VAX CDD/Plus User's Guide

Order Number: AA–KL46A–TE

August 1988

This manual provides tutorial information for VAX CDD/Plus. It supersedes the *VAX Common Data Dictionary User's Guide.*

Operating System and Version:	VMS
Software Version:	VAX CDD/Plus V4.0

digital equipment corporation, maynard, massachusetts

The postpaid Reader's Comments forms at the end of this document request the user's critical evaluation to assist in preparing future documentation.

Contents

Page

How to Use This Manual xv

Technical Changes and New Features xix

1 The CDD/Plus Dictionary System

1.1 How CDD/Plus Affects Current DMU Users 1–2
1.2 Overview 1–3
1.2.1 The CDO Interface 1–4
1.2.2 Distributed Dictionary Access 1–5
1.2.3 Field-Level Data Descriptions 1–5
1.2.4 Relationships 1–6
1.2.5 Pieces Tracking 1–6
1.2.6 Data Security and Integrity 1–7
1.2.7 CDD/Plus Call Interface 1–7
1.3 CDD/Plus Dictionary Naming Conventions 1–7
1.3.1 Parts of the Dictionary Definition Name 1–7
1.3.2 Accessing Dictionary Definitions 1–8
1.4 Emerging Support for CDO Dictionary Features 1–9

2 Using DMU with CDD/Plus

2.1 The CDD/Plus Compatibility Scheme 2–1
2.1.1 Translating Dictionary Formats 2–2
2.1.2 Accessing Definitions in CDD/Plus 2–4
2.1.3 Resolving Path Names to Dictionary Definitions 2–6
2.2 Creating, Reading, and Deleting Definitions in Compatibility Mode 2–7
2.2.1 Interpreting CDO Dictionary Features in DMU Format 2–8
2.2.2 Interpreting DMU Dictionary Features in CDO Format 2–10
2.2.3 Translating COBOL Level 88 Conditions into DMU and CDO Formats 2–12
2.3 Protecting Dictionary Definitions in Compatibility Mode 2–13
2.4 Converting Record Definitions from DMU to CDO Format 2–14
2.4.1 Sharing Fields After Record Conversion 2–17
2.4.2 Creating Directory Names for Definitions 2–18

3 Using the CDO Utility

3.1 CDO Utility Features 3-1
3.2 Logical Dictionary Structure 3-2
3.3 Getting Started with CDO 3-5
3.3.1 Invoking CDO 3-5
3.3.2 Running DCL Subprocesses 3-6
3.3.3 Creating a New Dictionary 3-7
3.3.4 Creating Dictionary Directories 3-8
3.3.5 Accessing Help in the CDO Environment 3-10
3.4 Editing Definitions in the CDO Environment 3-11
3.4.1 CDO Editor Key Definitions 3-11
3.4.2 Accessing Help in the CDO Editor 3-15
3.4.3 A Sample CDO Editing Session 3-15
3.4.4 CDO Editor Prompts 3-28
3.4.5 Editing Text Within the CDO Editor 3-28
3.4.6 Editing New Versions of Existing Definitions 3-29
3.4.7 Browsing Through Current Definitions 3-29
3.4.8 Exiting from the CDO Editor 3-29
3.5 Summary of CDO Environment Commands 3-30
3.6 Setting CDO Environment Characteristics 3-32
3.6.1 Sending Command Output to a File 3-32
3.6.2 Setting the Default Directory 3-32
3.6.3 Using Search Lists 3-34
3.6.4 Defining Terminal Keys 3-36
3.6.5 CDO Initialization Files 3-37
3.7 Manipulating Dictionary Definitions 3-38
3.7.1 Listing Dictionary Definitions 3-38
3.7.2 Showing Dictionary Definitions 3-41
3.7.3 Examining Current Definitions 3-43
3.8 Executing CDO Environment Command Procedures 3-44
3.9 Checking Your Dictionary Version 3-46

4 Populating Your Dictionary

4.1 Creating Dictionary Definitions 4-1
4.1.1 Documenting Dictionary Definitions 4-2
4.1.1.1 The DESCRIPTION Attribute 4-2
4.1.1.2 The AUDIT Attribute 4-2
4.1.2 Creating Field Definitions 4-4
4.1.3 Creating Record Definitions 4-7
4.1.3.1 Bottom-Up Definitions 4-7
4.1.3.2 Top-Down Definitions 4-8
4.1.4 Creating Relationships 4-9

4.2 Supported CDD/Plus Data Types 4–12
4.2.1 Character String Data Types 4–12
4.2.2 Fixed-Point Data Types 4–13
4.2.3 Floating-Point Data Types 4–13
4.2.4 Complex Numbers 4–14
4.2.5 Decimal String Data Types 4–15
4.2.6 Other Data Types 4–15
4.2.7 VAX Information Architecture Support for CDD/Plus Data Types 4–16
4.2.8 VAX Language Support for CDD/Plus Data Types 4–18
4.3 Copying Dictionary Definitions 4–22
4.3.1 Copying Relationships 4–24
4.3.2 Copying a Complete Dictionary 4–24
4.4 Changing Dictionary Definitions 4–24
4.4.1 Creating New Versions of Definitions with the DEFINE Command 4–25
4.4.2 Changing Original Definitions with the CHANGE Command 4–26
4.4.2.1 Receiving Messages About Changes 4–26
4.4.2.2 Changing Field Definitions 4–28
4.4.2.3 Changing Record Definitions 4–29
4.4.3 Changing Relationships 4–30
4.4.3.1 Changing a Member with the CHANGE Command 4–31
4.4.3.2 Changing a Member with the DEFINE Command 4–31
4.4.3.3 Changing Relationships in a Network 4–32
4.5 Deleting and Purging Dictionary Definitions 4–33
4.5.1 Deleting Dictionary Definitions 4–34
4.5.2 Purging Dictionary Definitions 4–35

5 Protecting Your Dictionary

5.1 Controlling Dictionary Access 5–1
5.1.1 User Identifiers 5–1
5.1.2 Access Control Entries 5–2
5.1.3 Determining Access to the Dictionary 5–4
5.2 CDO Dictionary Protection Provisions 5–6
5.2.1 Default Protection Provisions 5–7
5.2.2 Listing Protection Provisions 5–9
5.3 Protecting Dictionary Definitions 5–10
5.3.1 Adding New Access Control Entries 5–10
5.3.2 Changing Access Control Entries 5–14
5.3.3 Deleting Protection Provisions 5–15
5.3.4 Protecting Interrelated Definitions 5–16

5.3.5 Accessing Remote Dictionary Definitions 5–17
5.3.6 Protecting Your Dictionary from Corruption 5–17

6 Managing Dictionary Usage and Change

6.1 Tracking Dictionary Usage 6–1
6.1.1 Using the SHOW Commands 6–2
6.1.2 Generating Messages by Changing Definitions 6–4
6.1.3 Accessing Messages about Changes 6–6
6.1.4 Recompiling Application Programs 6–7
6.2 Confirming Dictionary Integrity 6–8
6.2.1 Journaling 6–8
6.2.2 Saving Copies of Your Dictionary 6–8
6.2.3 Verifying the Dictionary Condition 6–9
6.2.4 Changing the Location of a Dictionary 6–13
6.3 Enhancing Dictionary Performance 6–14
6.3.1 Removing Unused Definitions 6–14
6.3.2 Deleting a Dictionary 6–15
6.3.3 Structuring Your Dictionary 6–15
6.3.4 Improving Dictionary Performance over a Network 6–15

7 Using VAX Rdb/VMS with VAX CDD/Plus

7.1 Introduction to Using CDD/Plus with Rdb/VMS 7–2
7.1.1 Who Should Create Definitions in CDO? 7–3
7.1.2 Moving Between the RDO and CDO Utilities 7–4
7.2 Creating a Database with Shareable CDO Definitions 7–5
7.2.1 Defining Entities in CDO 7–6
7.2.2 Displaying Entities and Attributes Used By Databases 7–11
7.2.3 Defining a New Database 7–13
7.2.4 Requiring the Dictionary for Database Definitions 7–14
7.3 Including Dictionary Definitions in Your Database 7–15
7.3.1 Naming Database Definitions 7–17
7.3.2 Duplicate Processing Names 7–19
7.3.3 Defining Relations Using Dictionary Definitions 7–20
7.3.4 Completing Your Database Design with RDO Statements 7–22
7.3.5 Referencing Definitions from Programs 7–23
7.4 Changing and Integrating Definitions 7–25
7.4.1 Updating Dictionary Definitions with INTEGRATE IN 7–25
7.4.2 Updating Database Definitions with INTEGRATE FROM 7–26

7.4.3 Evaluating the Impact of Changes 7-27
7.4.4 Changing Definitions from CDO 7-28
7.4.4.1 The Impact of Immediate Changes 7-28
7.4.4.2 The Impact of Changes Over Time 7-29
7.4.5 Changing Shareable Definitions from RDO 7-31
7.4.6 Deleting Shared Definitions 7-34
7.5 Restoring a Database That Uses Shareable Dictionary Definitions 7-35
7.5.1 Backing Up and Restoring Databases 7-35
7.5.2 Restructuring and Reloading Databases 7-35
7.6 Converting Databases to CDO Format 7-36
7.6.1 Copying Definitions Into the Compatibility Dictionary 7-36
7.6.2 Sharing Converted Definitions 7-37
7.7 Deleting Databases 7-38

8 Managing RMS Files with CDO

8.1 Using CDD/Plus with RMS: An Overview 8-1
8.2 Creating RMS Database Definitions 8-3
8.3 Creating Physical Database Files 8-4
8.4 Using RMS File Definitions in Programs and Applications 8-7
8.4.1 Creating Relationships with REFERENCE FROM DICTIONARY 8-8
8.4.2 Tracking File Program Components 8-8
8.5 Showing Databases and Database Definitions 8-8
8.6 Moving Databases with CHANGE DATABASE 8-11
8.7 Deleting Databases and Database Definitions 8-11

A User's Guide to DMU Format Dictionaries

A.1 DMU Dictionary Structure A-2
A.1.1 Sample DMU Dictionary A-2
A.1.2 Subdictionary Directories in the DMU Dictionary A-5
A.1.3 CDD Types in the DMU Dictionary A-5
A.2 DMU Dictionary Paths A-6
A.2.1 The Given Name in the DMU Dictionary A-6
A.2.2 The Full DMU Dictionary Path Name A-7
A.2.3 Version Numbers in Object Names in the DMU Dictionary A-7
A.2.4 The Default DMU Dictionary Directory A-7
A.3 CDD Utilities in the DMU Dictionary A-8
A.3.1 Dictionary Management Utility (DMU) A-8
A.3.2 Dictionary Verify/Fix Utility (CDDV) A-10

A.3.3 Dictionary Data Definition Language Utility (CDDL) A-11
A.4 Using DMU and CDDV in the DMU Dictionary A-12
A.4.1 Issuing DMU and CDDV Commands A-12
A.4.1.1 System Level Commands A-12
A.4.1.2 DCL and Utility Command Procedures A-14
A.4.1.3 Halting a DMU Command Procedure A-15
A.4.2 Reading Command Lines in the DMU Dictionary A-16
A.4.2.1 Abbreviations A-16
A.4.2.2 Exclamation Point A-17
A.4.2.3 Hyphens in Utility Command Lines A-17
A.4.2.4 CTRL/C in DMU Commands A-17
A.4.3 Specifying Paths in the DMU Dictionary A-18
A.4.3.1 Path Name A-18
A.4.3.2 Given Name A-19
A.4.3.3 Version Numbers in Object Names A-19
A.4.3.4 Wildcard Characters A-20
A.4.3.5 Hyphens in Path Specifications A-22
A.4.3.6 Specifying Passwords in Path Names A-22
A.4.3.7 Using Logical Names in Path Names A-23
A.4.4 Defining Your Default Directory in the DMU Dictionary A-24
A.5 Using DMU Commands in the DMU Dictionary A-25
A.5.1 Checking Your Privileges in the DMU Dictionary A-25
A.5.2 Copying a Directory in the DMU Dictionary A-27
A.5.3 Changing Default Directories in the DMU Dictionary A-29
A.5.4 Creating History List Entries in the DMU Dictionary A-30
A.5.5 Listing the Contents of a DMU Dictionary Object A-31
A.5.6 Removing Obsolete DMU Dictionary Directories and Objects A-33
A.5.6.1 Backing Up Portions of the DMU Dictionary A-35
A.5.6.2 Purging DMU Dictionary Objects A-36
A.5.6.3 Deleting DMU Dictionary Objects A-36
A.5.7 Restoring Portions of the DMU Dictionary A-37
A.5.7.1 Checking the Contents of a Backup File . . . A-37
A.5.7.2 Restoring a Backup File A-38
A.5.8 Creating Multiple Versions of DMU Dictionary Objects A-39
A.5.8.1 Using RENAME/VERSION to Change a Version Number A-40
A.5.8.2 Using COPY/VERSION A-41
A.5.9 Exiting from DMU A-42
A.6 Security and Protection for the DMU Dictionary A-42

A.6.1 Access Control Lists and Access Control List Entries in the DMU Dictionary A–43
A.6.1.1 User Identification Criteria A–44
A.6.1.1.1 VMS User Name A–45
A.6.1.1.2 VMS User Identifier A–45
A.6.1.1.3 Terminal Number or Job Class A–46
A.6.1.1.4 Passwords A–46
A.6.1.2 Access Control Privileges A–47
A.6.1.3 Suggestions for Using Access Control Lists A–49
A.6.2 Sample Access Control Lists in the DMU Dictionary . . A–50
A.6.2.1 Accumulation of Privileges A–51
A.6.2.2 Combinations of ACL Entries A–52
A.6.3 Summary of ACL Results in the DMU Dictionary A–54
A.6.4 Modifying Access Control Lists in the DMU Dictionary A–54
A.6.5 Using VMS File Protection in the DMU Dictionary . . . A–63
A.6.5.1 The DCL SET PROTECTION Command A–64
A.6.5.2 Granting Access to Certain Users A–64
A.6.5.3 Controlling Access to Subdictionaries A–65
A.6.5.4 The DCL Access Control List Editor Utility A–66
A.6.6 Overriding Security in the DMU Dictionary A–68
A.7 Creating a CDDL Source File in the DMU Dictionary A–68
A.7.1 The CDDL Source File A–68
A.7.2 Creating a Record Definition in the DMU Dictionary A–69
A.7.3 Describing the Record Definition in the DMU Dictionary A–70
A.7.3.1 Field Description Statements A–70
A.7.3.2 Documenting Field Description Statements A–72
A.7.3.3 Field Attribute Clauses A–73
A.7.3.4 CDDL Data Types A–74
A.7.3.4.1 Character String Data Types A–75
A.7.3.4.2 Fixed Point Data Types A–75
A.7.3.4.3 Floating Point Data Types A–75
A.7.3.4.4 Decimal String Data Types A–76
A.7.3.4.5 Other Data Types A–77
A.7.3.4.6 Language Support for CDDL Data Types A–78
A.8 Compiling, Modifying, and Using CDDL Record Definitions in the DMU Dictionary A–83
A.8.1 Compiling a New Record Definition in the DMU Dictionary A–84

