DEC DATATRIEVE

New Features Guide

Order Number: AA–PVLZA–TE

June 1993

This manual describes the new features for DEC DATATRIEVE Version 6.1.

Operating System:	OpenVMS VAX Version 5.5 or higher
Operating System:	OpenVMS AXP Version 1.5
Software Version:	DEC DATATRIEVE Version 6.1

Digital Equipment Corporation, Maynard, Massachusetts

This document was prepared using VAX DOCUMENT, Version 2.1.

Contents

Tables

Preface

This manual describes new features for DEC DATATRIEVE Version 6.1. The new features relate to both DEC DATATRIEVE for OpenVMS AXP operating systems and DEC DATATRIEVE for OpenVMS VAX operating systems, which are referred to by their abbreviated name DEC DATATRIEVE. Sections specific to a particular platform are indicated using margin icons as shown in Conventions.

New features for DEC DATATRIEVE Version 6.1 are also described in online help. To read a description of new features from within DEC DATATRIEVE, enter the following command:

```
DTR> HELP New_Features
```

Intended Audience

This manual is intended for all DEC DATATRIEVE users. It assumes that you understand the concepts and terminology of the OpenVMS operating system.

Operating System Information

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

For information on the compatibility of other software products with this version of DEC 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 DEC DATATRIEVE.

Related Documents

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

- *DEC DATATRIEVE Installation Guide*

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

- *DEC DATATRIEVE Release Notes*

  Describes corrections to software, restrictions, workarounds, and known problems for DEC DATATRIEVE Version 6.1.

- *VAX DATATRIEVE User's Guide*

  Describes how to use DEC DATATRIEVE interactively.

- *VAX DATATRIEVE Guide to Interfaces*

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

- *VAX DATATRIEVE Reference Manual*

  Contains reference information for DEC DATATRIEVE.

- *VAX DATATRIEVE Guide to Programming and Customizing*

  Explains how to use the DEC DATATRIEVE Callable Interface. The manual also describes how to create user-defined keywords and user-defined functions to customize DEC DATATRIEVE and how to customize DEC DATATRIEVE help and message text.

Conventions

In this manual, every use of OpenVMS VAX indicates the OpenVMS VAX operating system, every use of OpenVMS AXP indicates the OpenVMS AXP operating system, and every use of OpenVMS indicates both the OpenVMS VAX operating system and the OpenVMS AXP operating system.

The following conventions are used to identify information specific to OpenVMS AXP or to OpenVMS VAX:

AXP	The AXP icon denotes the beginning of information specific to the OpenVMS AXP operating system.
VAX	The VAX icon denotes the beginning of information specific to the OpenVMS VAX operating system.
♦	The diamond symbol denotes the end of a section of information specific to the OpenVMS AXP or OpenVMS VAX operating system.

The following conventions are also used in this manual:

{ } (braces)	Braces enclose a clause from which you must choose one alternative.
[ ] (brackets)	Brackets enclose optional clauses from which you can choose one or none.
... (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.
. . (vertical . ellipsis)	A vertical ellipsis indicates you can repeat the line of the clause, statement, command, or expression immediately above the ellipsis.
bold	A bolded word indicates that a new term is being introduced
Color	Use of second color in examples shows user input. For online version of this manual, user input is shown in **bold**.

References to Products

The DEC DATATRIEVE documentation to which this manual belongs often refers to Digital products by their abbreviated names. Many of these names are in the process of changing to reflect their operating system platform, and thus may not be exactly correct.

- DEC DATATRIEVE refers to both DEC DATATRIEVE for OpenVMS AXP and DEC DATATRIEVE for OpenVMS VAX software.

- DEC Rdb refers to both DEC Rdb for OpenVMS AXP and DEC Rdb for OpenVMS VAX software.
- DEC DBMS refers to both DEC DBMS for OpenVMS AXP and DEC DBMS for OpenVMS VAX software.
- DEC FORTRAN refers to both DEC FORTRAN for OpenVMS AXP and DEC FORTRAN for OpenVMS VAX software.
- DEC C refers to both DEC C for OpenVMS AXP and DEC C for OpenVMS VAX software.
- DEC TPU refers to both DEC TPU for OpenVMS AXP and DEC TPU for OpenVMS VAX software.

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

- VAX Rdb/ELN
- DEC Rdb
- VIDA

1 New Features for DEC DATATRIEVE Version 6.1

This chapter describes new features for DEC DATATRIEVE Version 6.1. These include:

- Support of the Alpha AXP architecture.
- DECwindows Motif interface.
- Enhancements to the Report Writer.
- Improved integration with relational databases.
- New DEC DATATRIEVE functions.
- New DEC DATATRIEVE logical name.
- Improvements to the online help.

1.1 Using DEC DATATRIEVE on OpenVMS AXP Systems

DEC DATATRIEVE Version 6.1 runs and provides the same features on both OpenVMS VAX and OpenVMS AXP systems, provided that they are supported by the architecture of the system. In this section we will discuss only the changes that are visible in terms of functionality. For Version 6.1 most of the differences result from other layered products not being available on the OpenVMS AXP system.

1.1.1 Overview of the Alpha AXP Architecture

The Alpha AXP architecture is a 64-bit RISC architecture designed to avoid bias toward any particular operating system or programming language. It initially supports the OpenVMS AXP and DEC OSF/1 operating systems, and supports simple software migration from applications that run on those operating systems.

The Alpha AXP architecture differs fundamentally from the VAX architecture. Some of the more significant differences include:

– 64-bit virtual addressing capability.
– A new instruction set designed to optimize the design and operation of hardware. One aspect of the new instruction set is that the Alpha AXP architecture does not support VAX H_floating-point and packed-decimal data types.
– A larger memory page size which depends on processor design.
– An optimized memory/cache subsystem.

OpenVMS AXP is the code name of the version of the OpenVMS operating system that runs on processors built according to the Alpha AXP architecture.

1.1.2 Moving the DEC DATATRIEVE Environment Between Operating Systems

To move your DEC DATATRIEVE environment from an OpenVMS VAX system to an OpenVMS AXP system, do the following:

- Copy all DEC DATATRIEVE data files and procedures to the other system.
- If your application programs use the DEC DATATRIEVE Call Interface, or if you have customized DEC DATATRIEVE by modifying certain source files, you must recompile and relink the application programs and the DEC DATATRIEVE customization source files.
- For DEC DATATRIEVE related products (e.g. CDD/Repository, DECforms), refer to their specific documentation.

1.1.3 Floating-Point Data Types

There are some differences between the OpenVMS VAX and the OpenVMS AXP platforms, in terms of support for floating-point data types:

– The H_floating-point data type is not supported on OpenVMS AXP systems.
– Both the IEEE S_floating-point data type and the IEEE T_floating-point data type are supported on OpenVMS AXP systems, but not on OpenVMS VAX systems.

S_floating-point (the IEEE single-precision floating-point data type) indicates that the field is a floating-point number accurate to approximately 7 decimal digits. An S_floating-point field is 4 bytes long.

T_floating-point (the IEEE double-precision floating-point data type) indicates that the field is a floating-point number accurate to approximately 15 decimal digits. T_floating-point fields occupy 8 contiguous bytes in memory.

1.1.4 Packed-Decimal Data Type

The Alpha AXP architecture does not support the packed-decimal format (it is not mapped on machine instructions), hence the software performs some calculation in order to emulate it with the result that the packed-decimal field format is not so efficient on OpenVMS AXP systems as it is on OpenVMS VAX systems.

1.1.5 Data Definition

The new keywords S_FLOATING and T_FLOATING have been added to define the IEEE floating-point formats as values in a USAGE clause. The S_FLOATING and T_FLOATING keywords are supported on OpenVMS AXP systems, not on OpenVMS VAX systems. The H_FLOATING keyword is supported on OpenVMS VAX systems and not on OpenVMS AXP systems.

If you try to DEFINE or DECLARE data that the platform on which you are running does not support, DEC DATATRIEVE signals an error with the diagnostic message:

```
Unsupported data type on this platform.
```

If, when reading metadata (from CDD/Repository or relational databases), DEC DATATRIEVE detects a field of a type that is not supported on the platform it is running on, it issues the following warning message:

```
Field ... ignored because of unsupported data type.
```

The field is subsequently ignored.

1.1.6 Conversion in Computations

On both OpenVMS VAX and OpenVMS AXP systems, the internal arithmetic format chosen for computations involving floating-point data of different types is the one that reflects the highest possible accuracy.

For example, if you have an expression with one operand declared as a D_floating-point data type and the other declared as a G_ floating-point data type, the computation and the result will be in G_floating-point.

Table 1–1 shows the floating-point data types in order of precedence based on their precision.

Table 1–1 Floating-Point Data Types: Order of Precedence

1st Level (Lowest Precision)	2nd Level	3rd Level	4th Level	5th Level	6th Level (Highest Precision)
F	S	D	G	T	H

1.1.7 Remote Domains

When DEC DATATRIEVE accesses a remote domain, using DDMF, it is possible to operate between heterogeneous nodes (e.g. OpenVMS AXP and OpenVMS VAX nodes), but the following mechanisms apply, when readying the domain:

- Local DEC DATATRIEVE domain on OpenVMS VAX system, remote DEC DATATRIEVE domain on OpenVMS AXP system.

  If the remote DEC DATATRIEVE domain contains a field declared as S_floating-point, communication is handled in such a way that data is converted to F_floating-point. The local DEC DATATRIEVE user sees the field as an F_floating-point entity. No messages are issued.

  An identical behavior occurs with remote T_floating-point data fields, which are converted to G_floating-point.

  This behavior applies even when the local node runs VAX DATATRIEVE Version 6.0 or lower.

- Local DEC DATATRIEVE domain on OpenVMS AXP system, remote DEC DATATRIEVE domain on OpenVMS VAX system.

