHOW domain-name
Displays the name, the record definition name, and the file specification of the RMS domain. Displays the domain name, the associated VAX DBMS record, and the database name associated with the VAX DBMS domain. Displays the domain name, the associated relation name, and the database name associated with the relational domain.

SHOW DOMAINS
Displays the names of all domains cataloged in your default directory. DMU format domains are indicated by an asterisk (*).

SHOW EDIT
Indicates whether SET EDIT_BACKUP or SET NO EDIT_BACKUP is in effect in your DATATRIEVE session. SET EDIT_BACKUP is the default. If you enter the command SET NO EDIT_BACKUP and edit any definitions, DATATRIEVE deletes the highest version of the definitions when you exit the editor. SET NO EDIT_BACKUP automatically keeps outdated versions of definitions from piling up in your directory, but could erase the only definition you have to fall back on should you make a mistake in your editing.

SHOW FIELDS
Displays the names, data types, and index-key information of the fields of all domains you have readied. It also displays the names and data types of global variables. For RMS sources, the SHOW FIELDS command indicates whether or not a key field is the primary key or an alternate key. For fields from non-RMS sources, SHOW FIELDS indicates an indexed key.

SHOW FIELDS
FOR domain-name
dbms-record
rdb-relational-name

Displays the names, data types, and FOR domain-name index-key information of the fields in the domain you specify after FOR. You can only specify the name of a readied domain.

SHOW FORMS
Displays the form name, the form file, and the form product name of all loaded forms.

SHOW HELP
Indicates which of the settings for the HELP command is in effect. The settings are HELP_LINES, HELP_PROMPT, and HELP_WINDOW.

SHOW CAITIFFS
Shows all current key definitions in all states. A state allows the same key to be assigned multiple definitions by associating each definition with a different state key. In addition to SHOW CAITIFFS, you can use the function FN$SHOW_KEYDEFS to show all key definitions. This command is not available in a DECwindows environment.

SHOW path-name
Displays the text of the domain, record, procedure, or table definition specified by the dictionary path name.

SHOW PLOTS
Displays the names of the loaded plots from the directory specified in the SET PLOTS command.

SHOW PRIVILEGES
Displays the access privileges you have to the directory at which you are currently located.

SHOW PRIVILEGES
FOR path-name
Displays the access privileges you have to the directory or object you name in the FOR clause.

SHOW procedure-name
Displays the name of the procedure, the commands and statements contained in the procedure, and the END_PROCEDURE clause.

SHOW PROCEDURES
Displays the names of all procedures cataloged in the directory at which you are currently located.

SHOW READY
Displays for each readied RMS domain the full dictionary path name, the file organization of the associated data file, the access control option (EXCLUSIVE, PROTECTED, or SHARED), and the access mode (READ, WRITE, EXTEND, or MODIFY). For all readied relational and VAX DBMS domains, the SHOW READY command displays the name, type, access option, access mode, and the domain path. For individual relations and VAX DBMS records, the SHOW READY command displays the name, type, access option, access mode, and the dictionary path to the database. The most recently readied domain, relation, or VAX DBMS record is at the top of the displayed list. (See the section in this chapter on the READY command for further information.) The SHOW READY

command also displays the full dictionary path name and table type of all tables loaded in your workspace.

SHOW record-name
Displays the name, level numbers, fields, and field definitions of the record.

SHOW RECORDS
Displays the names of all record definitions cataloged at your current directory location. DMU format records are indicated by an asterisk (*).

SHOW SET_UP
Displays the current status of the options you can control with the SET command: ABORT/NO ABORT, APPLICATION_KEYPAD/ NO APPLICATION_KEYPAD, COLUMNS_PAGE, FORM/NO FORM, PROMPT/NO PROMPT, SEARCH/NO SEARCH, SEMICOLON/ NO SEMICOLON, and VERIFY/NOVERIFY.

SHOW SETS
Displays the names of any available VAX DBMS sets and their VAX DBMS insertion and retention classes.

SHOW SYNONYMS
Displays the names of any synonyms for DATATRIEVE keywords in effect during your DATATRIEVE session.

SHOW table-name
Displays the name of the table, the code and translation pairs, and the END_ TABLE clause.

SHOW TABLES
Displays the names of all dictionary tables and domain tables cataloged in your default directory.

SHOW VARIABLES
Displays the global variables in effect in the current DATATRIEVE session.

Restrictions

- In the SHOW command, you can use the arguments ALL, DOMAINS, RECORDS, PROCEDURES, TABLES, and DICTIONARIES to display the given names of the objects and directories listed in your default CDD/Repository directory.

- To establish a DMU format directory as your default directory, you must have at least P (PASS_THRU) access to the directory and all its ancestors. Thus, if you have P (PASS_THRU) access to a directory and its ancestors, you have access to the given names of all objects and directories cataloged in that directory.
- To establish a CDO format dictionary as your default dictionary, you must have S (SHOW) access to the dictionary.
- To use the SHOW path-name command to display information about a dictionary object in the DMU format dictionary, you must have the following access privileges:
  - P (PASS_THRU) access to it and its ancestors
  - S (SEE) and R (DTR_READ) access to the object
- To use the SHOW path-name command to display information about a dictionary object in the CDO format dictionary, you must have the following access privileges:
  - S (SHOW) access to the dictionary
  - S (SHOW) and R (READ) access to the object
- You cannot specify the dictionary path name of a CDD/Repository directory with the SHOW path-name command.
- When using the SHOW path-name command to display information about the definition of a domain, record, procedure, or table, that dictionary object must have been defined using DATATRIEVE.
- Plot definitions are stored only in the DMU format dictionary.
- You must have at least P (PASS_THRU) access to a dictionary object and its ancestors to display your access rights to it with a SHOW PRIVILEGES command.

Results

- DATATRIEVE displays the information you request, in the order requested.
- DATATRIEVE displays the objects you specify with version numbers at the end of the path name.
- You need only P (PASS_THRU) access to your default DMU directory or S (SHOW) access to your default CDO dictionary to use the SHOW command with these arguments: COLLECTIONS, collection-name, CURRENT, READY, FIELDS, VARIABLES, and SET_UP. Any other access privileges you may

have are irrelevant for these SHOW command options because DATATRIEVE does not access the data dictionary for the information you request.

- You do not need P (PASS_THRU) access to the objects or directories in your default DMU directory or S (SHOW) access to your default CDO dictionary to see the given names of objects when you enter a SHOW command with one of these options: ALL, DOMAINS, RECORDS, PROCEDURES, TABLES, and DICTIONARIES.
- If you do not have the access privileges needed to obtain information for the SHOW PRIVILEGES or SHOW path-name commands, DATATRIEVE displays an error message and returns you to DATATRIEVE command level.
- When you have more than one existing collection and you enter the SHOW COLLECTIONS command, the order in which DATATRIEVE displays the collections reverses the order in which you established them. The CURRENT collection is always at the top of the list, and the oldest existing collection is always at the bottom of the list.

  Knowing the order of the collections is useful because you can see the order in which DATATRIEVE searches for a selected record to establish a single record context for the MODIFY, LIST, PRINT, ERASE, and DISPLAY statements and for resolving field names in value expressions.
- The SHOW SET_UP command lets you check the current status of set options that affect your DATATRIEVE session. The second example in the Examples section shows the default settings for these options. The section in this chapter on the SET command discusses the meaning of these options.
- When you display objects in CDO directories using the CDO DIRECTORY command, not all objects created by DATATRIEVE appear in the directory listing. For example, while DATATRIEVE created a CDD$DATABASE object for the YACHTS_CDO domain, the CDD$DATABASE object for that domain is not displayed in a CDO directory listing. DATATRIEVE does not create directory entries for the special CDO objects it creates when it defines a domain. You can create these directory entries using the generic form of the CDO utility's ENTER command. To do this you must know the name of the object for which you want to create an entry. See the *VAX DATATRIEVE User's Guide* for information on the conventions DATATRIEVE uses when it creates these objects. See the CDD/Repository documentation for information on the ENTER command.

  A similar situation occurs for parts of record definitions that contain group fields. If you display the definition of YACHT_CDO_REC using the CDO SHOW RECORD record-name command, you see that the definition includes references to records called TYPE and SPECIFICATIONS. The reason for this is that the CDO utility translates DATATRIEVE group fields as record

definitions. However, entries for these definitions do not appear in a CDO directory listing. You can add these entries to the directory using the CDO utility's ENTER command. See the CDD/Repository documentation for more information.

Information on the SHOW PRIVILEGES command and on the DATATRIEVE CDO command can be found in the *VAX DATATRIEVE User's Guide*.

Examples

The following example displays the record definition OWNER_REC:

```
DTR>  SHOW OWNER_REC
RECORD OWNER_REC
01 OWNER.
   03 NAME PIC X(10).
   03 BOAT_NAME PIC X(17).
   03 TYPE.
      06 BUILDER PIC X(10).
      06 MODEL PIC X(10).
;

DTR>
```

The following example displays the status of the SET command options that affect your DATATRIEVE session:

```
DTR>  SHOW SET_UP
Set-up:
    Columns-page: 80
    No abort
    Prompt
    No search
    Form
    No verify
    No semicolon
    No lock wait
    Application keypad mode

DTR>
```

SHOWP Command

Displays the access control list (ACL) of an object or directory in the CDD/Repository data dictionary.

Format

SHOWP path-name

Argument

path-name
Is the given name, full dictionary path name, or relative path name of the dictionary object or directory whose access control list you want to display.

Restrictions

- To display the access control list of an object or directory in the DMU format data dictionary, you must have the following access privileges:
  - P (PASS_THRU) access to the parent directory of the object or directory
  - P (PASS_THRU) and C (CONTROL) access to the object or directory
- To display the access control list of an object or directory in the CDO format data dictionary, you must have S (SHOW) access privileges to the dictionary and the object.

Result

DATATRIEVE displays the entire access control list for the specified dictionary object or directory.

Usage Notes

- Before you use the DELETEP command to remove an entry from an access control list, use the SHOWP command to verify the sequence number of the entry you want to delete.
- To see what access privileges you have to an object, use the SHOW PRIVILEGES command. See the SHOW command for further information.
- The *VAX DATATRIEVE User's Guide* discusses ACLs and CDD/Repository protection. See also the VAX CDD/Repository documentation.