A.8.2 Using CDD Record Definitions in the DMU Dictionary A-87
A.8.2.1 Copying CDD Definitions into a Program A-87
A.8.2.2 Documenting the Use of Record Definitions A-88
A.8.3 Modifying CDDL Source Files in the DMU Dictionary A-89
A.8.3.1 Obtaining the Source File A-89
A.8.3.1.1 Extracting CDDL Source Text A-89
A.8.3.1.2 Extracting Record Definitions Without CDDL Source Text A-90
A.8.3.2 Replacing a CDDL Definition A-91
A.8.3.3 Creating an Additional Version of a Record Definition A-91
A.8.3.4 Recompiling Programs and Product Definitions A-92
A.8.3.5 Recompiling CDD Record Definitions A-92
A.8.3.5.1 Checking the Template Record's History List A-93
A.8.3.5.2 Checking Template Records with DMU EXTRACT A-94
A.8.3.5.3 Using CDDL/RECOMPILE A-96
A.9 Organizing and Maintaining Your Dictionary in the DMU Dictionary A-97
A.9.1 Organizing Your DMU Dictionary Directory Hierarchy A-98
A.9.1.1 By Organizational Entity A-98
A.9.1.2 By Application A-98
A.9.1.3 By Individual User A-98
A.9.1.4 By a Combination of Criteria A-99
A.9.1.5 Using Subdictionaries A-99
A.9.1.5.1 Creating Subdictionaries A-100
A.9.1.5.2 Deleting Subdictionaries A-101
A.9.1.5.3 Moving Subdictionary Files and Directories A-101
A.9.2 Organizing Your DMU Dictionary to Enhance Performance A-102
A.9.2.1 Using the Directory Hierarchy A-102
A.9.2.2 Reducing DMU Dictionary Size A-102
A.9.2.3 Limiting the Creation of Subdictionaries . . . A-103
A.9.2.4 Preventing the Creation of History Lists . . . A-103
A.9.3 Creating a Sample Hierarchy from a DMU Command Procedure in a DMU Dictionary A-103

A.9.4 Maintaining DMU Dictionary Files A-107
A.9.4.1 Verifying the Condition of a DMU Dictionary File A-107
A.9.4.2 Fixing a DMU Dictionary File A-108
A.9.4.3 Compressing a DMU Dictionary File A-110
A.10 Setting Up Your DMU Dictionary on a VAXcluster A-110
A.10.1 Installing Your First DMU Dictionary A-111
A.10.2 Moving a Single Existing DMU Dictionary to a Cluster Disk A-111
A.10.3 Merging Two or More DMU Dictionaries onto a Cluster Disk A-112
A.10.3.1 Creating a Single Logical DMU Dictionary A-114
A.10.3.2 Reorganizing and Eliminating Redundant Definitions A-116
A.10.3.2.1 Planning the Hierarchy of the Cluster Dictionary A-117
A.10.3.2.2 Identifying Redundant DMU Dictionary Elements A-117
A.10.3.2.3 Moving Portions of Former Subdictionaries Within the Cluster Dictionary A-118
A.10.3.3 Modifying Former Absolute Path Names . . . A-119

Glossary

Index

Examples

A-1 ADDRESS.DDL, Sample CDDL Source File A-69

Figures

1-1 Accessing CDD/Plus Dictionary Definitions with DIGITAL Products 1-4
1-2 Valid CDD/Plus Definition Names 1-8
1-3 Using CDO Metadata for Project Control 1-10
2-1 The CDD/Plus Access Routes 2-5
3-1 Sample Logical Dictionary Structure 3-4
3-2 CDO Editor Keypad Key Definitions 3-12
3-3 CDO Editor Menu Keypad Key Definitions 3-13
4-1 Creating Local Copies of Remote Dictionary Definitions 4-33
6-1 Placing Directories in Your Dictionary 6-16
7-1 Sharing Dictionary Definitions Among Database Products 7-3
7-2 Shareable CDO Definitions 7-10

A-1 Sample DMU Dictionary Hierarchy A-3
A-2 Portions of Hierarchies of Two DMU Dictionaries A-113
A-3 DMU Dictionaries on ALPHA and OMEGA Merged A-113

Tables

2-1 Accomplishing Tasks Through DMU and CDO Access Routes 2-6
2-2 Interpretation of CDO Definitions Through a DMU Access Route 2-8
2-3 How CDD/Plus Affects DMU RENAME 2-10
2-4 Translation of DMU Dictionary Protection Provisions to CDO Format 2-12
3-1 Suggested Process Resource Limits 3-7
3-2 Frequently Used CDO Editor Key Functions 3-14
3-3 Summary of CDO Commands 3-30
4-1 User-Specified Field Attributes 4-4
4-2 CDD/Plus Relationships 4-11
4-3 Fixed-Point Data Types 4-13
4-4 Floating-Point Data Types 4-14
4-5 Complex Numbers 4-14
4-6 Decimal String Data Types 4-15
4-7 Other Data Types 4-16
4-8 VAX Information Architecture Products Support for CDD/Plus Data Types 4-17
4-9 VAX BASIC, COBOL, DIBOL, and PL/I Support for CDD/Plus Data Types 4-19
4-10 VAX FORTRAN, C, PASCAL, and RPG II Support for CDD/Plus Data Types 4-21
5-1 VMS Access Rights 5-3
5-2 CDO Functional Access Rights 5-6
5-3 Protocol Protection 5-7
6-1 Summary of Usage Tracking Commands 6-2
6-2 Qualifiers to the VERIFY Command 6-12
8-1 Summary of Commands For Manipulating RMS Files 8-2
8-2 DEFINE_RMS_DATABASE Attribute Clauses 8-4
A-1 The Dictionary Management Utility Commands A-9
A-2 The Dictionary Verify/Fix Utility Commands A-11
A-3 The Dictionary Data Definition Language Utility Commands A-12
A-4 Specifying Version Numbers with CDD Objects in the DMU Dictionary A-19
A-5 Dictionary Management Utility Wildcard Characters A-21
A-6 Access Control Privileges in the DMU Dictionary A-48
A-7 General Field Attribute Clauses A-73
A-8 Facility-Specific Field Attribute Clauses in the DMU Dictionary . . . A-74
A-9 The Fixed-Point Data Types in the DMU Dictionary A-75
A-10 The Floating Point Data Types in the DMU Dictionary A-76
A-11 The Decimal String Data Types in the DMU Dictionary A-77

A-12 BASIC, COBOL, DIBOL, and PL/I Support for CDDL Data Types in the DMU Dictionary A-79
A-13 FORTRAN, C, PASCAL, and RPG II Support for CDDL Data Types in the DMU Dictionary A-80
A-14 VAX Information Architecture Products Support for CDDL Data Types in the DMU Dictionary A-82

How to Use This Manual

This manual provides tutorial material for users who plan to store, maintain, and analyze data descriptions in CDD/Plus using the CDO utility. Tutorial material is also provided for using the DMU utility.

Intended Audience

The audience for this manual consists of the following groups:

- The data administrator or system manager responsible for creating the dictionary hierarchy, setting up the security provisions, and maintaining the dictionary structure
- The database administrator responsible for creating standard definitions that can be shared among databases
- Programming supervisors responsible for maintaining portions of the dictionary hierarchy
- Programmers responsible for maintaining their own portions of the directory hierarchy, and for writing applications that use the definitions stored in CDD/Plus

Operating System Information

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

Structure

This manual has eight chapters, a glossary, and one appendix.

Chapter 1	Provides an overview of CDD/Plus and the features of CDO dictionaries.
Chapter 2	Explains how CDD/Plus operates when established DMU dictionaries and CDO dictionaries coexist.
Chapter 3	Helps you get started using the CDO utility.
Chapter 4	Describes how to define fields and records in a CDO dictionary.
Chapter 5	Describes how to protect CDO dictionary definitions.
Chapter 6	Describes how to analyze CDO dictionary usage and how to manage changes. It also discusses dictionary performance.
Chapter 7	Describes how to use shareable CDO dictionary definitions with Rdb/VMS.
Chapter 8	Describes how to create and use CDO dictionary definitions that represent VAX Record Management Services database entities.
Appendix A	Provides tutorial material on DMU dictionaries for users who access the dictionary with products that do not support the CDO features of CDD/Plus.
Glossary	Defines terms used in this manual and other manuals in the documentation set.

Related Documents

The other manuals in the CDD/Plus documentation set are:

- *VAX CDD/Plus Common Dictionary Operator Reference Manual*

  Provides reference material and syntax for all CDO commands

- *VAX Common Data Dictionary Data Definition Language Reference Manual*

  Describes the VAX Common Data Dictionary Data Definition Language Utility (CDDL), which manipulates definitions in DMU dictionaries

- *VAX Common Data Dictionary Utilities Reference Manual*

  Describes the Dictionary Management Utility (DMU) and the Dictionary Verify/Fix Utility (CDDV), utilities you use to manipulate DMU dictionaries

- *VAX CDD/Plus Call Interface Manual*

  Provides reference material for the system administrator on CDO dictionary architecture.

Conventions

The special symbols used in this book are:

Symbol	Meaning
$\boxed{\text{CTRL/x}}$	This symbol tells you to press the CTRL (control) key and hold it down while pressing the specified letter key.
KPn	Key names that begin with KP indicate keys on the numeric keypad on the right side of the terminal keyboard.
SHIFT	The CDO editor uses the PF1 key as a shift key.
SHIFT-KPn	The hyphen in key names means that you press the two keys in the order listed.
$\boxed{\text{RET}}$	This symbol indicates the RETURN key.
BOLD	Bold lettering indicates the definition of a new term.
. . .	Vertical ellipsis in an example means that information not directly related to the example has been omitted.
$	The dollar sign is used to indicate the DCL prompt. This prompt may be different on your system.
COLOR	Color in examples shows user input.

Technical Changes and New Features

CDD/Plus supports dictionary definitions in two distinct formats:

- DMU format—dictionary definitions that can be created and manipulated with the DMU, CDDL, and CDDV utilities.
- CDO format—dictionary definitions that can be created and manipulated with the CDO utility and the CDD/Plus call interface.

In addition to providing support for DMU dictionaries, CDD/Plus provides the following major features for CDO dictionaries:

An Easy-to-Use Interface

CDD/Plus provides a single user interface, known as CDO, where you can accomplish all data definition, administration, and management functions for CDO dictionaries. A flexible, menu-driven editor allows you to enter common field and record definitions easily. You can also read your DMU dictionary from CDO.

Distributed Dictionary Implementation

CDO dictionaries can be located on different devices on a single node, on different nodes on a VAXcluster, and on local or wide area networks. The CDO utility allows you to access all of these dictionaries, as well as your system DMU dictionary and subdictionaries, as one logical dictionary.

Field-Level Data Descriptions

You can create and access field definitions as the smallest fundamental data definitions in CDO dictionaries. Field definitions can be shared by many other dictionary definitions, thereby reducing redundancy of data.

Relationships

CDD/Plus allows you to connect CDO dictionary definitions in various ways. You do not need to define the relationships; CDD/Plus implicitly creates these relationships for you when you create CDO dictionary definitions. You can form relationships between definitions that are stored in different dictionaries across a network. CDD/Plus allows you to obtain information about the relationships between CDO dictionary definitions. You cannot create relationships between definitions in DMU dictionaries or between CDO dictionary definitions and DMU dictionary definitions.

Pieces Tracking

CDD/Plus keeps track of all dictionary usage in CDO dictionaries. With CDO commands, you can locate the definitions that would be affected if a particular definition were to be changed.

With CDO definitions you can make changes that take effect immediately, or changes that can be incorporated into related definitions over time. When an immediate change is made to a field definition, the appropriate changes are automatically made to record definitions that include the field definition. Alternatively, creating new versions provides you with a means of integrating changes over time. CDD/Plus attaches an informational message about the change to CDO definitions that do not automatically include the change.

Data Security and Integrity

CDO dictionaries implement protection provisions that are consistent with VMS and Rdb/VMS.

Integrity is a critical factor in the success of any dictionary operation. For this reason, CDD/Plus provides automatic journaling capabilities and commands you can use to verify the dictionary condition.

Call Interface

You can make direct calls from user programs to CDD/Plus routines that manipulate dictionaries with CDO format. This manual does not document how to make these calls; the call interface is documented in the *VAX CDD/Plus Call Interface Manual.*

Compatibility with DMU Dictionaries

CDD/Plus provides an automatic translation utility that allows you to read definitions in DMU dictionaries from the CDO interface.

The DMU, CDDL, and CDDV utilities are provided in addition to the CDO utility. You can continue to use these utilities to manipulate definitions in DMU dictionaries

- If you use VAX layered products that do not yet support CDO dictionary features
- If you need to perform write operations to DMU dictionaries

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

The VAX Information Architecture documentation explaining how these products interrelate is included with the CDD/Plus documentation. VAX Information Architecture documentation is also available separately. To order documentation, contact your DIGITAL representative.

The VAX CDD/Plus documentation to which this document belongs often refers to other DIGITAL products by their abbreviated names.

- VAX ACMS software is referred to as ACMS.
- VAX CDD software—released prior to VAX CDD/Plus—is referred to as CDD.
- VAX CDD/Plus software is referred to as CDD/Plus.
- VAX DEC/CMS Code Management System is referred to as CMS.
- VAX DATATRIEVE software is referred to as DATATRIEVE.
- VAX Rdb/VMS software is referred to as Rdb/VMS.
- VAX DBMS software is referred to as VAX DBMS.
- VAX TDMS software is referred to as TDMS.

The CDD/Plus Dictionary System 1

CDD/Plus is a data dictionary system that provides the ability to create, analyze, and administer metadata. **Metadata** describes data and describes how that data is used. Metadata includes the location, type, format, size, change history, and usage of the actual data.

CDD/Plus supports metadata stored in two formats: metadata that you manipulate with the Common Dictionary Operator (CDO dictionaries) and metadata that you manipulate with the Dictionary Management Utility (DMU dictionaries).

- CDO Dictionaries

  CDD/Plus dictionaries created in CDO can store not only definitions, but also information about how the definitions are related. When you install CDD/Plus, it creates a CDO compatibility dictionary on your system. The **compatibility dictionary** is a special overlapping CDO dictionary that coordinates DMU format definitions and CDO format definitions.

  After installation, you can create additional CDO dictionaries, and relate them to one another. When it is necessary to distinguish them from the compatibility dictionary, these are called **user-created** dictionaries.

  You can create and manipulate definitions in CDO dictionaries and read definitions in DMU dictionaries through the CDO utility. The body of this manual describes CDO dictionary features.

- DMU Dictionaries

  CDD/Plus continues to support DMU dictionaries. CDD/Plus installation installs the utilities that manipulate DMU dictionary definitions—DMU, CDDL, and CDDV. You can create and delete DMU dictionary definitions through the DMU utility; however, you can read DMU definitions through either DMU or CDO. Tutorial material for creating DMU dictionary definitions is provided in Appendix A.

You can access all of your physical dictionaries as one logical dictionary. (For more information about the logical dictionary, see Section 3.2.)

1.1 How CDD/Plus Affects Current DMU Users

For the most part, the features of CDO dictionaries do not affect established DMU dictionaries. This means that you can continue to access established DMU dictionaries with products such as DATATRIEVE and ACMS.

Since CDD/Plus provides the DMU, CDDL, and CDDV utilities, in addition to the CDO utility, you can perform all your dictionary work as with previous releases of CDD.

The special features of CDO dictionaries require definitions to be stored in a format completely different from that of DMU record definitions. CDD/Plus automatically translates definitions in DMU dictionaries so that you can read them from the CDO interface.