If the local DEC DATATRIEVE domain detects that the remote DEC DATATRIEVE domain has a field declared as H_floating-point, it handles the communication with the remote domain in such a way that data is converted to G_floating-point. The local DEC DATATRIEVE user sees the field as a G_floating-point entity.

A warning message is issued, with the diagnostic message:

```
Field ... converted to G_floating-point because of
unsupported data type.
```

Table 1–2 shows the data conversion rules between heterogeneous local and remote domains.

Table 1–2 Data Conversion Between Heterogeneous Local and Remote Domains

Location of Local DTR Domain	Location of Remote DTR Domain	Remote Data Type	Conversion to Local Data Type
OpenVMS VAX	OpenVMS AXP	S_floating-point	F_floating-point
OpenVMS VAX	OpenVMS AXP	T_floating-point	G_floating-point
OpenVMS AXP	OpenVMS VAX	H_floating-point	G_floating-point

1.2 DECwindows Motif Based User Interface

In DEC DATATRIEVE Version 6.1, the DECwindows interface has been substituted with a DECwindows Motif interface. The DECwindows Motif user interface has the same type of functionality as the previous DECwindows interface.

The following sections describe changes and features that have been added to the new interface.

1.2.1 Changes to the Main Window

The Main window has the following added features:

- The SHOW menu contains the additional Forms item.

  The Forms item displays the form name, the form file, and the form product name of all loaded forms.

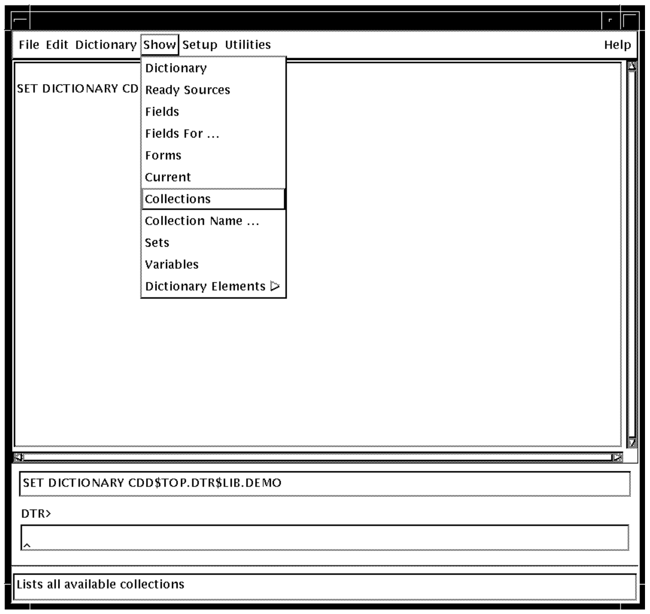
- Additional history area.

  The history area, located between the work area and the command line area, echoes commands and statements entered at the DTR> prompt.

- Additional help line at the bottom of the window.

  When you browse through the pull-down menus, the help line displays a short explanation of the item being highlighted by the pointer. See Figure 1–1.

Figure 1–1 DEC DATATRIEVE Main Window



SET DICTIONARY CD

SET DICTIONARY CDD$TOP.DTR$LIB.DEMO

DTR>

Lists all available collections

1.2.2 Changes to the SHOW Window

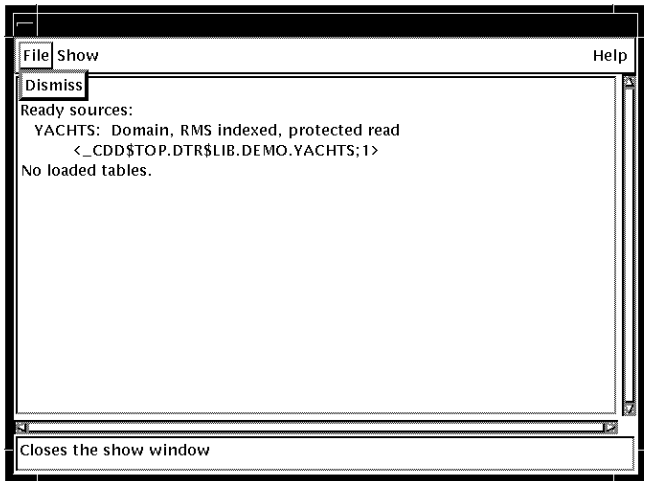
The SHOW window has a SHOW pull-down menu, which is identical to the one in the main window.

The DEC DATATRIEVE Show window menu bar now offers you three menu choices:

- File—provides you with access to the Dismiss item which dismisses the DEC DATATRIEVE Show window.
- Show—provides you with access to the same functionality as the Show menu options of the main application window.
- Help—provides you with access to general information on using DEC DATATRIEVE in a DECwindows Motif environment.

There is an additional help line at the bottom of the SHOW window which gives a short explanation of the item being highlighted by the pointer. See Figure 1–2.

Figure 1–2 DEC DATATRIEVE Show Window



```
Ready sources:
  YACHTS:  Domain, RMS indexed, protected read
        <_CDD$TOP.DTR$LIB.DEMO.YACHTS;1>
No loaded tables.
```

Closes the show window

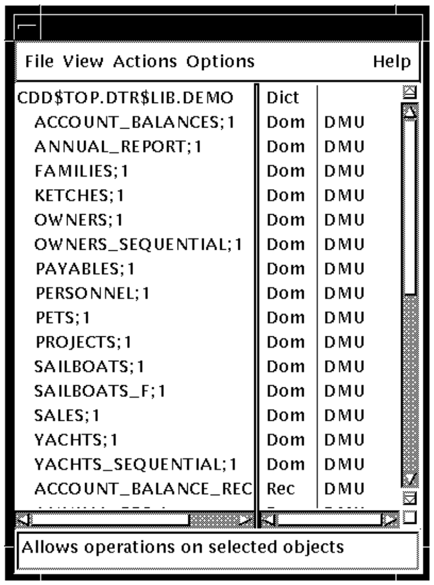
1.2.3 Changes to the Dictionary Navigator

The Dictionary Navigator window has the following added features:

- A new Column format has been added to the display mode item of the View Menu.

  The Columns format is like the Outline format, with additional vertical columns separating the objects from their type and format information. See Figure 1–3.

Figure 1–3 Dictionary Navigator: Columns Format



```
CDD$TOP.DTR$LIB.DEMO       Dict
  ACCOUNT_BALANCES;1       Dom   DMU
  ANNUAL_REPORT;1          Dom   DMU
  FAMILIES;1               Dom   DMU
  KETCHES;1                Dom   DMU
  OWNERS;1                 Dom   DMU
  OWNERS_SEQUENTIAL;1      Dom   DMU
  PAYABLES;1               Dom   DMU
  PERSONNEL;1              Dom   DMU
  PETS;1                   Dom   DMU
  PROJECTS;1               Dom   DMU
  SAILBOATS;1              Dom   DMU
  SAILBOATS_F;1            Dom   DMU
  SALES;1                  Dom   DMU
  YACHTS;1                 Dom   DMU
  YACHTS_SEQUENTIAL;1      Dom   DMU
  ACCOUNT_BALANCE_REC      Rec   DMU
```

Allows operations on selected objects

- The Set dictionary item has been added to the Actions pull-down menu.

  The Set dictionary item allows you to set your dictionary to the directory containing the object you have selected in the Navigator window. This item is active only when the object you select is of type dictionary.

  When you perform this operation, you execute a SET DICTIONARY command for the entire DEC DATATRIEVE session. This means that you affect the main window and the SHOW window, but not the displayed hierarchy of the Dictionary Navigator.

To affect the display of the Dictionary Navigator, but not the entire DEC DATATRIEVE session, select the Set Root item of the Navigator's View menu.

- The Options menu has been created.

  It allows you to view the object type and format of the Navigator's dictionary objects. The options are available only for the Outline and Columns format display.

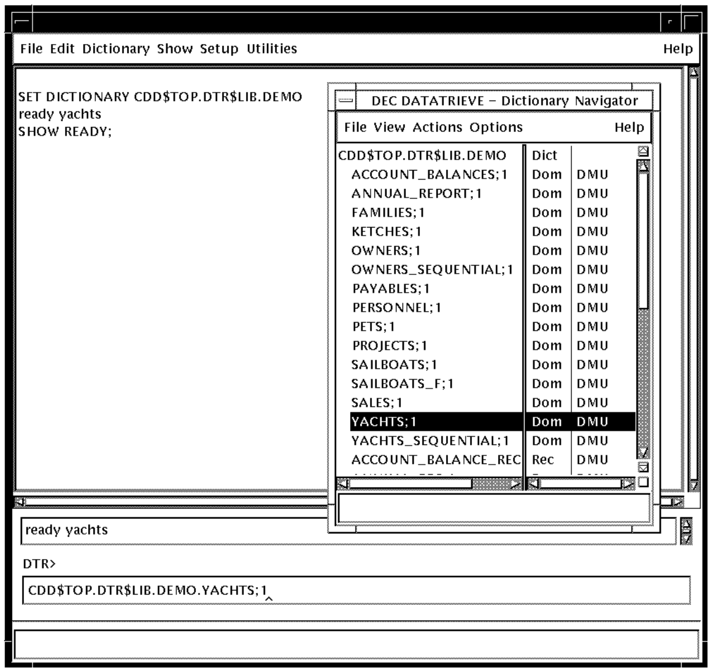
  The options are:

  - Show Object Type—It shows the object type of the displayed objects.
  - Show Object Format— It shows the object format of the displayed objects.

- A "select & paste" capability between the navigator and the command line area of the main window has been implemented.