Examples

The following example displays the access control list for the DMU YACHTS domain:

```
DTR>  SHOWP YACHTS
  1:    [*,*], Username: "JONES"
        Grant - CDEHMRSUW, Deny - none, Banish - none

DTR>
```

The following example uses a password to gain C (CONTROL) access to the DMU record YACHT and then display its access control list:

```
DTR>  SHOWP YACHT (*)
Enter password for YACHT:   !You enter CEP, which is not echoed.
DTR>  SHOWP YACHT
  1:    [*,*], Password: "P"
        Grant - P, Deny - CDESUX, Banish - none
  2:    [*,*], Password: "DP"
        Grant - DP, Deny - CESUX, Banish - none
  3:    [*,*], Password: "EP"
        Grant - EP, Deny - CDSUX, Banish - none
  4:    [*,*], Password: "PS"
        Grant - PS, Deny - CDEUX, Banish - none
  5:    [*,*], Password: "PU"
        Grant - PU, Deny - CDESX, Banish - none
  6:    [*,*], Password: "PX"
        Grant - PX, Deny - CDESU, Banish - none
  7:    [*,*], Password: "CEP"
        Grant - CEP, Deny - DSUX, Banish - none
  8:    [*,*], Password: "CDP"
        Grant - CDP, Deny - ESUX, Banish - none

DTR>
```

The following example displays the ACL of the CDO format dictionary SYS$COMMON:[CDDPLUS.PERSONNEL]:

```
DTR> SHOWP SYS$COMMON:[CDDPLUS.PERSONNEL]
 1: [*,*], Username: "CASADAY"
    GRANT - RWMESUDC, DENY - none
    ACCESS=READ+WRITE+MODIFY+EXTEND+SHOW+DEFINE+CHANGE+DELETE+CONTROL
 2: [*,*]
    GRANT - S, DENY - RWMEUDC
    ACCESS=SHOW

DTR>
```

SIGN Clause

Specifies the location and representation of a sign (+ or -) in a numeric elementary field.

Format

SIGN [IS] { LEADING / TRAILING } [SEPARATE]

Arguments

LEADING
TRAILING
Indicates that the sign is at the left (LEADING) or right (TRAILING) of the field value.

SEPARATE
Indicates that the sign occupies its own character position in the field. If this argument is omitted, the sign shares a character position with the field's leftmost (LEADING) or rightmost (TRAILING) digit.

Restrictions

- This clause can be used only with numeric elementary fields.
- A field definition cannot contain both a SIGN and a USAGE clause.
- Currently, the SIGN LEADING SEPARATE and SIGN TRAILING SEPARATE field clauses cannot be used in a record stored in a CDO format dictionary. If you include either of these clauses in a record definition, you receive the following message:

```
CDD-E-VALDEFAIL, entity definition failed validation CDD$DIGITS_LENGTH.
```

  This problem will be corrected in a future version of VAX CDD/Repository.

Result

If you do not include a SIGN clause with a numeric field, the sign shares a character position with the field's rightmost digit. You must include this clause if a program written in COBOL (or other language) uses the record and requires that the sign be a separate character or share the leftmost character position. A sign clause does not affect the output format of a field.

Examples

The following example defines the field CURRENT_BALANCE as a 6-digit signed field, with the sign sharing the leftmost character position:

```
03 CURRENT_BALANCE
   PIC IS S9999V99
   EDIT_STRING IS $$$$9.99-
   SIGN IS LEADING.
```

The following example defines the field NEW_PRICE as a 4-digit signed field. The sign is a separate character in the rightmost character position:

```
03 NEW_PRICE PIC S99V99
   SIGN TRAILING SEPARATE
   EDIT_STRING +++.99.
```

SORT Statement

Arranges a DATATRIEVE collection according to the order you specify for the contents of one or more fields in the records.

Format

SORT [collection-name] [BY] sort-key-1 [,...]

Arguments

collection-name

Is the name of the collection to be sorted.

BY

Is an optional language element you can use to clarify syntax.

sort-key

Is a field whose contents form the basis for the sort. You can also use a value expression as a sort key as long as the value expression does not contain a SORTED BY or REDUCED TO clause.

The sort key can be preceded or followed by a keyword that determines the order in which DATATRIEVE sorts the records in the collection. ASCENDING is the default order.

To specify the sort order for each sort key, use one of the following terms:

```
ASC[ENDING]
DESC[ENDING]
INCREASING
DECREASING
```

If you specify more than one sort key, use a comma to separate each sort key from the next.

Restrictions

- You can use the SORT statement only to rearrange a collection you have already formed with a FIND statement.
- You cannot use the SORT statement in a compound statement.
- You must specify at least one sort key.
- You cannot specify more than 255 sort keys in an RSE or in a SORT statement.
- A sort field cannot be a direct table lookup.

Results

- DATATRIEVE sorts the collection according to the order and fields you specify. DATATRIEVE sorts alphanumeric fields according to the order of the ASCII collating sequence. See the VAX/VMS documentation for more information on the DEC Multinational Character Set, which includes the ASCII character set.
- If you omit the collection name, DATATRIEVE sorts the current collection.
- If you specify ASC[ENDING] or INCREASING in the sort key, DATATRIEVE puts the record with the lowest value in the specified field first in the collection and the one with the highest value last. This is the default.

  If you specify DESC[ENDING] or DECREASING, DATATRIEVE puts the record with the highest value in the specified field first in the collection and the one with the lowest value last.
- If you specify more than one sort key, DATATRIEVE uses the first field name as the major sort key and each successive field name as an increasingly minor key.
- If, in the first sort key, you omit a keyword specifying the sort order (for example, ASCENDING or DESCENDING), DATATRIEVE arranges the collection according to the ascending order of the contents of that field.

  If, in the second or subsequent sort keys, you omit a keyword specifying the sort order, DATATRIEVE uses the sort order implied or specified for the preceding sort key.
- When DATATRIEVE executes the SORT statement, any selected record for the target collection is released, and the collection cursor does not point to any record in the collection. In addition, the number of records shown in the SHOW CURRENT display no longer includes records dropped from the collection.

Usage Notes

- To sort record streams, use the SORTED BY clause of the record selection expression that creates the record stream (see the chapter on RSE in the *VAX DATATRIEVE User's Guide*).
- To sort collections when you first form them, use the SORTED BY clause of the RSE in the FIND statement.

Examples

The following example sorts the first ten yachts by the ratio of PRICE to DISPLACEMENT:

```
DTR>  FIND FIRST 10 YACHTS
[10 records found]
DTR>  SORT BY PRICE/DISP
DTR>  PRINT ALL TYPE, PRICE, DISP,
[Looking for next element in list]
CON>  PRICE/DISP ("$/LB."/"RATIO") USING $$$.99

                                        $/LB.
MANUFACTURER   MODEL     PRICE  WEIGHT RATIO

 BLOCK I.    40                18,500   $.00
 BUCCANEER   270                5,000   $.00
 ALBERG      37 MK II   $36,951 20,000  $1.85
 AMERICAN    26          $9,895  4,000  $2.47
 BOMBAY      CLIPPER    $23,950  9,400  $2.55
 AMERICAN    26-MS      $18,895  5,500  $3.44
 BAYFIELD    30/32      $32,875  9,500  $3.46
 ALBIN       VEGA       $18,600  5,070  $3.67
 ALBIN       BALLAD     $27,500  7,276  $3.78
 ALBIN       79         $17,900  4,200  $4.26

DTR>
```

The following example forms a collection of FAMILIES CROSS KIDS and sorts the collection by decreasing age of the children:

```
DTR>  FIND FIRST 6 FAMILIES CROSS KIDS
[6 records found]
DTR>  SORT BY DESC AGE
DTR>  PRINT ALL FATHER, MOTHER, EACH_KID

                         KID
  FATHER     MOTHER      NAME    AGE

JIM         LOUISE     ANNE       31
JIM         LOUISE     JIM        29
JIM         LOUISE     ELLEN      26
JIM         LOUISE     DAVID      24
JIM         ANN        URSULA      7
JIM         ANN        RALPH       3
```

STORE Statement

Creates a record in a DATATRIEVE domain and stores values in one or more fields of the record. For information on using STORE with VAX DBMS and relational databases, see the section on the STORE statement for VAX DBMS and relational sources in this chapter.

Format

```
STORE  [context-variable IN]  domain-name
  [USING statement-1]

  [VERIFY [USING] statement-2]
```

Arguments

context-variable
Is the name of the record to be inserted.

domain-name
Is the given name or alias of the domain that contains the new record.

See the chapter on RSE in the *VAX DATATRIEVE User's Guide* for discussions of context variables.

USING statement-1
Specifies a DATATRIEVE statement that can store one value for one or more fields in the new record.

VERIFY [USING] statement-2
Specifies a statement DATATRIEVE executes just before storing the new record.

Restrictions

- The domain to contain the new record must be readied for WRITE or EXTEND access (see the section on the READY command.)
- You cannot store a new record in an RMS relative file or a view.
- With the USING clause you cannot store values in the fields of a list unless you include a MATCH statement (see the section on the MATCH statement). You can also store values into list fields by omitting the USING clause and having DATATRIEVE prompt you for a value for each field in the record. Or you can make no reference to the list fields in the USING clause, and use the MODIFY statement later to enter values in the list fields.

- If you include the keyword USING when specifying statement-1, you must put USING on the same input line as the domain name, unless you end the input line with a continuation character, hyphen (-). If you omit the keyword USING and the continuation character, you must put at least the first element of statement-1 on the same input line as the domain name.
- Unless you end your input line with the continuation character, hyphen (-), do not press the RETURN key immediately before typing the keyword VERIFY.
- For data contained in an RMS indexed file, you cannot store a new record that duplicates the information in either a primary or alternate key field with the NO DUP attribute.
- You cannot store a value in a COMPUTED BY field.
- You cannot store list fields or their subordinates in hierarchical records in remote domains.
- To use the name of a REDEFINES field when storing a record, you must specify the name in a USING clause. Otherwise, DATATRIEVE prompts you with the name of the elementary field from which the REDEFINES field takes its value.
- If you want to use a form to display the records you are storing with the STORE statement, you can use one of two methods:
  - Include a FORM clause in the definition of the domain you wish to access. Then you must omit the field list and the USING clauses of the STORE statement. That is, you must retrieve entire records from the target collection or record stream.
  - Use the DISPLAY_FORM statement for FMS and TDMS forms, or the WITH_FORM statement for DECforms forms.

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

    If you use the WITH_FORM statement to store, you must use its RECEIVE clause.

  See the sections on DEFINE DOMAIN and DISPLAY_FORM in this chapter. SET FORM must be in effect when you ready the domain and when you enter the STORE statement. Check the status of the SET FORM/NO FORM setting with the SHOW SET_UP command.

Results

- If you omit the USING statement-1 clause, DATATRIEVE prompts you to supply a value for the first elementary field in the record with the following message:

  ```
  ENTER field-name:
  ```

  After you enter a value, DATATRIEVE prompts you to supply a value for the next elementary field in the record, and so on, until you have entered a value for every elementary field in the record.

  After you supply a value to the last elementary field in the record, DATATRIEVE adds the record to the data file.