Chapter 2 explains how to continue using DMU with CDD/Plus. You should continue using DMU if:

- You have a large existing application that you do not want to convert to CDO format immediately
- You use *only* products that write definitions in DMU format

Most VAX languages and VAX Information Architecture products can *read* record definitions from the *compatibility dictionary*. Products that can also *read to and write from user-created CDO dictionaries* can take advantage of pieces tracking and other CDO features described in Section 1.2.

Depending on the product that you are using, you can do one or several of the following:

- Read record definitions from CDO dictionaries.
- Read and write record definitions in CDO dictionaries.

- Read data definitions other than records from CDO dictionaries. (For example, RALLY can read an RMS file, CDD$DATABASE.)
- Write data definitions other than records into CDO dictionaries. (For example, Rdb/VMS can write a CDD$DATA_AGGREGATE_CONTAINS relationship when defining a view in the dictionary.)

To determine the CDO support currently provided by a VAX product, consult the documentation for that product.

1.2 Overview

The CDD/Plus data dictionary:

- Ensures the integrity of shared metadata and the procedures used to analyze, maintain, manage, and design business metadata
- Provides a centralized repository for information management shops
- Offers a dynamic aid to software application development

CDD/Plus enables the dictionary administrator to manage information and application resources by allowing shared and controlled access to all metadata and by auditing the dictionary usage. Since the dictionary controls all changes to the metadata, the more the data administrator enforces dictionary usage, the more consistent and accurate data will be.

The dictionary contains metadata in the form of dictionary definitions. A CDD/Plus **dictionary definition** can contain various attributes and can be related to other CDD/Plus dictionary definitions. The most commonly used dictionary definitions are fields, records, and databases. Many products can share the metadata and data at one time.

You can store and maintain the actual data values outside the data dictionary in several ways; for example, with a database management system like Rdb/VMS, in RMS files, in CMS libraries, or even off line.

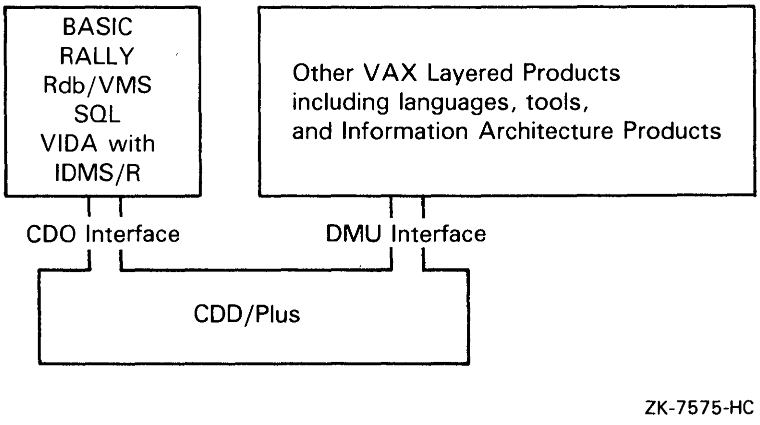
Figure 1-1 illustrates how different products currently access dictionary definitions through CDD/Plus.

CDD/Plus provides the following features:

- A single user interface
- Distributed dictionary access

- Field-level data descriptions
- Relationships between dictionary definitions
- Pieces tracking
- Data security and integrity
- Call interface

Figure 1–1: Accessing CDD/Plus Dictionary Definitions with DIGITAL Products


BASIC
RALLY
Rdb/VMS
SQL
VIDA with IDMS/R
Other VAX Layered Products including languages, tools, and Information Architecture Products
CDO Interface
DMU Interface
CDD/Plus
ZK-7575-HC

The following sections describe these features in detail. Section 1.4 discusses the emerging support and anticipated uses of these features.

1.2.1 The CDO Interface

The CDO utility provides a single user interface to CDD/Plus capabilities where you can accomplish all of the data description, administration, and dictionary management functions. The CDO utility provides a flexible, menu-driven editor that allows you to easily create common field and record definitions. CDO also supports the VMS style of command entry. Chapter 3 discusses how to create definitions with the CDO editor and summarizes CDO commands.

1.2.2 Distributed Dictionary Access

Through the CDO interface, you can access metadata in CDO dictionaries and directories

- On different devices on a single node
- On different nodes in a VAXcluster
- On nodes connected by a local or wide area network

You can access metadata in all these places as a single **logical dictionary**, provided that you have the appropriate access rights. You can also access your DMU dictionaries from CDO. This versatility affords you greater security for sensitive parts of a dictionary and greater flexibility for storing large dictionary files.

1.2.3 Field-Level Data Descriptions

A **field definition** is the smallest unit of metadata that can be created and accessed in the dictionary. Because each piece of metadata is a separately addressable entity, CDD/Plus is known as a field-level dictionary. Field definitions typically include information about the data type and size, and other optional attributes.

Field definitions can be simple data structures or complex subscripted structures. They can be combined to form various record definitions and can be accessed individually from several of the VAX layered products. You only need to store one copy of a particular definition that is used by various sources.

CDD/Plus keeps track of dictionary definition usage at the field level. Therefore, you can easily show which dictionary entities make use of a particular field definition. When you or someone else changes a field definition, you can identify which entities the change may affect and which entities need to be redefined in order to access the changed field. This ability to track entities is known as **pieces tracking**.

A **record definition** is a dictionary entity that typically consists of a grouping of field definitions. You can combine field and record definitions into complex record structures.

You organize your dictionary definitions by creating a dictionary directory structure. Directories map each definition name to a certain location. A **directory** is not a dictionary definition, but contains dictionary definitions and other directories. Field, record, and other data definitions are grouped in directories. Dictionary directories are similar in concept to VMS directories; they allow you to group definitions in your dictionary and enable you to organize the definitions hierarchically. You can use search lists and wildcards to specify definitions in directories.

1.2.4 Relationships

CDD/Plus creates **relationships** when you connect two CDO data definitions in some way.

For example, you can base the definition of a new field on a field definition that already exists in a CDO dictionary. You can also relate a group of field definitions to a record definition by including the field names in the record definition.

You do not need to define these relationships; CDO automatically creates them for you when you create your field and record definitions in CDO.

You can establish a relationship between two CDO definitions in different CDO dictionaries that are distributed on different devices on a single node, on different nodes in a VAXcluster environment, or on nodes connected by a local or wide area network. For example, you can create a record definition in one CDO dictionary that includes field definitions contained in another CDO dictionary. Chapter 4 discusses the most common relationships between CDO definitions.

1.2.5 Pieces Tracking

Because CDD/Plus keeps track of all CDO dictionary usage, you can find out which other dictionary entities make use of a particular field definition. Before you change a field definition, you can confirm which definitions the change may affect and which entities you must redefine to access the changed field definition.

For example, if you use a particular field definition in several different record definitions, and the record definitions are accessed in turn by other records and by an Rdb/VMS database, CDD/Plus can locate all the uses of the single CDO field definition. You find out about these relationships with the SHOW commands. The SHOW USES, SHOW USED_BY, and SHOW WHAT_IF commands help you to keep track of dependent and related definitions and to assess the impact of changes.

You can control changes to your definitions in two ways: you can change the original definition to take effect immediately or you can create a new version and allow users to incorporate the change over time. When you change or make a new version of a definition, dependent definitions that do not automatically include the change (such as Rdb/VMS databases) are flagged with an informational message about the change. Messages allow you to warn users when a new version of a dictionary definition exists or when inconsistencies may exist between the dictionary and external copies. Chapter 6 discusses how to keep track of dictionary usage and changes.

1.2.6 Data Security and Integrity

To protect dictionary files from unauthorized users, CDD/Plus provides the database administrator with the tools to grant or deny dictionary definition access rights. The CDD/Plus protection provisions for CDO definitions are consistent with Rdb/VMS and VMS protection schemes. Chapter 5 discusses how to protect the definitions in your dictionary.

Integrity, the completeness, accuracy, and consistency of definitions, is a critical factor in the success of any dictionary operation. For this reason, CDD/Plus provides journaling capabilities that automatically protect your dictionary sessions from system failures. Dictionary management features are discussed in Chapter 6.

1.2.7 CDD/Plus Call Interface

You can make direct calls to the CDD/Plus entry points from user programs. Using the call interface allows you to directly access CDO dictionaries without using the CDO utility. The call interface is documented in the *VAX CDD/Plus Call Interface Manual.*

1.3 CDD/Plus Dictionary Naming Conventions

In CDD/Plus, every dictionary definition has a full name that uniquely identifies it. The naming conventions for CDO definitions and DMU definitions differ only in their specification of the dictionary origin.

1.3.1 Parts of the Dictionary Definition Name

Definition names consist of three parts: the dictionary origin, the path, and the version number.

- The dictionary origin is the root of the dictionary. The CDO utility recognizes two representations of a dictionary origin: the anchor and CDD$TOP.

  To refer to CDO definitions, you specify the dictionary origin as a dictionary anchor. A dictionary **anchor** specifies the VMS directory where the CDO dictionary hierarchy is stored. It can optionally consist of node, device, and directory components; a *fully translated* anchor includes all three components. CENTRL::SYS$COMMON:[CDDPLUS] is an example of a fully translated anchor.

  To refer to DMU dictionary definitions, you specify the dictionary origin as CDD$TOP.

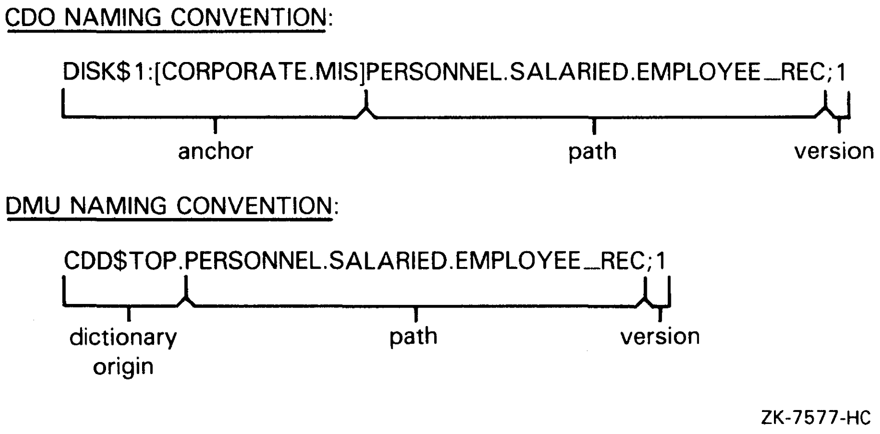
- A **path** consists of a list of dictionary directory names separated by periods and ending with an entity name. The path name specifies the path, or route, to the desired dictionary definition. EMPLOYEES.SALARIED.EMPLOYEE_REC, for example, represents the path that leads to the record definition EMPLOYEE_REC. Every name in the path except the last name is a dictionary directory. Path names reflect the hierarchy in your dictionary. A CDD/Plus path is similar to a VMS file specification.
- A **version** is similar to a VMS file version. The version number is always preceded by a semicolon (;). CDD/Plus allows you to create multiple versions of a dictionary definition. The maximum number of versions is 32,767.

Information about the maximum number of characters for a path and anchor can be found in *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

1.3.2 Accessing Dictionary Definitions

A **fully qualified name** specifies the complete dictionary location of a particular data definition. While fully qualified names are unique, the individual names of definitions in a dictionary need not be unique, provided that the path names are different. Although it is *not* recommended, you can create field and record definitions with the *same* names in *different* directories. When you create a field or record definition with the same name as an existing definition in the same directory, CDD/Plus creates a new version of the existing definition. Figure 1-2 shows two fully qualified CDD/Plus definition names.

Figure 1–2: Valid CDD/Plus Definition Names



You can use either the CDO or the DMU naming convention to access dictionary definitions from the CDO utility: Section 2.1 details the CDD/Plus compatibility provisions that enable you to do this.

Legal characters in a name include multinational alphanumeric characters such as the underscore character (_) and the dollar sign ($). You can use diacritical marks in the names of dictionary definitions such as fields and records, but not in the name of a dictionary directory.

1.4 Emerging Support for CDO Dictionary Features

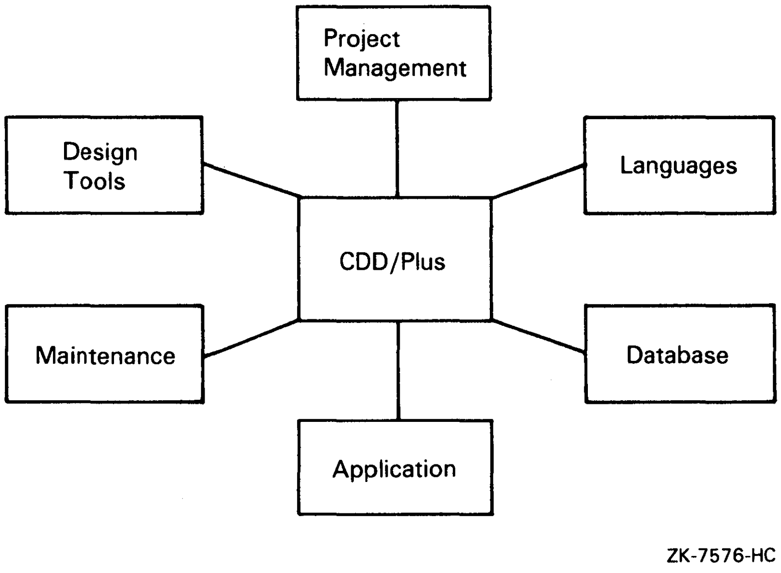
Managing data requires a significant amount of time and effort in application development, especially when two or more applications need to share common data in development. CDD/Plus enables data administrators to provide accurate and complete definitions for application developers. For information about application development using a dictionary as a common definition repository, see the VAX Information Architecture manual, *Introduction to Application Development.* Many products currently support the features of CDO definitions. These products include VIDA, a product that allows you to access data on IBM mainframes, and Rdb/VMS, a relational database system.

The VAX languages can include definitions stored in both the DMU and the CDO dictionaries.

For details on how to include dictionary definitions from a particular VAX language, see the documentation for that language.

As support for CDO dictionaries emerges, the CDO dictionary will play a more active role both in information management systems and in the development of software applications. CDO dictionary definitions are expected to be integrated with software tools, application development environments, database management systems, and information processing environments. Figure 1-3 shows the projected role of CDD/Plus in the life cycle of a product.

Figure 1–3: Using CDO Metadata for Project Control



CDD/Plus can be seen as the hub of the planning, design, and management cycle. While support for CDO dictionary features will emerge in stages, you should keep the support for the features you are using in mind when planning your own dictionary use. Section 2.1 describes the support that individual DIGITAL products currently provide for CDD/Plus.

Chapter 2 explains how CDD/Plus operates when you use DIGITAL products that currently access DMU dictionaries. It explains how to maintain dictionary definitions that can be accessed both from products that support user-created CDO dictionaries and those that still rely on DMU dictionaries.

Using DMU with CDD/Plus 2

If you do not currently maintain an established DMU dictionary, and if you intend to use CDD/Plus for only CDO dictionary definitions that are accessed by supporting products, you can skip this chapter and proceed to Chapter 3.