  When you select an object displayed in the Navigator window, then paste it in the command line area of the main window, the pasted object contains its complete path name. See Figure 1–4

Figure 1–4 Dictionary Navigator: Select and Paste


File Edit Dictionary Show Setup Utilities
Help
DEC DATATRIEVE – Dictionary Navigator
File View Actions Options
Help

```
SET DICTIONARY CDD$TOP.DTR$LIB.DEMO
ready yachts
SHOW READY;
```

CDD$TOP.DTR$LIB.DEMO	Dict	
ACCOUNT_BALANCES;1	Dom	DMU
ANNUAL_REPORT;1	Dom	DMU
FAMILIES;1	Dom	DMU
KETCHES;1	Dom	DMU
OWNERS;1	Dom	DMU
OWNERS_SEQUENTIAL;1	Dom	DMU
PAYABLES;1	Dom	DMU
PERSONNEL;1	Dom	DMU
PETS;1	Dom	DMU
PROJECTS;1	Dom	DMU
SAILBOATS;1	Dom	DMU
SAILBOATS_F;1	Dom	DMU
SALES;1	Dom	DMU
YACHTS;1	Dom	DMU
YACHTS_SEQUENTIAL;1	Dom	DMU
ACCOUNT_BALANCE_REC	Rec	DMU

```
ready yachts
DTR>
CDD$TOP.DTR$LIB.DEMO.YACHTS;1
```

- An additional help line at the bottom of the window has been created.

  When you browse through the pull-down menus, the help line displays a short explanation of the item being highlighted by the pointer.

1.2.4 Changes to the Help Menu

The Help menu lets you display general information on using DEC DATATRIEVE in a DECwindows Motif environment.

DECwindows Motif supports context sensitive help. When you choose On Context from the Help menu, the pointer changes to a question mark (?). You can then point to any object in the DEC DATATRIEVE window or dialog box and click MB1 (Mouse Button 1).

For example, to get help on a menu:

1. Choose On Context from the Help menu.

2. Position the question mark (?) on a menu from the DEC DATATRIEVE window and click MB1.

   Information on that menu is displayed in the help topic window. You might also see a list of additional topics that provide more information.

   Note

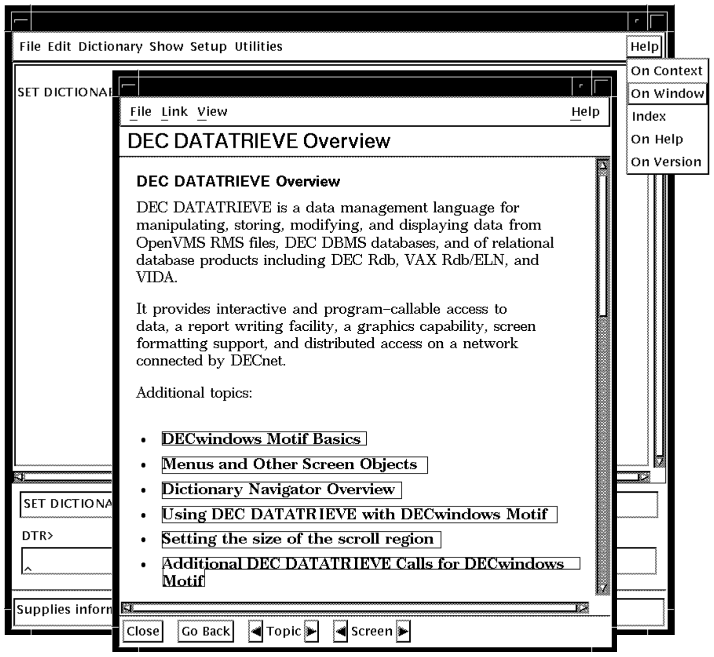
   To obtain help information on a menu item, point to the item, then press the Help key while you hold MB1.

The Help menu provides help that you can access by either

– Choosing an item from the Help menu. Double-click on any of these items to learn more:
  - On Context
  - On Window
  - On Index
  - On Help
  - On Version
– Clicking on the Help button in a dialog box

Help topics are displayed in a help topic window. If a topic is longer than the help window, use the scroll bars, or resize the help window to display the rest of the topic. See Figure 1–5.

Figure 1–5 DEC DATATRIEVE Help Window



DEC DATATRIEVE Overview

DEC DATATRIEVE Overview

DEC DATATRIEVE is a data management language for manipulating, storing, modifying, and displaying data from OpenVMS RMS files, DEC DBMS databases, and of relational database products including DEC Rdb, VAX Rdb/ELN, and VIDA.

It provides interactive and program–callable access to data, a report writing facility, a graphics capability, screen formatting support, and distributed access on a network connected by DECnet.

Additional topics:

- DECwindows Motif Basics
- Menus and Other Screen Objects
- Dictionary Navigator Overview
- Using DEC DATATRIEVE with DECwindows Motif
- Setting the size of the scroll region
- Additional DEC DATATRIEVE Calls for DECwindows Motif

For more information on how to use DECwindows Motif help, see the OpenVMS DECwindows Motif documentation.

You can get help on DEC DATATRIEVE-specific terms by typing the HELP command at the DTR> prompt. If you invoke help in this way, DEC DATATRIEVE spawns a separate DECterm window that displays a list of the DEC DATATRIEVE help topics (see Section 1.7).

1.3 Improvements to the Report Writer

The DEC DATATRIEVE Version 6.1 Report Writer allows you to improve the layout of a printed report (DDIF or PostScript® format) by giving you the following options:

– Printing a grid and boxes on the report page to determine where printed fields are positioned.

– Controlling the space allocation of string fields with the WIDTH clause.

1.3.1 Layout Representation

In a DEC DATATRIEVE report that uses DDIF or PostScript® format, the output report is divided into 80 columns (for portrait orientation), or 132 columns (for landscape orientation). With the COL, TAB, or SPACE print list element, followed by a column number, you can specify where the output of the next print list element begins. See the *VAX DATATRIEVE Reference Manual* for more information on print list elements.

However, in the DEC DATATRIEVE Report Writer, it is not easy to predict where a field will appear on the output report, because the space occupied by a given number of characters depends on the font you choose.

To help you understand where to position the printable objects of your report, DEC DATATRIEVE Version 6.1 gives you the option of printing a report showing the exact location of fields. Such a report consists of two parts:

- The first part containing boxes that show the actual space allocation of the printed fields.
- The second part containing the actual report printed on a background grid.

To see the boxes and the grid, you must define the logical name DTR$RW_DEBUG either from outside DEC DATATRIEVE, with the DEFINE command, or from inside DEC DATATRIEVE with the FN$CREATE_LOG function. When you define the DTR$RW_DEBUG logical name, you must specify an argument that determines whether you will print the grid, the layout, or both.

The argument must be enclosed in double quotes and must contain either the word GRID, the word LAYOUT, or both separated by a comma.

```
$ DEFINE DTR$RW_DEBUG "GRID,LAYOUT"
```

By choosing:

- "GRID", the output report is printed on a background grid.
- "LAYOUT", the output report is preceded by the layout boxes.

- "GRID,LAYOUT", the output report is printed on a background grid and is preceded by the layout boxes.

Layout Boxes

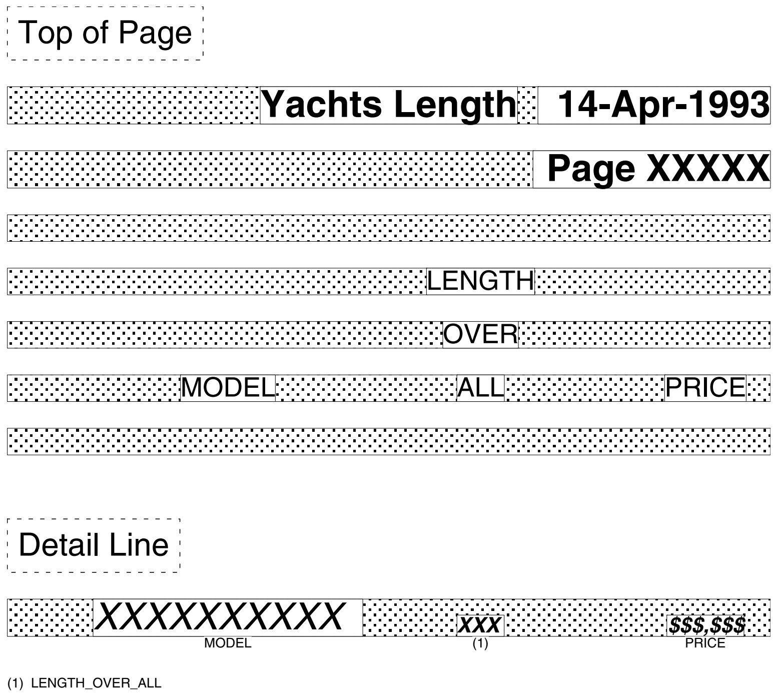
If you assign the argument "LAYOUT" to the DTR$RW_DEBUG logical name, the first part of your printed report consists of boxes showing the space allocation for printed fields. Example 1–1 produces an output report containing such boxes.

Example 1–1 Example Producing Layout Boxes

```
DTR> FN$CREATE_LOG ("DTR$RW_DEBUG","LAYOUT")
DTR> REPORT FIRST 10 YACHTS WITH PRICE NOT MISSING ON BOXES.PS -
CON> FORMAT PS
RW> DECLARE_ATT ATT1 BOLD, SIZE = 8
RW> DECLARE_ATT ATT2 ITALIC, NO BOLD, SIZE = 14
RW> SET REPORT_NAME = "Yachts Length"
RW> SET PAPER_SIZE = A5
RW> PRINT COL 10, ATT ATT2, MODEL, COL 45, ATT ATT1, LOA, -
CON> COL 70, ATT ATT1, PRICE
RW> END_REPORT
DTR>
```

The first part of the report produced by Example 1–1, is shown in Figure 1–6.

Figure 1–6 Layout Boxes



Print elements are represented on the report according to the following rules:

- Constant strings are represented in the size in which they would appear on the report.
- Fields are represented by their edit strings in their original size. The edit string characters correspond to those used in the record definition (X for alphanumeric fields, 9 for numeric fields, and so on)

- Field names are written at the bottom of the box, in small characters. If the field name is too big for the available space, DEC DATATRIEVE uses a legend.
- Unused spaces between print elements are filled with gray, and reflect the physical height of the actual output lines.