- If you press only the RETURN key in response to a prompt, DATATRIEVE repeats the prompt.
- To enter all spaces in an alphabetic or alphanumeric field or to enter all zeros in a numeric field, respond to the STORE statement's prompt with one or more spaces and press the RETURN key.
- If a field definition contains a DEFAULT VALUE clause and you enter one TAB when prompted for a value in that field, DATATRIEVE enters the default value in the field. DATATRIEVE enters the default value regardless of whether the field definition contains a MISSING VALUE clause.
- If you enter two or more tab characters to a prompt for an alphanumeric field, DATATRIEVE stores the tab characters in the field. If you enter two or more tab characters at a prompt for a numeric field, DATATRIEVE displays an error message and reprompts you for valid numeric data.
- If a field definition contains a MISSING VALUE clause and no DEFAULT VALUE clause, DATATRIEVE stores the missing value in the field when you respond to the prompt with one TAB.
- If you press CTRL/Z any time between the first prompt and entering your response to the last prompt, DATATRIEVE aborts the STORE statement and returns you to DATATRIEVE command level. No new record is stored.
- If in response to a prompt for a field value you enter more characters or digits than the field definition specifies, DATATRIEVE displays an error message and reprompts you for another value.
- If you enter data that conflicts with the conditions specified by a VALID IF clause in the record definition, DATATRIEVE reprompts you for valid data.

- If you specify a VERIFY USING clause, no data is stored until DATATRIEVE successfully executes statement-2. If the VERIFY USING clause contains an ABORT statement in an IF-THEN-ELSE statement and the abort conditions are met, DATATRIEVE aborts the STORE statement and returns you to DATATRIEVE command level. This abort occurs whether you have SET ABORT or SET NO ABORT in effect. When DATATRIEVE aborts a STORE statement, no record is stored.

Usage Notes

- In the USING statement-1 clause, you can include an Assignment statement or a BEGIN-END block containing a series of Assignment statements. You can use the following two forms of the Assignment statement:

  ```
  field-name = value-expression
  ```

  ```
  group-field-name-1 = group-field-name-2
  ```

  See the Assignment section of this chapter for more information on assigning a value to an elementary field and assigning a value to a group field.

- Each time DATATRIEVE completes the execution of a STORE statement, the new record is part of the data file. If a STORE statement is part of a loop and it creates a record each time DATATRIEVE executes the loop, DATATRIEVE enters each new record in the data file when it executes the STORE statement. If you abort the loop with a CTRL/Z, CTRL/C, or ABORT statement, the records already stored are unaffected.

  Once the records are in the data file, you must, when working with DATATRIEVE, use either the ERASE statement for indexed files or the MODIFY statement for sequential files to remove data from the files.

- To store more than one record in a domain, you can use the following form of the REPEAT statement:

  ```
  REPEAT n STORE domain-name
  ```

  The argument n represents the number of records to be stored in the domain. You can avoid counting the new records you want to store by making n much larger than the estimated number of new records. When you finish storing records, stop the prompts for additional data by entering a CTRL/Z in response to the prompt. DATATRIEVE then returns you to command level.

- You can use the VERIFY clause to check data before DATATRIEVE stores a new record. Put an ABORT statement in an IF-THEN-ELSE statement that establishes the conditions for the abort. In statement-2 of the VERIFY clause, you can also put a BEGIN-END block containing a series of IF-THEN-ELSE statements.

- The values you supply to the prompts of the STORE statement are first checked against any validation conditions specified for that field in the record definition. If the value conforms to the conditions specified in the appropriate VALID IF clause in the record definition, only then is it checked against the conditions in the VERIFY clause of the STORE statement.

  If you always use the same validation conditions when storing data, put those conditions in VALID IF clauses in the record definition. That way, DATATRIEVE reprompts you for a value if you enter invalid data. Otherwise, the ABORT statement in a VERIFY clause returns you to DATATRIEVE command level, and you have to reissue the STORE command to resume entering data.

- With the STORE statement, you can transfer information from one domain to another. Use the following syntax to nest the STORE statement in a FOR loop:

```
FOR domain-1
     STORE domain-2 USING
          group-field-name-2 = group-field-name-1
```

  The group fields need not contain identical elementary fields, but DATATRIEVE transfers values only between elementary fields with identical field names. You should use this method (instead of a RESTRUCTURE statement) when a record contains duplicate elementary field names subordinate to different group fields.

  Note, however, that DATATRIEVE still needs to know which elementary fields in the source domain are to be matched with the fields of the same names in the target domain. The specified group field names must unambiguously identify the fields in question. For example, do not specify a higher-level group field that includes multiple elementary fields of the same name; specify the lower-level group field that can distinguish the appropriate elementary field.

  You can use this method to transfer data between RMS sequential files and RMS indexed files. Domain-1 and domain-2 can share the same record definition, but the definition of one domain specifies a sequential file, and the other an indexed file. See the DEFINE FILE command for details on specifying types of file organization.

  In a similar manner, you can also transfer elementary field values from one domain to another. Put a STORE statement in a FOR statement whose RSE selected the desired source records, and put the necessary Assignment statements in the USING clause of the STORE statement.

If the definition of an elementary field of the target domain contains a DEFAULT VALUE clause and there is no matching field in the source domain, DATATRIEVE stores the default value in the field. This default value is stored whether or not the field definition contains a MISSING VALUE clause.

If the definition of an elementary field of the target domain contains a MISSING VALUE clause and no DEFAULT VALUE clause, and there is no matching field in the source domain, DATATRIEVE stores the missing value in the field.

If the definition of an elementary field contains neither a DEFAULT VALUE clause nor a MISSING VALUE clause and there is no matching field in the source domain, DATATRIEVE stores blanks in alphabetic and alphanumeric fields and zeros in numeric fields.

- You can also transfer information from one domain to another with the Restructure statement using the following format:

  ```
  domain-name = rse
  ```

  This form of the Assignment statement creates an implicit FOR loop and an implicit STORE statement with the appropriate USING clause. See the section in this chapter on the Restructure statement for further details about restructuring domains.

- Take special care when storing data in primary key fields of RMS indexed files and in other fields with the NO CHANGE attribute. Values in these fields cannot be changed with the MODIFY command, and errors can be corrected only by creating a new record with the correct value and erasing the old record.

- Use a context variable with the domain name in the STORE statement if you want to refer to the values you have entered before DATATRIEVE stores the record. You can use the values for VERIFY USING clauses (as in the second example of the Examples section), or you can use the values of one field to calculate the values of other fields in the record as you store them. For example, you want to store a yearly total after storing the quarterly quantities:

```
DTR> STORE X IN ACCOUNTS USING
CON> BEGIN
CON>     Q1 = *.Q1
CON>     Q2 = *.Q2
CON>     Q3 = *.Q3
CON>     Q4 = *.Q4
CON>     FY = X.Q1 + X.Q2 + X.Q3 + X.Q4
CON> END
Enter Q1: . . .
```

Examples

The following example stores two records in the FAMILIES domain:

```
DTR> READY FAMILIES WRITE
DTR> REPEAT 2 STORE FAMILIES
Enter FATHER: GEORGE
Enter MOTHER: SANDY
Enter NUMBER_KIDS: 2
Enter KID_NAME: DANA
Enter AGE: 12
Enter KID_NAME: DACIA
Enter AGE: 9

Enter FATHER: WAYNE
Enter MOTHER: SHEEL
Enter NUMBER_KIDS: 2
Enter KID_NAME: BETE
Enter AGE: 8
Enter KID_NAME: ALEX
Enter AGE: 5
DTR> FIND FAMILIES WITH MOTHER = "SANDY", "SHEEL"
[2 records found]
DTR> PRINT ALL

                      NUMBER    KID
  FATHER     MOTHER    KIDS     NAME    AGE

GEORGE     SANDY         2    DANA       12
                              DACIA       9
WAYNE      SHEEL         2    BETE        8
                              ALEX        5

DTR>
```

The following example stores a record in the YACHTS domain, using a context variable and a VERIFY clause:

```
DTR> SHOW HINKLEY_STORE
PROCEDURE HINKLEY_STORE
STORE A IN YACHTS USING
BEGIN
     BUILDER = "HINKLEY"
     MODEL = "BERMUDA 40"
     RIG = "YAWL"
     LOA = 40
     DISP = 20000
     BEAM = 12
     PRICE = 82000
END VERIFY USING
BEGIN
     PRINT A.BOAT, SKIP
     IF *.CONFIRMATION CONT "N" THEN
     PRINT SKIP THEN ABORT "BAD RECORD"
END
END_PROCEDURE

DTR> READY YACHTS WRITE
DTR> :HINKLEY_STORE

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 HINKLEY     BERMUDA 40 YAWL    40    20,000  12  $82,000

Enter CONFIRMATION: N

ABORT: BAD RECORD

DTR>
```

The following example defines a domain for single-digit integers and their squares, and uses a WHILE statement to control the number of records stored in the domain:

```
DTR> DEFINE DOMAIN SQUARES USING
DFN> SQUARES_REC ON SQUARES.DAT;
DTR> DEFINE RECORD SQUARES_REC USING
DFN> 01 SQUARES.
DFN>    03 NUMBER PIC 9.
DFN>    03 ITS_SQUARE PIC 99.
DFN> ;
[Record is 3 bytes long.]
DTR> DEFINE FILE FOR SQUARES;
DTR> READY SQUARES WRITE
DTR> DECLARE N PIC 99.
DTR> N = 0
DTR> WHILE N NE 10 STORE SQUARES USING
CON> BEGIN
CON>     NUMBER=N
CON>     ITS_SQUARE = N * N
CON>     N = N + 1
CON> END
DTR> FIND SQUARES
[10 records found]
DTR> PRINT ALL

        ITS
NUMBER SQUARE

  0      00
  1      01
  2      04
  3      09
  4      16
  5      25
  6      36
  7      49
  8      64
  9      81

DTR>
```

The following example stores data from the WORKER domain into the expanded NEW_WORKER domain. These domains have duplicate elementary field names that are subordinate to different group field names. The example uses a FOR statement to control the STORE statement so that DATATRIEVE stores the data from the elementary fields correctly. The record definitions and the STORE statement are as follows:

STORE Statement

```
DTR> SHOW WORK_REC
RECORD WORK_REC USING
01 WORK.
    03 LOCAL.
         05 CITY   PIC X(10).
         05 STATE  PIC X(2).
    03 REMOTE.
         05 CITY   PIC X(10).
         05 STATE  PIC X(2).
;
DTR> SHOW NEW_WORK_REC
RECORD NEW_WORK_REC USING
01 WORK.
    03 NEW_LOCAL.
         05 CITY   PIC X(10).
         05 STATE  PIC X(2).
    03 NEW_REMOTE.
         05 CITY   PIC X(10).
         05 STATE  PIC X(2).
    03 NAME.
         05 FIRST PIC X(10).
         05 LAST  PIC X(15).
;
DTR> READY NEW_WORKER WRITE
DTR> FOR WORKER
CON>   STORE NEW_WORKER USING
CON>     BEGIN
CON>       NEW_LOCAL = LOCAL
CON>       NEW_REMOTE = REMOTE
CON>     END
DTR>
```

STORE Statement (for VAX DBMS and Relational Sources)

Adds a new record to the database.