This chapter:

- Provides information about using the translation utility and compatibility dictionary to compensate for format differences between DMU definitions and CDO dictionary definitions
- Explains the different types of dictionary support provided by various VAX products
- Describes how to convert DMU definitions into CDO format and recommends when to convert

2.1 The CDD/Plus Compatibility Scheme

The CDD/Plus translation utility and compatibility dictionary make it possible for you to access your definitions without being concerned about which dictionary format the definitions are stored in. The **compatibility dictionary** is a special overlapping CDO dictionary that coordinates DMU format definitions and CDO format definitions.

The system logical name for the compatibility dictionary's file specification is CDD$COMPATIBILITY. The translation of CDD$COMPATIBILITY must include both a device and a directory.

The CDD/Plus compatibility dictionary provides you with the capability to:

- Continue to use your DMU dictionary definitions
- Create new definitions through CDO that can be read from products that support DMU dictionaries
- Create new definitions that can be accessed products beginning to support CDO dictionaries

You can access your DMU dictionary and all CDO dictionaries from the CDO utility, from RDO, and via the CDD/Plus call interface.

Definitions created in the compatibility dictionary can be interpreted by products that support DMU dictionaries and by those that support CDO dictionaries.

Your DMU directory structure is mapped to the directory structure in the compatibility dictionary, and vice versa. For example, when you create a new directory in the compatibility dictionary, the new directory is listed in your DMU directory hierarchy, just as your DMU directory structure is visible from the CDO utility.

You can refer to definitions in the compatibility dictionary with the naming conventions for either dictionary format. You can specify either an anchor or CDD$TOP as the dictionary origin for your compatibility dictionary.

Products that support DMU dictionaries and products that support CDO dictionaries can access definitions created (and stored) in the compatibility dictionary. Section 7.3.4.6 in Appendix A lists the VAX languages and VIA products that can read and write DMU dictionary definitions. Some of these products can read CDO definitions in user-created dictionaries as well as the compatibility dictionary. Other supporting products can both read definitions in any dictionary and write definitions in any CDO dictionary.

To determine the CDO support currently provided by a VAX product, consult the documentation for that product.

2.1.1 Translating Dictionary Formats

The **translation utility** translates definitions from CDO format into DMU format, and vice versa.

You refer to DMU dictionary definitions by a path name beginning with the origin CDD$TOP; you refer to CDO dictionary definitions by a path name beginning with an anchor. The translation utility translates CDD$TOP to be equivalent to the anchor of your compatibility dictionary (CDD$COMPATIBILITY).

For example, if your compatibility dictionary is installed in SYS$COMMON:[CDDPLUS], and you create a directory called PERSONNEL, you can refer to this directory with either of the following two naming conventions:

- CDD$TOP.PERSONNEL
- SYS$COMMON:[CDDPLUS]PERSONNEL

Since the system logical name CDD$COMPATIBILITY is equivalent to the anchor of the compatibility dictionary, you can confirm where your own compatibility dictionary is located by using the following SHOW LOGICAL command:

```
$ SHOW LOGICAL CDD$COMPATIBILITY
  "CDD$COMPATIBILITY" = "SYS$COMMON:[CDDPLUS]"  (LNM$SYSTEM_TABLE)
```

After installation, you can create other CDO dictionaries from the CDO utility. When you refer to definitions in such a user-created CDO dictionary, you must use the anchor for the dictionary origin, since the anchor in a user-created CDO dictionary is *not* associated with CDD$TOP.

When an application tries to access dictionary metadata, the translation utility automatically searches throughout your DMU dictionary as well as your compatibility dictionary. For example, VAX language programs currently support DMU dictionaries; however, the languages can also read definitions in the compatibility dictionary.

Specifying a path name beginning with CDD$TOP (either explicitly or by default) automatically ensures that both the DMU and the compatibility dictionaries are searched. The search is invisible to the user.

When a program attempts to include a definition, the translation utility searches for this definition in the DMU dictionary first. If this search fails, CDD/Plus replaces CDD$TOP with CDD$COMPATIBILITY and searches for the definition in the compatibility dictionary.

Caution

Your system manager has the ability to change the definition of the system logical name CDD$COMPATIBILITY to a different anchor from the one specified during the installation procedure. *DIGITAL strongly recommends that you do not redefine CDD$COMPATIBILITY once it contains definitions or directories.*

Such a change makes a different CDD/Plus dictionary equivalent to CDD$COMPATIBILITY. Once CDD$COMPATIBILITY is changed, you *must* specify CDD$TOP in your path names (either explicitly or by setting a default), or CDD/Plus searches for *only* a user-created CDO dictionary and will not locate definitions in the compatibility dictionary.

Changing CDD$COMPATIBILITY makes it difficult to maintain consistency of definitions within the DMU and compatibility dictionaries.

2.1.2 Accessing Definitions in CDD/Plus

Figure 2-1 illustrates the CDD/Plus architecture and the two different **access routes** into CDD/Plus: the CDO access route and DMU access route. The translation utility interprets definitions in the CDO format (in the compatibility dictionary) to DMU format, and vice versa.

Which access route you use depends on the product accessing CDD/Plus. Your access route controls where definitions are created and stored, as well as your write and delete access.

You use the CDO access route if you access with:

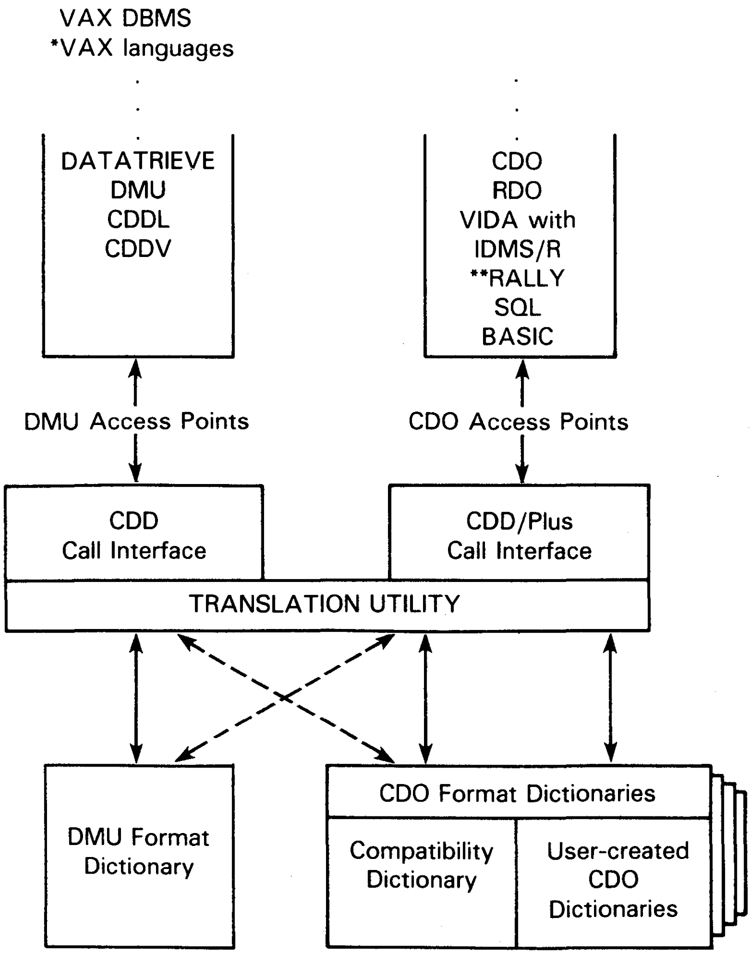
- CDO commands
- RDO commands (including callable RDO from a user program)
- Direct calls in a user program to the CDD/Plus entry points

You use the DMU access route if you access with:

- Any layered product that supports DMU dictionaries but not CDO dictionaries
- DMU, CDDL, and CDDV utility commands

You cannot access DMU dictionaries with direct calls to the dictionary in a user program. The call interface to CDO dictionaries is documented in the *VAX CDD/Plus Call Interface Manual.* Table 2-1 lists the tasks that you can and cannot accomplish through the two access routes. These restrictions *cannot* be overridden by protection provisions for individual definitions. For more information about protection provisions, see Chapter 5.

Figure 2–1: The CDD/Plus Access Routes



Key: Vertical lines indicate read, write, and delete access; dotted lines indicate read access only.

*Excluding BASIC

**You cannot create definitions in CDO dictionaries using the RALLY DATATRIEVE interface. For more information, see Table 2–1.

ZK-7578-HC

Table 2–1: Accomplishing Tasks Through DMU and CDO Access Routes

Dictionary Task	CDO Access Route	DMU Access Route
Create definitions in compatibility dictionary	YES	NO
Write history lists into compatibility dictionary	YES	YES
Delete definitions in compatibility dictionary	YES	NO
Read definitions in compatibility dictionary	YES	YES
Create definitions in a user-created CDO dictionary	YES	NO
Write history lists into a user-created CDO dictionary	YES	YES
Delete definitions in a user-created CDO dictionary	YES	NO
Read definitions in a user-created CDO dictionary	YES	YES
Create definitions in a DMU dictionary	NO	YES
Delete definitions in a DMU dictionary	NO	YES
Read definitions in a DMU dictionary	YES	YES
Create CDD/Plus messages[1]	YES	NO
Delete CDD/Plus messages	YES	NO
Read CDD/Plus messages	YES	NO

[1]CDO sends messages when a dictionary definition changes or when a new version of a definition is created. For more information about messages, see Section 4.4.2.1.

2.1.3 Resolving Path Names to Dictionary Definitions

You can create a *directory* in the DMU dictionary with the same name as a directory that exists in the compatibility dictionary, but you cannot create a *definition* in one dictionary with a pathname that matches that of a definition or directory in another dictionary. For example, a directory named CDD$TOP.EMPLOYEES.CONTRACT can exist in two dictionaries, but a field or record named CDD$TOP.EMPLOYEES.CONTRACT.RATE can exist in only one dictionary.

CDD/Plus can locate the definition regardless of which dictionary stores it. When you try to store new definitions in your compatibility dictionary, the translation utility searches through your DMU dictionary for an existing occurrence of the specified name. When a conflict exists, CDD/Plus signals an error and does not create the new definition.

When you create definitions with DMU, CDDL, DATATRIEVE, or with a VAX layered product that supports DMU dictionaries, your definitions are placed in a DMU dictionary, *not* in a CDO dictionary. The DMU, CDDL, and CDDV utilities can be invoked in the same manner as with previous versions of CDD/Plus. Although you cannot create a DMU definition through a CDO access route, such as RDO or CDO, definitions that you create in your DMU hierarchy can be read through the mapped hierarchy in your compatibility dictionary.

In summary:

- To create new definitions in CDO format, create your definitions in the CDO environment or in the RDO utility. Specify a path name beginning with CDD$TOP or the anchor for a user-created CDO dictionary.
- To create new definitions in DMU format, create your definitions through a DMU access route, such as CDDL or DATATRIEVE, and specify a path name beginning with CDD$TOP for the definition.

2.2 Creating, Reading, and Deleting Definitions in Compatibility Mode

Your access route to CDD/Plus affects whether or not you can create or delete dictionary definitions.

A useful rule of thumb is that you should delete definitions through the same access route as you create them: *delete where you create.* When you request a directory listing of definitions through a DMU access route, the record definitions in both DMU and CDO compatibility dictionaries are listed. You can read definitions in DMU and the compatibility dictionary regardless of your access route into CDD/Plus; however, you can delete definitions in DMU dictionaries only when you access the dictionary through a DMU access route. Similarly, you can delete definitions in compatibility and CDO dictionaries only when you access the dictionary through a CDO access route. Table 2-1 lists the tasks you can accomplish through each access route.

2.2.1 Interpreting CDO Dictionary Features in DMU Format

Because the features of CDO dictionaries extend beyond the features of DMU dictionaries, DMU access routes cannot accurately translate these extensions into DMU format. Therefore, when you read CDO definitions through a DMU access route, some attributes of the definition are altered.

Table 2-2 shows the DMU interpretation of the CDO features that it cannot accurately translate. The table lists alternative features you can use in the CDO definition that can more readily be translated to DMU format.

Table 2–2: Interpretation of CDO Definitions Through a DMU Access Route

Feature of Definitions Stored in CDO Format	Interpretation Through DMU Access Route	Alternative
Column major arrays	Displayed as row major	None
Expression in OCCURS...DEPENDING	Becomes "OCCURS n"	Use a simple name
Expression in OCCURS...INDEX	Ignored	Use a simple name
Expression in MISSING_VALUE	Ignored	Use a simple value
Expression in INITIAL_VALUE	Ignored	Use a simple value
ALPHABETIC data type	Translated as TEXT	None
Invalid DTR expression in VALID_IF	Ignored	Use valid DTR expression
Invalid DTR expression in COMPUTED_BY	Ignored	Use valid DTR expression

Table 2–2: Interpretation of CDO Definitions Through a DMU Access Route (Cont.)

Feature of Definitions Stored in CDO Format	Interpretation Through DMU Access Route	Alternative
Complex overlay expression in VARIANTS clause	Ignored	Use valid DMU dictionary tag
Access rights	Mapped to: PASS_THRU SEE DTR EXTEND DTR READ HISTORY DTR WRITE DTR MODIFY	See Table 2-4

If you are a CDDL user, you should be aware that the CDO attribute BASED ON is not equivalent to the CDDL COPY clause; there is no CDO equivalent to the CDDL COPY clause.

In special circumstances, DMU users may receive error messages related to the presence of CDO format definitions.

If you specify a name referencing multiple subdirectories in your CREATE command, the error message does not indicate which piece of the path is the name duplicated by a CDO format definition. You need to use the CDO DIRECTORY command to locate the CDO dictionary definition with the same name. For example:

```
DMU> CREATE PERSONNEL.SALES.EMPLOYEES.CONTRACT.SALARY.FY87
%DMU-E-CDDERROR, CDD error at "PERSONNEL.SALES.EMPLOYEES.CONTRACT.SALARY.FY87;1"
-CDD-E-RENTOCDO, A CDO dictionary entity already exists with this name
DMU> EXIT
$ CDO
CDO> DIRECTORY PERSONNEL.SALES.EMPLOYEES.*

Directory PERSONNEL.SALES.EMPLOYEES

ADDRESS_RECORD;1                        RECORD
CITY;1                                  FIELD
CONTRACT;1                              RECORD
FULL_NAME;1                             RECORD
PERFORMANCE_REVIEW                      FIELD
STATE;1                                 FIELD
STREET;1                                FIELD
ZIP_CODE;1                              FIELD
```

Since CDD/Plus does not let you create any new entity in a DMU directory that has the same name as an entity already existing in a CDO dictionary, CDD/Plus changes DMU command output in certain situations:

- A DMU LIST command may find no entities with a certain name, yet when you try to create a DMU entity with that name, you receive an error message saying that the operation cannot be performed. If you receive such a message, simply use CDO to check your CDD/Plus dictionary for the entity in question.
- When you specify a target that is not a DMU format entity in a DMU RENAME command, you may see certain error messages, shown in Table 2-3. If you receive a message that a CDO entity already exists for a certain target name, make certain that the existing entity is not what you want to create, then choose a different name.

Such error messages are generated because CDO performs checks to maintain consistency within the DMU dictionary and the CDO compatibility dictionary.