Grid

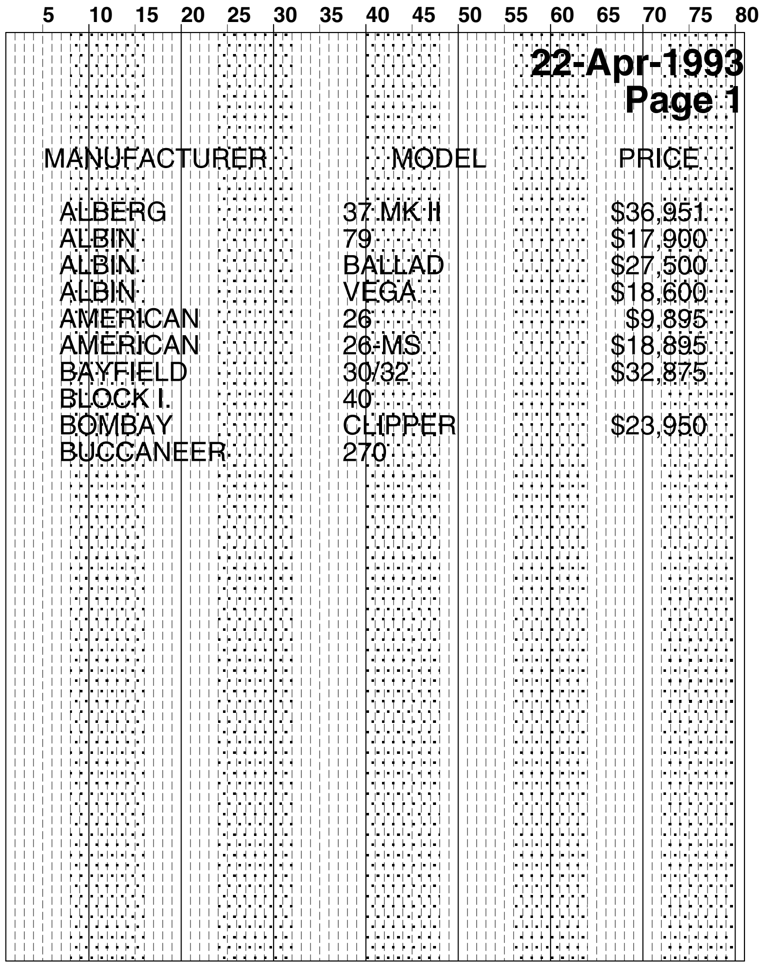
If you assign the argument "GRID" to the DTR$RW_DEBUG logical name, a grid appears on the background of each page of the printed report. The grid consists of dotted lines indicating columns, and continuous lines indicating intervals of ten columns. Tab stops are shown by shading the tab intervals. Example 1–2 produces an output report on a background grid.

Example 1–2 Example Producing Layout Grid

```
DTR> FN$CREATE_LOG ("DTR$RW_DEBUG","GRID")
DTR> REPORT FIRST 10 YACHTS ON GRID.PS FORMAT PS
RW> SET PAPER_SIZE = A5
RW> PRINT BUILDER,MODEL,PRICE
RW> END_REPORT
DTR>
```

The output generated by Example 1–2 is shown in Figure 1–7.

Figure 1–7 Report with Background Grid


5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80

```
                                                  22-Apr-1993
                                                       Page 1

 MANUFACTURER           MODEL            PRICE

   ALBERG             37 MK II          $36,951
   ALBIN              79                $17,900
   ALBIN              BALLAD            $27,500
   ALBIN              VEGA              $18,600
   AMERICAN           26                 $9,895
   AMERICAN           26-MS             $18,895
   BAYFIELD           30/32             $32,875
   BLOCK I.           40
   BOMBAY             CLIPPER           $23,950
   BUCCANEER          270
```

1.3.2 Space Allocation for Strings

String fields sometimes cause excessive space to be used on the printed page. This is because DEC DATATRIEVE calculates the space needed by considering the field length and the average character size of the chosen font.

In the past, users did not have control over space allocation. For example you could not force DEC DATATRIEVE to use less space than theoretically needed to represent a given field.

In DEC DATATRIEVE Version 6.1 a new WIDTH clause has been added to the Report Writer PRINT statement, to allow you to declare how much space (in columns) a string field can take. The syntax is:

```
          PRINT field_name WIDTH number_of_columns
```

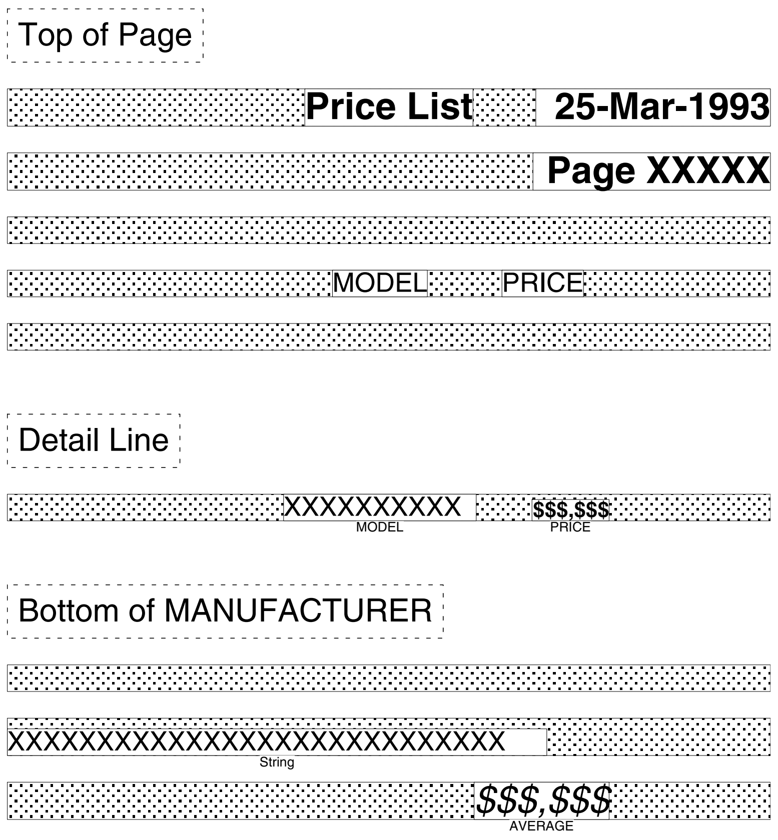
DEC DATATRIEVE allocates the space for the string field according to the WIDTH clause, instead of computing it from the field length and the font metrics. If a string takes more than the available space, it is compressed, and even truncated, if needed. However, if a string takes less than the allocated space it is not expanded. Example 1–3, Figure 1–8, and Figure 1–9 illustrate a situation where a WIDTH clause may be useful.

Example 1–3 Report with Long Fields

```
DTR> FN$CREATE_LOG ("DTR$RW_DEBUG","GRID,LAYOUT")
DTR> READY YACHTS
DTR> REPORT FIRST 15 YACHTS WITH PRICE NOT MISSING ON WIDTH.PS -
CON> FORMAT PS
RW> DECLARE_ATT ATT1 BOLD, SIZE = 8
RW> DECLARE_ATT ATT2 ITALIC, SIZE = 14
RW> SET REPORT_NAME = "Price List"
RW> SET PAPER_SIZE = A5
RW> PRINT
RW> COL 30, MODEL, COL 50, ATT ATT1, PRICE
RW> AT BOTTOM OF BUILDER PRINT
RW> SKIP, COL 1, "Average price for" ||| BUILDER,
RW> ATT ATT2, AVERAGE PRICE
RW> END_REPORT
DTR>
```

The output generated by Example 1–3 is shown in Figure 1–8.

Figure 1–8 Report Without WIDTH Clause

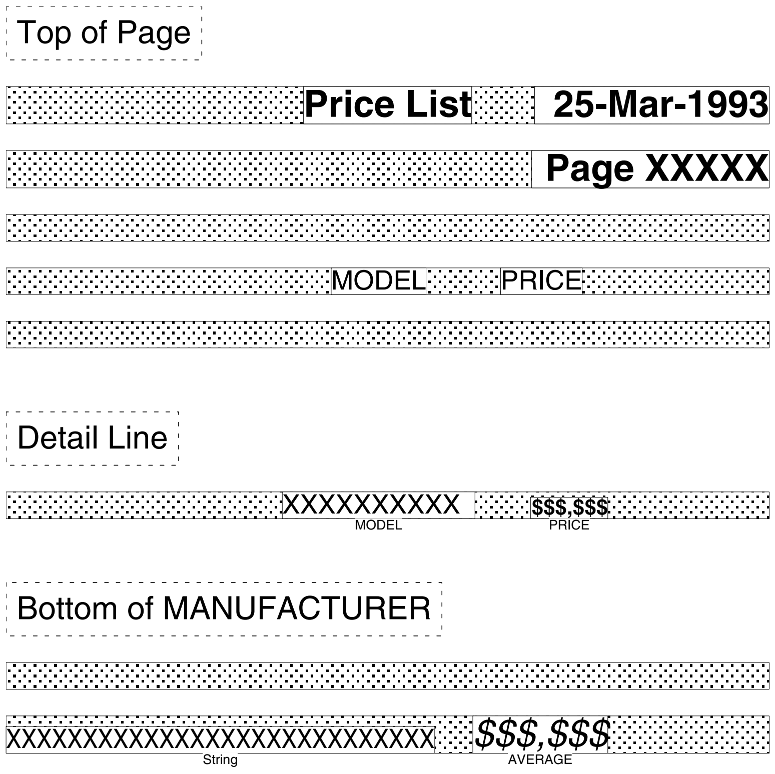


From Figure 1–8 you can see that the AVERAGE field is printed on a new report line. This happens because the edit string of the previous field ("Average price for" | | | BUILDER) is too long and the two fields would overlap if placed on the same report line. To resolve this problem you can add a WIDTH clause to the BUILDER field:

```
SKIP, COL 1, "Average price for" ||| BUILDER WIDTH 45,
```

The WIDTH clause forces DEC DATATRIEVE to compress the BUILDER field and to print the AVERAGE field on the same report line. See Figure 1–9.