Format

```
STORE { rdb-domain-name    }
      { rdb-relation-name  }
      { dbms-domain-name   }
      { dbms-record-name   }

    [USING   statement-1]

    [VERIFY   [USING]   statement-2]

    [ [CURRENCY]   [context-name.]set-name-1   [,...] ]
```

Arguments

rdb-domain-name
Is the given name of the domain to contain the new relation.

rdb-relation-name
Is the name assigned to the relation when the database was created. The relation name can be the name of a view relation.

dbms-domain-name
Is the given name of the domain to contain the new record.

dbms-record-name
Is the name assigned to the record when the database was created and to the record which will be stored.

USING statement-1
Specifies a DATATRIEVE statement that can store one value for one or more fields in the new record.

VERIFY [USING] statement-2
Specifies a statement DATATRIEVE executes just before storing the new record.

CURRENCY
Is a keyword that is used only with VAX DBMS records or domains.

context-name
Is the name of a valid context variable or the name of a collection with a selected VAX DBMS record. It must identify a record that participates in the specified set. If the SYSTEM owns the set, you do not need to establish a context for the set. If the set is not owned by the SYSTEM and the context name is not present, DATATRIEVE uses the most recent single record context of a domain with a record type that participates in the specified set type.

set-name
Is the name of a VAX DBMS set. The record being stored must be an AUTOMATIC member of that set. Use the CONNECT statement to connect the record to MANUAL insertion sets.

It is not necessary to put SYSTEM-owned sets in the currency list. For all other AUTOMATIC sets, you must establish currency, but it is not necessary to put them in the currency list.

Usage Notes

- When storing records in relations that are dependent on other relations because of a constraint definition, ready all the involved relations.
- If you use the optional CURRENCY clause, you must list one or more set names. For each set name, you can optionally specify a context name. You can put multiple set names within the CURRENCY list, but you can have only one CURRENCY clause in a STORE statement.

Examples

The following example stores a new record in the relation JOB_HISTORY and makes the change permanent by entering a COMMIT statement:

```
DTR> READY PERSONNEL USING JOB_HISTORY WRITE
DTR> STORE JOB_HISTORY
Enter DEPARTMENT_CODE: HENG
Enter EMPLOYEE_ID: 78645
Enter JOB_CODE: B-78
Enter JOB_END: 021783
Enter JOB_START: 052181
DTR> COMMIT
```

The following example adds a new COMPONENT record to the database. A valid context must be established for current records in the sets PART_USED_ON and PART_USES to store a COMPONENT. The STORE_COMP procedure prompts for values so that DATATRIEVE knows which record you want to be current. The nested FOR loop establishes the context, and the CURRENCY clause translates DATATRIEVE contexts into VAX DBMS currencies. After you

enter your response to the last prompt for the values in a record, VAX DBMS automatically inserts the record into the specified set occurrences:

```
DTR> DEFINE PROCEDURE STORE_COMP
DFN> DECLARE USED_ON PIC X(8).
DFN> DECLARE SUB_PART PIC X(8).
DFN> SUB_PART = *."I.D. number of the component part"
DFN> USED_ON = *."I.D. number of the part it is used on"
DFN> FOR A IN PART WITH PART_ID = USED_ON
DFN>     FOR B IN PART WITH PART_ID = SUB_PART
DFN>         STORE COMPONENTS USING

DFN>             BEGIN
DFN>                 COMP_SUB_PART = SUB_PART
DFN>                 COMP_OWNER_PART = USED_ON
DFN>                 COMP_MEASURE = *."component measure"
DFN>                 COMP_QUANTITY = *."quantity"
DFN>             END CURRENCY A.PART_USED_ON , B.PART_USES
DFN> END_PROCEDURE
DTR>
```

SUM Statement

Provides a summary of totals for one or more numeric fields in the current collection. The summary is sorted according to the values in one or more fields of the current collection. The summary includes subtotals for control groups. The summary can be written to a file or an output device.

Format

SUM print-list BY sort-list [ON { file-spec | *.prompt }]

Arguments

print-list

Is a list of one or more numeric fields, other value expressions, and modifiers. The print list has the following format:

```
{value-expression  [ {modifier} [...] ] }  [,...]
```

The section on the PRINT statement describes the value expressions and modifiers you can use in a print list.

sort-list

Is a list of one or more sort keys that determine the order in which DATATRIEVE presents the summary totals. An item in the sort list consists of the name of a field whose contents form the basis for the sort, preceded or followed by a keyword that determines the order DATATRIEVE uses to sort the data.

To specify the sort order for each sort key, use one of the following keywords:

```
ASC[ENDING]

DESC[ENDING]

INCREASING

DECREASING
```

If you specify more than one sort key, use a comma to separate each sort key from the next.

file-spec

Is the file specification to which you want to write the output of the statement.

A complete file specification has the following format:

```
node-spec::device:[directory]file-name.type;version
```

***.prompt-name**
Is the prompting value expression that prompts you for the file specification to which you want to write the output of the statement.

Restrictions

- You must have established a current collection before issuing this statement.
- You must include only numeric fields in the print list.
- You cannot use table translation values as sort keys in a SUM statement. You can declare a variable using the COMPUTED BY clause and a translation value and then use the variable as the sort key.
- When you use ON LP: to send SUM statement output directly to a line printer, DATATRIEVE assumes your system has a line printer. If your system does not have a device defined as LPA0:, the clause ON LP: will not work.

  Although this restriction applies to any system without a line printer, you may encounter it unexpectedly if your system is part of a VAXcluster with a common line printer. The ON LP: clause does not work in a VAXcluster that uses a common printer not directly connected to your system.

  If the nodes in the cluster are connected with DECnet, you can work around this restriction. To send output from a node without a line printer, you must include the node name of the system with the line printer in the LP: specification. For example, if the cluster's line printer is on a node named BIGVAX, the following SUM statement sends output to it:

```
DTR> SUM 1 ("NUMBER"/"OF YACHTS") USING 9,
CON>    PRICE USING $$$$,$$$ BY BUILDER ON BIGVAX::LP
```

  Note that you cannot directly specify the line printer on BIGVAX by using the cluster device-name BIGVAX$LPA0: in the ON clause.

Results

- DATATRIEVE creates a record stream based on the current collection. It sorts the records of the record stream according to the specifications in the sort list. It displays the summary or writes it to the device or file you specify in the ON clause.

  The summary consists of totals for the fields you specify in the print list. For each control group created by the sort list, there is a total for each field in the print list. At the end of the summary is a total for each value expression in the print list of the values for all the records in the record stream.

- The order and format of the data in the report depends on the arguments you select.
- If you omit the ON clause, DATATRIEVE displays the output.
- If you omit a field in the file specification, DATATRIEVE uses the defaults specified in Table 4–34.

Table 4–34 Output File Specification Defaults

Field	Default
node-spec::	Your local node
device:	Your default device
[directory]	Your default directory
file-name	Null string
.type	.LIS
;version	1 or next higher version number

The minimum file specification consists of a period (.). The specification of such a file stored in your default VMS directory ends with ".;n", where n is the version number and both the file name and the type are null strings.

Usage Note

Use edit strings to format the output of the totals for items in the print list.

Example

The following example forms a collection of yachts and uses the SUM statement to summarize the prices of yachts in the collection and to display the number of yachts built by each builder. Edit strings are used to format the values:

```
DTR> READY YACHTS; FIND FIRST 6 YACHTS
[6 records found]
DTR> SUM 1 ("NUMBER"/"OF YACHTS") USING 9,
CON>    PRICE USING $$$$,$$$ BY BUILDER

              NUMBER             NUMBER
MANUFACTURER OF YACHTS  PRICE   OF YACHTS  PRICE

 ALBERG          1      $36,951
 ALBIN           3      $64,000
 AMERICAN        2      $28,790
                                    6     $129,741

DTR>
```

SYNCHRONIZED Clause

Causes word boundary alignment of an elementary field.

Format

```
{ SYNCHRONIZED } [ LEFT  ]
{ SYNC        } [ RIGHT ]
```

Arguments

LEFT

Allows compatibility with COBOL record definitions. DATATRIEVE accepts the argument but ignores the boundary alignment specification that the field begin at the left boundary of the unit of storage the field occupies.

RIGHT

Allows compatibility with COBOL record definitions. DATATRIEVE accepts the arguments but ignores the boundary alignment specification that the field end at the right boundary of the unit of storage the field occupies.

Restriction

The SYNC clause forces word boundary alignment on elementary fields only if the allocation for the record is MAJOR_MINOR.

Results

- A SYNC clause can force word boundary alignment on an elementary field and thus increase the amount of memory needed to store the value. For compatibility with COBOL record definitions, DATATRIEVE accepts the optional arguments LEFT and RIGHT but ignores the distinction between the two types of word boundary alignment.
- The filler bytes created by word boundary alignment are added to the length of any group fields to which the elementary field belongs.

Example

This example shows the difference in storage allocation a SYNC clause can make. A field of USAGE type LONG occupies four bytes of storage (the equivalent of one longword):

```
DTR> DEFINE RECORD NOSYNC_REC USING
DFN> 01 TOP.
DFN>    03 ONE PIC X.
DFN>    03 TWO LONG.
DFN> ;
[Record is 5 bytes long.]
DTR> DEFINE RECORD SYNC_REC USING
DFN> 01 TOP.
DFN>    03 ONE PIC X.
DFN>    03 TWO SYNC RIGHT LONG.
DFN> ;
[Record is 8 bytes long.]
DTR>
```

THEN Statement

Joins two or more DATATRIEVE statements into a compound statement.

Format

statement-1 {THEN statement-2} [...]

Argument

statement
Is a DATATRIEVE statement.

Restrictions

- You must observe all restrictions on the statements included in the compound statement. Restrictions are listed in the descriptions of each statement.
- You cannot use DATATRIEVE commands in a compound statement.
- A procedure you invoke in a compound statement cannot contain DATATRIEVE commands.
- Do not include FIND and SELECT in the same compound statement.
- Do not include FIND and SORT in the same compound statement.
- Do not include SELECT and DROP in the same compound statement.

Results

- DATATRIEVE executes the individual statements of the compound statement in the order you enter them.
- If any statement in the compound statement contains a syntax error, DATATRIEVE does not execute any of the statements in the compound.

Usage Notes

- You can use a compound statement anywhere you can use a single statement.
- Use THEN statements rather than BEGIN-END blocks to form short compound statements.
- If you use any statement that creates a context in the compound statement, do not use any other statement in that compound that refers to or depends on that context information.