Table 2–3: How CDD/Plus Affects DMU RENAME

RENAME input	Error Message	Explanation
Source is CDDL created and target name is unused	No error message	Entity is renamed successfully
Source is nonexistent entity	-CDD-E-NODNOTFND, directory or object not found	Entity is not found
Source is CDDL created and target is DMU known CDO field	-CDD-E-RENTOCDO, A CDO dictionary entity already exists with this name	You cannot give the same name to two entities
Source is CDDL created and target is DMU unknown CDO entity	-CDD-E-RENTOCDO, A CDO dictionary entity already exists with this name	You cannot give the same name to two entities
Source is CDO created	-CDD-E-CDDONLY, you cannot perform this call on a CDO dictionary node	You cannot rename an existing CDO entity

2.2.2 Interpreting DMU Dictionary Features in CDO Format

CDO interprets a DMU record definition that contains only one field as a field definition. CDO interprets a single DMU field defined within a structure as a field.

Many features of DMU definitions can be displayed accurately in CDO format.

For example, in CDO, you cannot define fields within record definitions. However, you can use the /FULL qualifier with the CDO SHOW RECORD command to display the attributes of included field names in a CDO record or a DMU record displayed in CDO format.

```
CDO> SHOW RECORD /FULL ADDRESS_RECORD
Definition of record ADDRESS_RECORD;1

|    Contains field   STREET
|    |  Datatype              text size is 30 characters
|    Contains field   CITY
|    |  Datatype              text size is 30 characters
|    Contains field   STATE
|    |  Datatype              text size is 2 characters
|    Contains record  ZIP_CODE
|    |   Contains field   NEW
|    |   |  Datatype              unsigned numeric digits 4
|    |   Contains field   OLD
|    |   |  Datatype              unsigned numeric digits 5
```

Such a display does not indicate that a DMU definition is actually converted to CDO format; the process of display is read-only. You can include DMU definitions in programs without converting them.

To convert definitions from DMU to CDO format, you must use the CONVERT command, as described in Section 2.4.

Note

If you request a CDO directory listing of DMU definitions, the listing omits definitions that are currently locked. A later directory listing might differ, because it could include definitions in directories that subsequently became unlocked. CDO does not have a locking mechanism, and has no means of reporting why a locked DMU definition is inaccessible.

There is no CDO equivalent for some of the DMU access rights, so CDD/Plus must interpret some rights in order to display them in CDO format. Table 2-4 shows how DMU protection provisions are interpreted to be the closest CDO protection provisions. DMU protection provisions that are not listed in the table are not translated.

Table 2–4: Translation of DMU Dictionary Protection Provisions to CDO Format

DMU Protection	Equivalent CDO Protection
CONTROL	CONTROL
DELETE Local or global	DELETE
MODIFY or DTR MODIFY	MODIFY (confirms that CHANGE access can be granted)
READ or DTR READ	READ (confirms that SHOW access can be granted)
WRITE or DTR WRITE	WRITE (confirms that DEFINE access can be granted)
SEE	SHOW
UPDATE	CHANGE + DEFINE

2.2.3 Translating COBOL Level 88 Conditions into DMU and CDO Formats

CDD/Plus translates COBOL level 88 conditions from DMU format to CDO format and vice versa. The following example shows COBOL syntax for a record containing level 88 definitions:

```
01  COB88.
    03  C               USAGE IS COMP PIC 9(4).
    88  C_ONE           VALUE 1.
    88  C_FIVE_TEN      VALUES ARE 5 THRU 10.
    88  C_OTHER         VALUES ARE 2 THRU 4
                                   11 THRU 20.
```

The equivalent CDDL syntax for this record definition is:

```
DEFINE RECORD COB88.
    COB88 STRUCTURE.
       C DATATYPE SIGNED WORD
           CONDITION FOR COBOL IS C_ONE VALUE IS 1
           CONDITION FOR COBOL IS C_FIVE_TEN
               COBOL NAME IS "C_5_10"
               VALUE IS 5 THRU 10
           CONDITION FOR COBOL IS C_OTHER
               VALUES ARE 2 THRU 4, 11 THRU 20.
    END.
END RECORD.
```

You can define the same record in CDO format using the DEFINE FIELD command and COMPUTED BY clause with a conditional value expression. The following CDO syntax definition is equivalent to the preceding CDDL definition:

```
DEFINE FIELD C
        DATATYPE SIGNED WORD.

DEFINE FIELD C_ONE
        COMPUTED BY IF C EQ 1 THEN 1 ELSE 0.

DEFINE FIELD C_FIVE_TEN
        NAME FOR COBOL IS C_5_10
        COMPUTED BY IF C GE 5 AND C LE 10 THEN 1 ELSE 0.

DEFINE FIELD C_OTHER
        COMPUTED BY
            IF (C GE 2 AND C LE 4) OR (C GE 11 AND C LE 20) THEN 1 ELSE 0.

DEFINE RECORD COB88.
    C.
    C_ONE.
    C_FIVE_TEN.
    C_OTHER.
END RECORD.
```

2.3 Protecting Dictionary Definitions in Compatibility Mode

When you define a directory, subdictionary, or data definition in a DMU dictionary, it automatically inherits the protection of its parent directory. When entities are defined in a compatibility dictionary, they acquire the CDO default protection provisions. It is possible that some protection inconsistencies exist unless you explicitly specify protection provisions for CDO definitions to be as near as possible to your DMU protection.

When a directory exists in both the DMU dictionary and the compatibility dictionary, only the DMU definitions in this directory inherit the directory's protection. When a dictionary item is created in CDO format with the same directory path as a definition in your DMU dictionary, these items do not necessarily have the same access control lists attached to them. By default, CDO grants the owner of a definition all access rights including CONTROL; all other users have only SHOW access. (For an explanation of the other default access rights listed in the example, see Chapter 5.)

Although the following record definitions share the same directory path, CDD$TOP.CORPORATE.PERSONNEL, they do not have the same protection provisions. The first entity, ADDRESS_REC, a DMU record definition, inherited protection from the parent directory. The second entity, FULL_NAME, a compatibility definition, is protected by CDO default provisions.

```
CDO> !
CDO> !Record in DMU dictionary with inherited protection
CDO> !
CDO> SHOW PROTECTION FOR RECORD CDD$TOP.CORPORATE.PERSONNEL.ADDRESS_REC

CDD$TOP.CORPORATE.PERSONNEL.ADDRESS_REC
Access control rights:
(IDENTIFIER=DBA,ACCESS=READ+WRITE+MODIFY+ERASE+
                 CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[12,5],ACCESS=READ+SHOW+CREATE+CHANGE)
(IDENTIFIER=[12,9],ACCESS=READ+SHOW)
CDO> !
CDO> !Record in compatibility dictionary with default protection
CDO> !
CDO> SHOW PROTECTION FOR RECORD CDD$TOP.CORPORATE.PERSONNEL.FULL_NAME

CDD$TOP.CORPORATE.PERSONNEL.FULL_NAME
Access control rights:
(IDENTIFIER=DBA,ACCESS=READ+WRITE+MODIFY+ERASE+CREATE+CHANGE
                     +DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=WORLD,ACCESS=READ+WRITE+MODIFY+ERASE+SHOW
                              +OPERATOR+ADMINISTRATOR)
CDO>
```

To avoid the problem of inconsistent protection schemes, you can explicitly define the protection scheme for the definition stored in your CDO dictionary to match as closely as possible the protection scheme for the DMU record definition. You can add entries and change the access control list for individual definitions in a compatibility dictionary with the DEFINE PROTECTION command (see Chapter 5).

2.4 Converting Record Definitions from DMU to CDO Format

DMU dictionary definitions are not automatically converted to CDO format when you install CDD/Plus; however, you can convert DMU record definitions to CDO format with the CONVERT command.

Software products with read-only support for CDO field and record definitions create definitions in DMU format. See Table 2-1.

Note

DIGITAL does not recommend that you convert your entire DMU dictionary to CDO definitions at one time. It is better to create new applications using definitions in CDO format and convert definitions in existing applications to CDO format in cases where a new application uses part of an existing one.

The CONVERT command copies a record definition from a DMU dictionary or from an Rdb/VMS database into CDO format. (You cannot currently use CONVERT to copy database definitions.) When the new definition is created in CDO format, the old one is *not* deleted from the DMU dictionary. Since definitions with the same name and the same path name cannot exist in both a DMU and a compatibility dictionary, you must do one of the following:

- Place the converted definitions into a different directory structure
- Rename the definitions you are converting

To be consistent, you should choose one of these two methods and always use it.

The following command copies the record definition EMP_REC from a DMU dictionary into a compatibility dictionary and renames it EMPLOYEE_RECORD. No path names are included; therefore, the default path name is used for both the DMU and the converted record definitions.

```
CDO> CONVERT EMP_REC EMPLOYEE_REC
```

To keep the same record definition name, you must specify a different directory path. For example, the following command copies a DMU record definition, ADDRESS_RECORD, in the default directory into a new compatibility directory, CONVERTED. A wildcard is used so that the definition in the directory CONVERTED is also named ADDRESS_RECORD.

```
CDO> CONVERT ADDRESS_RECORD CDD$TOP.CORPORATE.CONVERTED.*
```

If, in your destination path, you specify a directory that *does not* exist in your CDO dictionary but that *does* exist in a DMU dictionary, CDD/Plus creates it in the CDO dictionary for you.

When you do not specify a version number, the highest version of a record definition is converted. The converted definition is created as version 1 in the new directory.

The following three step process allows you to convert your definitions to CDO format and continue to use the same path and definition names:

1. Copy the DMU definition to a new DMU dictionary directory using DMU
2. Delete the original definition using DMU
3. Convert the copied definition to the original path name using CDO

After converting a record definition, you should confirm the new CDO record definition to ensure that it reflects the layout you intended.

The following example:

1. Displays an existing DMU record definition, CDD$TOP.CORPORATE.ADDRESS_ RECORD, using the DMU LIST/FULL command.
2. Exits DMU and enters CDO.
3. Converts the DMU ADDRESS_RECORD into a CDO ADDRESS_RECORD in a new compatibility dictionary, CONVERTED.

   CDD$TOP.CORPORATE.CONVERTED.ADDRESS_RECORD, the path name for the CDO record definition, is different from the path name for the DMU record definition, so the definition can retain the name ADDRESS_RECORD.
4. Displays the converted record definition in CDO using the /FULL qualifier with the SHOW RECORD command.

```
DMU> SET DEFAULT CDD$TOP.CORPORATE
DMU> LIST/FULL ADDRESS_RECORD
CDD$TOP.CORPORATE.ADDRESS_RECORD;1 <CDD$RECORD>
    Created by VAX CDD Data Definition Language Version V3.3-1
        on 12-MAY-1986 12:00:24.62 using protocol version 4.
    Source:
        DEFINE RECORD _CDD$TOP.CORPORATE.ADDRESS_RECORD
            DESCRIPTION IS
                /* This record contains the standard format
                for addresses.  It provides the source from which all
                address fields in other record descriptions are copied. */.
            ADDRESS STRUCTURE.
                STREET              DATATYPE IS TEXT
                                    SIZE IS 30 CHARACTERS.
                CITY                DATATYPE IS TEXT
                                    SIZE IS 30 CHARACTERS.
                STATE               DATATYPE IS TEXT
                                    SIZE IS 2 CHARACTERS.
                ZIP_CODE STRUCTURE.
                    NEW             DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 4 DIGITS
                                    BLANK WHEN ZERO.
                    OLD             DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 5 DIGITS.
                END ZIP_CODE STRUCTURE.
            END ADDRESS STRUCTURE.
        END ADDRESS_RECORD.
    Description:
         This record contains the standard format
                for addresses.  It provides the source from which all
                address fields in other record descriptions are copied.
DMU> EXIT
```

```
$ DICTIONARY
CDO> SET DEFAULT CDD$TOP.CORPORATE
CDO> CONVERT ADDRESS_RECORD CONVERTED.ADDRESS_RECORD
CDO> SHOW RECORD CONVERTED.ADDRESS_RECORD
Definition of record ADDRESS_RECORD;1
|   Contains field      STREET
|   Contains field      CITY
|   Contains field      STATE
|   Contains field      ZIP_CODE
CDO> DIRECTORY CONVERTED.*

Directory CDD$TOP.CORPORATE.CONVERTED
!
! Converted fields are not listed in the contents
!
ADDRESS_RECORD;1                       RECORD

CDO> SHOW RECORD /FULL CONVERTED.ADDRESS_RECORD
Definition of record SYS$COMMON:[CDDPLUS]CORPORATE.CONVERTED.ADDRESS_RECORD;1
|   Contains field          STREET
|   |  Datatype                 text size is 30 characters
|   Contains field          CITY
|   |  Datatype                 text size is 30 characters
|   Contains field          STATE
|   |  Datatype                 text size is 2 characters
|   Contains record         ZIP_CODE
|   |   Contains field          NEW
|   |   |  Datatype                 unsigned numeric digits 4
|   |   Contains field          OLD
|   |   |  Datatype                 unsigned numeric digits 5
```

After the conversion, notice that the DIRECTORY command does not list the included field definitions individually. Definitions of fields in DMU dictionaries are accessible only through their respective record definitions. However, a DMU record containing only one field is converted to a CDO field.

2.4.1 Sharing Fields After Record Conversion

After being converted, a record definition includes the same fields with their respective attributes as it did before. However, a *field* definition that was specified as *part of a DMU record* cannot be shared by other applications; only the record is shareable.

When you create a definition in CDO, you specify the exact dictionary directory and the name for the definition; this is known as the *directory name*. The directory name acts as a pointer to the storage location of the actual entity definition. The name of the entity itself is known as the **processing name**. For different applications to share definitions, the definitions must be named and placed in a CDO dictionary directory. Applications access an entity definition by specifying the full dictionary path name and the definition's directory name—for example, CDD$TOP.PERSONNEL.EMPLOYEE_ID.

Fields internal to converted records cannot be referred to by name and, as a result, cannot be listed individually by the DIRECTORY command. After conversion, such an internal field continues to have only a processing name, not a directory name.

If you plan to create definitions through the call interface, you need to be aware of the differences between the two names.

- The **processing name** is the name that all layered products use when processing with the entity. All dictionary entities have a processing name: it is part of the definition.
- The **directory name** is the given name that identifies the entity within the dictionary structure. It enables layered products to find the definition.

If you need to know the processing name of a definition, use the SHOW FIELD or SHOW RECORD command; all other CDO commands display the directory name, if one exists.

Not all dictionary entities have a directory name. For example, when you store fields in the dictionary using RDO (the Rdb/VMS Relational Database Operator), the fields do not have directory names. You must locate such fields relative to the database structure, as in the SHOW FIELD FROM DATABASE command described in Section 7.2.2.

A definition without a directory name cannot be directly named and, therefore, is difficult to share among several applications. If a definition will be shared among applications, use one of the methods described in Section 2.4.2 to give the definition a directory name.

2.4.2 Creating Directory Names for Definitions

If you want the internal field definitions from a converted record to be shareable, you must

- Create a directory name for each existing field with the ENTER ... FROM RECORD command
- Create a new dictionary definition for each field with the DEFINE command

For example, you can explicitly define the field STREET after converting ADDRESS_RECORD:

```
CDO> DEFINE FIELD STREET
cont> DATATYPE IS TEXT
cont> SIZE IS 30
```

A STREET field created with the DEFINE command is a different data definition from the STREET field that previously existed in ADDRESS_RECORD.