Figure 1–9 Report with WIDTH Clause



Note the following:

- The WIDTH clause is not valid outside of the Report Writer PRINT statement.

- If the number specified in the WIDTH clause is not in the range allowed by the SET PAPER_ORIENTATION statement (80 columns for portrait orientation and 132 columns for landscape orientation), DEC DATATRIEVE displays an error message similar to the following:

```
WIDTH value of 83 not in range for PORTRAIT paper orientation.
```

1.4 Improvements to Relational Databases Support

DEC DATATRIEVE Version 6.1 has improved its interface to relational databases by supporting null values and providing limited help for dbkeys (with the new DEC DATATRIEVE functions FN$LASTREC_DBKEY, FN$DBKEY_LENGTH, and FN$FORMAT_DBKEY)

See the following sections for more information.

1.4.1 Null Values

Previous versions of DEC DATATRIEVE recognized values that were absent or unknown through the "missing" mechanism: the MISSING VALUE clause in a record field definition defines a value that flags the absence of data in that field.

Relational databases, on the other hand, use null values to notify the absence of a value in a field.

Previous versions of DEC DATATRIEVE did not support relational databases null values because of the difference between missing values and null values—missing values are actual values (zeros or others) representing the absence of data in a field, while null values are distinct values (different from all the other "not null" values).

Version 6.1 of DEC DATATRIEVE supports null values of SQL-compliant relational databases. DEC DATATRIEVE now recognizes columns with null values and displays them according to the edit string for the missing value. Section 1.4.3 shows the different behavior of VAX DATATRIEVE Version 6.0 and DEC DATATRIEVE Version 6.1 when dealing with relational null values. DEC DATATRIEVE performs the following:

- Queries relational databases containing null values.
- Selects table rows with columns containing null values when a relational operator MISSING is used.

- Displays properly null values.
- Stores and modifies null values in record fields of relational domains.
- Performs operations on null values.
- Supports null values in DEC DATATRIEVE variables, arithmetic and string expressions, Boolean expressions, and FORMAT value expressions.
- Handles null values occurring in collections based on relational tables or views based on relational values.

Note

- At present, null values are not supported for remote domains based on relational domains.
- Null values used with forms products or with DEC DATATRIEVE functions lead to unpredictable results.

1.4.1.1 Performing Queries on Relational Databases

A query is a request for DEC DATATRIEVE to identify all the records in a database that satisfy a specified condition. You can perform queries with RSEs and Boolean expressions. When querying a relational database, you can identify null values by using the MISSING and NOT MISSING clauses in RSEs and Boolean expressions (see Section 1.4.1.5 for more information).

```
DTR> READY CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL
DTR> PRINT JOB_HISTORY WITH JOB_END MISSING

EMPLOYEE JOB      JOB         JOB     DEPARTMENT SUPERVISOR
   ID    CODE    START        END        CODE        ID

 00168   SPGM 18-Feb-1982 NULL           MGVT      00267
 00172   SANL 28-Oct-1980 NULL           ELEL      00200
 00188   EENG  8-Apr-1982 NULL           ADMN      00225
             .
             .
             .
DTR> FOR JOB_HISTORY
CON> IF JOB_END MISSING
CON> PRINT EMPLOYEE_ID | " Still with us"
```

```
00168 Still with us
00172 Still with us
00188 Still with us
      .
      .
      .
DTR>
```

1.4.1.2 Displaying Null Values

In DEC DATATRIEVE Version 6.1, the display of null values depends on the field's edit string. It can be NULL or other than NULL, according to the following considerations:

- DEC DATATRIEVE uses the edit string of the field if available and complete. For this reason the edit string must fall into one of the following categories:
  - Be explicitly defined.

    ```
    DTR> PRINT REL_TABLE USING 9999?"NOT AVAILABLE"
    ```

  - Be defined in the database.

    ```
    SQL> CREATE TABLE T1
                      (C1 REAL
         EDIT STRING IS '9999.99?"NOT AVAILABLE"');
    ```

  - Be defined as a default edit string in DTRTEXT.MAR. For example, date fields have a default edit string of DD-MMM-YYYY?" ". Note that, in this case, null values will not be displayed because the missing edit string is defined to be a blank space (the value defined between double quotes).
- DEC DATATRIEVE assigns a default edit string to those relational fields whose edit string is totally or partially absent. The default edit string has the format:

  ```
  basic-edit-string?"missing-edit-string"
  ```

  The question mark (**?**) divides the edit string into two parts. The first part (**basic-edit-string**) of the edit string applies to the output of the field value if that value is not missing. The second part (**missing-edit-string**) is the part of the edit string that applies if the field contains a null value. This string must be enclosed in double quotes.

When a default edit string is assigned, the first part is taken from the field definition, and the second part contains the word NULL. This default edit string is defined in the file DTRTEXT.MAR, located in DTR$LIBRARY, and it can be customized by the user. (See the *VAX DATATRIEVE Guide to Interfaces* for more information on how to customize DEC DATATRIEVE text).

Note

When DEC DATATRIEVE assigns a default edit string:

- NULL strings for numeric fields are right-justified, while NULL strings for text and date fields are left-justified.
- If you print a field shorter than 4 characters and containing a null value, the word NULL will not be fully displayed.

See the examples in Section 1.4.1.7.

1.4.1.3 Storing Null Values

DEC DATATRIEVE Version 6.1 inserts (with STORE statements or Restructure statements) null values into fields of a record belonging to a relational domain when no value is explicitly inserted.

Restructure Statement

The Restructure statement transfers data from the fields of records in a record source stream to fields with the corresponding name in a target domain. The Restructure statement has the following behavior:

- When the source field contains a null value, the target field will contain a null value.
- When transferring data from an OpenVMS RMS domain (or another source not supporting null values) to a relational target domain, if a source field contains a missing value, the corresponding target field will contain a null value. In addition, if the relational target domain has more fields than the source domain, those fields will contain null values. See the following example.

```
DTR> SHOW DEPARTMENT43
DOMAIN DEPARTMENT43 USING DEPARTMENT43_REC ON DEPARTMENT43.DAT;
DTR> SHOW DEPARTMENT43_REC
RECORD DEPARTMENT43_REC
01 DEPARTMENT43.
03 EMP_ID PIC X(10) MISSING VALUE "missing".
03 DEPT_CODE PIC X(10).;
DTR> PRINT DEPARTMENT43

 EMP_ID    DEPT_CODE

missing    23456789
DTR> SHOW FIELDS FOR DEPARTMENT_SUPERVISOR_TABLE
DEPARTMENT_SUPERVISOR_TABLE
   DEPARTMENT_SUPERVISOR_TABLE
     EMP_ID         <Number>
     DEPT_CODE      <Number>
     SUPERVISOR_ID  <Number>

DTR> DEPARTMENT_SUPERVISOR_TABLE = DEPARTMENT43
DTR> PRINT DEPARTMENT_SUPERVISOR_TABLE

    EMP_ID    DEPT_CODE  SUPERVISOR_ID

       NULL   23456789            NULL
DTR>
```

- When transferring data from a relational source domain to a target domain not supporting null values, if a relational field contains a null value, and the corresponding target field has a MISSING VALUE clause, then the target field will contain the missing value (or default value if there is a DEFAULT VALUE clause). In addition, if the target domain has more fields than the relational source, those fields will contain the value specified by the MISSING VALUE clause or by the DEFAULT VALUE clause (should the clauses be absent, text fields will be initialized to spaces and numeric fields to zero).

```
DTR> SHOW DEPARTMENT10
DOMAIN DEPARTMENT10 USING DEPARTMENT10_REC ON DEPARTMENT10.DAT;
DTR> SHOW DEPARTMENT10_REC
RECORD DEPARTMENT10_REC
01 DEPARTMENT10_REC.
05 EMP_ID PIC X(4) MISSING VALUE 1 .
05 MGR_ID PIC X(10) DEFAULT VALUE "ID"
   MISSING VALUE "Not there".
05 DEPT_NAME PIC X(10) MISSING VALUE "Admin.".
05 DEPT_CODE PIC X(10) DEFAULT VALUE "AD32".
05 PROJECT PIC X(10) DEFAULT VALUE "Support"
   MISSING VALUE "T.B.A.".
05 SECRETARY PIC X(10).;
DTR> PRINT DEPARTMENT10
DTR> DEPARTMENT10 = EMPLOYEE_MANAGER_TABLE
DTR> PRINT EMPLOYEE_MANAGER_TABLE

   EMP_ID       MGR_ID

      10        NULL
      15          20
      30        NULL
    NULL          40

DTR> PRINT DEPARTMENT10

EMP_ID  MGR_ID    DEPT_NAME  DEPT_CODE   PROJECT     SECRETARY

0010   Not there   Admin.     AD32      Support
0015   20          Admin.     AD32      Support
0030   Not there   Admin.     AD32      Support
0001   40          Admin.     AD32      Support
DTR>
```

STORE Statement

Notice that the STORE statement has two different types of behavior when used for storing null values in relational sources:

- When you use a STORE USING statement, those fields that are not specified by the statement receive null values.

```
DTR> STORE EMPLOYEE_MANAGER_TABLE USING EMP_ID = 2345
DTR> PRINT EMPLOYEE_MANAGER_TABLE

    EMP_ID         MGR_ID

       2345        NULL
DTR>
```

  In the previous example the field MGR_ID of the relation EMPLOYEE_MANAGER_TABLE contains a null value.

