September 1982 PUB-3022 c.1

DCN/SEEDIS

The Distributed Computer Network (DCN) and Socio-Economic-Environmental Demographic Information System (SEEDIS)

An Introduction to the Distributed Computer Network

Virginia A. Sventek

For Reference

Not to be taken from this room

RECEIVED
LAWRENCE
BERKELEY LABORATORY

FEB 25 1983

LIBRARY AND
DOCUMENTS SECTION

Computer Science and
Mathematics Department
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

PUB-3022 c.1

Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

An Introduction to the Distributed Computer Network

September, 1982

Virginia A. Sventek
Computer Science and Mathematics Department
Lawrence Berkeley Laboratory
Berkeley, California
FTS 451-5216

This work was supported by the Applied Mathematical Sciences Program of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098 and the U.S. Department of Labor.

Table of Contents

Introduction to DCN	3	Accessing the DCN
		Terminals
		Connecting Terminals to DCN
	4	*Keyboards*
		Special Characters
	5	Logging on to DCN
	6	VMS Directory Structure
		Definitions
	7	File Names
	8	Using File Specifications
	10	The Directory Command
	12	File Manipulation
		Type Command
		Print Command
	13	*Removing Files*
	14	*Copying Files*
	16	Editing Files
	18	*Positioning*
	20	*Substitutions*
	25	*Adding Text*
	27	*Terminating Text Additions*
	28	*Deleting Text*
	30	*Leaving the Editor*

Introduction to Datatrieve	1	Accessing Datatrieve
	1	Leaving Datatrieve
	2	Help
	4	The Common Data Dictionary
	5	Set Dictionary Command
	6	Show Command
	8	Parts of a Datatrieve Domain
	9	Record Definition
	18	Field Definition Clause
	20	Record Definition Syntax

Table of Contents (Continued)

Introduction to Datatrieve 21 Domain Definition

21 Domain Definition Syntax

22 File Definition

23 File Definition Syntax

24 Datatrieve Editor

28 Extract Command

29 Accessing a Datatrieve Domain

31 Establishing Collections

33 Examples of the Find Command

35 Print Command

40 Sort Command

43 Store Command

45 Erase Command

47 Modify Command

50 Assignment Statements

52 For Statement

54 Then Statement

56 Begin-End Statement

58 Selecting Records

61 Prompting for Values

64 Statistical Functions

67 Tables

71 Report Writer
72 *Basic Commands*
73 *Report Command*
74 *Set Statement*
76 *Control Groups*
77 *At Statement*
79 *Print Statement*
81 *Examples*

90 Datatrieve Procedures

Syllabus:

"An Introduction to the Distributed Computer Network" was designed to serve as support documentation for a five day course presented to DOL/ETA at Regional Offices by LBL staff. At these presentations, new users of the DCN receive instruction on the basic components of the VAX 11/780 minicomputers of which the DCN is comprised, including VMS (the VAX 11/780 Operating System), use of interactive terminals with VMS, an overview of the VMS directory structure, introduction to a text editor, and an introduction to Datatrieve (a data entry and retrieval system developed by Digital Equipment Corporation).

Specific topics to be presented include:

Use of the Terminal Keyboard;

Logging on to VMS (the VAX 11/780 Operating System);

VMS Directory Structure;

Files Manipulation;

Introduction to Datatrieve.

The nature of these presentations is intended to be highly interactive, with questions addressed and answered as they arise. New users attending these presentations should expect, at the end of the five day period, to be able to log onto DCN machines, create and edit files, and execute pre-defined Datatrieve procedures.

The presentation and this booklet draw heavily on material supplied in the Digital Equipment Corporation Datatrieve Documentation and readers are referred to that documentation for further information.

An Introduction to the DCN: Summary Documentation

Accessing the DCN through an interactive terminal:

The Distributed Computer Network is composed of Digital Equipment Corporation VAX 11/780 machines at the ETA National Office, the Region IX and X Regional Offices, and Lawrence Berkeley Laboratory. The VAXs are connected together through a network over which they communicate.

The VAX is an interactive computer which enables users to communicate with its operating system (VMS) and various utilities and programs through terminals. The concept of interactive computing is a simple one: the user types a command telling the system which operation to perform; the operating system then performs the operation, or in the case of an error, tells the user why the operation cannot be performed. In this way the machine and the user interact with each other.

The following paragraphs explain the basic tools used in interactive computing:

Terminals:
Terminals fall into roughly three categories: hardcopy terminals which print directly on paper, video display terminals which have a television-like screen, and graphics terminals which are capable of drawing images and which usually have a hardcopy device attached to them to allow the user to make hardcopies of the material shown on the terminal screen.

Connecting a Terminal to the DCN:
On the DCN terminals can be connected to any of the five VAXs in one of two ways: through a wire which goes directly from the terminal into the computer or through a telephone modem to which the terminal is connected.

In order for the terminal to communicate with the computer it is necessary that:

- It be turned on; make sure that the terminal has been plugged in and that the power is on.
- If the terminal has a "local/remote" switch or button, make sure that it is set to "remote."
- If you are using a modem to communicate with the computer, make sure that it is plugged in and turned on.

Keyboards:
The keyboards on the interactive terminals are similar to typewriter keyboards in the placement of the alphabetic and numeric keys. Because the minicomputer is capable of performing many more functions than the standard typewriter, the keyboards of the interactive terminals contain special keys which can be used in conjunction with the standard keys to produce needed effects.

Special Characters:
The table below lists the special keys and their functions:

Special Keys	
Keys	Effect
Return	Sends a line to the computer for processing
Shift Rub Rub Out Delete	Erases characters, one character at a time
Control U	Erases an entire line
Control R	Reprints the line being typed
Control S	Suspends printing
Control Q	Resumes printing
Control O	Terminates printing
Control Z	Stops a running program
Control Y	Aborts a running program
Control C	Aborts a command

Logging on to the DCN:

Before it is possible for the user to issue commands to the system, it is necessary to log in. This is done by giving the system a valid *user name* and a valid *password*. The user name must be unique among all the user names that the system knows. Frequently user's last names are assigned as their user name. Passwords are used to keep others from gaining illegal access to a user's account and should be kept private.

Because the password is the first level of security associated with VMS, the operating system attempts to maintain that security by not printing the password on the terminal as it is being entered.

The following illustrates the log in sequence. (The user's entries are shown in italic.)

<RETURN>

USERNAME: *virginia <RETURN>*
PASSWORD: <RETURN>

Welcome to VAX/VMS Version 2.4 on node _RIX::
RIX VAX-11/780 Machine

$

The notation *<RETURN>* indicates that it is necessary to press the *return* key to alert the system to the user's presence. The return key is preseed whenever a line has been completed and is ready to be sent to the computer. However, the rest of this documentation will not show the *<RETURN>* at the end of each line.

After the user has been logged in, the system prints out a welcome, typically followed by a "message of the day."

The $ (dollar sign) is the VMS operating system "prompt;" the prompt is issued by the system whenever it is ready for input from the user. The portion of the operating system which listens to the user is often called DCL (Digital Command Language Interpreter). At this point in the example the system is ready to accept commands from the user.

VMS Directory Structure:

Files:
All data belonging to a user is kept in *files*. No data exists on the machine unless it is in a file.

These files can be located on a disk, or a magnetic tape. For the purposes of this presentation, we will assume that all files exist on disks.

Disks
A disk is a set of magnetic platters resembling large bronze phonograph records. Twelve disk platters are stacked one atop the other in a disk pack. This disk pack is then mounted on a disk drive. Information is written on both sides of the platters in the disk drive. Each node on the DCN has a minimum of two disks.

Directories
Since a disk can contain files belonging to many different users, each disk has a set of files called directories. A directory is simply a catalog of the files on the disk that belong to a particular user.

File Specification:
Each disk, directory, and file on the DCN has a name by which it can be accessed by the user.

The combination of disk name, directory name, and file name is known as a *file specification.*

Defaults:
When a user logs onto a DCN machine, the operating system looks at the user name and *defaults* to a specific directory on a specific disk based on that user name. Any commands which access files, or request information about a directory will be performed on this *default directory* unless otherwise specified.

In the following example, the command *directory* will display contents of the default directory for the user *virginia.* Because the directory is a catalog of file names, only the name and not the contents of the files will be displayed.

The most common analogy given by computer users for the combination of disks, directories, and files is that of the office file cabinet: Each office typically has one or more file cabinets; each file cabinet contains four drawers; each drawer contains several folders. Each cabinet, drawer, and folder is labeled to identify the contents.

```
$ directory

Directory DRC0:[VIRGINIA]

ADMIN.MEM;1        PERSONNEL.DAT;1    PERSONNEL.DOM;1    PERSONNEL.IDX;1
PERSONNEL.REC;1    REPORT.DAT;3       REPORT.DAT;2       REPORT.DAT;1

Total of 8 files.

$
```

File Names:
A VMS file name has three parts as illustrated below using the file ADMIN.MEM;1 as an example:

ADMIN	The file name - The name the user chooses for the file should in some way describe the contents of the file;
MEM	The file extension - The file extension describes the type of information contained in the file. The file name and extension are separated by a period (.)
1	indicates the file version. Most programs, with notable exceptions, make a new copy of the file whenever they are run. The extension and version number are separated by a semi-colon (;) You can usually omit the semicolon and version number from a file name.

There are three rules which the user must follow when naming a new file. They are:

Names	Can be one to nine characters in length. Either alphabetic characters or numeric characters can be used.
Extensions	Must be three characters in length.
Versions	Version numbers are assigned by the system

A set of conventions have been developed for DCN file extensions. They are:

dat	Data
for	Fortran source code
seq	Sequential files
idx	Indexed files
def	Datatrieve record definitions
pro	Datatrieve procedures
dom	Datatrieve domains
log	Datatrieve logs
scr	Chart scripts
mem	Memos
let	Letters

While VMS will accept any three character combination as a file extension, it is in the best interest of the user to assign descriptive names and extensions. *Letter.let* may mean something to the user today, but more than likely will be less than clear three months from now.

Using File Specifications:
As explained above, a file specification is the complete address of a file, including the disk, directory, and file names.

If the user is working in his default disk and directory (as is usually the case) VMS does not require that the disk and directory names be specified when accessing a file.

To make file access easier for the user, VMS employs the concept of *wildcards*. A wildcard is a notation used to match any of the three parts of the file name. The symbol the user uses to indicate a wildcard is the asterisk (*).

Example:

To illustrate the use of wildcards we will use them with the directory command.

(1) Notice that the following directory command can be used to obtain information on the files *REPORT.DAT;3, REPORT.DAT;2,* and *REPORT.DAT;1* in the default directory *virginia* by listing all the file names separated by commas:

$ directory report.dat;1,report.dat;2,report.dat;3

Directory DRC0:[VIRGINIA]

REPORT.DAT;1 REPORT.DAT;2 REPORT.DAT;3

Total of 3 files.

---

(2) This operation can be simplified by using a wildcard as follows:

---

*$ directory report.**

Directory DRC0:[VIRGINIA]

REPORT.DAT;3 REPORT.DAT;2 REPORT.DAT;1

Total of 3 files.

Notice that all files which begin with the name "REPORT" are listed.

(3) The following example illustrates that wildcards can be used at any point in the file name, and will match any set of characters:

---

*$ directory *.dat*

Directory DRC0:[VIRGINIA]

PERSONNEL.DAT;1 REPORT.DAT;3 REPORT.DAT;2 REPORT.DAT;1

Total of 4 files.

The Directory Command:

The *directory command* is used to examine the attributes of files contained in the user's directory. These attributes include file name, file size, date of file creation, and file type.

Examples of this command's most typical uses are shown below.

By issuing the *directory* command without any qualifiers will produce a list of file names for the default directory.

$ *directory*

Directory DRC0:[VIRGINIA]

ADMIN.MEM;1	PERSONNEL.DAT;1	PERSONNEL.DOM;1	PERSONNEL.IDX;1
PERSONNEL.REC;1	REPORT.DAT;3	REPORT.DAT;2	REPORT.DAT;1

Total of 8 files.

The *directory/size* command produces a list of file names and file sizes for the files in the default directory:

$ *directory/size*

Directory DRC0:[VIRGINIA]

ADMIN.MEM;1	1
PERSONNEL.DAT;1	10
PERSONNEL.DOM;1	1
PERSONNEL.IDX;1	30
PERSONNEL.REC;1	10
REPORT.DAT;3	5
REPORT.DAT;2	10
REPORT.DAT;1	15

Total of 8 files, 82 blocks.

The *directory/date* command produces a list of file names and and the date of file creation for the files in the default directory:

$ *directory/date*

Directory DRC0:[VIRGINIA]

```
ADMIN.MEM;1          26-APR-1982 15:41
PERSONNEL.DAT;1      26-APR-1982 15:43
PERSONNEL.DOM;1      26-APR-1982 15:42
PERSONNEL.IDX;1      26-APR-1982 15:42
PERSONNEL.REC;1      26-APR-1982 15:42
REPORT.DAT;3         26-APR-1982 15:43
REPORT.DAT;2         26-APR-1982 15:43
REPORT.DAT;1         26-APR-1982 15:43
```

Total of 8 files, 8 blocks.

The *directory/full* command produces a complete list of file attributes for the files in the default directory:

$ *directory/full*

Directory DRC0:[VIRGINIA]

```
ADMIN.MEM;1   Size:      1/5        Created: 26-APR-1982 15:41
              Owner:     [213,036]  Revised: 26-APR-1982 15:41 (1)
              File ID:   (11368,7,0) Expires: <None specified>
                                     Backup:  <No backup done>

File protection:      System:RWED, Owner:RWED, Group:RWED, World:RE
File organization:    Sequential
File attributes:      Allocation = 5, Extend = 0
Record format:        Variable length
Record attributes:    Carriage return
```

(Note that all file attributes for all files are not shown in this example in the interest of space.)

File Manipulation

Examining the Contents of a File:

There are three ways in which to examine the contents of a file: editing a file, *typing* a file, and *printing* a file.