Alternatively, you can create a directory name for existing definitions with the ENTER command:

```
CDO> ENTER FIELD STREET FROM RECORD ADDRESS_RECORD
```

After you create a directory name for each converted field definition with the ENTER command, you can list the individual field definitions with the other contents of a dictionary directory:

```
CDO> SET DEFAULT CDD$TOP.CORPORATE.CONVERTED
CDO> CONVERT CDD$TOP.CORPORATE.ADDRESS_RECORD ADDRESS_RECORD
CDO> ENTER FIELD STREET FROM RECORD ADDRESS_RECORD
CDO> ENTER FIELD STATE FROM RECORD ADDRESS_RECORD
CDO> ENTER FIELD CITY FROM RECORD ADDRESS_RECORD
CDO> DIRECTORY CONVERTED.*

Directory CDD$TOP.CORPORATE.CONVERTED
!
! Fields are now listed in the directory
!
ADDRESS_RECORD;1                        RECORD
CITY;1                                  FIELD
STATE;1                                 FIELD
STREET;1                                FIELD
```

You cannot use wildcard characters with the ENTER command, so you must supply the entity's processing name in the command line.

Note

If you use a logical name that is defined for your system or your process as the directory name for an entity, the name will be translated before use.

You can use the ENTER command to create directory names for entities that are directly related to the named record. Consider a record—OUTSIDE—that includes another record—INSIDE, which includes fields A, B, and C. Fields A, B, and C can be accessed only after you have created a directory name for the record definition INSIDE.

First, create a directory name for INSIDE with the statement ENTER RECORD INSIDE FROM RECORD OUTSIDE. Once the record INSIDE has a directory name, you can enter the command ENTER FIELD A FROM RECORD INSIDE.

To create a directory name for a field within a STRUCTURE clause of a record, you must first create a directory name for the structure, then you can address the fields within the structure. The ENTER command cannot create directory names for fields in a VARIANTS clause. For more information about the ENTER command, see Chapter 7.

The DMU history list is converted to a CDO definition and can be viewed with the /AUDIT qualifier to the SHOW RECORD command.

An unstructured DMU record that contains a single field is converted to a CDO field. The following DMU definition of the record SINGLE contains only one field. The DIRECTORY command lists SINGLE as a field before conversion. After conversion, SINGLE is a field definition.

```
CDO> CONVERT SINGLE CONVERTED.SINGLE
CDO> SHOW FIELD CONVERTED.SINGLE
Definition of field SYS$COMMON:[CDDPLUS]CORPORATE.CONVERTED.SINGLE;1
|   Datatype     text size is 12 characters
CDO>
```

In CDO definition format, DESCRIPTION attributes are attached to definitions, not directory list entries. Therefore, when a DMU definition is converted to CDO format, any descriptions *attached to the directory access* are not converted. Descriptions that are *attached to named definitions* are included in the DESCRIPTION attribute for the converted definition.

When a DMU record definition is converted to CDO format, CDD/Plus matches the protection provisions according to the scheme shown in Table 2-4. To fine tune the protection provisions for converted DMU record definitions, use the CHANGE PROTECTION and DEFINE PROTECTION commands as described in Chapter 5.

The *VAX CDD/Plus Common Dictionary Operator Reference Manual* provides the syntax diagram and rules for the CONVERT command.

Chapter 3 describes how to use logical names and search lists, and how to set your default directory. These actions allow you to use abbreviated path names and to easily search through your dictionaries from the CDO utility.

Using the CDO Utility 3

The Common Dictionary Operator (CDO) utility is the user interface to CDD/Plus. This chapter helps you to get started using CDO and shows you how to create common dictionary definitions with the CDO editor.

3.1 CDO Utility Features

The CDO utility supports many familiar VMS features. In CDO you can:

- Access any CDO or DMU dictionary that you have privileges to
- Create field and record definitions easily with the CDO editor
- Execute all CDO commands
- Learn CDO commands using the online help system
- Execute frequently used series of commands in CDO command procedures
- Manipulate several definitions by using wildcard characters (*, %, and ...) in command lines
- Search through any number of physical dictionaries with search lists
- Enter frequently used commands quickly by defining key sequences
- Work in a subprocess without disturbing your dictionary session by using the CDO commands SPAWN and ATTACH
- Recall and alter commands with VMS line editing features:
  - Erase the current command with CTRL/U

- Cancel the current command with CTRL/C
- Edit lines with the DELETE key and the terminal left (←) and right (→) arrow keys
- Recall previous commands with CTRL/B and the terminal up (↑) and down (↓) arrow keys

3.2 Logical Dictionary Structure

The physical location of definitions in CDO dictionaries can be distributed on different devices on a single node, on different nodes in a VAXcluster, and on local or wide area networks. The CDO utility allows you to access all of these dictionaries, as well as your system DMU dictionary, as one logical dictionary.

Your logical dictionary can include the physical dictionaries created during installation plus those created by users.

A logical dictionary has no implicit structure: it is a set of disjoint hierarchies, like the logical structure of files and directories on a disk. You impose your own hierarchical structure when you create dictionary directories and group entity definitions within these directories. You can use both explicit naming and search lists, which are described in Section 3.6.3 to create the logical dictionary structure.

You can create a logical dictionary that consists of:

- One physical dictionary
- More than one physical dictionary
- One or more subsets of one physical dictionary

To define a physical dictionary, use the DEFINE DICTIONARY command, described in Section 3.3.3. When you define a physical dictionary, you specify an anchor that describes that dictionary's origin. (For details about naming conventions, see Section 1.3.) You can define CDO dictionary directories and subdirectories by using the DEFINE DIRECTORY command, described in Section 3.3.4.

Before creating your dictionary directory structure, consider what type of structure might lend itself to the anticipated dictionary usage.

There are a number of ways to organize your dictionary:

- By application

In an organization where the departments must share most of the data, you can structure your dictionary according to application areas. You can set aside one dictionary directory for storing those definitions that are shared throughout the organization. You can then create application-specific directories for definitions that are less widely shared.

- By organizational entity

  Organizing your dictionary by organizational entity is most useful in situations where differences in data and security needs are sharply defined between different divisions of the organization. For example, a company with two separate departments might choose to create two separate physical dictionaries. Selected employees might have access to the definitions in both dictionaries. A similar effect can be achieved by setting up two dictionary subdirectories in the dictionary hierarchy and protecting the definitions in each subdirectory from unqualified users.

- By individual user

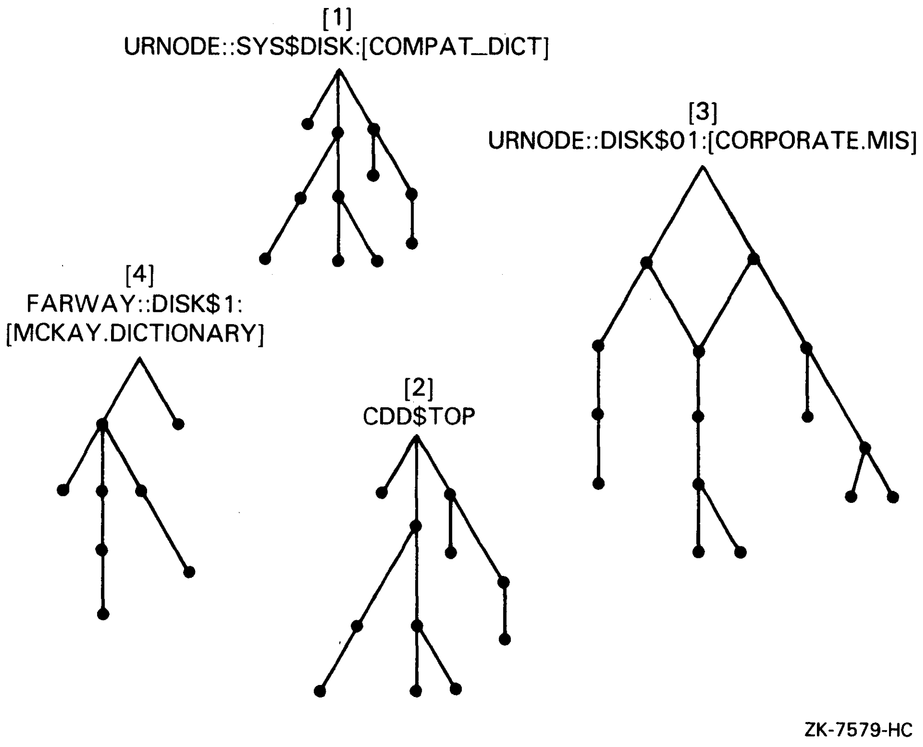
  When individuals work separately on independent projects, they may need a personal dictionary, or a personal directory structure within a large dictionary. You can reflect this structure in the dictionary hierarchy by assigning directories to individuals for their own use, or allowing individuals to create their own dictionaries on the system.

In many cases, the needs of an organization are served best by a combination of these criteria. If your logical dictionary includes personal, department, and system-wide dictionaries, you can store definitions that change frequently in a personal directory and definitions that are permanent in a widely-used system location. You may have special needs in your organization, such as security or maintenance requirements, that indicate a need for multiple dictionaries. Good planning prior to setting up your dictionary can result in a more secure and consistent dictionary.

Within the CDO interface, you can switch from one physical dictionary to another and gain access to definitions in any of the dictionaries to which you have access. Figure 3-1 shows the structure of one logical dictionary consisting of four physical dictionaries. Information about each dictionary is provided following the figure.

CDD/Plus naming conventions are explained in Chapter 1. To recap, an anchor is a VMS directory specification where a CDO dictionary is located. CDO allows you to use CDD$TOP as the dictionary origin of your DMU dictionary *and* as an alternative to specifying the anchor for definitions in the compatibility dictionary. You must specify only the dictionary anchor to refer to definitions in a user-created CDO dictionary.

Figure 3–1: Sample Logical Dictionary Structure



1. The compatibility dictionary—created in the installation procedure. (See Section 2.1). The anchor for this special CDO dictionary is the VMS directory selected during the installation period—URNODE::SYS$DISK:[COMPAT_DICT], in this case. This anchor can be interchanged with CDD$TOP.
2. The DMU dictionary—created when CDD/Plus is installed, unless one already exists on the system. This dictionary directory structure is mapped to the directory structure in the compatibility dictionary. The origin for the DMU dictionary is CDD$TOP. CDD$TOP can be interchanged with the anchor for the compatibility dictionary, URNODE::SYS$DISK:[COMPAT_DICT], in this case.
3. A new CDO dictionary—created by a user in the CDO environment. This dictionary is the example dictionary that is created in this chapter. URNODE::DISK$01:[CORPORATE.MIS] is the anchor for this dictionary.
4. A CDO dictionary created on a remote node. The anchor for this dictionary is FARWAY::DISK$1:[MCKAY.DICTIONARY].

3.3 Getting Started with CDO

On installation of CDD/Plus, two physical dictionaries are created that you can access as one logical dictionary from CDO: the compatibility dictionary (a CDO dictionary) and a DMU dictionary. You can also create and access other CDO dictionaries through the CDO interface, all as members of one logical dictionary.

This section shows you how to:

1. Invoke CDO.
2. Create a new dictionary in a dedicated VMS directory. (You can eliminate this step if you plan to work only in the compatibility dictionary.)
3. Create dictionary directories—either in the compatibility dictionary or another CDO dictionary—where your definitions can reside.

3.3.1 Invoking CDO

To enter the CDO environment, type the DICTIONARY OPERATOR command at the DCL prompt. The system responds with a brief introductory message and the CDO> prompt as shown:

```
$ DICTIONARY OPERATOR
Welcome to CDO V1.0
The CDD/Plus V4.0 User Interface
Type HELP for help
CDO>
```

You can abbreviate the DICTIONARY command to its first four characters, like other DCL commands. You can optionally omit the OPERATOR parameter because the current default invokes CDO.

The DICTIONARY command accepts foreign command lines, so after you enter the DICTIONARY command, you can specify any CDO command. After the CDO command executes, you are returned to the DCL prompt.

To connect a CDO command that continues on more than one line, you can use the hyphen (-). The DCL prompt for a continued line is preceded by an underscore.

In the following example, a user defines the field CITY using DICTIONARY OPERATOR and a foreign command.

```
$ DICTIONARY OPERATOR DEFINE FIELD CITY -
_$ DATATYPE IS TEXT SIZE IS 20.
$
```

To end your dictionary session, type EXIT or CTRL/Z at the CDO> prompt:

```
CDO> EXIT
$
```

3.3.2 Running DCL Subprocesses

You may wish to issue DCL commands without exiting from CDO. If you are in the CDO environment, you can use the SPAWN command to create a DCL subprocess. You can use a subprocess to work at DCL level without interrupting your dictionary session. Section 3.3.3 shows how you might use a subprocess when creating a new dictionary.

When you finish working in your subprocess, you can enter the DCL command ATTACH, specify the name of your original process, and return to your dictionary session. Unless you have specifically renamed the process that is running CDO, the process name is your system user name, as in the following example:

```
CDO> SPAWN
$
$ SHOW DEFAULT
DISK$01:[SMITH]
    .
    .
    .
$ ATTACH SMITH
CDO>
```

If you spawn more than two subprocesses from one process, you may exceed your default quotas for certain process resources, generating an error message. If you do not have system privileges, ask your system manager to increase your process resource limits. Suggested values for the limits affected appear in Table 3-1. If you use remote access to the dictionary, the DECnet account or the proxy account on your remote node should also have these quotas. (The default DECnet account has a default FILLM quota of less than 50.)

Table 3–1: Suggested Process Resource Limits

Limit	Description	Value
BYTLM	Buffered I/O count quota	20000
ENQLM	Enqueue quota	600
FILLM	Open file quota	60
PRCLM	Subprocess quota	5

3.3.3 Creating a New Dictionary

If you use only DMU or products that have read-only access to CDO format field and record definitions, you will only need to use the compatibility dictionary, created for you during the installation of CDD/Plus. (To determine the CDO support currently provided by a VAX product, consult the documentation for that product.) If you plan to work only in the compatibility dictionary, skip this section and go to Section 3.3.4.

Before creating a CDO dictionary, you must create a VMS directory where the dictionary can reside. You should create only one dictionary for each VMS directory. This VMS directory should remain dedicated to your CDO dictionary; do not store other VMS files in this directory. If you delete the dictionary later, all files stored in this directory will be deleted. You can set your process default to any other VMS directory, invoking CDO and referring to your new dictionary from there.

Caution

Do not store any other files in a dictionary anchor directory. If your dictionary is to be used by the public, be sure to set the VMS protection on your CDD/Plus anchor directory so that all users of the system have read, write, and execute privileges (W:RWE). (Processes that write to DMU-based dictionaries must also be able to write to a run-unit journal file created in the anchor directory.) For details about setting VMS directory protection, see Chapter 5.

To create a new VMS directory, you can spawn to a subprocess and use the DCL command CREATE. After creating the directory, you can use the DCL command ATTACH to return to your CDO session.

The following example starts a subprocess to work at the DCL level, then creates a new VMS directory, MIS, under the main account CORPORATE, then resumes the CDO session:

```
CDO> SPAWN
$ SHOW DEFAULT
DISK$01:[CORPORATE]
$ CREATE/DIRECTORY [.MIS]
$ ATTACH CORPORATE
CDO>
```

Use the DEFINE DICTIONARY command to create a new dictionary in the VMS directory. The following example creates a dictionary in the VMS directory [CORPORATE.MIS]. Before you begin, confirm that you have enough free disk space to create a new dictionary (the amount required is listed in the *VAX CDD/Plus Installation Guide*). This operation may take a few minutes depending on your system resources. The operation returns the CDO prompt when it has completed successfully. To see proof that your dictionary was created, you can use the CDO DIRECTORY command, as shown below.

```
CDO> DEFINE DICTIONARY [CORPORATE.MIS].
CDO> DIRECTORY [CORPORATE.MIS]*

Directory DISK$01:[CORPORATE.MIS]

CDD$PROTOCOLS                                           DIRECTORY
CDO>
```

The directory CDD$PROTOCOLS appears in any directory where you create a dictionary.