- When you use a STORE statement and you enter a TAB character in response to a DEC DATATRIEVE prompt, DEC DATATRIEVE stores the default value of that field—the value specified by the DEFAULT VALUE clause. If the DEFAULT VALUE clause is absent, DEC DATATRIEVE initializes a text field to spaces and a numeric field to zero, regardless of whether the field definition contains a MISSING VALUE clause.

```
DTR> STORE EMPLOYEE_MANAGER_TABLE
Enter EMP_ID: 4332
Enter MGR_ID: TAB
DTR> PRINT EMPLOYEE_MANAGER_TABLE

    EMP_ID          MGR_ID

     4332             0
DTR>
```

1.4.1.4 Modifying Null Values

DEC DATATRIEVE Version 6.1 allows you to modify null values stored in record fields of relational domains.

```
DTR> PRINT City, Code OF CITY_CODE_TABLE

   City    Code

NULL       NULL
DTR> DECLARE TEMP PIC X(10).
DTR> TEMP = 17
DTR> MODIFY CITY_CODE_TABLE USING Code = TEMP
DTR> PRINT City, Code OF CITY_CODE_TABLE

   City    Code

NULL         17
DTR>
```

You can also modify records in a collection:

```
DTR> FIND CITY_CODE_TABLE
[1 record found]
DTR> SELECT 1
DTR> MODIFY USING Code = 18
DTR> PRINT City, Code OF CITY_CODE_TABLE

   City    Code

NULL         18
DTR>
```

Note that you cannot use a MODIFY statement to replace directly the current value of a field with a null value.

```
DTR> MODIFY CITY_CODE_TABLE USING City = NULL
"NULL" is undefined or used out of context
DTR>
```

To obtain positive results, use a variable containing a MISSING VALUE clause:

```
DTR> DECLARE VAR PIC X(10) MISSING VALUE 0.
DTR> MODIFY CITY_CODE_TABLE USING Code = VAR
DTR> PRINT City, Code OF CITY_CODE_TABLE

   City    Code

NULL       NULL
DTR>
```

1.4.1.5 Performing Operations with Null Values

DEC DATATRIEVE can now perform arithmetic, relational, boolean and statistical operations on null values according to SQL standards. For example:

- An arithmetic or string operation generates null when one of the operands is null.

  ```
  DTR> FOR X IN EMPLOYEE_MANAGER_TABLE PRINT  EMP_ID

   EMP_ID

       10
       15
       30
     NULL
     NULL

  DTR> FOR X IN EMPLOYEE_MANAGER_TABLE PRINT  EMP_ID + 1000

      1010
      1015
      1030
      NULL
      NULL
  DTR>
  ```

- Statistical expressions and operations ignore null elements.

```
DTR> PRINT TOTAL EMP_ID OF EMPLOYEE_MANAGER_TABLE

   TOTAL
  EMP_ID

[Function computed using 3 of 5 values.]
      55
DTR>
```

- Statistical expressions and operations generate null as a result when operating on empty data streams.

```
DTR> FIND EMPTY IN CITY_CODE_TABLE WITH Code < 1
[0 records found]
DTR> PRINT Code OF EMPTY
DTR> PRINT TOTAL Code OF EMPTY

  TOTAL
   Code

    NULL
DTR>
```

1.4.1.6 Null Values and Variables

When transferring a missing value from a DEC DATATRIEVE variable to a relational field, the field receives a null value.

```
DTR> DECLARE VAR PIC X(10) MISSING VALUE "77".
DTR> PRINT VAR

VAR

77

DTR> STORE EMPLOYEE_MANAGER_TABLE USING EMP_ID = VAR
DTR> PRINT EMPLOYEE_MANAGER_TABLE

 EMP_ID          MGR_ID

    NULL        NULL
DTR>
```

When transferring a null value from a relational field to a DEC DATATRIEVE variable, the variable receives either its missing value (if a MISSING VALUE clause is specified in the variable definition), or a null value (if the variable does not contain a MISSING VALUE clause).

```
DTR> DECLARE EMPLOYEE PIC X(10) MISSING VALUE 8999.
DTR> FOR X IN EMPLOYEE_MANAGER_TABLE
CON> EMPLOYEE = EMP_ID
DTR> PRINT EMPLOYEE

 EMPLOYEE
```

```
8999
DTR> DECLARE MANAGER PIC X(10).
DTR> FOR X IN EMPLOYEE_MANAGER_TABLE
CON> MANAGER = MGR_ID
DTR> PRINT MANAGER

 MANAGER

NULL
DTR>
```

1.4.1.7 Null Values and FORMAT Value Expression

The FORMAT value expression of a null value does not evaluate to the null value but to the string "NULL" (or to the edit string you supply instead of "NULL").

```
DTR> PRINT CITY_CODE_TABLE

       Code      City

      NULL ROME
      0031 NULL

DTR> PRINT (FORMAT City USING XX) OF
CON> CITY_CODE_TABLE

RO
NU
DTR>
```

1.4.1.8 Null Values and Boolean Expressions

A Boolean expression is the logical representation of a relationship between value expressions. When dealing with relational sources, the value of a Boolean expression is either TRUE, FALSE, or **UNKNOWN** according to SQL standards. The UNKNOWN boolean value may result if either value expression in a Boolean expression is null. (See Appendix A for more information on how relational databases evaluate predicates combined with Boolean operators). As a consequence, the DEC DATATRIEVE CHOICE and IF THEN ELSE value expressions, and the CHOICE and IF statements can contain an additional ELSE clause to support the UNKNOWN boolean value.

In an IF THEN ELSE value expression, the ELSE part of the expression returns a value if the boolean expression evaluates to a truth value of either FALSE or UNKNOWN. See the following example.

```
DTR> PRINT CITY_CODE_TABLE

 Code      City

NULL ROME
0031 NULL
NULL PARIS
0001 BERN

DTR> FOR X IN CITY_CODE_TABLE
CON> IF Code = 1 THEN PRINT "1"  ELSE
CON> PRINT "2"

2
2
2
1
DTR>
```

You can add an additional ELSE part of the expression to handle the truth value of UNKNOWN:

```
DTR> FOR X IN CITY_CODE_TABLE
CON> IF MISSING Code THEN PRINT "3" ELSE
CON> IF Code = 1 THEN PRINT "1" ELSE PRINT "2"

3
2
3
1
DTR>
```

The same thing happens with the CHOICE value expression.

```
DTR> FOR X IN CITY_CODE_TABLE
CON> CHOICE Code = 1 THEN PRINT "1" ELSE
CON> CHOICE NOT Code = 1 THEN PRINT "2" ELSE
CON> PRINT "3"
CON> END_CHOICE
CON> END_CHOICE

3
2
3
1

DTR>
```

1.4.2 Dbkeys

Dbkeys are binary values that identify rows in a database by indicating the address of the table row. When you access a row by dbkey, the database system can retrieve, delete or update that row directly, without accessing an index or sequentially scanning a table, row by row.

Some languages and packages may use record dbkeys for specific operations. For example, SQL multimedia routines accept a record dbkey to identify the table row containing the data to be managed.

DEC DATATRIEVE Version 6.1 has implemented three functions (FN$LASTREC_DBKEY, FN$DBKEY_LENGTH, and FN$FORMAT_DBKEY) that support dbkeys when dealing with relational databases.

Note

These functions have been implemented for DEC Rdb dbkeys only.

1.4.2.1 FN$LASTREC_DBKEY

FN$LASTREC_DBKEY returns the dbkey of the last record read in a currently readied relational domain or table.

Format:

```
FN$LASTREC_DBKEY ("domain-name")
```

Input:

domain-name
A string expression providing the name of the domain or table.

Output:

The record dbkey as a byte string. The function returns an empty string (zero length string) if:

- There is no dbkey for the record (e.g. certain views).
- The domain has never been accessed.

DEC DATATRIEVE returns an error message if:

- The domain is not currently readied.
- The domain specified as argument is not a relational domain.

Example:

See the examples in Section 1.4.2.3.

1.4.2.2 FN$DBKEY_LENGTH Function

FN$DBKEY_LENGTH returns the byte length of dbkeys for the relational domain or table specified as argument.

Format:

```
FN$DBKEY_LENGTH ("domain-name")
```

Input:

domain-name
A string expression providing the name of the domain or table.

Output:

The byte length of the dbkey. An error message is returned if the domain is unknown, or if the domain is not a relational one.

Example:

See the examples in Section 1.4.2.3.

1.4.2.3 FN$FORMAT_DBKEY

FN$FORMAT_DBKEY returns a character string containing the dbkey of the readied relational domain, table, or view.

Format:

```
FN$FORMAT_DBKEY (dbkey)
```

Input:

dbkey
A string containing the dbkey in its binary format.

Output:

The character string of the dbkey. This function assumes that the dbkey is internally structured as an array of elementary dbkeys and that each elementary dbkey is 8 bytes long.

Examples:

In the following example the FN$FORMAT_DBKEY function returns the dbkey of the last record read from CITY_CODE_ TABLE in its string format.

```
DTR> PRINT FN$FORMAT_DBKEY(FN$LASTREC_DBKEY("CITY_CODE_TABLE"))

          FN$FORMAT
            DBKEY

11:149:26
DTR>
```

In the following example the variable VAR prints the dbkey of each record in the relational view C in their string format. The length of the dbkeys in this view is 16 bytes (8 bytes by the number of tables in the view).

```
DTR> DECLARE VAR PIC X(80).;
DTR> FOR C BEGIN
CON> VAR = FN$LASTREC_DBKEY("C");
CON> PRINT -
CON> FN$FORMAT_DBKEY(FN$STR_EXTRACT(VAR,1,FN$DBKEY_LENGTH("C")))
CON> END;

          FN$FORMAT
            DBKEY

39:512:0 40:518:0
39:512:1 40:518:0
39:512:2 40:518:0
39:512:3 40:518:0
DTR>
```

The FN$FORMAT_DBKEY function gives you the same results as SQL does in the following example:

```
SQL> SELECT DBKEY FROM C;

                                      DBKEY
               39:512:0             40:518:0
               39:512:1             40:518:0
               39:512:2             40:518:0
               39:512:3             40:518:0
4 rows selected
```

1.4.3 Differences Between Version 6.0 and 6.1

This section illustrates the behavior of VAX DATATRIEVE Version 6.0 and DEC DATATRIEVE Version 6.1 when dealing with relational null values.