The use of the editor will be explained later in this document.

The Type Command
The *type* command followed by a file name is used to display the contents of a file on a terminal. It displays the entire file. As an example, the command *type admin.mem* displays the contents of the file ADMIN.MEM.

$ type admin.mem

The Computer Science and Mathematics Department conducts basic and applied research in a variety of computer science and mathematics disciplines. The research program encompasses four basic research areas: (1) advanced analytical and numerical methods for solving partial differential equations related to energy technology; (2) innovative approaches to the manipulation, analysis, and display of large, complex data sets; (3) investigation of advanced computer concepts, distributed architectures, and computer networks; and (4) the development and evaluation of analytical and statistical methodologies and techniques used for specific epidemiological studies investigating the impact of environmental pollution upon human health.
$

The Print Command:
The *print* command followed by a file name will send a copy of the file to a line printer. Obviously, this command is only useful if the line printer is within walking distance of the user. It is used as follows:

$ print admin.mem
Job 1019 entered on queue SYS$PRINT
$

The print command has routed the file to the line printer and it is waiting in a queue for its turn to be printed.

Removing Files from a Directory:

Deleting Files
Users will frequently find that they no longer have use for a file which resides in their directory. The *delete* command is used to remove unwanted files. It should be noted that once a file has been deleted it is no longer accessible by the system unless a copy has been put on tape as a backup by the DCN system manager.

The delete command requires that the full file name be used. Wild cards can be used as part of the file name.

$ delete report.dat;3

$ directory

Directory DRC0:[VIRGINIA]

ADMIN.MEM;1	PERSONNEL.DAT;1	PERSONNEL.DOM;1	PERSONNEL.IDX;1
PERSONNEL.REC;1	REPORT.DAT;2	REPORT.DAT;1	

Total of 7 files.

Purging Files:
The *purge* command will remove redundant versions of files from the user's directory: (In this example the reader should assume that the contents of the directory before the purge are as they are shown on page 10.)

$ purge report.dat

$ directory

Directory DRC0:[VIRGINIA]

ADMIN.MEM;1	PERSONNEL.DAT;1	PERSONNEL.DOM;1	PERSONNEL.IDX;1
PERSONNEL.REC;1	REPORT.DAT;3		

Total of 6 files.

Notice that the highest version (latest version) of the file is the one that is kept.

Copying Files

A copy of a file can be made using the *copy* command.

Copies of a file can be given the same name as the original file, or they can be assigned a new name. If the same name is used, VMS creates a new, higher version number for the file.

In the example below a directory command is issued; the copy command is then issued to copy the file *REPORT.DAT* to a new version of *REPORT.DAT*

$ directory

Directory DRC0:[VIRGINIA]

ADMIN.MEM;1	PERSONNEL.DAT;1	PERSONNEL.DOM;1	PERSONNEL.IDX;1
PERSONNEL.REC;1	REPORT.DAT;3		

Total of 6 files.

$ copy
$_From: *report.dat*
$_To: *report.dat*

$ directory

Directory DRC0:[VIRGINIA]

ADMIN.MEM;1	PERSONNEL.DAT;1	PERSONNEL.DOM;1	PERSONNEL.IDX;1
PERSONNEL.REC;1	REPORT.DAT;4	REPORT.DAT;3	

Total of 7 files.

Notice that the new copy of *REPORT.DAT* was assigned the next highest version number.

In this example a copy of the file *REPORT.DAT* is made on a file named *NEWREPORT.DAT*

$ copy

$_From: *report.dat*
$_To: *newreport.dat*

$ directory

Directory DRC0:[VIRGINIA]

ADMIN.MEM;1	NEWREPORT.DAT;1	PERSONNEL.DAT;1	PERSONNEL.DOM;1
PERSONNEL.IDX;1	PERSONNEL.REC;1	REPORT.DAT;4	REPORT.DAT;3

Total of 8 files.

Editing Files:

Text editors are the most frequently used program on any interactive computing system. They are used to put programs, letters, reports, etc. into the machine. There are four text editors on the DCN; they are EDT, SOS, TECO, and the Software Tools editor, ED. The Software Tools editor will be shown during this presentation.

Basic Editing Operations:

Three basic editing operations are illustrated in this document. They are: adding text to a file, deleting text from a file, and changing text in an existing file.

The following table lists the basic editor commands.

Command	Function
a	Append text
i	Insert text
d	Delete line
p	Print line
<return>	Print next line
c	Change line
s	Substitute existing text with new text
//	Find text after current line
\\	Find text before current line
•	Terminate appending or inserting text
w	Write file to disk (save changes)
q	Quit the editor (use w followed by q to save changes)

Sample File

The following paragraph will be used to demonstrate the editor:

> The **Compter** Science and Mathematics Department
> conducts basic and applied research in a
> variety of computer science and mathematics disciplines.
> The research **progrm** encompasses four basic
> research areas: (1) advanced analytical and
> numerical methods for solving partial differential
> equations related to energy technology;
> (**2**(innovative approaches to the manipulation,
> analysis, and **disply** of large, complex data sets;
> (3) investigation of advanced computer concepts,
> distributed architectures, and computer networks;
> and (4) the **developmnt** and evaluation of
> analytical and statistical methodologies and
> techniques used for specific epidemiological
> studies investigating **th** impact of environmental
> pollution upon human health.

Notice that several typographical errors exist in the file and are printed in bold face.

Positioning:

The editor commands // and \\ are used to locate text within a file. The command // searches downward through the file while the command \\ searches upward through the file.

In the following example, the text **progrm** is located with the command **/progrm/**

/progrm/

```
  The Compter Science and Mathematics Department
  conducts basic and applied research in a
  variety of computer science and mathematics disciplines.
☞ The research progrm encompasses four basic
  research areas: (1) advanced analytical and
  numerical methods for solving partial differential
  equations related to energy technology;
  (2( innovative approaches to the manipulation,
  analysis, and disply of large, complex data sets;
  (3) investigation of advanced computer concepts,
  distributed architectures, and computer networks;
  and (4) the developmnt and evaluation of
  analytical and statistical methodologies and
  techniques used for specific epidemiological
  studies investigating th impact of environmental
  pollution upon human health.
```

The little hand in the example above points to the line on the file that the command **/progrm/** located. That line is now the **current** line. If the **p** command is given, it will print that line. If a substitute command is given it will change that line, and only that line; if text is added to the file it will be added after that line; if a command to delete a line is given, that line will be deleted.

Simple Substitutions:

Substitutions are used to change text. The syntax, or way in which the user requests the change is: **s/oldthing/newthing/p**

In English, this command reads: substitute the old thing with the new thing and print the line after the substitution has been made.

It is important that the line on which the error is to be corrected be the **current** line.

The following example illustrates this command:

s/Compter/Computer/p
The Computer Science and Mathematics Department

In the next set of examples the use of the //, \\, and s/oldthing/newthing/p are shown in combination:

/Compter/

☞ The **Compter** Science and Mathematics Department conducts basic and applied research in a variety of computer science and mathematics disciplines. The research **progrm** encompasses four basic research areas: (1) advanced analytical and numerical methods for solving partial differential equations related to energy technology; (**2**(innovative approaches to the manipulation, analysis, and **disply** of large, complex data sets; (3) investigation of advanced computer concepts, distributed architectures, and computer networks; and (4) the **developmnt** and evaluation of analytical and statistical methodologies and techniques used for specific epidemiological studies investigating **th** impact of environmental pollution upon human health.

s/Compter/Computer/p

☞ The **Computer** Science and Mathematics Department conducts basic and applied research in a variety of computer science and mathematics disciplines. The research **progrm** encompasses four basic research areas: (1) advanced analytical and numerical methods for solving partial differential equations related to energy technology; (**2**(innovative approaches to the manipulation, analysis, and **disply** of large, complex data sets; (3) investigation of advanced computer concepts, distributed architectures, and computer networks; and (4) the **developmnt** and evaluation of analytical and statistical methodologies and techniques used for specific epidemiological studies investigating **th** impact of environmental pollution upon human health.

/progrm/

The **Computer** Science and Mathematics Department conducts basic and applied research in a variety of computer science and mathematics disciplines.
☞ The research **progrm** encompasses four basic research areas: (1) advanced analytical and numerical methods for solving partial differential equations related to energy technology; (**2**(innovative approaches to the manipulation, analysis, and **disply** of large, complex data sets; (3) investigation of advanced computer concepts, distributed architectures, and computer networks; and (4) the **developmnt** and evaluation of analytical and statistical methodologies and techniques used for specific epidemiological studies investigating **th** impact of environmental pollution upon human health.

s/progrm/program/p

The **Computer** Science and Mathematics Department conducts basic and applied research in a variety of computer science and mathematics disciplines.
☞ The research **program** encompasses four basic research areas: (1) advanced analytical and numerical methods for solving partial differential equations related to energy technology; (**2**(innovative approaches to the manipulation, analysis, and **disply** of large, complex data sets; (3) investigation of advanced computer concepts, distributed architectures, and computer networks; and (4) the **developmnt** and evaluation of analytical and statistical methodologies and techniques used for specific epidemiological studies investigating **th** impact of environmental pollution upon human health.

/(2(/

The **Computer** Science and Mathematics Department conducts basic and applied research in a variety of computer science and mathematics disciplines. The research **program** encompasses four basic research areas: (1) advanced analytical and numerical methods for solving partial differential equations related to energy technology;
☞ (**2**(innovative approaches to the manipulation, analysis, and **disply** of large, complex data sets; (3) investigation of advanced computer concepts, distributed architectures, and computer networks; and (4) the **developmnt** and evaluation of analytical and statistical methodologies and techniques used for specific epidemiological studies investigating **th** impact of environmental pollution upon human health.

s/(2(/(2)/p

The **Computer** Science and Mathematics Department conducts basic and applied research in a variety of computer science and mathematics disciplines. The research **program** encompasses four basic research areas: (1) advanced analytical and numerical methods for solving partial differential equations related to energy technology;
☞ (**2**) innovative approaches to the manipulation, analysis, and **disply** of large, complex data sets; (3) investigation of advanced computer concepts, distributed architectures, and computer networks; and (4) the **developmnt** and evaluation of analytical and statistical methodologies and techniques used for specific epidemiological studies investigating **th** impact of environmental pollution upon human health.

/disply/

The **Computer** Science and Mathematics Department
conducts basic and applied research in a
variety of computer science and mathematics disciplines.
The research **program** encompasses four basic
research areas: (1) advanced analytical and
numerical methods for solving partial differential
equations related to energy technology;
(2) innovative approaches to the manipulation,
☞ analysis, and **disply** of large, complex data sets;
(3) investigation of advanced computer concepts,
distributed architectures, and computer networks;
and (4) the **developmnt** and evaluation of
analytical and statistical methodologies and
techniques used for specific epidemiological
studies investigating **th** impact of environmental
pollution upon human health.

s/disply/display/p

The **Computer** Science and Mathematics Department
conducts basic and applied research in a
variety of computer science and mathematics disciplines.
The research **program** encompasses four basic
research areas: (1) advanced analytical and
numerical methods for solving partial differential
equations related to energy technology;
(2) innovative approaches to the manipulation,
☞ analysis, and **display** of large, complex data sets;
(3) investigation of advanced computer concepts,
distributed architectures, and computer networks;
and (4) the **developmnt** and evaluation of
analytical and statistical methodologies and
techniques used for specific epidemiological
studies investigating **th** impact of environmental
pollution upon human health.

Adding Text to a File:

Text can be added to an existing file with either of two commands:

a for append
i for insert

While both commands will allow the user to add new text, their important difference lies in the positioning of the new text.

The **a** command will add text **after** the current line.

The **i** command will add text **before** the current line.

The following example illustrates the **a** command. The current line is marked by the hand while the new line is marked by the dot.

```
a
The CSAM department is part of Lawrence Berkeley Laboratory.
•
```

```
  The Compter Science and Mathematics Department
  conducts basic and applied research in a
☞ variety of computer science and mathematics disciplines.
• The CSAM department is part of Lawrence Berkeley Laboratory.
  The research progrm encompasses four basic
  research areas: (1) advanced analytical and
  numerical methods for solving partial differential
  equations related to energy technology;
  (2( innovative approaches to the manipulation,
  analysis, and disply of large, complex data sets;
  (3) investigation of advanced computer concepts,
  distributed architectures, and computer networks;
  and (4) the developmnt and evaluation of
  analytical and statistical methodologies and
  techniques used for specific epidemiological
  studies investigating th impact of environmental
  pollution upon human health.
```

The following example shows the use of the **i** command. Again the current line is marked by the hand and the new line is marked by the dot.

```
i
The CSAM department is part of Lawrence Berkeley Laboratory.
•
```

```
•   The CSAM department is part of Lawrence Berkeley Laboratory.
☞   The Compter Science and Mathematics Department
    conducts basic and applied research in a
    variety of computer science and mathematics disciplines.
    The research progrm encompasses four basic
    research areas: (1) advanced analytical and
    numerical methods for solving partial differential
    equations related to energy technology;
    (2( innovative approaches to the manipulation,
    analysis, and disply of large, complex data sets;
    (3) investigation of advanced computer concepts,
    distributed architectures, and computer networks;
    and (4) the developmnt and evaluation of
    analytical and statistical methodologies and
    techniques used for specific epidemiological
    studies investigating th impact of environmental
    pollution upon human health.
```

Terminating the a and i Commands:

Any text typed by the user after the **a** and **i** commands will be added to the file.

You can add as many lines of input as you want.

A period (.) as the first character on a line, followed immediately by a carriage return signals the end of input.

Deleting Text:

The **d** command is used to delete a line of text. It is necessary that the current line be the one to be deleted.

The following examples illustrate the positioning associated with deleting lines.

In the first example the hand points to the current line before the delete command is issued.