3.3.4 Creating Dictionary Directories

Dictionary **directories** are named sections of a dictionary that you use to organize field and record definitions, and other directories. Before defining your dictionary entities, you should create dictionary directories to group related definitions, keeping your planned structure in mind.

After you invoke CDO, you should set a default dictionary directory. The **default dictionary directory** (CDD$DEFAULT) is the one in which you plan to work during the current session. All the definitions that you create will be stored in the default directory that you specify, until you specify a different default directory. You can designate a default directory with the SET DEFAULT command, as in the following example:

```
CDO> SET DEFAULT [CORPORATE.MIS]PERSONNEL
CDO>
```

For more information about setting default dictionary directories, see Section 3.6.2.

A dictionary directory is similar to a VMS directory in terms of hierarchical purpose. Although a CDO dictionary itself has no implicit hierarchical structure, dictionary directories allow you to group related entity definitions and to use the resulting hierarchy in definition names. They act as markers or pointers to the definitions you store in them; they are not dictionary definitions themselves.

From the CDO environment, you can create dictionary directories in the compatibility dictionary or in a user-created CDO dictionary. You create dictionary directories with the DEFINE DIRECTORY command. In the following example, two dictionary directories are created: BUDGET and PERSONNEL. Note that the names of these dictionary directories are appended to the dictionary anchor, which appears in square brackets. You can supply an asterisk (*) with the DIRECTORY command to display a list of the contents of the dictionary so far:

```
CDO> DEFINE DIRECTORY [CORPORATE.MIS]BUDGET.
CDO> DEFINE DIRECTORY [CORPORATE.MIS]PERSONNEL.
CDO> DIRECTORY [CORPORATE.MIS]*

Directory DISK$01:[CORPORATE.MIS]

CDD$PROTOCOLS                                          DIRECTORY
PERSONNEL                                              DIRECTORY
BUDGET                                                 DIRECTORY
CDO>
```

In the previous example, the dictionary contains only directories. If your dictionary contains other data elements, such as fields, records, or databases, the DIRECTORY command will display these also.

CDD/Plus creates the directory CDD$PROTOCOLS automatically when you create a new CDO dictionary. It contains definitions that describe the types of entities and attributes that you use in your data descriptions. These definitions are essential to the functioning of your dictionary and should not be changed or deleted.

The following example sets the default directory to be the PERSONNEL directory in the compatibility dictionary and creates two subdirectories. The subsequent DIRECTORY command confirms that the subdirectories were created.

```
CDO> SET DEFAULT CDD$TOP.PERSONNEL
CDO> DEFINE DIRECTORY SALARIED.
CDO> DEFINE DIRECTORY CONTRACT.
CDO> DIRECTORY *

Directory DISK$01:[CORPORATE.MIS]PERSONNEL

CONTRACT                                               DIRECTORY
SALARIED                                               DIRECTORY
CDO>
```

Within these dictionary directories, you create the field and record definitions you need.

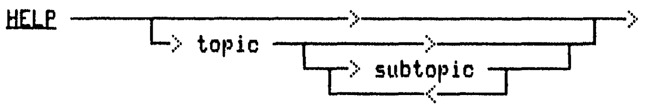
Note

Before creating field and record definitions, you must make sure that you have set the default dictionary directory to the location where you want them created, or you must specify the full path. For more information about setting the default dictionary directory, see Section 3.6.2.

3.3.5 Accessing Help in the CDO Environment

To access online help for CDO commands, type HELP at the CDO> prompt, as shown in the following example. The HELP utility displays a list of the available help topics.

```
CDO> HELP
HELP
Provides help on CDO commands and CDD/Plus concepts.
```



```
Additional information available:

@(At_Sign) ATTACH     CHANGE     CLEAR      CONVERT    COPY       DEFINE
DELETE     DIRECTORY  EDIT       Edit_str   ENTER      EXIT
Expressions           EXTRACT    Field_attr HELP       MOVE       ON
PURGE      Record_attr           SET        SHOW       SPAWN      VERIFY

Topic?
```

To save keystrokes, you can type the exact topic you want help on at the CDO> prompt. For example, the following command displays help text on the /AUDIT qualifier to the SHOW FIELD command:

```
CDO> HELP SHOW FIELD/AUDIT
```

3.4 Editing Definitions in the CDO Environment

The CDO editor is a flexible, menu-driven tool that is available on DIGITAL video terminals. The editor is useful for entering common field and record definitions. (Chapter 4 describes how to use the DEFINE command to create field and record definitions.) In the CDO editor you can:

- Create new field and record definitions
- Create new versions of previous definitions
- Browse through your current definitions

You can manipulate definitions by selecting items from menus and entering values from the keyboard. Among the features that make defining fields and records within the editor easy are:

- Convenient menu displays of possible attributes, relationships, and allowed values
- Useful browsing capability
- Dynamic data type validation
- Easy cursor movement between field attributes
- Online help on keypad keys and field attributes
- Easy keypad access to a text editor when necessary

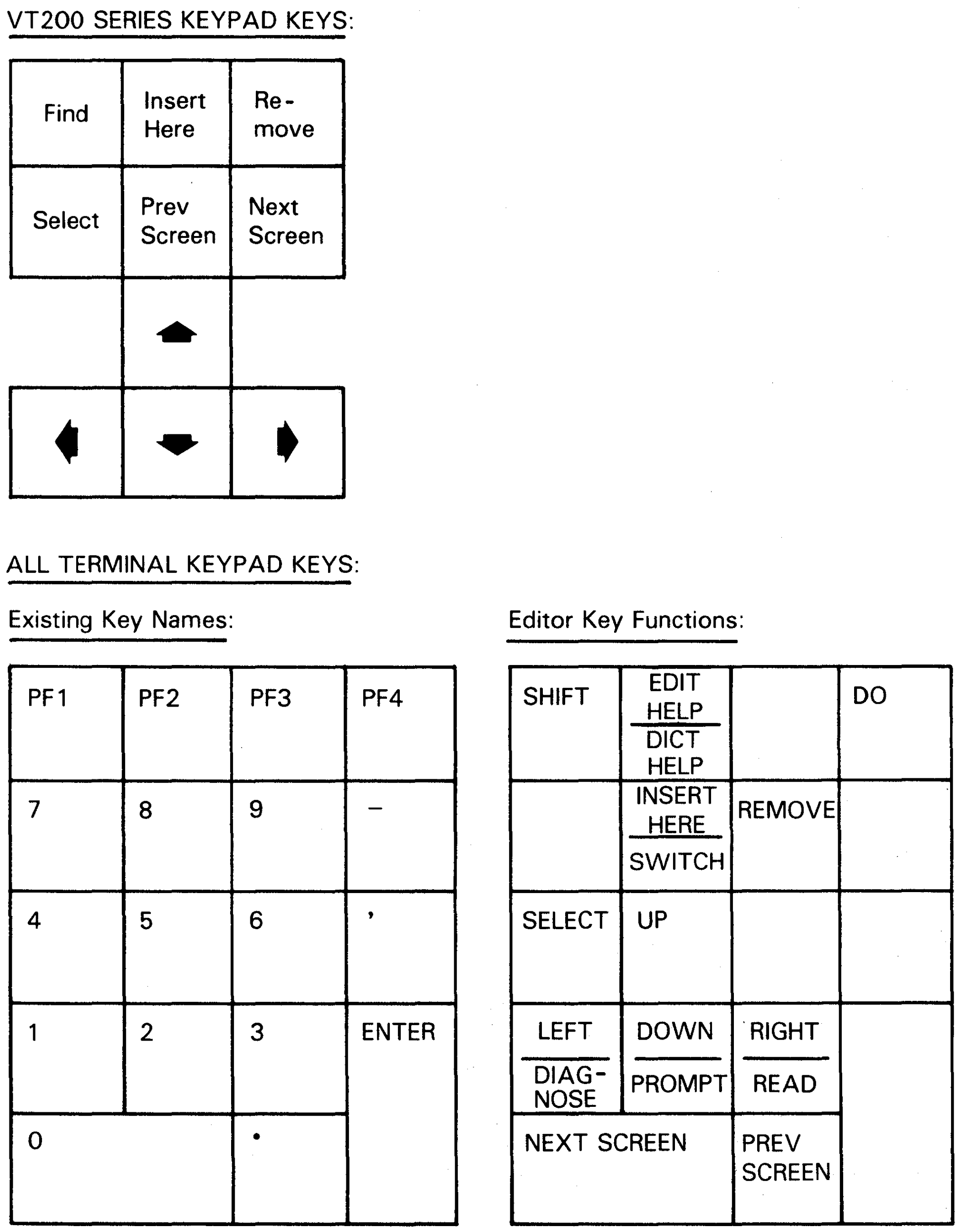
Section 3.4.3 provides a sample CDO editing session that you can follow step-by-step.

3.4.1 CDO Editor Key Definitions

During an editing session, the keypad keys are defined to aid faster editing. Figure 3-2 illustrates the key functions on all keyboard keypads as well as the special keypad on an LK201 keyboard. The shaded key functions are available when you press the shaded key after pressing PF1 (the editor's SHIFT key).

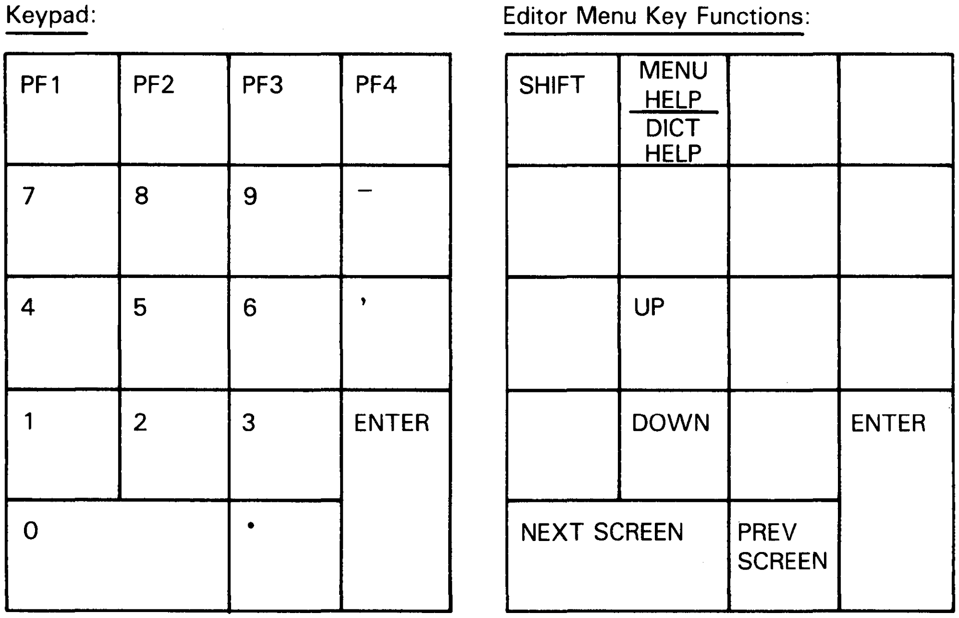
When the editor displays a menu, the keypad keys assume slightly different functions to help you select menu items. Figure 3-3 illustrates the set of key functions available when an editor menu is in effect.

Figure 3–2: CDO Editor Keypad Key Definitions



ZK-7580-HC

Figure 3–3: CDO Editor Menu Keypad Key Definitions



ZK-7581-HC

In addition to the keypad keys displayed, users with LK201 keyboards can use the DCL supported line-editing keys, such as the HELP and DO keyboard keys. The F10 key allows you to exit from the CDO editor. Table 3-2 summarizes the most frequently used key functions for the CDO editor. For a full list of the DCL line-editing keys, see the VMS documentation set.

If you make a mistake entering text, you can erase incorrect characters with the DELETE key. To move the cursor from one attribute to another, use the arrow keys and enter the values as you choose. Enter CTRL/Z to cancel a menu display. To remove an attribute that you inserted by mistake, press the REMOVE key or KP9. When you press RETURN, the next appropriate menu is displayed.

If you decide to change from editing a field definition to a record definition, or vice versa, you can make this switch by pressing the PF1 key followed by 8 on the numeric keypad (the PF1-KP8 combination). The editor responds by making the appropriate changes in the display and in the menu items.

To access help on the editor key functions, press the PF2 key. If you want help on a particular attribute or menu item, press PF1-PF2 while the cursor is positioned on the item that you need help on. Section 3.4.3 provides a sample CDO editing session that can help you become familiar with the key functions.

Table 3–2: Frequently Used CDO Editor Key Functions

LK201 Keyboard Keypad Key	Any DIGITAL Keyboard Keypad Key	Function
INSERT	KP8[1]	Displays menu of attributes
SELECT	KP4[1]	Displays menu of current definitions
REMOVE	KP9[1]	Deletes a selected menu item
DO	PF4	Enters a definition in the dictionary
HELP	PF2	Displays the keypad diagram
PF1-PF2	PF1-PF2	Displays help for menu items while the cursor is positioned on the item
PF1-KP8[1]	PF1-KP8[1]	Switches from a record definition to a field definition, or vice versa
	Left arrow or KP1[1]	Moves the cursor to the left
	Right arrow or KP3[1]	Moves the cursor to the right
	Up arrow or KP5[1]	Moves the cursor up
	Down arrow or KP2[1]	Moves the cursor down
CTRL/Z	CTRL/Z	Exit from an editor menu
F10	CTRL/Z	Exit from the editor

[1]Keys that begin with KP indicate keys on the numeric keypad on the right side of the terminal keyboard.

3.4.2 Accessing Help in the CDO Editor

Two kinds of help are available during an editing session:

- Help on the editing key functions
- Help on the attributes and menu items displayed

To access online help on any of the key functions from within the editor, press the PF2 or HELP key. This displays a diagram of the keypad keys that are available at the time you request help. You can then request help on a particular key function by pressing that key. Figure 3-2 and Figure 3-3 provide diagrams of the two sets of key functions available.

You can access help on attributes and items in a menu by pressing PF1-PF2 while the editor cursor is positioned on the item you want help on.

3.4.3 A Sample CDO Editing Session

This section provides a sample CDO editing session on a VT200-series terminal with an LK201 keyboard. You may want to refer to Figure 3-2 and Figure 3-3 while following the steps outlined. (If you have a VT100-series terminal, use Table 3-2 to locate the equivalent keys on your keyboard.) The purpose of this exercise is to help you become familiar with the CDO editor. If you follow the steps outlined here, you will create three new field definitions and one record definition that uses the fields. The editor screen is updated with each step in the procedure. If you already feel comfortable with the editor, you can skip this section.