1.4.3.1 Query Operations

VAX DATATRIEVE Version 6.0

```
DTR> PRINT CITY_CODE_TABLE WITH Code MISSING
MISSING VALUE not defined for Code, using default value.

 Code      City

0000 BOSTON
0000 LONDON
DTR>
```

DEC DATATRIEVE Version 6.1

```
DTR> PRINT CITY_CODE_TABLE WITH Code MISSING

 Code      City

NULL BOSTON
NULL LONDON
DTR>
```

VAX DATATRIEVE Version 6.0

```
DTR> PRINT CITY_CODE_TABLE

 Code      City

0001
0002
0000 BOSTON
0000 LONDON
DTR> FIND CITY_CODE_TABLE
DTR> PRINT CURRENT WITH Code LE 0

 Code      City

0000 BOSTON
0000 LONDON
DTR> PRINT CITY_CODE_TABLE WITH Code LE 0
DTR>
```

DEC DATATRIEVE Version 6.1

```
DTR> PRINT CITY_CODE_TABLE

 Code      City

0001 NULL
0002 NULL
NULL BOSTON
NULL LONDON
DTR> FIND CITY_CODE_TABLE
DTR> PRINT CURRENT WITH Code LE 0
DTR> PRINT CITY_CODE_TABLE WITH Code LE 0
DTR>
```

VAX DATATRIEVE Version 6.0

```
DTR> PRINT Code + 10 OF CITY_CODE_TABLE

        11
        12
        10
        10
DTR>
```

DEC DATATRIEVE Version 6.1

```
DTR> PRINT Code + 10 OF CITY_CODE_TABLE

        11
        12
      NULL
      NULL
DTR>
```

1.4.3.2 Display Operations

VAX DATATRIEVE Version 6.0

```
DTR> PRINT City USING X(10)?"Not available",
CON>       Code USING 9(10)?"   not here",
CON>       City USING X(2),
CON>       Code USING 9(10)?"not here" OF CITY_CODE_TABLE;

     City       Code     City   Code

              0000000001     0000000001
              0000000002     0000000002
BOSTON        0000000000  BO 0000000000
LONDON        0000000000  LO 0000000000
DTR>
```

DEC DATATRIEVE Version 6.1

```
DTR> PRINT City USING X(10)?"Not available",
CON>       Code USING 9(10)?"   not here",
CON>       City USING X(2),
CON>       Code USING 9(10)?"not here" OF CITY_CODE_TABLE;

     City        Code    City   Code

Not available 0000000001  NU 0000000001
Not available 0000000002  NU 0000000002
BOSTON           not here BO not here
LONDON           not here LO not here
DTR>
```

1.4.3.3 Store Operations

VAX DATATRIEVE Version 6.0

```
DTR> STORE CITY_CODE_TABLE USING Code = 1
DTR> PRINT CITY_CODE_TABLE

 Code      City

0001
DTR>
```

DEC DATATRIEVE Version 6.1

```
DTR> STORE CITY_CODE_TABLE USING Code = 1
DTR> PRINT CITY_CODE_TABLE

 Code      City

0001 NULL
DTR>
```

1.4.3.4 Boolean Expressions

VAX DATATRIEVE Version 6.0

```
DTR> PRINT CITY_CODE_TABLE

 Code      City

0001
0002
0000 BOSTON
0000 LONDON
DTR> FOR CITY_CODE_TABLE
CON> BEGIN
CON> IF Code EQ 1
CON>    THEN PRINT "Code EQ 1 "
CON>    ELSE IF NOT Code EQ 1
CON>       THEN PRINT "Code NE 1"
CON>       ELSE PRINT "Code UNKNOWN" ;
CON> IF City EQ "BOSTON"
CON>    THEN PRINT "City EQ BOSTON"
CON>    ELSE IF NOT City EQ "BOSTON"
CON>       THEN PRINT "City NE BOSTON"
CON>       ELSE PRINT "City UNKNOWN" ;
CON> END ;
```

```
Code EQ 1
City NE BOSTON
Code NE 1
City NE BOSTON
Code NE 1
City EQ BOSTON
Code NE 1
City NE BOSTON
DTR>
```

DEC DATATRIEVE Version 6.1

```
DTR> PRINT CITY_CODE_TABLE

 Code      City

0001 NULL
0002 NULL
NULL BOSTON
NULL LONDON
DTR> FOR CITY_CODE_TABLE
CON> BEGIN
CON> IF Code EQ 1
CON>    THEN PRINT "Code EQ 1 "
CON>    ELSE IF NOT Code EQ 1
CON>       THEN PRINT "Code NE 1"
CON>       ELSE PRINT "Code UNKNOWN" ;
CON> IF City EQ "BOSTON"
CON>    THEN PRINT "City EQ BOSTON"
CON>    ELSE IF NOT City EQ "BOSTON"
CON>       THEN PRINT "City NE BOSTON "
CON>       ELSE PRINT "City UNKNOWN" ;
CON> END ;

Code EQ 1
City UNKNOWN
Code NE 1
City UNKNOWN
Code UNKNOWN
City EQ BOSTON
Code UNKNOWN
City NE BOSTON
DTR>
```

VAX DATATRIEVE Version 6.0

```
DTR> PRINT ( CHOICE
CON>            Code EQ 1 THEN "Code EQ 1" ;
CON>            NOT Code EQ 1 THEN "Code NE 1" ;
CON>            ELSE "Code UNKNOWN"
CON>         END-CHOICE ) ,
CON>       ( CHOICE
CON>            City EQ "BOSTON" THEN "City EQ BOSTON" ;
CON>            NOT City EQ "BOSTON" THEN "City NE BOSTON" ;
CON>            ELSE "City UNKNOWN"
CON>         END-CHOICE ) OF CITY_CODE_TABLE ;

Code EQ 1    City NE BOSTON
Code NE 1    City NE BOSTON
Code NE 1    City EQ BOSTON
Code NE 1    City NE BOSTON
DTR>
```

DEC DATATRIEVE Version 6.1

```
DTR> PRINT ( CHOICE
CON>            Code EQ 1 THEN "Code EQ 1" ;
CON>            NOT Code EQ 1 THEN "Code NE 1" ;
CON>            ELSE "Code UNKNOWN"
CON>         END-CHOICE ) ,
CON>       ( CHOICE
CON>            City EQ "BOSTON" THEN "City EQ BOSTON" ;
CON>            NOT City EQ "BOSTON" THEN "City NE BOSTON" ;
CON>            ELSE "City UNKNOWN"
CON>         END-CHOICE ) OF CITY_CODE_TABLE ;

Code EQ 1    City UNKNOWN
Code NE 1    City UNKNOWN
Code UNKNOWN City EQ BOSTON
Code UNKNOWN City NE BOSTON
DTR>
```

1.4.3.5 Variables

VAX DATATRIEVE Version 6.0

```
DTR> DECLARE VAR1 PIC 9999.
DTR> DECLARE VAR2 PIC X(10).
DTR> FIND CITY_CODE_TABLE WITH Code EQ 1
DTR> SELECT 1
DTR> VAR1 = Code
DTR> VAR2 = City
DTR> PRINT VAR1, VAR2

VAR1    VAR2

0001

DTR> STORE CITY_CODE_TABLE USING Code = VAR1 + 10
DTR> PRINT CITY_CODE_TABLE
```

```
 Code      City

0001
0002
0000 BOSTON
0000 LONDON
0011
DTR>
```

DEC DATATRIEVE Version 6.1

```
DTR> DECLARE VAR1 PIC 9999.
DTR> DECLARE VAR2 PIC X(10).
DTR> FIND CITY_CODE_TABLE WITH Code EQ 1
DTR> SELECT 1
DTR> VAR1 = Code
DTR> VAR2 = City
DTR> PRINT VAR1, VAR2

VAR1    VAR2

0001 NULL

DTR> STORE CITY_CODE_TABLE USING Code = VAR1 + 10
DTR> PRINT CITY_CODE_TABLE

 Code      City

0001 NULL
0002 NULL
NULL BOSTON
NULL LONDON
0011 NULL
DTR>
```

1.5 New DEC DATATRIEVE Functions

DEC DATATRIEVE Version 6.1 provides the following new functions:

- FN$SHOWDEF
- FN$SETDEF
- FN$SHOWDEFPROT
- FN$SETDEFPROT
- FN$PROCESS_INFO
- FN$HEX_TO_DEC

See the following sections for more information.

1.5.1 FN$SHOWDEF

FN$SHOWDEF displays the current disk directory for the process.

Format:

```
FN$SHOWDEF
```

Input:

None.

Output:

A text string containing the name of the current disk directory.

Example:

The following example shows how to display the current working directory from DEC DATATRIEVE.

```
$ SHOW DEFAULT
  DISK:[SMITH.WORK]
$ DATATRIEVE
DTR> PRINT FN$SHOWDEF USING X(80)

                                    FN$SHOWDEF

[SMITH.WORK]
DTR>
```

1.5.2 FN$SETDEF

FN$SETDEF allows you to change the default disk directory for the process.

Format:

```
FN$SETDEF ("new-directory")
```

Input:

new-directory
A string expression containing the new default disk directory.

Output:

None.

Usage Notes:

DEC DATATRIEVE displays an error message if the argument you supply to the FN$SETDEF function contains an illegal directory specification.

The new directory setting is saved when you exit DEC DATATRIEVE. Therefore, you should restore the old default directory to its original status, if you want to return to the original directory setting.