Example

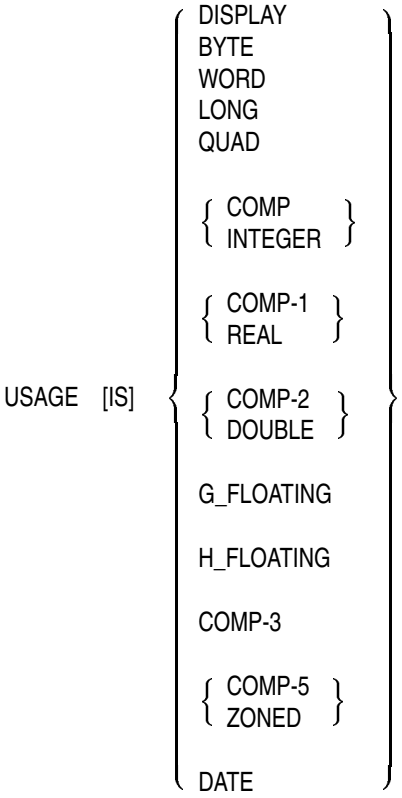
The following example uses a THEN statement to join a PRINT statement and a MODIFY statement in a FOR loop:

```
DTR> SET NO PROMPT
DTR> READY YACHTS MODIFY
DTR> FOR YACHTS WITH BUILDER EQ "ALBIN"
CON>     PRINT THEN MODIFY

                              LENGTH
                               OVER
MANUFACTURER   MODEL     RIG   ALL   WEIGHT BEAM  PRICE

 ALBIN       79         SLOOP  26     4,200  10  $17,900
Enter MANUFACTURER: CTRL/Z
Execution terminated by operator

DTR>
```

USAGE Clause

Specifies the internal format of a numeric field or specifies a date field.

Format



Arguments

DISPLAY

Indicates that each digit occupies one byte of storage. DISPLAY is the default if you do not include a USAGE clause.

BYTE

Indicates that field value is stored in binary format and that the value is stored in one byte of storage.

WORD
Indicates that field value is stored in binary format and that the value is stored in one word (two bytes) of storage.

LONG
Indicates that field value is stored in binary format and that the value is stored in one longword (four bytes) of storage.

QUAD
Indicates that field value is stored in binary format and that the value is stored in one quadword (eight bytes) of storage.

COMP
INTEGER
Indicates that the field value is stored in binary format. INTEGER is a synonym for COMP.

COMP-1
REAL
Indicates that the field value is stored in single-precision real format. REAL is a synonym for COMP-1.

COMP-2
DOUBLE
Indicates that the field value is stored in double-precision real format. DOUBLE is a synonym for COMP-2.

G_FLOATING
Indicates that a field is a floating point number with precision to approximately 15 decimal digits.

H_FLOATING
Indicates that a field is a floating point number with precision to 33 decimal digits.

COMP-3
Indicates that the field value is stored in packed-decimal format.

COMP-5
ZONED
Indicates that the field value is stored in signed decimal format. ZONED is a synonym for COMP-5.

DATE
Indicates that the field is a date field.

Restrictions

- This clause is valid for elementary numeric or date fields only.
- A field definition cannot contain both a USAGE clause and a SIGN clause.

Results

- The internal storage format of a numeric field is determined by the USAGE clause specified. If a numeric field definition does not have a USAGE clause, each digit in the field value occupies one character position in the record.
- If USAGE IS DATE is specified in a field definition clause, the field is identified as a DATATRIEVE date field.

Usage Notes

- Use the appropriate form of the USAGE clause when a program written in COBOL, BASIC-PLUS-2, or other language uses the record and requires a different internal format.
- The USAGE clause can cause DATATRIEVE to align fields on hardware storage boundaries. See the section in this chapter on the ALLOCATION clause.
- In order to avoid the inaccuracies of floating point comparisons, when possible you should declare fields or variables USAGE INTEGER or QUAD rather than USAGE REAL, DOUBLE, G_FLOATING, or H_FLOATING.
- A COMP (or INTEGER) field stores its value in binary format. The size of a COMP (or INTEGER) field depends on the number of digit positions specified in its PICTURE clause. You can avoid having both a USAGE IS COMP clause and a PICTURE clause by using the keywords WORD, LONG, and QUAD to specify the three types of storage allocation available with COMP. Table 4–35 shows the COMP storage allocation types.

Table 4–35 COMP Storage Allocation Types

PIC Clause	Size of Field	Alternate USAGE Type
9(1) to 9(4)	2 bytes	WORD
9(5) to 9(9)	4 bytes	LONG
9(10) to 9(18)	8 bytes	QUAD

- VAX DATATRIEVE has always treated fields of the COMP data type as SIGNED. DATATRIEVE does not support the UNSIGNED data types during data definition.

  For example, a record containing a COMP field with an UNSIGNED data type might indicate that the record was originally defined using the CDDL utility of CDD/Repository. If such a record were later edited through DATATRIEVE, the data type of the COMP field would automatically be converted to SIGNED.

  This is the expected behavior of DATATRIEVE.

  To ensure that a COMP field is stored with the proper sign, you should edit or redefine the field using only the utility with which the definition was originally created.

COMP-1 (or REAL) Fields

A COMP-1 (or REAL) field stores its value in single-precision real (floating-point) format. COMP-1 fields are four bytes long.

Example

The following example defines the field SALE_PRICE as a REAL (COMP-1) field:

```
05 SALE_PRICE PIC 9(5)
   USAGE REAL
   EDIT_STRING IS $(6).
```

COMP-2 (or DOUBLE) Fields

A COMP-2 (or DOUBLE) field stores its value in double-precision real (floating-point) format. COMP-2 fields are eight bytes long.

G_FLOATING Fields

A G_FLOATING field is an extended range 64-bit floating-point number with precision to approximately 15 decimal digits.

H_FLOATING Fields

An H_FLOATING field is an extended range 128-bit floating-point number with precision to approximately 33 decimal digits.

COMP-3 Fields

A COMP-3 (packed) field stores its value in packed-decimal format. The value is stored two digits per byte. The value of a COMP-3 field must contain a sign. The sign occupies the four low-ordered bits in the rightmost byte. The size of the field depends on the number of digit positions specified by the field's PICTURE clause:

$$size\ (in\ bytes)\ =\ \frac{(digit - positions + 1)}{2}$$

For example, a field with three digit positions is two bytes long. If the field contains an even number of digits, the size is rounded up. Thus, a 6-digit field is stored in four bytes.

COMP-5 (or ZONED) Fields

A COMP-5 (or ZONED) field stores its value in signed decimal format. A value in a COMP-5 field is stored one digit per byte. Therefore, the size of a COMP-5 field is the number of digit positions specified in its PICTURE clause.

The sign of a COMP-5 value shares the rightmost byte with the lowest-valued digit of the value. The lowercase letters P through Y represent a negative sign for the values 0 through 9.

DATE Fields

A DATATRIEVE date field stores a date as an 8-byte binary value. Other languages may not interpret the date field correctly. The date is expressed as the number of clunks (100-nanosecond units) since the base date of 00:00:00 AM on November 17, 1858.

When you print a date field, DATATRIEVE translates the number of clunks to the format you specify in the EDIT_STRING clause (or to the default format if no EDIT_STRING clause is included in the field definition). The default format is DD-MMM-YYYY.

When you enter a date value, DATATRIEVE translates the input value to clunks before storing it.

The default edit string for date fields is 11 characters. When you concatenate two fields defined as USAGE IS DATE, DATATRIEVE converts the date values to 23-character string literals. This results in longer concatenated strings than you may have expected. To force the use of the default when concatenating date fields, use a FORMAT value expression or specify an edit string for the concatenated fields in the PRINT statement.

Example

The following example defines the field SALE_DATE as a date field, to be printed in the default format for date fields:

```
06 SALE_DATE USAGE IS DATE.
```

VALID IF Clause

Validates a field value before it is stored in the record.

Format

VALID IF boolean-expression

Argument

boolean-expression
Is a DATATRIEVE Boolean expression.

Restrictions

- A field definition cannot contain both a VALID IF and a COMPUTED BY clause.
- The Boolean expression of a VALID IF clause cannot refer to a list field from the same domain.
- When you assign a value to a variable or a field that contains a VALID IF clause, the variable or field can end up containing an invalid value under the following conditions:
  - If you prompt for the value, enter an invalid value, and then stop the statement execution with CTRL/Z, the invalid value may be stored in the variable.
  - If you assign a value directly to a variable and the value is invalid, DATATRIEVE issues an invalid value message but still assigns the invalid value.

Result

When a value is entered for the field with a MODIFY or STORE statement, DATATRIEVE evaluates the Boolean expression. If the Boolean expression is true, DATATRIEVE stores the value in the field. If it is false, DATATRIEVE prints an error message and reprompts for the field value.

Usage Note

When a VALID IF clause in one field refers to a list field in another domain, you must ready the domain with the list field before you ready the domain with the VALID IF clause. You must finish the domains in the reverse order: the domain with the VALID IF clause first and the domain with the list field last.

Examples

The following example compares the value entered for the RIG field to the character strings SLOOP, KETCH, MS, and YAWL and stores the value in the field if it is one of those character strings:

```
06 RIG PIC X(6)
       VALID IF RIG EQ "SLOOP", "KETCH", "MS", "YAWL".
```

The following example stores a value in the LOA field if it is between 15 and 50:

```
06 LENGTH_OVER_ALL PIC XXX
        VALID IF LOA BETWEEN 15 AND 50
        QUERY_NAME IS LOA.
```

The following example stores a value in the PRICE field if it is greater than 1.3 times the displacement or if it is zero:

```
06 PRICE PIC 99999
       VALID IF PRICE>DISP*1.3 OR PRICE EQ 0
       EDIT_STRING IS $$$,$$$.
```

WHILE Statement

Causes DATATRIEVE to repeat a statement as long as the condition specified in the Boolean expression is true.

Format

WHILE boolean-expression statement

Arguments

boolean-expression
Is a Boolean expression. (See Chapter 1). In the WHILE statement, Boolean expressions are limited to the following format: { variable-name / *.prompt } boolean-operator value-expression

statement
Is a simple or compound statement you want DATATRIEVE to execute if the Boolean expression evaluates to true.

Restrictions

- The restrictions for particular statements are documented in this chapter in the section for each statement. You must observe these restrictions for any statements you include in the WHILE statement.
- You cannot use a field name as a value expression on the left side of the Boolean expression of a WHILE statement.

Result

DATATRIEVE repeats the specified statement as long as the Boolean example evaluates to true. If the relationship between the two values in the Boolean never changes (for example, if 2 GT 1), the loop repeats infinitely until you end it with a CTRL/C or CTRL/Z.

Usage Notes

- Use the WHILE statement to form and control the execution of loops.
- You can use a prompting value expression or a variable as the first member of the Boolean expression.