In the second example the hand points to the new current line.

```
  The Compter Science and Mathematics Department
  conducts basic and applied research in a
  variety of computer science and mathematics disciplines.
☞ The CSAM department is part of Lawrence Berkeley Laboratory.
  The research progrm encompasses four basic
  research areas: (1) advanced analytical and
  numerical methods for solving partial differential
  equations related to energy technology;
  (2( innovative approaches to the manipulation,
  analysis, and disply of large, complex data sets;
  (3) investigation of advanced computer concepts,
  distributed architectures, and computer networks;
  and (4) the developmnt and evaluation of
  analytical and statistical methodologies and
  techniques used for specific epidemiological
  studies investigating th impact of environmental
  pollution upon human health.
```

d

```
  The Compter Science and Mathematics Department
  conducts basic and applied research in a
  variety of computer science and mathematics disciplines.
☞ The research progrm encompasses four basic
  research areas: (1) advanced analytical and
  numerical methods for solving partial differential
  equations related to energy technology;
  (2( innovative approaches to the manipulation,
  analysis, and disply of large, complex data sets;
  (3) investigation of advanced computer concepts,
  distributed architectures, and computer networks;
  and (4) the developmnt and evaluation of
  analytical and statistical methodologies and
  techniques used for specific epidemiological
  studies investigating th impact of environmental
  pollution upon human health.
```

Notice that the line following the deleted line becomes the new current line.

The commands **d** and **p** can be used in combination to delete the current line and print the new current line. This is done by typing the command as

dp

Leaving the Editor:

There are two ways in which to leave the editor. The first keeps the changes made to the file; the second throws them away.

Saving Changes:

To leave the editor and make any changes made to the file permanent, use the **w** command. (The w is an abbreviation for write.)

After this command is issued, the editor will print out the number of lines currently contained in the file.

Once the **w** command has been issued the user terminates the editing session by typing the command **q.**

Discarding Changes:

To leave the editor without keeping the changes made during the editing session, use the **q** command without preceding it by the **w** command.

If you issue the **q** command without first writing the file with the **w** command, the editor will request that you confirm that you do not wish to keep changes made to the file.

Accessing VAX-11 Datatrieve

VAX-11 Datatrieve (also known as DTR32) is accessible either through a DCL command procedure or at the DCL command level. Most users will access the Datatrieve at the DCL command level.

As seen in the previous sections, DCL command level is indicated by a dollar sign ($). When this prompt is received at the terminal, Datatrieve can be accessed by typing *DTR32,* as in

$ DTR32

Datatrieve announces itself by typing:

VAX-11 Datatrieve T1.2-2
DEC Query and Report System
Type HELP for help
DTR>

Notice that the Datatrieve prompt is *DTR>*.

At this point commands can be given to Datatrieve.

Leaving Datatrieve

Terminating a Datatrieve session is done by typing the command *exit.*

When a Datatrieve session is terminated, all readied domains are closed; additionally, any collections or declared variables are deleted.

The Help Command

On-line help is available within Datatrieve. Typically the information given in the on-line help is intended to be a quick-reference guide to the most frequently used commands. It certainly should not be relied upon to act as a comprehensive resource for information.

The following examples show various forms of the help command. The user's input is shown in italic.

DTR> *help*

HELP

To get help for DATATRIEVE type HELP followed by the names of the statement or topic for which help is required:

HELP topic [subtopic]...

For more information about HELP available with Datatrieve type HELP HELP.

Additional information available:

ABORT	ADT	ASSIGNMENT	BEGIN_END	BOOLEAN	CLOSE
CROSS	DBMS	DECLARE	DEFINE	DEFINEP	DELETE
DICTIONARY	DISPLAY	DROP	EDIT	ERASE	EXIT
FIND	FINISH	FOR	FORMS	FUNCTIONS	GUIDE
IF	LIST	MISSING	MODIFY	OPEN	
PLOT	PRINT	PROCEDURE	QUIT	READY	RELEASE
REPORT	RESTRUCTURE	RSE	SEARCH	SELECT	
SHOW	SHOWP	SORT	STARTUP	STORE	SUM
THEN	VALUE	WHILE	WOMBAT		

DTR> *help print*

PRINT

The PRINT statement is used to print fields from records. The form of the statement is:

PRINT [ALL] [print_list] [OF rse] [ON output_file]

where: print_list if it is present, it lists fields and computations to be printed, separated by commas. If a print_list is not specified, all fields contained in the record are printed.

rse is a record selection expression (see HELP RSE)

file_name is a file name to which the output is directed.

Additional information available:

INNER_PRINT_LIST PRINT_LIST Examples

DTR> *help print inner-print-list*

PRINT

INNER_PRINT_LIST

An inner print list can be used anywhere a print list item is used.
The format of an inner print list is:

ALL print_list OF rse

This is especially useful for printing lists from hierarchical records. For example, to print the name of the father and each of his children from the hierarchical family record, we would use:

PRINT FATHER, ALL KID_NAME OF KIDS OF FAMILIES

The Common Data Dictionary

Datatrieve uses the Common Data Dictionary as a central storage facility for the definitions of domains, records, procedures, and tables.

The CDD has a hierarchical structure similar to that of VAX/VMS subdirectories.

The hierarchical structure of the CDD allows the user of a VAX/VMS system to centralize data definitions, to make logical divisions among those data definitions, and to provide a central mechanisms to control access to them.

When combined in one category, the data descriptions stored in the CDD are called *dictionary objects.*

The definitions of domains, records, procedures, and tables are types of dictionary objects that Datatrieve can store in and retrieve from the CDD.

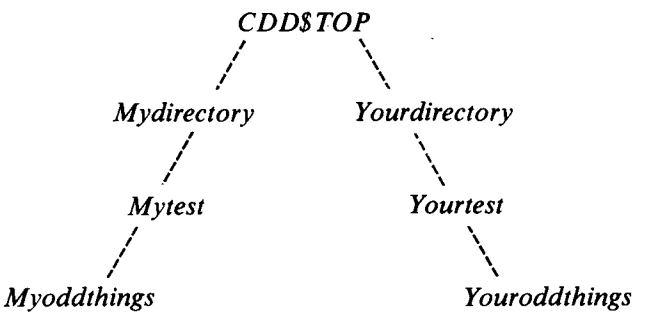
The catalogs of these dictionary objects are called *dictionary directories.*

The CDD is a hierarchy of dictionary directories and dictionary objects.

The top of the CDD hierarchy is the directory named *CDD$TOP.* While dictionary objects can be stored in this directory, it is recommended that user's define their own directories as subdirectories of this directory.

Accessing CDD directories and subdirectories is described in the next section.

The following diagram illustrates the hierarchical structure of the Common Data Dictionary:



The Set Dictionary Command

Whenever you enter Datatrieve, you are automatically positioned in the default directory called "CDD$TOP." This is the top level directory of the Common Data Dictionary.

The command to move between CDD directories is *set dictionary*.

Two very simplistic rules for moving between dictionaries are presented here. If you wish to examine the more detailed information provided by the Datatrieve documentation refer to Pages 2-1 through 3-24 in the Reference Manual.

(1) Relative path names can be used when moving down the CDD hierarchy. This is because each dictionary contains the names of its subdictionaries.

For example, assume that your directory (called "Yourdirectory") is a subdirectory of the top level directory (called "CDD$TOP").

You would then give the command

set dictionary yourdirectory to position yourself in that directory.

(2) Full path names must be used when moving up or across the CDD hierarchy. This is because dictionaries do not contain the names of their parents.

For example, assume that your directory (called "Yourdirectory") contains a sub-subdirectory called "Yourtest." Also assume that you are working in "Yourtest." If you wish to move back into "Yourdirectory" you must use the command

set dictionary cdd$top.yourdirectory

Notice that the symbol for separating node names is the period (.).

The Show Command

The *show* command is used to show

(1) the contents of the current CDD subdirectory;

(2) the status of the current Datatrieve session.

The following table contains a partial list of the *show* commands available to the user.

Command	Effect
Show all	Shows the names of all CDD objects in the current CDD subdirectory (referred to as the default directory), and the status of currently readied domains, names of current collections selected records, declared variables, readied tables;
Show records	Shows the names of all record definitions stored in the current CDD subdirectory;
Show domains	Shows the names of all domain definitions stored in the current CDD subdirectory;
Show collections	Shows the names of all collections; also shows any selected records in a collection;
Show tables	Shows the names of all table definitions stored in the current CDD subdirectory;
Show procedures	Shows the names of all procedure definitions stored in the current CDD subdirectory;
Show ready	Shows the names of any currently readied domains; also lists the type of access specified as well the type of file associated with the domain;
Show (object)	Shows the definitions of any objects stored in the current CDD subdirectory;
Show fields	Shows the names and data types of fields contained in any readied domain as well as which field in the record has been defined as a key field.

The Personnel Domain:

The following example illustrates the contents of the *Personnel Domain.*

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Susan	Anthony	$ 580.00	40	02	01
John	Bach	$ 450.00	40	03	02
Joe	Blow	$ 375.00	40	00	00
Humphrey	Bogart	$ 450.00	00	03	05
James	Bond	$ 480.00	40	00	00
Don	Bumble	$ 322.00	35	01	00
Fredric	Chopin	$ 460.00	40	02	02
Aaron	Copland	$ 525.00	35	02	00
Herbert	Hoover	$ 375.00	40	15	05
John	Lennon	$ 500.00	40	03	07
Judy	Brown	$ 400.00	30	20	10
Abraham	Lincoln	$ 600.00	40	00	00
Mary	Lamb	$ 500.00	40	05	00
Misty	Skies	$ 500.00	40	00	00
Stormy	Skies	$ 475.00	40	02	01
George	Washington	$ 520.00	40	00	01

As with the manual experiment conducted earlier, this *domain* is composed of three parts:

(1) A *name;*

(2) A *definition* for reading the *fields* on the records.

(3) A *file* containing the *records* of all the employees;

We will examine these three parts individually:

The Three Parts of a Datatrieve Domain

(1) The *Name:*

When you use Datatrieve you must give a name to the relationship that exists between the file containing your records and the definition used to read the fields on the records.

By giving a name to this relationship you have defined a Datatrieve *domain.*

(2) *The Definition* for Reading the Fields on the Records:

In the manual experiment the blue explanation sheet defined the contents of each of the records. You used this explanation sheet whenever you were asked to read data from those records.

When using Datatrieve, you must define the contents of the records for Datatrieve by creating a *record definition.*

Datatrieve will use this record definition whenever it accesses the records.

(3) The *File* Containing the Records:

In the manual experiment we kept all personnel records in a box. We called this box the "personnel file."

When you use Datatrieve you do not keep the records in a box; instead you have the computer put them in a file which is kept on the disk.

When using Datatrieve the structure of the file is determined by the record definition.

The following sections describe the procedures used to define the three parts of a Datatrieve domain.

The Datatrieve Record Definition

In the manual experiment conducted earlier, a sheet of blue paper defined contents of each of the records. That definition specified:

(4) The name of each field;

(5) What could be entered into each field;

The following several pages illustrate the way in which these records are defined in Datatrieve.

A Record Definition for the Personnel Domain:

Every record stored in the personnel domain will follow the specifications laid out in the record definition below.

The bullets in the example below indicate the minimum specifications required to define the records in the Personnel Domain.

```
• RECORD PERSONNEL_RECORD USING
• 01 EMPLOYEE-INFORMATION.
      02    NAME.
    • 03    FIRST_NAME              PIC X(15).
    • 03    LAST_NAME               PIC X(15).
      02    STATUS.
    • 03    SALARY                  PIC 9999V99
            EDIT-STRING IS $ZZZZ.99.
    • 03    HOURS_WORKED            PIC 99.
    • 03    VACATION_TAKEN          PIC 99.
    • 03    SICK_LEAVE_TAKEN        PIC 99.
• ;
```

The record definition is automatically installed in the Common Data Dictionary.

Specifying Datatrieve Fields In a Datatrieve Record Definition

Datatrieve supports two types of fields: group and elementary. Group fields are names assigned to groups of fields in order to simplify data access. Elementary fields are those individual fields which contain data.

To aid in understanding the concept of group and elementary fields, we show the following table:

EMPLOYEE-INFORMATION

NAME		STATUS			
FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Susan	Anthony	$ 580.00	40	02	01
John	Bach	$ 450.00	40	03	02
Joe	Blow	$ 375.00	40	00	00
Humphrey	Bogart	$ 450.00	00	03	05
James	Bond	$ 480.00	40	00	00
Don	Bumble	$ 322.00	35	01	00
Fredric	Chopin	$ 460.00	40	02	02
Aaron	Copland	$ 525.00	35	02	00
Herbert	Hoover	$ 375.00	40	15	05

This table contains three types of labeling information.

(1) A table heading: Employee Information.

When you refer to the table "employee information" the implication is that you are referring to all the columns in the table.

(2) Table subheadings: Name, Status.

When you refer to the subheading "name" the implication is that you are referring to the columns first name and last name. Likewise referring to the subheading "status" implies that you are referring to the columns listed under "status."

(3) Column headings: First Name, Last Name, Salary, Hours Worked, Vacation Taken, Sick Leave Taken. These headings refer uniquely to each column.

To relate the concept of a table to a record definition, we must understand the following:

(1) Just as every table must have a title (explicitly or otherwise) each record must have a *top-level group field.* The top-level group field serves the same function as the table title in that it refers to all the fields in the record.

In the example record definition, the top-level group field is named "Employee-Information."

(2) Just as tables can have subheadings which refer to groups of columns, records can have *group fields* which refer to groups of fields.

In the example record definition the two group fields are "Name," and "Status."

(3) Just as every column within a table must have a name, each *field* in a record must have a name.

(4) Information within a column in a table must have the same type of information on every row: A salary figure cannot be printed in the column "first name." The same applies within records in a Datatrieve domain.

The Components of a Record Definition:

A Datatrieve Record Definition is composed of four parts:

(1) A name,
(2) A top-level group field,
(3) Group or elementary field definition clauses, and a
(4) A semicolon to terminate the record definition.