Example:

The following example shows how to change the default directory from within your DEC DATATRIEVE process and how to restore the old default directory to its original status before exiting DEC DATATRIEVE.

```
$ SHOW DEFAULT
  DISK:[SMITH]
DTR> DECLARE OLDDIR PIC X(80).
DTR> OLDDIR = FN$SHOWDEF
DTR> PRINT OLDDIR

                      OLDDIR

[SMITH]
DTR> FN$SETDEF ("[SMITH.WORK]")
DTR> PRINT FN$SHOWDEF

          FN$SHOWDEF

[SMITH.WORK]
DTR>
             .
             .
             .
DTR> FN$SETDEF (OLDDIR)
DTR> PRINT FN$SHOWDEF

          FN$SHOWDEF

[SMITH]
DTR> EXIT
$ SHOW DEFAULT
  DISK:[SMITH]
$
```

1.5.3 FN$SHOWDEFPROT

FN$SHOWDEFPROT displays the current file protection for the process.

Format:

```
FN$SHOWDEFPROT
```

Input:

None.

Output:

A text string containing the current default file protection.

Example:

The following example shows how to display the current default file protection.

```
DTR> PRINT FN$SHOWDEFPROT USING X(80)

                                           FN$SHOWDEFPROT

SYSTEM:RWED,OWNER:RWED,GROUP:RWED,WORLD:
DTR>
```

1.5.4 FN$SETDEFPROT

FN$SETDEFPROT allows you to change the default file protection for the process.

Format:

```
FN$SETDEFPROT ("new-default-protection")
```

Input:

new-default-protection
A string expression containing the new default file protection specification. The format is the standard OpenVMS format (e.g. "SYSTEM:RWED,OWNER:RWED,GROUP:R,WORLD:R").

Output:

None.

Usage Notes:

The new default file protection is saved when you exit DEC DATATRIEVE. Therefore, you should restore the old default file protection to its original status unless you want the new default to be saved when you exit the image.

Note that if a file already exists, OpenVMS RMS maintains the protection of the existing file when creating a higher version, regardless of the changes you made to the file protection.

Example:

The following example shows how to change the default file protection from within your DEC DATATRIEVE process and how to restore the old default file protection to its original status before exiting DEC DATATRIEVE.

```
DTR> PRINT FORMAT FN$SHOWDEFPROT USING X(80)
SYSTEM:RWED,OWNER:RWED,GROUP:RWED,WORLD:

DTR> DECLARE OLDPROT PIC X(80).
DTR> OLDPROT = FN$SHOWDEFPROT
DTR> PRINT OLDPROT

                                    OLDPROT

SYSTEM:RWED,OWNER:RWED,GROUP:RWED,WORLD:
DTR> FN$SETDEFPROT ("S:RWED,O:RWED")
DTR> PRINT FORMAT FN$SHOWDEFPROT USING X(80)
SYSTEM:RWED,OWNER:RWED,GROUP:,WORLD:

DTR> DEFINE DOMAIN H USING H_REC ON H.DAT;
DTR> DEFINE RECORD H_REC USING
DFN> 01 H_REC.
DFN> 03 F1 PIC X(7).
DFN> 03 F2 PIC X(10). ;
DTR> DEFINE FILE FOR H;
DTR> FN$DCL ("DIR/PROT *.DAT")

Directory DISK:[DALFY.NEWUSER]

EMPLOYEES.DAT;1      10-FEB-1993 11:06:03.39  (RWED,RWED,RWED,)
FAMILY.DAT;108       25-MAR-1992 21:44:53.30  (RWED,RWED,RWED,)
FAMS.DAT;1            6-AUG-1992 15:33:51.24  (RWED,RWED,RWED,)
H.DAT;1              24-MAR-1993 09:47:39.31  (RWED,RWED,,)

Total of 4 files.
DTR> FN$SETDEFPROT (OLDPROT)
DTR> PRINT FORMAT FN$SHOWDEFPROT USING X(80)

SYSTEM:RWED,OWNER:RWED,GROUP:RWED,WORLD:

DTR> EXIT
$ SHOW PROTECTION
  SYSTEM=RWED, OWNER=RWED, GROUP=RWED, WORLD=NO ACCESS
$
```

1.5.5 FN$PROCESS_INFO

FN$PROCESS_INFO returns information on the current process.

Format:

```
FN$PROCESS_INFO (item-code)
```

Input:

item-code
An integer value specifying the item of information that the FN$PROCESS_INFO function is to return; the integer value is one of the values used by the $GETJPI system service.

DEC DATATRIEVE supplies a table associating symbolic strings to values of the $GETJPI system service. The table is called JPI_CODES, located in CDD$TOP.DTR$LIB. If you use the JPI_CODES table with this function, the syntax is:

```
FN$PROCESS_INFO ("symbolic-string" VIA CDD$TOP.DTR$LIB.JPI_CODES)
```

Where **symbolic-string** is a text string defined in JPI_CODES.

For a complete description of the item codes, see the $GETJPI service documented in the *OpenVMS System Services Reference Manual*.

Output:

A text string containing the information requested by the item-code.

Usage Notes:

Note that the symbolic-string is a table element, therefore it is case sensitive.

Examples:

The following examples shows how the FN$PROCESS_INFO function returns information about the username; the first example uses an integer value, the second example uses the JPI_CODES table.

```
DTR> PRINT FN$PROCESS_INFO (514)

          FN$PROCESS
             INFO

DALFY

DTR>
DTR> PRINT -
CON> FN$PROCESS_INFO ("USERNAME" VIA CDD$TOP.DTR$LIB.JPI_CODES)

          FN$PROCESS
             INFO

DALFY

DTR>
```

1.5.6 FN$HEX_TO_DEC

FN$HEX_TO_DEC calculates the integer equivalent of a hexadecimal value.

Format:

```
FN$HEX_TO_DEC ("hexadecimal-string")
```

Input:

hexadecimal-string
A string expression containing the hexadecimal character string.

Output:

A longword integer equivalent to the hexadecimal value.

Examples:

The following example returns the integer equivalent to the hexadecimal value FF.

```
DTR> PRINT FN$HEX_TO_DEC ("FF")

  FN$HEX
    TO
   DEC

        255

DTR>
```

The following example shows how the FN$HEX_TO_DEC function can be used to calculate the numeric value of the address of the first free page at the end of the program region of the process.

```
DTR> PRINT FN$HEX_TO_DEC (FN$PROCESS_INFO ("FREP0VA" VIA -
CON> JPI_CODES))

  FN$HEX
    TO
   DEC

    7991296

DTR>
```

1.6 New DEC DATATRIEVE Logical Name

The DEC DATATRIEVE Version 6.1 installation kit provides a new logical name:

- DTR$BANNER_DISABLE

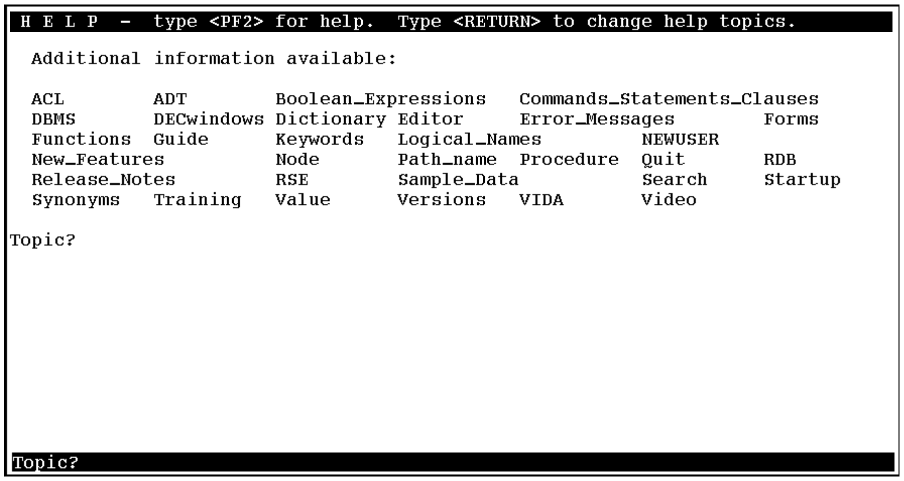
If you define the DTR$BANNER_DISABLE logical name to true, the DEC DATATRIEVE banner will not be displayed when you invoke the application.

```
$ DEFINE DTR$BANNER_DISABLE TRUE
$ DATATRIEVE
DTR>
```

1.7 Improvements to the Online Help

In DEC DATATRIEVE Version 6.1, the main screen of the online help has changed. When you invoke the DEC DATATRIEVE Help facility with the HELP command, you will see a different menu. (See Figure 1–10.)

Figure 1–10 DEC DATATRIEVE Main Help Menu



```
 H E L P  -  type <PF2> for help.  Type <RETURN> to change help topics.

  Additional information available:

  ACL        ADT        Boolean_Expressions   Commands_Statements_Clauses
  DBMS       DECwindows Dictionary Editor     Error_Messages        Forms
  Functions  Guide      Keywords   Logical_Names          NEWUSER
  New_Features          Node       Path_name  Procedure  Quit       RDB
  Release_Notes         RSE        Sample_Data           Search     Startup
  Synonyms   Training   Value      Versions   VIDA       Video

Topic?







Topic?
```

A
SQL Truth Tables

This appendix illustrates how SQL compliant relational databases evaluate predicates combined with Boolean operators (AND, OR, NOT). Such tables are also called **truth tables**. See Table A–1, Table A–2, and Table A–3.

Table A–1 Boolean Operators: AND

A	B	A and B
True	False	False
True	True	True
False	False	False
False	True	False
True	Unknown	Unknown
False	Unknown	False
Unknown	True	Unknown
Unknown	False	False
Unknown	Unknown	Unknown

Table A–2 Boolean Operators: OR

A	B	A or B
True	False	True
True	True	True
False	False	False
False	True	True
True	Unknown	True
False	Unknown	Unknown
Unknown	True	True
Unknown	False	Unknown
Unknown	Unknown	Unknown

Table A–3 Boolean Operators: NOT

A	NOT A
True	False
False	True
Unknown	Unknown

Index