Example

The following example groups the boats with LOA less than 35 according to the value of BEAM and displays the TYPE, LOA, and BEAM of the shortest boat from each group of boats with the same value for BEAM:

```
DTR> SHOW WHILE_EX
PROCEDURE WHILE_EX
BEGIN
DECLARE X PIC 99.
X = 0
FOR YACHTS WITH LOA < 35 AND
    BEAM NE 0 SORTED BY BEAM, LOA
    WHILE X < BEAM
    BEGIN
        PRINT TYPE, LOA, BEAM, X
        X = BEAM
    END
END
END_PROCEDURE

DTR> :WHILE_EX

                         LENGTH
                          OVER
MANUFACTURER   MODEL      ALL    BEAM X

CAPE DORY    TYPHOON      19      06  00
WINDPOWER    IMPULSE      16      07  06
ERICSON      23/ SPECIA   23      08  07
EASTWARD     HO           24      09  08
ALBIN        79           26      10  09
BOMBAY       CLIPPER      31      11  10
IRWIN        25           25      12  11

DTR>
```

WITH_FORM Statement

Sends and receives data to and from a DECforms form. Control text items can be also sent and received.

Format

```
WITH_FORM  form-name  IN  file-name
  SEND FROM datatrieve-source [USING exchange-record] [,]
                    .           .           .
                    .           .           .
                    .           .           .
     TO decforms-name
  RECEIVE FROM decforms-name
     TO datatrieve-destination [USING exchange-record] [,]
                    .           .           .
                    .           .           .
                    .           .           .
  [SEND_CONTROL_TEXT send-control-item [,...]]
  [RECEIVE_CONTROL_TEXT receive-control-item [,...]]
```

Arguments

form-name
Is the name of the DECforms form in the form file. The form name is syntactically required but ignored when the file name is a .FORM file.

file-name
Is the DECforms file name. It can either be a .FORM or a .EXE file (the default file extension is .EXE).

decforms-name
Is a DECforms record name or list name. A list is a DECforms structure. If a list name is specified as decforms-name, all the equivalent datatrieve-sources or datatrieve-destinations must be specified.

datatrieve-source/datatrieve-destination
Is the name of a DATATRIEVE data element; a data element can be represented by a DATATRIEVE record, by a group field, by a field, or by a variable. When datatrieve-source and datatrieve-destination can be used interchangeably, referred to as "DATATRIEVE data element". The DATATRIEVE data element can not be a COMPUTED BY field (for more information see the chapter on DECforms in the *VAX DATATRIEVE Guide to Interfaces*). When the decforms-name is a list name, specify a list of DATATRIEVE data elements separated by commas.

exchange-record
Is the CDD/Repository path name of a record used to send and receive data with DECforms. It is an optional argument of the datatrieve-source and datatrieve-destination structures. For more information on the exchange record see the chapter on DECforms in the *VAX DATATRIEVE Guide to Interfaces*.

send-control-item
Is the input parameter (variable or string literal) that activates the proper actions defined by the user inside the .IFDL file. If you specify more than one input parameter, separate them with commas. Every control-item has to be from one to five characters long. For more details on how to build and use a SEND_ CONTROL_TEXT see the DECforms documentation.

receive-control-item
Is a DATATRIEVE variable previously declared which receives a text control item returned by the form. The content of the variable will be loaded by the form depending on the rules established by the user in the .IFDL file. If you specify more than one output parameter, separate them with commas. Every receive-control-item should be five characters long. For more details on how to build and use a RECEIVE_CONTROL_TEXT see the DECforms documentation.

Restrictions

- The SEND and RECEIVE clauses are both optional, but either one or the other has to be present in the WITH_FORM statement.
- When both SEND and RECEIVE clauses appear, a DECforms TRANSCEIVE operation is performed. If you want two distinct SEND and RECEIVE operations, you must use two distinct WITH_FORM statements, each with the appropriate clause.
- If you press RETURN after the SEND clause, DATATRIEVE will not allow you to enter the RECEIVE clause in the WITH_FORM statement, because it considers the statement as being completed.

- The records listed in the SEND and RECEIVE clauses and the records declared inside the form must have the same structure. They should also be based on the same dictionary definition. It is not required that they have the same name. If an exchange record is specified, the form record must have the same structure as the exchange record.
- 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 that you want to send or receive.

Usage Notes

- It is not mandatory to define all the record fields of the DATATRIEVE data element in the related exchange record structure. It is possible to define, also with different data types, only the fields needed by the form. See the *VAX DATATRIEVE Guide to Interfaces* for more information on the use of the exchange record.
- When executing a WITH_FORM statement, DATATRIEVE does not apply any data conversion on the data sent to or received from DECforms. Only if there is an exchange record linked to a record does DATATRIEVE perform the conversions between the two structures. All the other data conversions must be done within DECforms.

Example

In the following example the WITH_FORM statement causes DATATRIEVE to send records contained in the YACHTS domain to the form record named BOAT. The form responds to the send operation by displaying the form record data (one record at a time) on the screen.

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

A

VAX DATATRIEVE Keywords

This appendix lists the VAX DATATRIEVE keywords. You should not use these names when you are naming items.

A.1 DATATRIEVE Keywords

The following list includes all VAX DATATRIEVE keywords.

*#(asterisk)
@#(at sign)
:#(colon)
,#(comma)
**#(double asterisk)
"#(double quotation mark)
=#(equal sign)
!#(exclamation point)
>#(greater than sign)
-#(hyphen or minus sign)
(#(left parenthesis)
<#(less than sign)
.#(period)
+#(plus sign)
?#(question mark)
)#(right parenthesis)
;#(semicolon)
/#(slash)
_#(underscore)
| #(vertical bar)
ABORT
ADT
ADVANCED
AFTER
ALIGNED_MAJOR_MINOR
ALIN_MAJ_MIN
ALL

VAX DATATRIEVE Keywords
A.1 DATATRIEVE Keywords

ALLOCATION
AND
ANY
APPLICATION_KEYPAD
ARGUMENTS
AS
ASC
ASCENDING
AT
ATT
AVERAGE
BANISH
BATCH
BEFORE
BEGIN
BETWEEN
BLANK
BOOLEAN
BOTTOM
BT
BUT
BY
BYTE
CDO
CHANGE
CHARACTER
CHARACTERS
CHOICE
CLOSE
COL
COLLECTIONS
COLUMN
COLUMN_HEADER
COLUMNS_PAGE
COMMIT
COMP
COMP_1
COMP_2
COMP_3
COMP_5
COMP_6
COMPUTED
CONCURRENCY

CONNECT
CONSISTENCY
CONT
CONTAINING
COUNT
CROSS
CURRENCY
CURRENT
DATABASE
DATABASES
DATA TYPE
DATE
DEBUG
DECIMAL
DECLARE
DECLARE_ATT
DECREASING
DEFAULT
DEFINE
DEFINEP
DELETE
DELETEP
DENY
DEPENDING
DESC
DESCENDING
D_FLOATING
DICTIONARY
DICTIONARIES
DIGIT
DIGITS
DISCONNECT
DISPLAY
DISPLAY_FORM
DO
DOMAIN
DOMAINS
DOUBLE
DROP
DUP
EDIT
EDIT_BACKUP
EDIT_STRING

ELSE
END
END_CHOICE
END_PLOT
END_PROCEDURE
END_REPORT
END_TABLE
ENDING
ENTRY
EQ
EQUAL
ERASE
EXCLUSIVE
EXECUTE
EXIT
EXTEND
EXTRACT
F_FLOATING
FIELD
FIELDS
FILE
FILL
FILLER
FIND
FINISH
FIRST
FOR
FORM
FORMAT
FORMS
FROM
GE
GET_FORM
G_FLOATING
GRANT
GREATER_EQUAL
GREATER_THAN
GROUP
GT
GUIDE
HELP
HELP_LINES
HELP_PROMPT

HELP_WINDOW
H_FLOATING
IF
IN
INCR
INCREASING
INIT_VECTOR
INSERT
INTEGER
IS
JUST
JUSTIFIED
JUSTIFY
KEEP
KEY
KEYDEFS
KEYWORD
LAST
LE
LEADING
LEAVE
LEFT
LEFT_RIGHT
LESS_EQUAL
LESS_THAN
LINES_PAGE
LIST
LOCAL
LOCK_WAIT
LONG
LONGWORD
LT
MAJOR_MINOR
MATCH
MAX
MAX_LINES
MAX_PAGES
MEMBER
MIN
MISSING
MODIFY
NE
NETWORK

NEW_PAGE
NEW_SECTION
NEXT
NO
NONE
NONLOCAL
NOT
NOT_EQUAL
NOVERIFY
NUMBER
NUMERIC
OCCURS
OCTA
OCTAWORD
OF
ON
OPEN
OPTIMIZE
OR
OTHERWISE
OVER
OVERPUNCHED
OWNER
PACKED
PAGE
PATH
PIC
PICTURE
PLOT
PLOTS
PORT
PRINT
PRIOR
PRIVILEGES
PROCEDURE
PROCEDURES
PROMPT
PROTECTED
PURGE
PUT_FORM
PW
QUAD
QUADWORD

QUERY_HEADER
QUERY_NAME
READ
READY
REAL
RECEIVE
RECONNECT
RECORD
RECORDS
RECOVER
REDEFINE
REDEFINES
REDUCE
REDUCED
RELATIONSHIPS
RELEASE
REPEAT
REPORT
REPORT_HEADER
REPORT_NAME
RETRIEVE
RIGHT
ROLLBACK
RSE
RUNNING
SCALE
SCHEMA
SCHEMAS
SEARCH
SELECT
SEMICOLON
SEND
SEPARATE
SET
SETS
SET_UP
SHARED
SHOW
SHOWP
SIGN
SIGNED
SIZE
SKIP

SNAPSHOT
SORT
SORTED
SOURCE
SPACE
STARTING
STD_DEV
STORE
STRING
STRUCTURE
SUBSCHEMA
SUM
SUPERCEDE
SUPERSEDE
SYNC
SYNCHRONIZED
SYNONYM
SYNONYMS
TAB
TABLE
TABLES
TERMINAL
TEXT
THE
THEN
TIMES
TO
TOP
TOTAL
TRAILING
UIC
UNSIGNED
USAGE
USER
USING
VALID
VALUE
VARIABLES
VARYING
VECTOR
VERIFY
VIA
WHEN

WHILE
WITH
WITH_FORM
WITHIN
WORD
WRITE
ZERO
ZONED

B

Access Privileges Tables

The access privileges available to users of DATATRIEVE are listed in the following tables. Table B–1 gives a brief description of the privileges available for use in DMU format dictionaries. Table B–2 lists privileges available for use in CDO format dictionaries. Table B–3 cross-references the DMU and CDO privileges.