In turn, each group or elementary field is composed of three or more parts:

(1) A level number:

Level numbers are used to define field hierarchies within the record. Field level numbers are assigned in descending order: those with lower numbers are higher in the hierarchy than those with higher numbers. This is consistent with the ordering of labels on tables as is shown below:

(Label Level 1) EMPLOYEE-INFORMATION					
(Label Level 2) NAME		STATUS			
(Label Level 3) FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN

(2) A field name:

A field name must be assigned to each group or elementary field. Datatrieve names (names of domains, tables, record definitions, fields, procedures, etc.) cannot contain spaces. In the case where you wish to assign a two or more word name you must connect the words with a hyphen (-) or an underscore (-).

Datatrieve names must be fewer than 32 characters in total length.

Note: This does *not* mean that the data stored in domains cannot contain spaces; it only means that *names* must not contain spaces.

(3) A field definition clause:

Field definition clauses define the type and size of the data values to be stored in a given field. There are three types of field definition clauses: those which describe alphanumeric or numeric data, those which are stored as floating point or packed decimal, and those which rather than being stored, are computed at the time of access.

In this introduction we describe the first type: alphanumeric and numeric data: Picture. A detailed description of the picture clause is given in the next section.

(4) An optional set of field qualifiers:

Field qualifiers are used to alter either the labeling or format of the data at print time, and to provide synonyms for field names which can be used when querying the domain.

A list of field qualifiers appears at the end of this section.

(5) A period to end the field definition:

Each field definition must be terminated with a period (.). The period must follow the last descriptive statement regarding the field.

Group fields consist of only a level number and a field name. The period would then follow the field name.

Elementary fields consist of a level number, a field name, a field definition clause, and (optionally) one or more field qualifiers. The period would follow the field definition clause, or, in the case where field qualifiers are used, after the last field qualifier.

An Annotated Record Definition:

The following example contains an annotated record definition for the personnel domain:

```
RECORD PERSONNEL_RECORD (1) USING

01 EMPLOYEE-INFORMATION.(2)
      02(3)   NAME(4).
      03      FIRST_NAME(5).            PIC X(15)(6).(7)
      03      LAST_NAME                 PIC X(15).
      02      STATUS.
      03      SALARY                    PIC 9999V99
              EDIT-STRING IS $ZZZZ.99(8).
      03      HOURS_WORKED              PIC 99.
      03      VACATION_TAKEN            PIC 99.
      03      SICK_LEAVE_TAKEN          PIC 99.
;(9)
```

Key:

(1) Record name
(2) Top-level group field
(3) Level number
(4) Group field name
(5) Elementary field name
(6) Field definition clause
(7) Period ending field definition
(8) Field qualifier
(9) Semicolon ending record definition

Field Qualifier and Edit-string Tables:

Edit-strings are field qualifiers which alter the format of the data at print time. They can be included within the record definition (as shown in the above example) or as part of the "print" command. The following table lists the most commonly used edit-string characters.

Edit-String Characters:

Edit-string Character	Effect
.	A decimal point is placed in that character position
,	A comma is placed in that character position
$	A dollar sign is placed in that character position
Z	If a Z matches a leading zero in the field it is replaced by a space. If not, Z is replaced by a digit from the field's content.
9	Each 9 is replaced by one digit from the field's content. The leading character positions are filled with zeros.
X(number)	The number of spaces indicated in the edit-string will be used to print the contents of the field.

Field Qualifiers:

Field qualifiers are used to alter either the labeling or format of the data at print time and to provide synonyms for field names which can be used when querying the domain.

The following table lists the most commonly used field qualifiers.

Qualifier	Effect
query-header is "string"	The set of characters inside the quotes would be used to label each field at print time.
query-name is string	The set of characters specified could be used instead of the field-name at access time. Note: Quotation marks are not allowed on query-names.
edit-string is	The format specified in the edit-string would be used when the data is formatted at print time. Examples include:
edit-string is zzz9	Suppress leading zeros at print time (z). Print any digits present in the field value in the last position, even if it is a zero (9).
edit-string is zzz9.99	Suppress leading zeros at print time (z). Print any digits present in the field value of the last position. Display two digits of accuracy to the right of the decimal point.
edit-string is x(10)	Allow ten character spaces for alphanumeric field values at print time.

Field Definition Clause:

Two field definition clauses are described in this introductory document: Picture (or pic), and usage.

(1) *The Picture Clause:*

In the example record definition each elementary field is defined with a picture clause. There are two types of picture clauses: Alphanumeric which contain any of the characters on the terminal keyboard, and numeric which contain numbers, more specify numbers on which calculations will be performed.

(While an employee identification number is a number, you probably would not total them up to print a grand total of employee numbers.)

Numeric fields may contain signed numbers (+ or -).

The following table shows several examples of valid picture clause fields:

Picture Clause	Meaning
PIC X	Alphanumeric field; Maximum size: one character
PIC XXXXX	Alphanumeric field; Maximum size: five characters
PIC X(15)	Alphanumeric fields; Maximum size: fifteen characters
PIC 9	Numeric field; Maximum size: one character
PIC 99999	Numeric field; Maximum size: five characters Negative numbers cannot be stored
PIC 9(5)	Numeric field; Maximum size: five characters Negative numbers cannot be stored
PIC S9(10)	Numeric field; Maximum size ten characters; Negative numbers can be stored
PIC S9(10)V99	Numeric field; Maximum size twelve characters; Negative numbers can be stored; Numbers stored in this field will have ten places to the left of an implied decimal point and two to the right

Numeric fields are automatically right justified while alphanumerics are left justified unless specified otherwise via an edit-string.

(2) *The Usage Clause*

The usage clause is used to specify the way in which data will be stored in a field. While Datatrieve uses several different usage clauses, only one will be described here and that is the "usage is date" clause.

Usage is Date:
The "usage is date" clause is used to describe dates; you can store dates between November 17, 1858 and December 31, 9999. These values of these dates can then be used in some types of computation. For example, you can subtract one date from another to get the number of days elapsed between the two dates, or add ten to an existing date to calculate deadline dates. For further discussion see page 11-23 of the Datatrieve manual.

Datatrieve allows dates to be printed using several types of output formats. It also allows dates to be entered in several different ways (3-Mar-81, March 3, 1981, 3/3/81, etc.)

When specifying that a field's contents are to be stored as a date field you do not specify the size of the field (PIC).

Syntax for Defining a Datatrieve Record Definition

The syntax to install a Datatrieve record definition in the Common Data Dictionary is:

```
DEFINE RECORD PERSONNEL_RECORD USING
01 EMPLOYEE-INFORMATION.
    02  NAME.
    03  FIRST_NAME              PIC X(15).
    03  LAST_NAME               PIC X(15).
    02  STATUS.
    03  SALARY                  PIC 9999V99
        EDIT-STRING IS $ZZZZ.99.
    03  HOURS_WORKED            PIC 99.
    03  VACATION_TAKEN          PIC 99.
    03  SICK_LEAVE_TAKEN        PIC 99.
;
```

If the record definition has been correctly defined, Datatrieve will respond by typing the record length (in bytes) at the terminal.

If the record definition has not been correctly defined, Datatrieve will list out the part of the definition that is correct, following by the line on which the error occurs. An arrow will be positioned at the point at which Datatrieve thinks the error occurred.

The Datatrieve Domain Definition

A Datatrieve domain definition contains three things:

(1) A *name* to be assigned to the domain in the Common Data Dictionary;

(2) The name of the *record definition* to be used when reading fields on the records;

(3) The name of the *file* on which to store records and from which to retrieve records.

Syntax for Defining a Datatrieve Domain

The syntax used to install a Datatrieve domain definition in the Common Data Dictionary is:

DEFINE DOMAIN personnel USING personnel-record ON personnel.idx;

Defining a Datatrieve File

A Datatrieve file definition contains:

(1) The name of the *domain* for which the file is being defined;

(2) The field in the record to be used as the index *key;*

(3) A specification for whether the key field is to allow *duplicate* entries.

If items 2 and 3 are not specified when the file is defined, the file will automatically become a sequential file.

If items 2 and 3 are specified when the file is defined, the file will automatically become an indexed file.

File Restrictions:

(1) Records cannot be removed from a *sequential* file; this is a VMS restriction.

(2) The entries in a primary key field on an indexed file cannot be changed; this also is a VMS restriction.

(3) If the duplication specification is not set (Item 3), key field values must be unique.

Syntax for Defining a Datatrieve File

Examples of the syntax used to define a Datatrieve file are:

DEFINE FILE personnel;

(Creates sequential file for the personnel domain)

DEFINE FILE personnel key = last-name;

(Creates an indexed file for the personnel domain; duplicates are not allowed)

DEFINE FILE personnel key = last-name (dup);

(Creates an indexed file for the personnel domain; duplicates are allowed)

The Datatrieve Editor

The Datatrieve editor provides a convenient way to edit

(1) Datatrieve commands:

Datatrieve maintains a command buffer which contains the last command or portion of a syntactically incorrect command entered.

This command, or part of a command, is then available for editing.

When the editor command *exit* is issued, the contents of the command buffer are executed.

Issuing the editor command *quit* terminates the editing session, but the last command is not executed.

The syntax used to edit commands is:

edit

(2) Datatrieve record, domain, and table definitions:

These dictionary objects can not be edited directly, however changes can be made to these objects by using the *extract* command to extract the object to a VMS file. It is then possible to read the VMS file into the editor buffer.

When the editing session is terminated with the *exit* command, the object is automatically redefined in the Dictionary.

The format for commands to extract and edit record, table, and domain procedures is:

edit

and reading the VMS file into the editor buffer by using the editor command:

include VMS.filename

(3) Datatrieve procedures:

Datatrieve procedure can be edited directly. The format of the command to edit a procedure is:

edit procedure-name

Basic Editor Commands:

The following list includes the basic Datatrieve editor commands; while this list is certainly not exhaustive, it should provide enough information to make necessary corrections.

%be	Go to beginning of buffer
%wh	Type out entire contents of buffer
f'string'	Find next occurrence of string searches downward only
s/old/new/	Substitute old with new on current line
s/old/new/wh	Substitute old with new on entire buffer
Number	Go to line number (number)
d	Delete current line
i	Insert new line into buffer above current line (Note: This editor does not have an "a" command to append lines after the current line)
<return>	Advance to next line
•	Print current line
help	Get help regarding editor
include VMS.fil	Include VMS.fil into buffer
write VMS.fil	Write entire contents of buffer to VMS.fil
exit	Leave editor and execute contents of buffer
quit	Leave editor and do not execute contents of buffer

The following example illustrates the way in which commands can be edited, and uses the creation of a record definition as an example.

```
(A record definition is being entered:)

DTR> define record personnel-record using
DFN> 01 employee-information.
DFN>     02 name.
DFN>        03 first-name          pic x(15).
DFN>        03 first-name          pic x(15).
DFN>        03 salary              pic 9999v99.
DFN>           edit-string is      $zzzz.99.

(An error is made; DTR reprints the record definition on the
terminal and places an "up arrow" under the error.)

DEFINE RECORD PERSONNEL_RECORD USING
01 EMPLOYEE_INFORMATION .
02 NAME .
03 LAST_NAME  PIC X(15) .
03 FIRST_NAME PIC X(15) .
02 STATUS.
03 SALARY PIC 9999V99 .
EDIT_STRING IS  $ZZZZ.99.
^
Expected number, encountered "EDIT_STRING".
```

```
(The editor is invoked and the current command is
edited.)

DTR> edit

    1            DEFINE RECORD PERSONNEL_RECORD USING
* <carriage return>
    2            01 EMPLOYEE_INFORMATION.
* <carriage return>
    3                02 NAME.
* <carriage return>
    4                   03 FIRST_NAME      PIC X(15).
*s/first/last/
    4                   03 last_NAME       PIC X(15).
1 substitution
* <carriage return>
    5                   03 FIRST_NAME      PIC X(15).
* <carriage return>
    6                02 STATUS.
* <carriage return>
    7                   03 SALARY          PIC 9999V99.
*s/.//
    7                   03 SALARY          PIC 9999V99
1 substitution
* <carriage return>
    8                      EDIT_STRING IS          $ZZZZ.99.
* <carriage return>
[EOB]

*exit
(The editor is exited.  Datatrieve then executes the portion
of the command within the buffer and continues to prompt the
user for the remainder of the command.)

DFN>          03 hours-worked pic 99.
DFN>          03 vacation-taken pic 99.
DFN>          03 sick-leave-taken pic 99.
DFN> ;
[Record is 42 bytes long.]
```

The Extract Command

The *extract* command makes a copy of a dictionary object in the Common Data Dictionary and stores the copy in a file in your VMS directory.

There are several reasons for extracting dictionary objects:

(1) To make objects available for editing which cannot be edited directly; these objects are: record, domain, and table definitions.

(2) To provide a backup copy of dictionary objects on your VMS directory.

(3) To make objects available for distribution to other DCN nodes.

The syntax of the command is:

extract object file.nam

For example:

(1) *extract personnel-record personnel.rec*

A copy of the record definition "personnel-record" will be put on a VMS file named "personnel.rec" in your VMS directory.

(2) *extract personnel personnel.dom*

A copy of the domain definition "personnel" will be put on a VMS file named "personnel.dom" in your VMS directory.

(3) *extract personnel-table personnel.tab*

A copy of the table definition "personnel-table" will be put on a VMS file named "personnel.tab" in your VMS directory.

Accessing a Datatrieve Domain

In order to use Datatrieve to retrieve, print, modify, delete, or store new data you must first tell the Datatrieve which domain you wish to use.

You must also tell Datatrieve how you intend to use the data: whether for printing, modifying, deleting or storing purposes.

The command used to access a domain is *ready domain-name (type of access).*

When the ready domain-name command is issued, Datatrieve does three things:

(1) Checks to see if there is a domain with the name you have specified located in the Common Data Dictionary.

(2) Compares the record definition contained in the domain definition against the file residing on the disk.

(3) Checks to see if you have authorization to perform the type of access specified.

If any of the items listed above do not work, Datatrieve will print an error message.

If the items listed above work, Datatrieve will respond by prompting "DTR>." In this way you know that the domain is ready for use.

The table on the next page illustrates the variety of ready commands available and their effects:

Types of Ready Commands

READY DOMAIN-NAME	Readies the domain for read access only; records can be found and printed. You cannot modify existing records. You cannot store new records. You cannot erase existing records.
READY DOMAIN-NAME EXTEND	Readies the domain for store operations only; "Extend" means that you are extending the data base: storing more records. You cannot use the find command. You cannot modify existing records. You cannot erase existing records
READY DOMAIN-NAME MODIFY	Readies the domain for modify operations only; Find and print commands can be issued. The modify command can be used. You cannot store new records. You cannot erase old records.
READY DOMAIN-NAME WRITE	Readies the domain with all access: Records can be modified. Records can be erased. New records can be stored.

Establishing Collections

As was demonstrated in the manual experiment, very little can be done with Datatrieve unless a current collection of records has been established.

The command for extablishing a collection is the command

find domain-name (record selection expression).

The records matching the criteria given in the record selection expression compose the current collection of records (otherwise known as the current collection).

All data manipulation operations, with the exception of the store command, work on the current collection.

Record selection expressions (which determine which records are to be included in the current collection) are expressed using any of the boolean expressions shown on the next page:

Boolean Expression Format	Evaluation of Expression
Field = value-expression	True if the value of the field is equal to any value expression in the list
Field EQ value-expression	Same as above
Field EQUAL value-expression	Same as above
Field NE value-expression	True if the value of the field is not equal to any value expression in the list
Field NOT EQUAL value-expression	Same as above
FIELD > value-expression	True if the value of the field is greater than value expression in the list
FIELD GT value-expression	Same as above
FIELD < value-expression	True if the value of the field is less than the value expression in the list
FIELD LT value-expression	Same as above
FIELD GE value-expression	True if the value of the field is greater than or equal to the value expression in the list
FIELD LE value-expression	True if the value of the field is less than or equal to the value expression in the list
FIELD BT value-expression 1 and value-expression 2	True if the value of the field is between value expression 1 and value expression 2
FIELD NOT BT value-expression 1 and value expression 2	True if the value of the field is not between value expression 1 and value expression 2
FIELD CONTAINING value-expression	True if the value of the field does contain value-expression
FIELD NOT CONTAINING value-expression	True if the value of the field does not contain value-expression

Examples of the Find Command

The following examples illustrate the results of using the Find command on the personnel domain:

(1) Find Personnel
This command would establish a collection consisting of all the records in the personnel data base.

(2) Find Personnel with last-name = "Skies"
This command would establish a collection consisting of all the records in the personnel data base where the value of the last-name field was equal to Skies. The collection would consist of two records..

Note: If you want to match upper and lower case characters, or have special characters such as slashes as values in your records you will have to enclose the value expression in quotes.

(3) Find personnel with salary = 500
This command would establish a collection consisting of all the records in the personnel data base where the value of the salary field was equal to 500.

(4) Find personnel with hours-worked < 36
This command would establish a collection consisting of all the records in the personnel data base where the value of the hours-worked field was less than 36.

Note: 36 is not less than 36.

(5) Find personnel with hours-worked = sick-leave-taken + vacation-taken
This command would establish a collection consisting of all the records in the personnel data base where the value of the hours-worked field is equal to the sick-leave-taken field plus the value of the vacation-taken field.

(6) Find personnel with last-name = "Skies" and first-name = "Misty"
This command would establish a collection consisting of all the records in the personnel data base where the value of the last-name is equal to the value "Skies" *and* the value of the first-name field is equal to the value "Misty."

Using the Current Collection

Once a collection of records has been established, it is possible to work with those records: to print them, modify them, erase them, and report on them.

Example of the ways in which you can work with the current collection of records follow over the next few pages.

The Print Command

The print command is used to print the records in the current collection

(1) at your terminal;
(2) on a file;
(3) in a report;
(4) or on the line printer.

The print command can be used to calculate data for special purposes. However, the results of this calculation are not stored (unless you are printing to a file).

If you are printing more than one field at a time from the current collection, the fields must be separated by commas on the command line.

The following examples illustrate the use of the print command; they assume that the find command which established the current collection was

Find personnel with sick-leave-taken > 0

Note: The qualifier *all* on the examples refers to "all" the records in the current collection.

(1) Print all first-name, last-name

FIRST NAME	LAST NAME
Susan	Anthony
John	Bach
Humphrey	Bogart
Fredric	Chopin
Herbert	Hoover
John	Lennon
Judy	Brown
Stormy	Skies
George	Washington

(2) Print all last-name, salary on money.dat
(Note: money.dat is a VMS File name)

(3) Print all first-name, sick-leave-taken + vacation-taken

FIRST NAME	
Susan	03
John	05
Humphrey	08
Fredric	04
Herbert	20
John	10
Judy	30
Stormy	03
George	01

(4) Print all last-name, sick-leave-taken + vacation-taken on timeoff.dat
(Note that "timeoff.dat" is a VMS File name.)

(5)

Print all name

FIRST NAME	LAST NAME
Susan	Anthony
John	Bach
Humphrey	Bogart
Fredric	Chopin
Herbert	Hoover
John	Lennon
Judy	Brown
Stormy	Skies
George	Washington.

(6)

Print all last-name, salary

LAST NAME	SALARY
Antony	$ 580.00
Bach	$ 450.00
Bogart	$ 450.00
Chopin	$ 460.00
Hoover	$ 375.00
Lennon	$ 500.00
Brown	$ 400.00
Skies	$ 475.00
Washington	$ 520.00

(7) Print all last-name, salary, sick-leave-taken on lp:
(Note that lp: is the device name for the line printer.)

Formatting the Results of the Print Command:

Formatting instructions for printing data are called "edit-strings." Edit-strings can be included within the record definition or specified as part the print command to alter the format of the data at print time.

If an edit-string is included as part of the print command, the Datatrieve keyword *using* must be added along with the edit-string.

The following examples show the formatting being performed on the print command.

The examples assume that the current collection was established with the command:

Find personnel with last-name = "Anthony"

(1)

Print all last-name, salary using zzz9.99

LAST NAME	SALARY
Anthony	580.00

(2)

Print all last-name, sick-leave-taken

LAST NAME	SICK LEAVE TAKEN
Anthony	01

(3)

Print all last-name, sick-leave-taken using z9

LAST NAME	SICK LEAVE TAKEN
Anthony	1

(4)

Print all last-name, sick-leave-taken + vacation-taken ("Time Off")

LAST NAME	Time Off
Anthony	03

(5)

Print all last-name, sick-leave-taken + vacation-taken ("Time"/"Off")

LAST NAME	Time Off
Anthony	03

(6)

Print all last-name (-), salary (-) using zzz9.00, hours-worked (-)

Anthony	580.00	40

(7)

Print all last-name, "Makes more than I do:", salary using zzz9.99

LAST NAME		SALARY
Anthony	Makes more than I do:	580.00

(8)

Print all last-name (-), "Makes more than I do:", salary (-) using zzz9.99

Anthony	Makes more than I do:	580.00

The Sort Command:

The sort command is used to sort the contents of the current collection.

The command can be given at any time to reorder the current collection.

The command can be used in conjunction with the **find** command to sort the collection as it is established.

One of two types of sort order can be specified: ascending or descending.

Examples of the sort command are:

```
DTR> ready personnel
DTR> find personnel
[17 records found]
DTR> print all last-name, first-name, salary, hours-worked, salary
```

LAST NAME	FIRST NAME	HOURS WORKED	SALARY
Anthony	Susan	40	$ 638.00
Bach	John	40	$ 450.00
Blow	Joe	35	$ 375.00
Bogart	Humphrey	40	$ 450.00
Bond	James	35	$ 480.00
Brown	Judy	40	$ 400.00
Bumble	Don	30	$ 322.00
Chopin	Fredric	40	$ 460.00
Copland	Aaron	30	$ 577.50
Hoover	Herbert	40	$ 375.00
Lamb	Mary	40	$ 500.00
Lennon	John	35	$ 500.00
Lincoln	Abraham	30	$ 660.00
Skies	Misty	35	$ 500.00
Skies	Stormy	40	$ 475.00
Sventek	Virginia	55	$ 333.00
Washington	George	35	$ 572.00

```
DTR> sort by first-name
DTR> print all last-name, first-name, hours-worked, salary
```

LAST NAME	FIRST NAME	HOURS WORKED	SALARY
Copland	Aaron	30	$ 577.50
Lincoln	Abraham	30	$ 660.00
Bumble	Don	30	$ 322.00
Chopin	Fredric	40	$ 460.00
Washington	George	35	$ 572.00
Hoover	Herbert	40	$ 375.00
Bogart	Humphrey	40	$ 450.00
Bond	James	35	$ 480.00
Blow	Joe	35	$ 375.00
Bach	John	40	$ 450.00
Lennon	John	35	$ 500.00
Brown	Judy	40	$ 400.00
Lamb	Mary	40	$ 500.00
Skies	Misty	35	$ 500.00
Skies	Stormy	40	$ 475.00
Anthony	Susan	40	$ 638.00
Sventek	Virginia	55	$ 333.00

```
DTR> sort by hours-worked
DTR> print all last-name, first-name, hours-worked, salary
```

LAST NAME	FIRST NAME	HOURS WORKED	SALARY
Bumble	Don	30	$ 322.00
Copland	Aaron	30	$ 577.50
Lincoln	Abraham	30	$ 660.00
Blow	Joe	35	$ 375.00
Bond	James	35	$ 480.00
Skies	Misty	35	$ 500.00
Washington	George	35	$ 572.00
Lennon	John	35	$ 500.00
Anthony	Susan	40	$ 638.00
Bach	John	40	$ 450.00
Bogart	Humphrey	40	$ 450.00
Chopin	Fredric	40	$ 460.00
Hoover	Herbert	40	$ 375.00
Skies	Stormy	40	$ 475.00
Brown	Judy	40	$ 400.00
Lamb	Mary	40	$ 500.00
Sventek	Virginia	55	$ 333.00

DTR> sort by hours-worked, salary
DTR> print all last-name, first-name, hours-worked, salary

LAST NAME	FIRST NAME	HOURS WORKED	SALARY
Bumble	Don	30	$ 322.00
Copland	Aaron	30	$ 577.50
Lincoln	Abraham	30	$ 660.00
Blow	Joe	35	$ 375.00
Bond	James	35	$ 480.00
Skies	Misty	35	$ 500.00
Lennon	John	35	$ 500.00
Washington	George	35	$ 572.00
Hoover	Herbert	40	$ 375.00
Brown	Judy	40	$ 400.00
Bach	John	40	$ 450.00
Bogart	Humphrey	40	$ 450.00
Chopin	Fredric	40	$ 460.00
Skies	Stormy	40	$ 475.00
Lamb	Mary	40	$ 500.00
Anthony	Susan	40	$ 638.00
Sventek	Virginia	55	$ 333.00

DTR> find personnel sorted by hours-worked
[17 records found]
DTR> print
No record selected, printing whole collection

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Don	Bumble	$ 322.00	30	01	00
Aaron	Copland	$ 577.50	30	00	00
Abraham	Lincoln	$ 660.00	30	00	00
Joe	Blow	$ 375.00	35	00	00
James	Bond	$ 480.00	35	00	00
Misty	Skies	$ 500.00	35	00	00
George	Washington	$ 572.00	35	00	00
John	Lennon	$ 500.00	35	03	07
Susan	Anthony	$ 638.00	40	00	00
John	Bach	$ 450.00	40	03	02
Humphrey	Bogart	$ 450.00	40	03	05
Fredric	Chopin	$ 460.00	40	02	02
Herbert	Hoover	$ 375.00	40	15	05
Stormy	Skies	$ 475.00	40	02	01
Judy	Brown	$ 400.00	40	20	10
Mary	Lamb	$ 500.00	40	05	00
Virginia	Sventek	$ 333.00	55	33	22

The Store Command:

The store command is used to extend the domain by adding new records to the file.

The domain must be readied as *extend* or *write* before new records can be stored.

The store command can be used to enter one new record, or if used in conjunction with the *repeat (number)* command, to store multiple records.

Storing new records through an input procedure will be shown later.

The following examples illustrate the use of the store command:

(1) To store one new record:

```
Ready personnel write
Store personnel
Enter FIRST_NAME: Virginia
Enter LAST_NAME: Sventek
Enter SALARY: 400
Enter HOURS_WORKED: 90
Enter VACATION_TAKEN: 0
Enter SICK_LEAVE_TAKEN: 0
```

(2) To store more than one new record:

```
Repeat 2 store personnel
Enter FIRST_NAME: Virginia
Enter LAST_NAME: Sventek
Enter SALARY: 400
Enter HOURS_WORKED: 90
Enter VACATION_TAKEN: 0
Enter SICK_LEAVE_TAKEN: 0
Enter FIRST_NAME: Nora
Enter LAST_NAME: Charles
Enter SALARY: 400
Enter HOURS_WORKED: 90
Enter VACATION_TAKEN: 0
Enter SICK_LEAVE_TAKEN: 0
```

The Erase Command:

The *erase* command operates on the current collection of records just as the print and modify commands do.

Therefore it is necessary to establish a collection of records containing those records you wish to delete (erase) from the domain.

No prompting is given by Datatrieve for this command.

The "erased" records will be removed from the file immediately. The mechanism for erasing selected records will be shown later.

The following examples illustrate the *recommended* use of the erase command:

(1) The careful way (printing before erasing):

Find personnel with last-name = "Anthony"
Print all

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Susan	Anthony	$ 580.00	40	02	01

Erase all

(2) The careless way:

Find personnel with last-name = "Anthony"
Erase all

This is careless because, unless you are paying very close attention, the following can happen:

Find personnel
Erase all
Find personnel
[0 records found]

All the records in the file have been erased.

The Modify Command

The *modify* command operates on the current collection of records just as the print command does.

It is therefore necessary to establish a collection of records before using the modify command.

The modify command will prompt for the new value to be inserted in the records.

The new value will be inserted into all the records in the current collection.