Table B–1 Access Privileges for DMU Format Dictionaries

Privilege	Allows you to:
C (CONTROL)	Read, create, modify, and delete ACL entries. You cannot deny yourself CONTROL privilege.
D (LOCAL_DELETE)	Delete dictionary objects and directories and subdictionaries with no children. Also lets you edit, replace, or recompile definitions stored in the dictionary.
E (DTR_EXTEND/EXECUTE)	Ready a domain for EXTEND access, access a table, or invoke a procedure.
F (FORWARD)	Create subdictionaries.
G (GLOBAL_DELETE)	Delete dictionary directories and subdictionaries, including any children they may have, with a single command.
H (HISTORY)	Add entries to CDD/Repository history lists with the Dictionary Management Utility (DMU).
M (DTR_MODIFY)	Ready a domain for READ and MODIFY access.
P (PASS_THRU)	Use a dictionary directory, subdictionary, or object in a path name. You cannot deny yourself PASS_THRU privilege.

(continued on next page)

Access Privileges Tables

Table B–1 (Cont.) Access Privileges for DMU Format Dictionaries

Privilege	Allows you to:
R (DTR_READ)	Ready a domain for READ access, display dictionary definitions with a SHOW command, use the EDIT command, and copy them into a command file with an EXTRACT command.
S (SEE)	See the definition of a dictionary object. SEE access to a domain definition and its associated record definition is necessary to define a data file and then to ready the domain.
U (UPDATE)	Update the definition of a dictionary object.
W (DTR_WRITE)	Ready a domain for WRITE access.
X (EXTEND)	Create children of dictionary directories and subdictionaries.

Table B–2 Access Privileges for CDO Format Dictionaries

Privilege	Allows you to:
ADMINISTRATOR	Not used.
CHANGE	Change the definition of a dictionary object. Define new objects or delete objects in the dictionary. [1]
CONTROL	Create, modify, and delete ACL entries.
DELETE	Delete and purge dictionary objects. Delete empty dictionary directories.
DEFINE	Create new definitions and new versions of definitions. [1]
EXTEND	Ready a domain for EXTEND access. [2]
MODIFY	Ready a domain for READ and MODIFY access.
OPERATOR	Not used.

[1]DATATRIEVE does not recognize either the CDO CHANGE privilege or the CDO DEFINE privilege when used alone; they are only recognized when used together. The DATATRIEVE UPDATE privilege identifies this combination of CHANGE and DEFINE.

[2]DATATRIEVE renames the CDO privilege ERASE to be EXTEND. If you acquire the EXTEND privilege through DATATRIEVE or acquire the ERASE privilege through CDO, DATATRIEVE will include EXTEND in your list of privileges. However, CDO does not convert the name, so if you look at the same privileges through CDO, you will see ERASE listed.

(continued on next page)

Table B–2 (Cont.) Access Privileges for CDO Format Dictionaries

Privilege	Allows you to:
READ	Ready a domain for READ access, display dictionary definitions with a SHOW command, use the EDIT command, and copy them into a command file with an EXTRACT command.
SHOW	On an object, show the definition of a dictionary object. SHOW access to a domain is necessary to define a data file and then ready the domain. On a dictionary, show the contents of the dictionary.
WRITE	Ready a domain for WRITE access.

Table B–3 DMU/CDO ACL Privilege Equivalents

DATATRIEVE Code	DMU Dictionary	CDO Dictionary
C	CONTROL	[NO]CONTROL
D	LOCAL_DELETE	[NO]DELETE
E	DTR_EXTEND/EXECUTE	EXTEND
F	FORWARD	*** no equivalence ***
G	GLOBAL_DELETE	*** no equivalence ***
H	HISTORY	*** no equivalence ***
M	DTR_MODIFY	[NO]MODIFY
P	PASS_THRU	*** no equivalence ***
R	DTR_READ	[NO]READ
S	SEE	[NO]SHOW
U	UPDATE	[NO]CHANGE+[NO]DEFINE
W	DTR_WRITE	[NO]WRITE
X	EXTEND	*** no equivalence ***

B.1 Access Privilege Requirements

Privileges affect the DATATRIEVE statements and commands you can use.

Table B–4 lists the statements and commands that require privileges, and the privileges you need to use them. Any statements not listed in the table require no DMU privileges except P (PASS_THRU) access to a default dictionary directory, and no CDO privileges except the VMS privileges needed to access the VMS

dictionary directory. If you do not have P (PASS_THRU) access to a DMU default dictionary directory nor VMS directory access to a CDO dictionary directory, you cannot use VAX DATATRIEVE.

Commands which are not available in the CDO format dictionary are identified as "DMU Only."

Table B–4 Access Privilege Requirements

Data Description Commands	DMU Privileges Needed	CDO Privileges Needed
DEFINE DICTIONARY DEFINE DOMAIN DEFINE PORT DEFINE RECORD	To Parent Directory P (PASS_THRU) X (EXTEND)	To Directory VMS Directory Access To Dictionary S (SHOW) U (CHANGE)
DEFINE PROCEDURE DEFINE TABLE	To Parent Directory P (PASS_THRU) X (EXTEND)	To Directory VMS Directory Access To Dictionary S (SHOW) U (CHANGE)
REDEFINE DOMAIN REDEFINE PORT REDEFINE RECORD	To Parent Directory P (PASS_THRU) X (EXTEND) To Dictionary and To Highest Existing Version S (SEE) P (PASS_THRU) U (UPDATE)	To Directory and To Highest Existing Version S (SHOW) U (CHANGE+DEFINE)
REDEFINE PROCEDURE REDEFINE TABLE	To Parent Directory P (PASS_THRU) X (EXTEND) To Highest Existing Version S (SEE) P (PASS_THRU) U (UPDATE)	To Directory and To Highest Existing Version S (SHOW) U (CHANGE+DEFINE)

(continued on next page)

Table B–4 (Cont.) Access Privilege Requirements

Data Description Commands	DMU Privileges Needed	CDO Privileges Needed
DEFINE FILE	To Domain Definition P (PASS_THRU) S (SEE) W (DTR_WRITE) R (DTR_READ) To Record Definition P (PASS_THRU) S (SEE) W (DTR_WRITE) R (DTR_READ) E (DTR_EXTEND/ EXECUTE)	To Dictionary S (SHOW) To Domain and Record Definitions S (SHOW) W (WRITE)
Data Protection Commands	**DMU Privileges Needed**	**CDO Privileges Needed**
DEFINEP DELETEP	To Dictionary Object or Directory P (PASS_THRU) C (CONTROL)	To Dictionary Containing the Object S (SHOW) U (CHANGE+DEFINE) To Dictionary Object or Dictionary S (SHOW) U (CHANGE+DEFINE) C (CONTROL)
SHOWP	To Dictionary Object or Directory P (PASS_THRU) C (CONTROL)	To Dictionary Object or Dictionary S (SHOW)

(continued on next page)

Table B–4 (Cont.) Access Privilege Requirements

Dictionary Manipulation Commands	DMU Privileges Needed	CDO Privileges Needed
SET DICTIONARY	To Dictionary Directory P (PASS_THRU)	To Dictionary Directory S (SHOW)
SHOW path-name	To Dictionary Object P (PASS_THRU) S (SEE) R (DTR_READ)	To Dictionary S (SHOW) To Dictionary Object S (SHOW) R (READ)
EDIT path-name (if SET NO EDIT_ BACKUP)	To Parent Directory P (PASS_THRU) X (EXTEND) To Dictionary Object P (PASS_THRU) S (SEE) R (DTR_READ) and either D (LOCAL_DELETE) or G (GLOBAL_DELETE)	To Dictionary S (SHOW) U (CHANGE+DEFINE) To Dictionary Object S (SHOW) R (READ) D (DELETE) U (CHANGE+DEFINE)
EDIT path-name (if SET EDIT_BACKUP)	To Parent Dictionary P (PASS_THRU) X (EXTEND) To Dictionary Object P (PASS_THRU) S (SEE) R (DTR_READ) U (UPDATE)	To Dictionary S (SHOW) U (CHANGE+DEFINE) To Dictionary Object S (SHOW) R (READ) U (CHANGE+DEFINE)
EXTRACT	To Dictionary Object P (PASS_THRU) S (SEE) R (DTR_READ)	To Dictionary S (SHOW) To Dictionary Object S (SHOW) R (READ)

(continued on next page)

Table B–4 (Cont.) Access Privilege Requirements

Dictionary Manipulation Commands	DMU Privileges Needed	CDO Privileges Needed
DELETE	To Parent Directory P (PASS_THRU) X (EXTEND) To Dictionary Object P (PASS_THRU) and either D (LOCAL_DELETE) or G (GLOBAL_DELETE)	To Dictionary S (SHOW) U (CHANGE+DEFINE) To Dictionary Object S (SHOW) D (DELETE)
PURGE	To Parent Directory P (PASS_THRU) X (EXTEND) To Dictionary Object P (PASS_THRU) S (SEE) R (DTR_READ) and either D (LOCAL_DELETE) or G (GLOBAL_DELETE)	To Dictionary S (SHOW) U (CHANGE+DEFINE) To Dictionary Object S (SHOW) D (DELETE)
Data Manipulation Commands and Statements	**DMU Privileges Needed**	**CDO Privileges Needed**
READY for All Modes	To Domain, Database, Relation, and VAX DBMS Record Definition P (PASS_THRU) S (SEE)	To Dictionary S (SHOW) To Domain, Database, Relation, and VAX DBMS Record Definition S (SHOW)
	To Record Definitions of RMS Domains E (DTR_EXTEND/ EXECUTE)	To Record Definitions Of RMS Domains E (EXTEND)
for SNAPSHOT Access	To Domain Definition R (DTR_READ)	To Domain Definition R (READ)

(continued on next page)

Table B–4 (Cont.) Access Privilege Requirements

Data Manipulation Commands and Statements	DMU Privileges Needed	CDO Privileges Needed
for READ Access	To Domain Definition R (DTR_READ) or W (DTR_WRITE) or M (DTR_MODIFY)	To Domain Definition R (READ) or W (WRITE) or M (MODIFY)
for WRITE Access	To Domain Definition W (DTR_WRITE)	To Domain Definition W (WRITE)
for MODIFY Access	To Domain Definition M (DTR_MODIFY) or W (DTR_WRITE)	To Domain Definition M (MODIFY) or W (WRITE)
for EXTEND Access	To Domain Definition E (DTR_EXTEND/ EXECUTE) or W (DTR_WRITE)	To Domain Definition E (EXTEND) or W (WRITE)

(continued on next page)

Table B–4 (Cont.) Access Privilege Requirements

Invocations	DMU Privileges Needed	CDO Privileges Needed
Procedure	To Procedure Definition P (PASS_THRU) S (SEE) E (DTR_EXTEND/ EXECUTE)	To Dictionary S (SHOW) To Procedure Definition S (SHOW)
Dictionary Table	To Table Definition P (PASS_THRU) S (SEE) E (DTR_EXTEND/ EXECUTE)	To Dictionary S (SHOW) To Table Definition S (SHOW)
Domain Table	To Table Definition P (PASS_THRU) S (SEE) E (DTR_EXTEND/ EXECUTE) To Domain Definition P (PASS_THRU) S (SEE) and either R (DTR_READ) or W (DTR_WRITE) or M (DTR_MODIFY) To Record Definition P (PASS_THRU) S (SEE) E (DTR_EXTEND/ EXECUTE)	To Dictionary S (SHOW) To Table Definition S (SHOW) To Domain Definition R (READ) To Record Definition E (EXTEND)

B.2 Default Access Control Lists

When you create a dictionary directory, subdictionary, or object, the default ACL entry grants privileges to your username. Table B–5 and Table B–6 summarize these privileges.