The mechanism for assigning a unique value to every record will be shown later.

The following examples illustrate the use of the modify command:

(1) Modifying the value of a field in all records:

```
Find personnel
Modify all sick-leave-taken
Enter SICK_LEAVE_TAKEN: 0
Print all last-name, sick-leave-taken
```

LAST NAME	SICK LEAVE TAKEN
Anthony	00
Bach	00
Blow	00
Bogart	00
Bond	00
Brown	00
Bumble	00
Chopin	00
Copland	00
Hoover	00
Lennon	00
Lincoln	00
Lamb	00
Skies	00
Skies	00
Washington	00

(2)

Modifying the value of a field in one record:

```
Find personnel with last-name = "Anthony"
[1 record found]
Modify all sick-leave-taken
Enter SICK_LEAVE_TAKEN: 6
Print all last-name, sick-leave-taken
```

LAST NAME	SICK LEAVE TAKEN
Anthony	06

Find personnel
Print all last-name, sick-leave-taken

LAST NAME	SICK LEAVE TAKEN
Anthony	06
Bach	00
Blow	00
Bogart	00
Bond	00
Brown	00
Bumble	00
Chopin	00
Copland	00
Hoover	00
Lennon	00
Lincoln	00
Lamb	00
Skies	00
Skies	00
Washington	00

Using an Assignment Statement in Conjunction With the Modify Command:

An assignment statement is used to assign a value to a field at Datatrieve command level rather than through the normal mode of responding to prompts. Assigned values can be absolute values (a value assigned to the field) or computed values.

The form of the assignment statement is:

field = value

or

field = computation

When using the *modify* command to change the value of one field in all the records in the current collection, the syntax becomes:

modify all using assignment statement,

which means that you type:

modify all field1 = the new value for the field

The problem of modifying more than one field at a time is described in a later section.

The following two examples illustrate the use of the assignment statement with the modify command:

(1) Assigning a common value to a field:

```
Find personnel
Modify all using sick-leave-taken = 1
Print all last-name, sick-leave-taken
```

LAST NAME	SICK LEAVE TAKEN
Anthony	01
Bach	01
Blow	01
Bogart	01
Bond	01
Brown	01
Bumble	01
Chopin	01
Copland	01
Hoover	01
Lennon	01
Lincoln	01
Lamb	01
Skies	01
Skies	01
Washington	01

(2) Assigning a computed value to a field:

```
Find personnel
Modify all sick-leave-taken using sick-leave-taken = sick-leave-taken + 4
Print all last-name, sick-leave-taken
```

LAST NAME	SICK LEAVE TAKEN
Anthony	05
Bach	05
Blow	05
Bogart	05
Bond	05
Brown	05
Bumble	05
Chopin	05
Copland	05
Hoover	05
Lennon	05
Lincoln	05
Lamb	05
Skies	05
Skies	05
Washington	05

The For Statement:

The Datatrieve *for* statement causes Datatrieve to execute a statement or group of statements once for each record in the record stream created by the *record selection expression.*

The statement provides repeating loops for Datatrieve operations as will be seen in greater detail in the next few sections.

This means that Datatrieve will perform all the operations requested on the first record before going on to the next; it will then perform all the operations for that record before going on to the third, *ad nauseum.*

These actions will continue until the set of records specified by the record selection expression is exhausted.

(Remember that the record selection expression is the set of criteria which specifies the records on which an operation is to occur. If necessary refer to the section on the Find command.)

It should be remembered that when used with the find command, the record selection expression is formed by:

with field = value expression

as in *find domain with field = value expression*

That is, the records to be included in the collection are those where the value of the field matches the value expression as in

find personnel with salary > 500

Likewise, the *for* statement specifies those records on which an operation is to occur. The syntax is:

for current with salary > 599

This syntax will cause Datatrieve to operate only on those records in the current collection where the value of the salary field is greater than the value 599.

The statement *for record selection expression* is followed by the Datatrieve statements to be executed for that group of records.

The statements can be simple or compound statements. (Compound statements are formed with *begin-end, if-then-else,* or *then* statements).

Begin-end statements will be shown in a later section.

Assuming that the current collection was established with the find command

find personnel with salary > 500

then an appropriate *for* statement might be:

*for current with salary > 599 print name, salary, salary * 1.10*

In this example, the value of the name and salary fields, along with the value of the salary field multiplied by 1.10 will be printed for every record in the current collection where the value of the salary field is greater than 599.

FIRST NAME	LAST NAME	SALARY	
Abraham	Lincoln	$ 600.00	660.00

The Then Command:

The *then* statement is used to connect two simple statements to form one compound statement.

The *then* statement is used rather than *begin-end* statements to form short compound statements.

For example, assume we wished to modify the salaries of the records in the current collection where the value of the salary field is greater than 519. Also, assume that we need to see the name of each employee before assigning a new value.

This would be done by issuing one command to:

(1) print the value of the name field and *then*

(2) modify the salary field

for every record in the current collection where the salary is greater than 519.

To do this, we would issue the command

for current with salary > 519 print name then modify salary

which would result in:

```
DTR> for current with salary > 519 print name,
salary then modify salary

   FIRST        LAST
   NAME         NAME       SALARY

Susan        Anthony      $ 580.00
Enter SALARY: 600
Aaron        Copland      $ 525.00
Enter SALARY: 550
Abraham      Lincoln      $ 600.00
Enter SALARY: 625
George       Washington   $ 520.00
Enter SALARY: 530
```

```
DTR> for current with salary > 519 print last-name, salary

                LAST
               NAME

Anthony        $ 600.00
Copland        $ 550.00
Lincoln        $ 625.00
Washington     $ 530.00
```

Note: If you do not wish to change the value of a particular field, enter a tab (control I) character when the new value for the field is requested by Datatrieve.

The Begin-End Statement:

The *begin-end* statement groups a set of Datatrieve statements into a single compound statement called a *begin-end* block.

We will examine the *begin-end block* by using it to modify records.

In the previous examples we have seen how to assign a new value to a single field by using an *assignment statement.*

If more than one field is to be assigned a value in an assignment statement the assignment statements must be enclosed within a *begin-end* block.

In this example new values are assigned to more than one field.

(First list those records with salary greater than 500 just to see who they are:)

```
DTR> for personnel with salary > 500 print employee-information
```

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Anthony	Susan	$ 580.00	40	02	01
Copland	Aaron	$ 525.00	35	02	00
Lincoln	Abraham	$ 600.00	40	00	00
Washington	George	$ 520.00	40	00	01

(Modify the fields with the following for statement and modify command:)

```
DTR> for personnel with salary > 500 modify using
begin
        salary = salary * 1.10
        hours-worked = 40
        vacation-taken = 0
        sick-leave-taken = 0
end
```

(Print those records with salary greater than 500 to see the changes:)

DTR> for personnel with salary > 500 print first-name,
last-name, salary, hours-worked, vacation-taken, sick-leave-taken

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Anthony	Susan	$ 638.00	40	00	00
Copland	Aaron	$ 577.50	40	00	00
Lincoln	Abraham	$ 660.00	40	00	00
Washington	George	$ 572.00	40	00	00

Selecting Records in the Current Collection

The *select* command is used when you want to do something with a single record that is inappropriate or not desirable for all the records in current collection.

The qualifier "all" is not used when working with selected records. This is because the commands are not intended to have an effect on all the records in the current collection, but only on the selected record.

Remember that the qualifier *all* refers to *all* records in the current collection.

There are several qualifiers which enable you to select a specific record. These qualifiers are listed at the end of this section.

The following example illustrates the use of the select command in conjunction with the modify command. It assumes that the current collection of records was established with the command:

Find personnel with sick-leave-taken > 0

```
Select 1
Modify sick-leave-taken
Enter SICK-LEAVE-TAKEN: 0
Print last-name, sick-leave-taken
            SICK
 LAST      LEAVE
 NAME      TAKEN

Anthony        0


Select next
Modify sick-leave-taken
Enter SICK-LEAVE-TAKEN: 5
Print last-name, sick-leave-taken
            SICK
 LAST      LEAVE
 NAME      TAKEN

Bach           5
```

```
Select next
Modify sick-leave-taken
Enter SICK-LEAVE-TAKEN: 6
Print last-name, sick-leave-taken
            SICK
 LAST      LEAVE
NAME       TAKEN

Bogart        6


Select next
Modify sick-leave-taken
Enter SICK-LEAVE-TAKEN: 0
Print last-name, sick-leave-taken
            SICK
 LAST      LEAVE
NAME       TAKEN

Chopin        0
```

The following table shows the most frequently used valid select statements.

Command	Qualifier	Effect
Select		Selects the first record in the current collection.
Select	#	Selects record with number specified
Select	Next	Selects the next record in the current collection. If you have not selected a record in a collection and you enter "select next", the first record in the collection becomes the selected record.
Select	Prior	Selects the previous record (if there is no "previous" record, Datatrieve prints an error message.)
Select	First	Selects the first record in the current collection.
Select	Last	Selects the last record in the current collection.
Select	First with Boolean	Causes Datatrieve to select the first record that satisfies the boolean expression
Select	Last with Boolean	Causes Datatrieve to select the last record that satisfies the boolean expression

Prompting for Values

Datatrieve makes provision for prompting for values in assignment statements and record selection expressions by way of *value expression prompts.*

Prompting is especially useful within procedures because it allows you to enter a different value each time the procedure is invoked.

Datatrieve recognizes two types of prompting expressions: the ***.prompt* and the ****.prompt.*

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.

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

The following examples show the difference between the ***.prompt* and the ****.prompt* value expressions.

Although prompts are used with both the modify and store commands, we will examine their use in conjunction with the store command.

The ***.prompt* value expression:

In the example below, four new records are stored and the values are assigned in the following way:

(1) the value for the sick-leave-taken, vacation-taken and hours-worked fields will be set to 0 for the new records;

(2) the value of the name and salary fields will be prompted for:

```
Repeat 4 store personnel using
begin
        first-name = *.first-name
        last-name = *.last-name
        salary = *.salary
        hours-worked = 40
        sick-leave-taken = 0
        vacation-taken = 0
end;
```

Using this command results in:

```
Enter FIRST_NAME: Mary
Enter LAST_NAME: Blye
Enter SALARY: 450
Enter FIRST_NAME: Stan
Enter LAST_NAME: Smith
Enter SALARY: 500
Enter FIRST_NAME: Shirley
Enter LAST_NAME: Browne
Enter SALARY: 560
Enter FIRST_NAME: Frank
Enter LAST_NAME: Jones
Enter SALARY: 570
```

And the new records look like:

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Frank	Jones	$ 570.00	40	00	00
Mary	Blye	$ 450.00	40	00	00
Shirley	Brown	$ 560.00	40	00	00
Stan	Smith	$ 500.00	40	00	00

The **.*prompt* value expression:

In the next example three new records are stored and are assigned values in the following way:

(1) the values for the vacation-taken and sick-leave-taken fields are assigned a specific constant value (0);

(2) the value for the hours-worked field is a constant value obtained through a prompt;

(3) the names and salary fields are assigned a unique value (as they were in the example above).

```
Repeat 3 store personnel using
begin
        First-name = *.first-name
        last-name = *.last-name
        salary = *.salary
        hours-worked = **.hours-worked
        sick-leave-taken = 0
        vacation-taken = 0
end;
```

Using this command results in:

```
Enter FIRST_NAME: Ed
Enter LAST_NAME: Whipple
Enter SALARY: 300
Enter HOURS_WORKED: 35
Enter FIRST_NAME: Nikki
Enter LAST_NAME: Isaacs
Enter SALARY: 560
Enter FIRST_NAME: Jeff
Enter LAST_NAME: Lawrence
Enter SALARY: 580
```

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Ed	Whipple	$ 300.00	35	00	00
Jeff	Lawrence	$ 580.00	35	00	00
Nikki	Isaacs	$ 560.00	35	00	00

Notice that the hours-worked value (hours-worked = **.hours-worked) was prompted for only once: for the first record; the first-name, last-name, and salary fields were prompted for on every record (first-name = *.first-name, last-name = *.last-name, salary = *.salary).

Datatrieve Statistical Functions

Datatrieve provides several statistical functions which can be used wherever the syntax of a Datatrieve statement allows a value expression.

The table below lists the most commonly used statistical functions:

Function	Value of Function
MAX	The largest value of the value expression
MIN	The smallest value of the value expression
AVERAGE	The average value of the value expression
TOTAL	The total value of the value expression
COUNT	The number of records in the current collection or in a specified collection or record stream
RUNNING COUNT	The current number of the record
RUNNING TOTAL	Cumulative totals

The following examples illustrate the use of these statistical functions.

(1) Find personnel
Print total salary

TOTAL
SALARY

7512

(2) Find personnel
Print min salary

MIN
SALARY

469.50

(3) Find personnel
Print max salary

MAX
SALARY

MAX
SALARY

600

(4) Find personnel
Print count

COUNT

16

(5) Find personnel
Print all running count, last-name

RUNNING COUNT	LAST NAME
1	Anthony
2	Bach
3	Blow
4	Bogart
5	Bond
6	Brown
7	Bumble
8	Chopin
9	Copland
10	Hoover
11	John
12	Lincoln
13	Mary
14	Skies
15	Skies
16	Sventek
17	Washington

(6)
Find personnel with salary > 500
Print all running count, last-name

RUNNING COUNT	LAST NAME
1	Anthony
2	Copland
3	Lincoln
4	Washington

(7)

Print all last-name, salary, running total salary

LAST NAME	SALARY	RUNNING TOTAL SALARY
Anthony	$ 638.00	638.00
Copland	$ 577.50	1215.50
Lincoln	$ 660.00	1875.50
Washington	$ 572.00	2447.50

(8)

Find personnel
Print all last-name, salary, running total salary

LAST NAME	SALARY	RUNNING TOTAL SALARY
Anthony	$ 638.00	638.00
Bach	$ 450.00	1088.00
Blow	$ 375.00	1463.00
Bogart	$ 450.00	1913.00
Bond	$ 480.00	2393.00
Brown	$ 400.00	2793.00
Bumble	$ 322.00	3115.00
Chopin	$ 460.00	3575.00
Copland	$ 577.50	4152.50
Hoover	$ 375.00	4527.50
Lamb	$ 500.00	6187.50
Lennon	$ 500.00	5027.50
Lincoln	$ 660.00	5687.50
Skies	$ 500.00	6687.50
Skies	$ 475.00	7162.50
Sventek	$ 333.00	7495.50
Washington	$ 572.00	8067.50

Datatrieve Tables

Datatrieve tables can be used to perform two functions:

(1) You can specify one value in a value expression and retrieve the translation of that value from a table;

(2) You can validate data according to the presence or absence of data items in the table.

There are two types of tables: dictionary and domain.

The use of dictionary tables is be described here.

Datatrieve dictionary tables are defined with the *define table* command.

Tables are invoked with the keywords *via* or *in.*

Table definitions contain

(1) a field value, and

(2) a translation string;

When the table is invoked, the value of the specified field is translated to the contents of the translation string in the table.

The following example illustrates the definition of a dictionary table:

```
Define table employee-table
"Anthony"     :   "Program Manager"
"Bogart"      :   "Program Manager"
"Copland"     :   "Technical Personnel"
"Brown"       :   "Support Personnel"
"Washington"  :   "Program Manager"
Else "Scientific Staff"
End-table;
```

Notice that

(1) field values are entered on the left side of the table definition, and the translation strings are entered on the right;

(2) values and translation strings are separated by a colon (:);

(3) table definitions must include a default translation string for records without values in the table (Else "Something").

(4) If values or strings in either the field value or translation string contain spaces, upper and/or lower case characters, or special characters such as $, -, %, they must be contained within double quotation marks.

Using Datatrieve Tables

The following examples illustrate typical uses of dictionary tables:

Print all last-name, last-name via employee-table using x(25)

LAST NAME	
Anthony	Program Manager
Bach	Scientific Staff
Blow	Scientific Staff
Bogart	Program Manager
Bond	Scientific Staff
Brown	Support Personnel
Bumble	Scientific Staff
Chopin	Scientific Staff
Copland	Technical Personnel
Hoover	Scientific Staff
Lamb	Scientific Staff
Lennon	Scientific Staff
Lincoln	Scientific Staff
Skies	Scientific Staff
Skies	Scientific Staff
Washington	Program Manager

Find personnel sorted by last-name via employee-table
Print all last-name, last-name via employee-table using x(25)

LAST NAME	
Anthony	Program Manager
Bogart	Program Manager
Washington	Program Manager
Bach	Scientific Staff
Blow	Scientific Staff
Bond	Scientific Staff
Bumble	Scientific Staff
Chopin	Scientific Staff
Hoover	Scientific Staff
Lamb	Scientific Staff
Lennon	Scientific Staff
Lincoln	Scientific Staff
Skies	Scientific Staff
Skies	Scientific Staff
Brown	Support Personnel
Copland	Technical Personnel

Find personnel with last-name in employee-table
Print all

FIRST NAME	LAST NAME	SALARY SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Susan	Anthony	$ 580.00	40	02	01
Humphrey	Bogart	$ 450.00	40	03	05
Judy	Brown	$ 400.00	30	20	10
Aaron	Copland	$ 525.00	35	02	00
George	Washington	$ 520.00	40	00	01

The Datatrieve Report Writer

The Datatrieve report writer allows the user to create reports in standard report format on

(1) the terminal
(2) the line printer

or

(3) a file.

Report statements are available to

(1) specify page layout,
(2) specify groups of records on which a variety of operations should occur,
(3) and perform statistical functions.

The next several pages provide an introduction to the formatting of Datatrieve reports.

Basic Datatrieve Report Writer Commands:

In addition to the data manipulation facilities seen previously, the report writer can

(1) Divide your report into pages (numbered or not, as desired);
(2) Print the date at the top of each report page;
(3) Display a report name at the top of each page (as desired);
(4) Adjust the page size for the report;
(5) Create header and summary lines for the top and bottom of pages;
(6) Display data sorted by several fields and grouped according to the changes in the specified fields;
(7) Display header and summary lines above these groups of sorted data;
(8) Display, at top and bottom of the report and tops and bottoms of pages and groups of sorted data, the results of statistical and mathematical evaluation of data in those sections of the report.

The basic report writer statements and their effects are listed in the table below:

Statement	Effect
Report	Invokes the report writer;
Set (parameter)	The set command is used to specify page layout;
At (parameter)	Indicates action to take place at specific point in report;
Print (list)	Indicates fields to be printed in the report body;
End-Report	Terminates report writer;

The Report Command:

This is a required statement.

The report statement invokes the report writer and must be the first entry in a report specification. In the *report* statement you can specify:

(1) the data you want to report

(2) the output device for the report.

After the *report* statement is entered at Datatrieve command level, the prompt changes to *RW>* showing that the report writer is ready to accept report writer statements.

Examples of the Report Statement:

report (record selection expression) on file.nam

If a VMS *"file.nam"* is given as the output destination for the report, all report output will be written to that file.

report (record selection expression) on *.prompt-name

If a **.prompt-name* is given, the report can be sent to the output device specified by the user.

Output destinations include:

a) A VMS file
b) A output device: line printer or terminal; a line printer is known as "lp:" to the system, and a terminal as "tt:."

The *record selection expression* can be

(1) a readied domain or

(2) a collection.

The Set Statement:

The set statements control the report header and define the size of report pages and the length of the report.

With the report writer *set* statements you can specify:

(1) Report header:

report name

report date

format of date

page numbering

(2) Size of report pages:

number of columns per page

number of lines per page

(3) Length of report:

maximum number of lines

maximum number of pages

Examples of the Set Command:

Set report-name = "Report on Current Personnel"/ "Fourth Quarter 1981"

Notice that the conventions for multi-line column heading seen previously in the *print* command apply here.

Set date = "July 21, 1982"

The date specified on the *set date* statement will be printed as entered.

If the *set date* statement is ommitted, the current date will be supplied by the system.

Set no date

If a date is not desired, use the *set no date* statement.

Set number

The *set number* statement causes pagination to occur. The default action is to number pages, so this statement does not need to be specified.

Set no number

If pagination is not desired use the *set no number* statement.

Set columns-page = 80

The *set columns-page* statement specifies the width of the report page in columns. The maximum value for columns-page is 255 columns.

The width of normal line-printer paper is 132 columns.

Set lines-page = 60

The set lines-page statement specifies the number of lines per report page. (Page ejects can be specified to occur at control group changes.)

The default value is 60 lines per page.

Control Groups:

Control groups are created by sorting a collection of records by the values in a field.

In our example, the personnel domain could be sorted by the hours-worked field; at that point at least three categories of records would occur in the collection:

(1) those people who work 40 hour weeks,

(2) those people who work 35 hour weeks, and

(3) those people who work 30 hour weeks.

These categories of sorted records are called control groups. A *control group* is a series of sorted data records that have the same value in one or more fields.

The At Statement

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

To maximize the usefulness of the *at* statement, the current collection of records must be sorted by the field on which the *at* statements are to have an effect.

The two forms of the *at* statement (at top, at bottom) specify the value, position, and format of the print objects on the header and summary lines.

(1) The *at top* statement prints header or summary lines at the top of reports, report pages, and control groups. In these header or summary lines print objects are specified in the same manner as the *print* statement.

(2) The *at bottom* statement prints summary lines at the bottoms of reports, report pages, and control groups. In these summary lines, you can specify the same type of print objects as you can in the *print* statement.

At top of hours-worked print new-page

This *at top of* statement would start a new page at the beginning of each hours-worked control group, i.e., whenever the value of the hours-worked field changed.

At top of hours-worked print hours-worked

This statement would print the value of the hours-worked control group at the top of that control group, i.e., whenever the value of the hours-worked field changed.

At bottom of hours-worked print total hours-worked

This statement would print the total of the values in the current hours-worked control group; only the records in the control group would be used to calculate the total. This can be thought of as a subtotal.

The total would be placed under the hours-worked field if that field were listed as part of the *print* command.

The total for the hours-worked field will be shown as the last print line for the control group.

All other statistical functions can be used in the *at bottom of* statement.

At bottom of hours-worked print total hours-worked, total salary

Summary statistical information can be obtained for more than one field as is shown in this example. The totals for the hours-worked field and the salary field in the control group will be shown as the last print line for the control group. This can be thought of as a subtotal.

At bottom of report print total hours-worked, total salary

The *at bottom of report* statement calculates a grand total of the fields specified in the statement.

The Print Command in the Report Writer

The report writer *print* statement specifies the format for the detail lines in a report. Detail lines are the formatted data lines of a report. With *print* you specify three characteristics of the detail lines:

(1) The content of the detail lines:

Values of fields from records in the current collection or the record stream specified by the record selection expression in the *report* command;

Other desired values and text strings;

(2) The format of the fields in the detail lines:

Order of print objects within the line;

Position of print objects within the line;

Edit-string for each print object (if desired);

(3) Column headers for the print objects in the detail line.

The report writer *print* statement produces a detail line in the report for every data record in the current collection or in the specified record stream. A detail line can cover several lines on the report page, depending on the format and columns per page you specify.

You can include only one *print* statement in a report specification. If the specification does not contain an *at* statement, it must contain a *print statement*.

If your report specification contains an *at* statement it need not contain a *print* statement.

(One use of a report specification containing only *at* statements is in the creation of a report summary page.

The *print* statement in the report writer follows the rules laid out for the *for* statement in that reporting is done for every record in the record stream, and therefore the qualifier *all* is not needed.

Examples of the Print statement:

print name, salary, salary * 1.10

The results will be the first and last name values, the salary, and the salary multiplied by 1.10.

print name (-), salary (-), salary * 1.10 ("After"/"Increase")

The result of this statement will be a detail line for every record in the current collection, or every record specified by the record selection expression, containing the first and last name fields, the salary field, and the salary field multiplied by 1.10. Only the "salary * 1.10" field will be labeled.

Example Report Writer Statement Sequences

This report simply produces all the information in the personnel domain in standard report format:

```
DTR> report
RW>  set report-name = "Employee Report"/
[Looking for header segment]
RW>  "First Quarter 1982"
RW>  print employee-information
RW>  end-report
```

Employee Report
First Quarter 1982

26-Jul-1982
Page 1

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Susan	Anthony	$ 638.00	40	00	00
John	Bach	$ 450.00	40	03	02
Joe	Blow	$ 375.00	35	00	00
Humphrey	Bogart	$ 450.00	40	03	05
James	Bond	$ 480.00	35	00	00
Judy	Brown	$ 400.00	40	20	10
Don	Bumble	$ 322.00	30	01	00
Fredric	Chopin	$ 460.00	40	02	02
Aaron	Copland	$ 577.50	30	00	00
Herbert	Hoover	$ 375.00	40	15	05
Mary	Lamb	$ 500.00	40	05	00
John	Lennon	$ 500.00	35	03	07
Abraham	Lincoln	$ 660.00	30	00	00
Misty	Skies	$ 500.00	35	00	00
Stormy	Skies	$ 475.00	40	02	01
George	Washington	$ 572.00	35	00	00

This report prints totals for all the numeric fields in the personnel domain.

Notice that the totals are printed using the default edit-strings:

```
DTR> report
RW>  set report-name = "Employee Report"/
[Looking for header segment]
RW>  "First Quarter 1982"
RW>  print employee-information
RW>  at bottom of report print total salary, total hours-worked,
[Looking for next element in list]
RW>  total vacation-taken, total sick-leave-taken
RW>  end-report;
```

Employee Report
First Quarter 1982

26-Jul-1982
Page 1

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Susan	Anthony	$ 638.00	40	00	00
John	Bach	$ 450.00	40	03	02
Joe	Blow	$ 375.00	35	00	00
Humphrey	Bogart	$ 450.00	40	03	05
James	Bond	$ 480.00	35	00	00
Judy	Brown	$ 400.00	40	20	10
Don	Bumble	$ 322.00	30	01	00
Fredric	Chopin	$ 460.00	40	02	02
Aaron	Copland	$ 577.50	30	00	00
Herbert	Hoover	$ 375.00	40	15	05
Mary	Lamb	$ 500.00	40	05	00
John	Lennon	$ 500.00	35	03	07
Abraham	Lincoln	$ 660.00	30	00	00
Misty	Skies	$ 500.00	35	00	00
Stormy	Skies	$ 475.00	40	02	01
George	Washington	$ 572.00	35	00	00
		7734.50	585	54	32

This report writer sequence assigns an edit-string to the total salary:

```
DTR> report
RW>  set report-name = "Employee Report"/
[Looking for header segment]
RW>  "First Quarter 1982"
RW>  print employee-information
RW>  at bottom of report print total salary using $zzzz.99, total hours-worked,
[Looking for next element in list]
RW>  total vacation-taken, total sick-leave-taken
RW>  end-report;
```

Employee Report
First Quarter 1982

26-Jul-1982
Page 1

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Susan	Anthony	$ 638.00	40	00	00
John	Bach	$ 450.00	40	03	02
Joe	Blow	$ 375.00	35	00	00
Humphrey	Bogart	$ 450.00	40	03	05
James	Bond	$ 480.00	35	00	00
Judy	Brown	$ 400.00	40	20	10
Don	Bumble	$ 322.00	30	01	00
Fredric	Chopin	$ 460.00	40	02	02
Aaron	Copland	$ 577.50	30	00	00
Herbert	Hoover	$ 375.00	40	15	05
Mary	Lamb	$ 500.00	40	05	00
John	Lennon	$ 500.00	35	03	07
Abraham	Lincoln	$ 660.00	30	00	00
Misty	Skies	$ 500.00	35	00	00
Stormy	Skies	$ 475.00	40	02	01
George	Washington	$ 572.00	35	00	00
		$7734.50	585	54	32