Table B–5 Default Access Control Lists for DMU Format Dictionaries

For Dictionary or Subdictionary Directories	For Dictionary Objects
C (CONTROL)	C (CONTROL)
D (LOCAL_DELETE)	D (LOCAL_DELETE)
H (HISTORY)	E (DTR_EXTEND/EXECUTE)
P (PASS_THRU)	H (HISTORY)
S (SEE)	M (DTR_MODIFY)
X (EXTEND)	R (DTR_READ)
	S (SEE)
	U (UPDATE)
	W (DTR_WRITE)

Table B–6 Default Access Control Lists for CDO Format Dictionaries

For Dictionaries	For Dictionary Objects
U (CHANGE+DEFINE)	U (CHANGE+DEFINE)
C (CONTROL)	C (CONTROL)
D (DELETE)	D (DELETE)
S (SHOW)	E (EXTEND)
	M (MODIFY)
	R (READ)
	S (SHOW)
	W (WRITE)

C

Logical Names for the DATATRIEVE Environment

Table C–1 lists and describes all the logical names you can define for your DATATRIEVE environment. These logical names must be defined at DCL level, before invoking the DATATRIEVE image. DATATRIEVE then translates the logical names at runtime.

Table C–1 Logical Name Assignments

Logical Name	Example
CDD$DEFAULT Default CDD/Repository directory	`$ DEFINE CDD$DEFAULT "CDD$TOP.DTR$LIB"`
DTR$CAPTIVE_ALLOWED Access to DCL and SPAWN commands by captive accounts	`$ DEFINE DTR$CAPTIVE_ALLOWED FN$DCL, FN$SPAWN`
DTR$COMMAND_LINES Number of command recall lines	`$ DEFINE DTR$COMMAND_LINES "25"`
DTR$DATE_INPUT Default date input interpretation; formats: “MDY” “DMY” “YDM” “YMD” “DYM” “MYD”	`$ DEFINE DTR$DATE_INPUT "MDY"`
DTR$EDIT EDT, VAXTPU, or LSE as editor (Note: Use one of the following 3-letter codes: EDT, TPU, LSE)	`$ DEFINE DTR$EDIT "TPU"`
DTR$KEYDEFS File of key definitions	`$ DEFINE DTR$KEYDEFS "DSK1:[ROY]KEYDEFS.COM"`
DTR$NOWINDOWS	

(continued on next page)

Table C–1 (Cont.) Logical Name Assignments

Logical Name	Example
Use of the DECwindows interface	`$ DEFINE DTR$NOWINDOWS TRUE`
DTR$PROMPT_LINES	
Number of prompt recall lines	`$ DEFINE DTR$PROMPT_LINES "30"`
DTR$READY_MODE	
Default access option	`$ DEFINE DTR$READY_MODE "SHARED"`
DTR$RW_BODY_ATTRIBUTES	
Default rendition attributes for the body of a report	`$ DEFINE DTR$RW_BODY_ATTRIBUTES "FAMILY=NC_SCHOOLBOOK, SIZE=10"`
DTR$RW_HEADER_ATTRIBUTES	
Default rendition attributes for the header of a report	`$ DEFINE DTR$RW_HEADER_ATTRIBUTES "FAMILY=COURIER, SIZE=24, ITALIC"`
DTR$RW_INITIAL_FF	
FF (Form Feed) at the beginning of TEXT format reports	`$ DEFINE DTR$RW_INITIAL_FF TRUE`
DTR$STACK_SIZE	
DATATRIEVE stack size	`$ DEFINE DTR$STACK_SIZE "200"`
DTR$STARTUP	
Startup command file	`$ DEFINE DTR$STARTUP "DTRSTART.COM"`
DTR$SYNONYM	
File containing keywords and their corresponding abbreviations	`$ DEFINE DTR$SYNONYM SYNONYMS_LIST.TXT`
DTRADT	
New or modified ADT text file	`$ DEFINE DTRADT "NEWADT.DAT"`
DTRHELP	
New or modified help library	`$ DEFINE DTRHELP "NEWDTRLIB.HLB"`
DTRMSGS	
New or modified message file	`$ DEFINE DTRMSGS "NEWMSGS.EXE"`

D

Edit String Characters

Table D–1 lists and describes all the edit string characters you can use in the EDIT_STRING clause of a field definition or in the USING clause of the PRINT statement.

The table heading "Character Type" indicates what type of field you can edit with the character. Do not use editing characters designated for alphabetic or alphanumeric on numeric fields (or vice versa). If you do, you can get unexpected results. Remember that field type is determined by the PIC or USAGE clause, not the value in the field. A field defined as PIC X(10) might contain only numbers, for example, but you should use only alphanumeric editing characters to format the way you want to display those numbers.

Table D–1 Edit String Characters

Character Type	Edit String Character	Description
Alphabetic Replacement	A	Each A is replaced by an alphabetic character from the field's content. An asterisk is placed in the position of each digit or nonalphabetic character in the field's content.
Alphanumeric Replacement	X	Each X is replaced by one character from the field's content.
	T	Indicates text. Each T reserves a column on a line for the associated print list element. For example, PRINT "1234567890" USING T(5) displays: 12345 in the first five columns of one line and 67890 in the first five columns of the next line. Edit strings containing a T cannot contain other characters.

(continued on next page)

Edit String Characters

Table D–1 (Cont.) Edit String Characters

Character Type	Edit String Character	Description
	A	Each A is replaced by an alphabetic character from the field's content. An asterisk is placed in the position of each digit or nonalphabetic character in the field's content.
Numeric Replacement	9	Each 9 is replaced by one digit from the field's content. Nondigit characters are ignored, and the digits are right justified in the output and the leading character positions (if any) are filled with zeros.
	Z	If a Z matches a leading zero in the field's content, it is replaced by a space. If not, Z is replaced by a digit from the field's content.
	* (asterisk)	If an asterisk (*) matches a leading zero in the field's content, an asterisk is placed in that character position. If not, it is replaced by a digit from the field's content.
	. (period)	A period specifies the character position of the decimal point.
Alphanumeric Insertion	+ (plus)	A plus is inserted in that character position unless more than one plus sign is specified.
	- (hyphen)	A hyphen is inserted in that character position.
	. (period)	A period is inserted in that character position.
	, (comma)	A comma is inserted in that character position.
Numeric Insertion	+ (plus)	If only one plus sign is specified, it is replaced by either a plus sign, if the field's content is positive, or a minus sign if it is negative.
	- (minus)	If only one minus sign is specified, it is replaced by either a blank, if the field's content is positive, or a minus sign if it is negative.
	. (decimal)	A decimal point is inserted in that character position. Put only one decimal point in a numeric edit string.
	, (comma)	If all the digits to the left of the comma are suppressed zeros, the comma is replaced by a blank. If not, a comma is inserted in that character position.

(continued on next page)

Table D–1 (Cont.) Edit String Characters

Character Type	Edit String Character	Description
	CR	If the field's content is negative, the letters CR are inserted. If the field's content is positive, CR is replaced by two blanks. Put only one CR in an edit string, either at the far right or the far left.
	DB	If the field's content is negative, the letters DB are inserted. If the field's content is positive, DB is replaced by two blanks. Put only one DB in an edit string, either at the far right or the far left.
	( ) (parentheses)	If the field's content is negative, single left and right parentheses are inserted before and after the field value.
Alphanumeric and Numeric Insertion	B	A space is inserted in that character position.
	0 (zero)	A zero is placed in that character position.
	$ (dollar sign)	If only one dollar sign is specified, it is printed in that character position.
	% (percent)	A percent sign is inserted in that character position.
	/ (slash)	A slash is inserted in that character position.
	"literal"	The character string literal enclosed in single or double quotation marks is inserted at that position. The outer quotation marks are not inserted in the output.
Numeric Floating Insertion	$ (dollar sign)	If more than one dollar sign is specified to the left of the other edit string characters, leading zeros are suppressed and one dollar sign is displayed to the immediate left of the leftmost digit.
	+ (plus)	If more than one plus sign is specified to the left of the other edit string characters, any leading zeros are suppressed, and the sign of the field's value (plus or minus) is displayed to the immediate left of the leftmost character position determined by the other edit string characters.

(continued on next page)

Edit String Characters

Table D–1 (Cont.) Edit String Characters

Character Type	Edit String Character	Description
	- (minus)	If more than one minus sign is specified to the left of the other edit string characters, any leading zeros that the minus sign matches are suppressed. If the value of the field is negative, a minus sign is displayed to the immediate left of the leftmost character position determined by the other edit string characters.
Floating-Point Edit String	E	The E divides the edit string into two parts for floating-point or scientific notation. The first part is the mantissa edit string and the second part is the exponent edit string.
Missing Value Edit String	?	If the field has a MISSING VALUE clause, the question mark separates two edit strings. If the field value is not the missing value, the first edit string controls the output of the field. If the field value is the missing value, the second edit string controls the output of the field.
Date Replacement	D	Each D is replaced by the corresponding digit of the day of the month, starting with the first letter. Put no more than two Ds in a date edit string; the use of DD is recommended.
	M	Each M is replaced by the corresponding letter of the name of the month. An edit string of M(9) prints the entire name of the month.
	N	Each N is replaced by a digit of the number of the month. Put no more than two Ns in a date edit string; the use of NN is recommended.
	Y	Each Y is replaced by the corresponding digit of the numeric year. Put no more than four Ys in a date edit string; the use of YY or YYYY is recommended.
	J	Each J is replaced by the corresponding digit of the Julian date. Put no more than three Js in a date edit string; the use of JJJ is recommended.

(continued on next page)

Table D–1 (Cont.) Edit String Characters

Character Type	Edit String Character	Description
	W	Each W is replaced by the corresponding letter from the name of the day of the week, starting with the first letter. An edit string of W(9) prints the entire day. Put no more than 9 Ws in a date edit string.
	B	Each B is replaced by a space in that character position.
	/ (slash)	A slash is inserted in that character position.
	- (hyphen)	A hyphen is inserted in that character position.
	. (period)	A period is inserted in that character position.

Index

A

B

C

D

E

F

G

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z