This report writer sequence prints a blank line before the summary total line:

```
DTR> report
RW>  set report-name = "Employee Report"/
[Looking for header segment]
RW>  "First Quarter 1982"
RW>  print employee-information
RW>  at bottom of report print skip 1, total salary using $zzzz.99,
RW>  total hours-worked, total vacation-taken, total sick-leave-taken
RW>  end-report;
```

Employee Report
First Quarter 1982

26-Jul-1982
Page 1

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Susan	Anthony	$ 638.00	40	00	00
John	Bach	$ 450.00	40	03	02
Joe	Blow	$ 375.00	35	00	00
Humphrey	Bogart	$ 450.00	40	03	05
James	Bond	$ 480.00	35	00	00
Judy	Brown	$ 400.00	40	20	10
Don	Bumble	$ 322.00	30	01	00
Fredric	Chopin	$ 460.00	40	02	02
Aaron	Copland	$ 577.50	30	00	00
Herbert	Hoover	$ 375.00	40	15	05
Mary	Lamb	$ 500.00	40	05	00
John	Lennon	$ 500.00	35	03	07
Abraham	Lincoln	$ 660.00	30	00	00
Misty	Skies	$ 500.00	35	00	00
Stormy	Skies	$ 475.00	40	02	01
George	Washington	$ 572.00	35	00	00
		$7734.50	585	54	32

The following display shows the contents of the current collection when sorted by the hours-worked field. Horizontal lines have been drawn to artifically mark the control group boundaries:

DTR> find personnel sorted by hours-worked
[16 records found]
DTR> print all

FIRST NAME	LAST NAME	SALARY	HOURS WORKED	VACATION TAKEN	SICK LEAVE TAKEN
Don	Bumble	$ 322.00	30	01	00
Aaron	Copland	$ 577.50	30	00	00
Abraham	Lincoln	$ 660.00	30	00	00
Joe	Blow	$ 375.00	35	00	00
James	Bond	$ 480.00	35	00	00
John	Lennon	$ 500.00	35	03	07
Misty	Skies	$ 500.00	35	00	00
George	Washington	$ 572.00	35	00	00
Susan	Anthony	$ 638.00	40	00	00
John	Bach	$ 450.00	40	03	02
Humphrey	Bogart	$ 450.00	40	03	05
Fredric	Chopin	$ 460.00	40	02	02
Herbert	Hoover	$ 375.00	40	15	05
Mary	Lamb	$ 500.00	40	05	00
Stormy	Skies	$ 475.00	40	02	01
Judy	Brown	$ 400.00	40	20	10

The following report writer sequence prints the values of the control groups "hours-worked;" it also requests that the value of the hours-worked field be printed for each detail record.

Notice that when a field's value is printed both as part of a control group and as a detail line, the values on the detail lines are printed at the same column position as the control group value.

```
DTR> report
RW>  set report-name = "Employee Report"/
RW>  "First Quarter 1982"
RW>  print last-name, first-name, salary, vacation-taken, sick-leave-taken,
RW>  hours-worked
RW>  at top of hours-worked print hours-worked
RW>  at bottom of report print skip 1, total salary using $zzzz.99,
RW>  total hours-worked, total vacation-taken, total sick-leave-taken
RW>  end-report;
```

Employee Report
First-Quarter 1982

26-Jul-1982
Page 1

HOURS WORKED	LAST NAME	FIRST NAME	SALARY	VACATION TAKEN	SICK LEAVE TAKEN
30					
	Bumble	Don	$ 322.00		
30				01	00
	Copland	Aaron	$ 577.50		
30				00	00
	Lincoln	Abraham	$ 660.00		
30				00	00
35					
	Blow	Joe	$ 375.00		
35				00	00
	Bond	James	$ 480.00		
35				00	00
	Lennon	John	$ 500.00		
35				03	07
	Skies	Misty	$ 500.00		
35				00	00
	Washington	George	$ 572.00		
35				00	00
40					
	Anthony	Susan	$ 638.00		
40				00	00
	Bach	John	$ 450.00		
40				03	02
	Bogart	Humphrey	$ 450.00		
40				03	05
	Chopin	Fredric	$ 460.00		
40				02	02
	Hoover	Herbert	$ 375.00		
40				15	05
	Lamb	Mary	$ 500.00		
40				05	00
	Skies	Stormy	$ 475.00		
40				02	01
	Brown	Judy	$ 400.00		
40				20	10
			$7734.50	54	32

The following report writer sequence requests that the hours-worked value be printed for each control group and that it *not* be printed for each detail record:

```
DTR> Report
RW> Set Report-name = "Employee Report"/
RW> "First Quarter 1982"
RW> print first-name, last-name, salary, vacation-taken, sick-leave-taken
RW> At top of hours-worked print hours-worked
RW> At bottom of report print skip 1, total salary using $zzzz.99,
RW> total vacation-taken, total sick-leave-taken
RW> end-report;
```

Employee Report 26-Jul-1982
Page 1

HOURS WORKED	FIRST NAME	LAST NAME	SALARY	VACATION TAKEN	SICK LEAVE TAKEN
30					
	Don	Bumble	$ 322.00	01	00
	Aaron	Copland	$ 577.50	00	00
	Abraham	Lincoln	$ 660.00	00	00
35					
	Joe	Blow	$ 375.00	00	00
	James	Bond	$ 480.00	00	00
	John	Lennon	$ 500.00	03	07
	Misty	Skies	$ 500.00	00	00
	George	Washington	$ 572.00	00	00
40					
	Susan	Anthony	$ 638.00	00	00
	John	Bach	$ 450.00	03	02
	Humphrey	Bogart	$ 450.00	03	05
	Fredric	Chopin	$ 460.00	02	02
	Herbert	Hoover	$ 375.00	15	05
	Mary	Lamb	$ 500.00	05	00
	Stormy	Skies	$ 475.00	02	01
	Judy	Brown	$ 400.00	20	10
			$7734.50	54	32

The following report writer sequence requests that the hours-worked value be printed at the top of each control group. It also requests that the totals for the numeric fields be printed at the bottom of each control group:

```
DTR> Report
RW> Set Report-name = "Employee Report"/
RW> "First Quarter 1982"
RW> print first-name, last-name, salary, vacation-taken, sick-leave-taken
RW> At top of hours-worked print hours-worked
RW> At bottom of hours-worked print total salary using $zzzz.99,
RW> total vacation-taken, total sick-leave-taken
RW> At bottom of report print skip 1, total salary using $zzzz.99,
RW> total vacation-taken, total sick-leave-taken
RW> end-report;
```

Employee Report
First Quarter 1982

26-Jul-1982
Page 1

HOURS WORKED	FIRST NAME	LAST NAME	SALARY	VACATION TAKEN	SICK LEAVE TAKEN
30					
	Don	Bumble	$ 322.00	01	00
	Aaron	Copland	$ 577.50	00	00
	Abraham	Lincoln	$ 660.00	00	00
			$1559.50	01	00
35					
	Joe	Blow	$ 375.00	00	00
	James	Bond	$ 480.00	00	00
	John	Lennon	$ 500.00	03	07
	Misty	Skies	$ 500.00	00	00
	George	Washington	$ 572.00	00	00
			$2427.00	03	07
40					
	Susan	Anthony	$ 638.00	00	00
	John	Bach	$ 450.00	03	02
	Humphrey	Bogart	$ 450.00	03	05
	Fredric	Chopin	$ 460.00	02	02
	Herbert	Hoover	$ 375.00	15	05
	Mary	Lamb	$ 500.00	05	00
	Stormy	Skies	$ 475.00	02	01
	Judy	Brown	$ 400.00	20	10
			$3748.00	50	25
			$7734.50	54	32

The following report writer sequence creates a report consisting of one summary page for the domain:

```
DTR>report
RW> set report-name = "Employee Report"/
[Looking for header segment]
RW> "First Quarter 1982"
RW> At bottom of hours-worked print hours-worked,
RW> total salary using $zzzz.99, total vacation-taken, total sick-leave-taken
RW> At bottom of report skip 1, col 1, "Group Totals:",
RW> total salary using $zzzz.99, total vacation-taken, total sick-leave-taken,
RW> col 1, "Group Averages:", average salary, average vacation-taken,
RW> average sick-leave-taken
RW> end-report;
```

Employee Report 26-Jul-1982
First-Quarter 1982 Page 1

HOURS WORKED	TOTAL SALARY	TOTAL VACATION TAKEN	TOTAL SICK LEAVE TAKEN
30	$1559.50	1	0
35	$2427.00	3	7
40	$3748.00	50	25
Group Totals:	$7734.50	54	32
Group Averages:	$ 483.41	03	02

The following report writer sequence prints the last name values both as they exist in the records and as translated through the employee-table:

```
Report
Set report-name = "Employee Report"/
"First Quarter 1982"
At top of hours-worked print hours-worked
print first-name, last-name, last-name via employee-table, salary
end-report;
```

Employee Report
First Quarter 1982

27-Jul-1982
Page 1

HOURS WORKED	FIRST NAME	LAST NAME		SALARY
30				
	Don	Bumble	Scientific Staff	$ 322.00
	Aaron	Copland	Technical Personnel	$ 577.50
	Abraham	Lincoln	Scientific Staff	$ 660.00
35				
	Joe	Blow	Scientific Staff	$ 375.00
	James	Bond	Scientific Staff	$ 480.00
	John	Lennon	Scientific Staff	$ 500.00
	Misty	Skies	Scientific Staff	$ 500.00
	George	Washington	Program Manager	$ 572.00
40				
	Susan	Anthony	Program Manager	$ 638.00
	John	Bach	Scientific Staff	$ 450.00
	Humphrey	Bogart	Program Manager	$ 450.00
	Fredric	Chopin	Scientific Staff	$ 460.00
	Herbert	Hoover	Scientific Staff	$ 375.00
	Mary	Lamb	Scientific Staff	$ 500.00
	Stormy	Skies	Scientific Staff	$ 475.00
	Judy	Brown	Support Personnel	$ 400.00

Defining Datatrieve Procedures

Many interactions with Datatrieve involve recurring sequences of commands and statements. Retyping such sequences can be avoided by storing them in the Common Data Dictionary *procedures*.

Define procedures with the *define procedure* command by

(1) giving the sequence of commands and statements a name and

(2) listing the commands and statements.

To use a Datatrieve procedure, invoke it by typing a colon (:) followed by the procedure name.

The following examples illustrate procedure definition and use:

A Procedure to Modify Records:

A procedure to modify existing records; prompts the user for the new value for the vacation-taken field

```
Define procedure modify-loop
for current modify using
Begin
        Print last-name
        Vacation-taken = *."Vacation Time Taken"
End
End-procedure;
```

The procedure is invoked by typing :modify. The user is then prompted for the new values for the vacation-taken field:

```
:modify-loop

    LAST
    NAME

Anthony
Enter Vacation Time Taken: 4
Bach
Enter Vacation Time Taken: 3
Blow
Enter Vacation Time Taken: 2
Bogart
Enter Vacation Time Taken: 5
Bond
Enter Vacation Time Taken: 3
Brown
Enter Vacation Time Taken: 2
Bumble
Enter Vacation Time Taken: 1
Chopin
Enter Vacation Time Taken: 5
Copland
Enter Vacation Time Taken: 4
Hoover
Enter Vacation Time Taken: 3
John
Enter Vacation Time Taken: 2
Lincoln
Enter Vacation Time Taken: 1
Lamb
Enter Vacation Time Taken: 5
Skies
Enter Vacation Time Taken: 4
Skies
Enter Vacation Time Taken: 3
Washington
Enter Vacation Time Taken: 2
```

A Procedure to Store New Records

```
Define procedure store-loop
Ready personnel write
Repeat *.number-of-records-to-be-stored store personnel
end-procedure;


:store-loop
Enter number of records to be stored: 1
Enter FIRST_NAME: Virginia
Enter LAST_NAME: Sventek
Enter SALARY: 400
Enter HOURS_WORKED: 40
Enter VACATION_TAKEN: 0
Enter SICK_LEAVE_TAKEN: 0
```

A Procedure to Store New Records and Generate a Report:

```
Define procedure store-report
Ready personnel write
Repeat *.number-of-records-to-be-stored store personnel
Report
Set report-name = "Employee Report"/
"First Quarter 1982"
At top of hours-worked print hours-worked
print first-name, last-name, last-name via employee-table, salary
end-report;
end-procedure;
```

```
:store-report

Enter number of records to be stored: 1
Enter FIRST_NAME: Virginia
Enter LAST_NAME: Sventek
Enter SALARY: 400
Enter HOURS_WORKED: 40
Enter VACATION_TAKEN: 0
Enter SICK_LEAVE_TAKEN: 0
```

Employee Report
First Quarter 1982

27-Jul-1982
Page 1

HOURS WORKED	FIRST NAME	LAST NAME		SALARY
30				
	Don	Bumble	Scientific Staff	$ 322.00
	Aaron	Copland	Technical Personnel	$ 577.50
	Abraham	Lincoln	Scientific Staff	$ 660.00
35				
	Joe	Blow	Scientific Staff	$ 375.00
	James	Bond	Scientific Staff	$ 480.00
	John	Lennon	Scientific Staff	$ 500.00
	Misty	Skies	Scientific Staff	$ 500.00
	George	Washington	Program Manager	$ 572.00
40				
	Susan	Anthony	Program Manager	$ 638.00
	John	Bach	Scientific Staff	$ 450.00
	Humphrey	Bogart	Program Manager	$ 450.00
	Fredric	Chopin	Scientific Staff	$ 460.00
	Herbert	Hoover	Scientific Staff	$ 375.00
	Mary	Lamb	Scientific Staff	$ 500.00
	Stormy	Skies	Scientific Staff	$ 475.00
	Virginia	Sventek	Scientific Staff	$400.00
	Judy	Brown	Support Personnel	$ 400.00

This report was done with support from the Department of Energy. Any conclusions or opinions expressed in this report represent solely those of the author(s) and not necessarily those of The Regents of the University of California, the Lawrence Berkeley Laboratory or the Department of Energy.

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.

TECHNICAL INFORMATION DEPARTMENT
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720