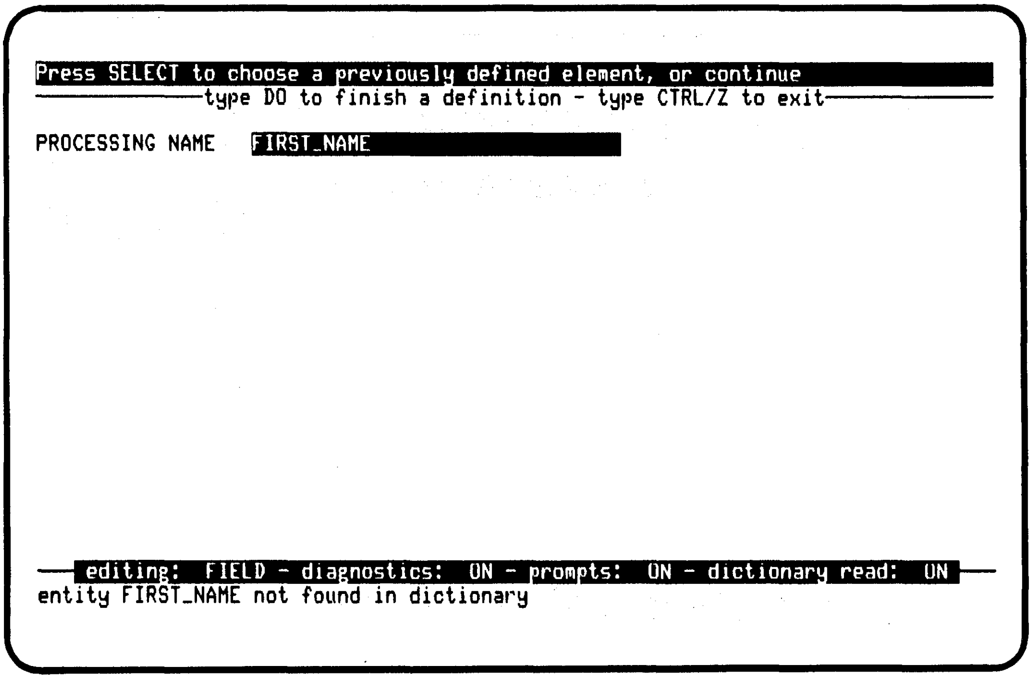
1. Create a dictionary directory to group the example definitions together and set your default work area to the new directory. Enter the following commands at the CDO> prompt:

   ```
   CDO> DEFINE DIRECTORY EDITOR_EXAMPLES.
   CDO> SET DEFAULT YOUR$DISK:[YOUR_DICTIONARY]EDITOR_EXAMPLES
   ```

2. You can invoke the editor at the CDO> prompt with the EDIT FIELD or EDIT RECORD command. If you do not include the definition name on the command line, you can enter the name once you are within the editor. Be aware that you cannot name a definition with a logical name that is defined for your process or your system. Invoke the editor to begin creating a new field definition FIRST_NAME. Enter the following command at the CDO> prompt:

   ```
   CDO> EDIT FIELD FIRST_NAME
   ```

The editor responds with the following display:

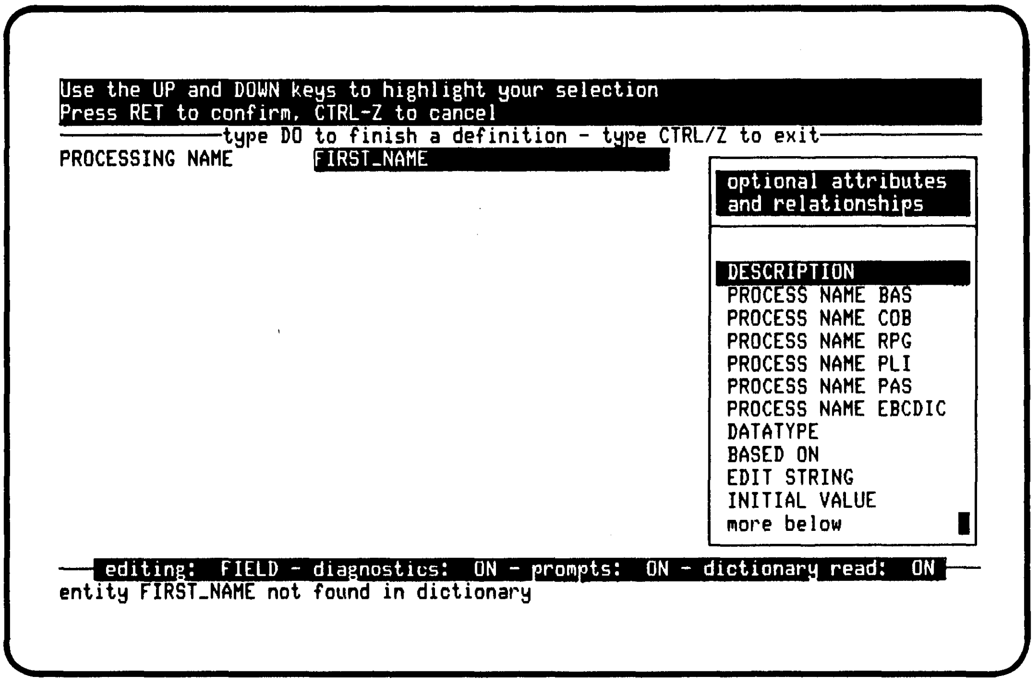


```
Press SELECT to choose a previously defined element, or continue
─────────────type DO to finish a definition - type CTRL/Z to exit─────────────

PROCESSING NAME   FIRST_NAME















─── editing:  FIELD - diagnostics:  ON - prompts:  ON - dictionary read:  ON ───
entity FIRST_NAME not found in dictionary
```

ZK-7466-HC

The processing name of the definition you are creating is displayed and the cursor is positioned after the name. (For information about processing names, see Chapter 1.) The message at the bottom of the screen indicates that the editor cannot find FIRST_NAME in the dictionary; you are creating a new field definition. If another user on your system has already created this field definition, you will see a different message.

3. Press the INSERT key to view possible field attributes. The editor responds with the following display:

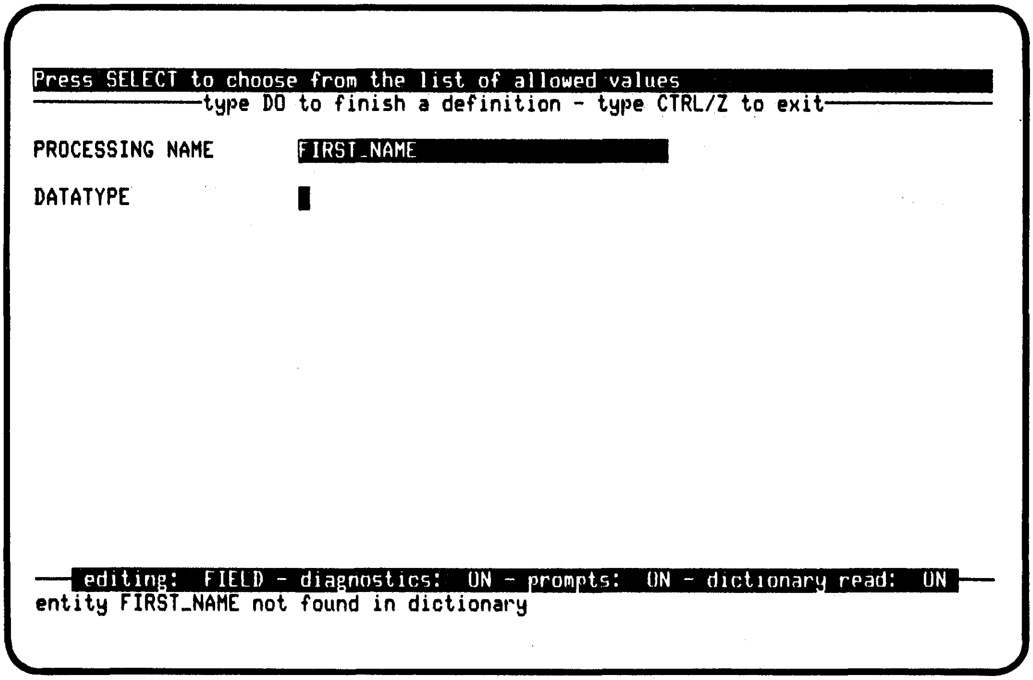


```
Use the UP and DOWN keys to highlight your selection
Press RET to confirm, CTRL-Z to cancel
─────────────type DO to finish a definition - type CTRL/Z to exit──────────
PROCESSING NAME        FIRST_NAME
                                                    optional attributes
                                                    and relationships

                                                    DESCRIPTION
                                                    PROCESS NAME BAS
                                                    PROCESS NAME COB
                                                    PROCESS NAME RPG
                                                    PROCESS NAME PLI
                                                    PROCESS NAME PAS
                                                    PROCESS NAME EBCDIC
                                                    DATATYPE
                                                    BASED ON
                                                    EDIT STRING
                                                    INITIAL VALUE
                                                    more below
── editing: FIELD - diagnostics: ON - prompts: ON - dictionary read: ON ──
entity FIRST_NAME not found in dictionary
```

ZK-7467-HC

You must select at least one attribute. Press the down arrow key (↓) until the DATATYPE attribute is highlighted.

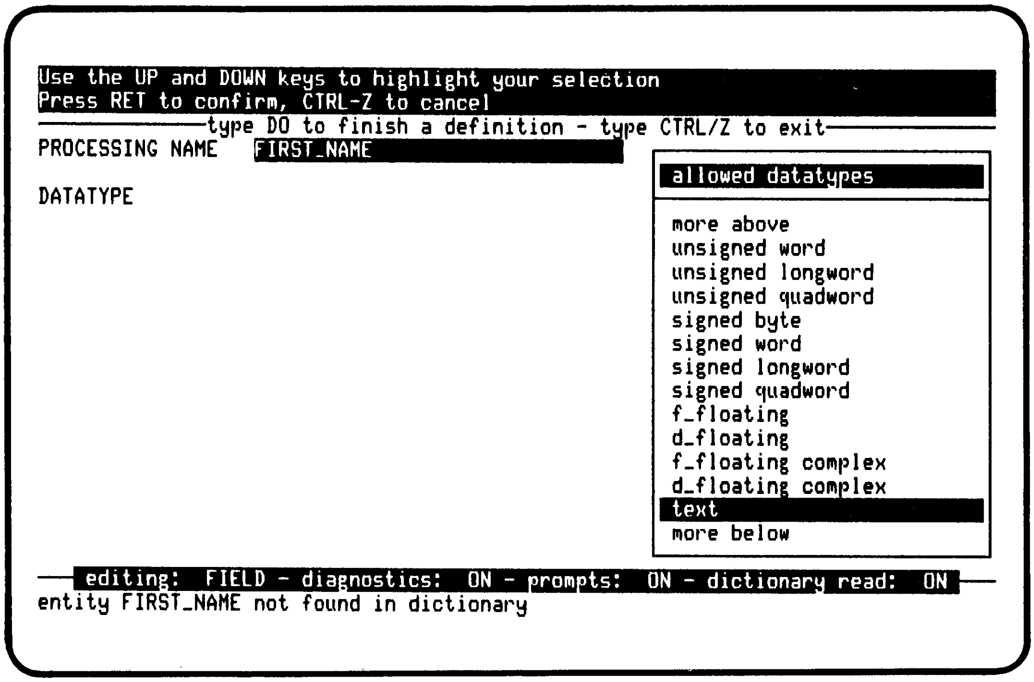
4. Press RETURN to select DATATYPE. DATATYPE is added to the screen display, as shown:



```
Press SELECT to choose from the list of allowed values
--------------type DO to finish a definition - type CTRL/Z to exit---------------

PROCESSING NAME        FIRST_NAME

DATATYPE               █










--- editing:  FIELD - diagnostics:  ON - prompts:  ON - dictionary read:  ON ---
entity FIRST_NAME not found in dictionary
```

ZK-7468-HC

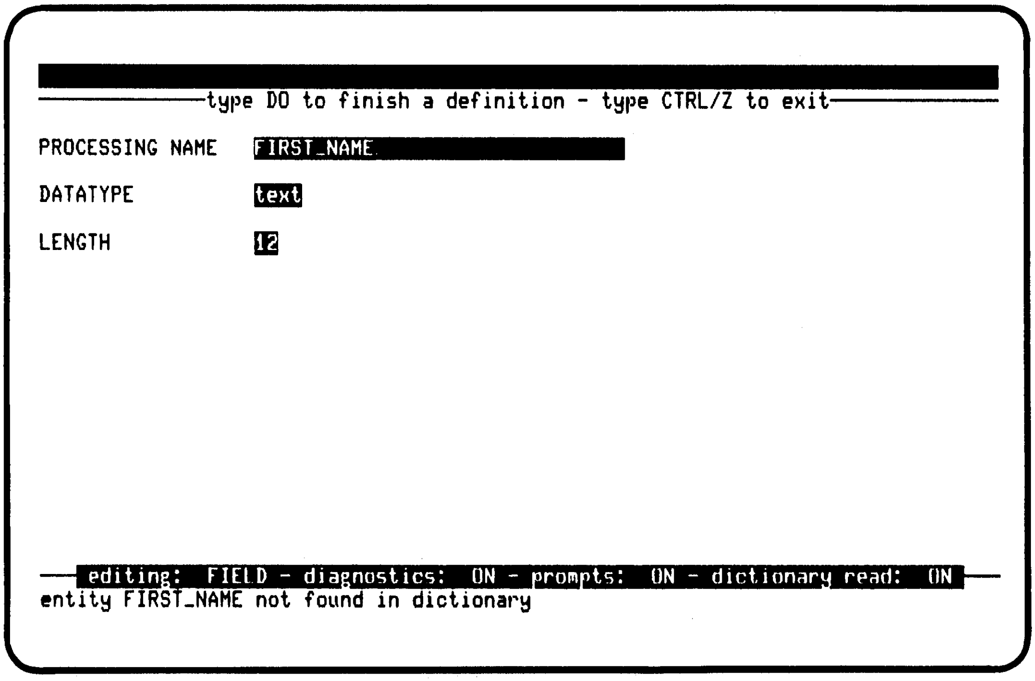
5. Press the SELECT key to view a menu of possible data types. Press the down arrow key until the data type *text* is highlighted, as shown:



```
Use the UP and DOWN keys to highlight your selection
Press RET to confirm, CTRL-Z to cancel
─────────────type DO to finish a definition - type CTRL/Z to exit────────────
PROCESSING NAME   FIRST_NAME
                                                   ┌──────────────────────────┐
                                                   │ allowed datatypes        │
DATATYPE                                           ├──────────────────────────┤
                                                   │ more above               │
                                                   │ unsigned word            │
                                                   │ unsigned longword        │
                                                   │ unsigned quadword        │
                                                   │ signed byte              │
                                                   │ signed word              │
                                                   │ signed longword          │
                                                   │ signed quadword          │
                                                   │ f_floating               │
                                                   │ d_floating               │
                                                   │ f_floating complex       │
                                                   │ d_floating complex       │
                                                   │ text                     │
                                                   │ more below               │
                                                   └──────────────────────────┘
── editing:  FIELD - diagnostics:  ON - prompts:  ON - dictionary read:  ON ──
entity FIRST_NAME not found in dictionary
```

ZK-7469-HC

6. Press RETURN to select text; text is inserted as the data type in the screen display. You are required to enter the length of the text for the field; therefore, the editor displays LENGTH on your screen. Use the down arrow key to move the cursor to the length clause. Type a value, such as 12. Your screen should look like this:

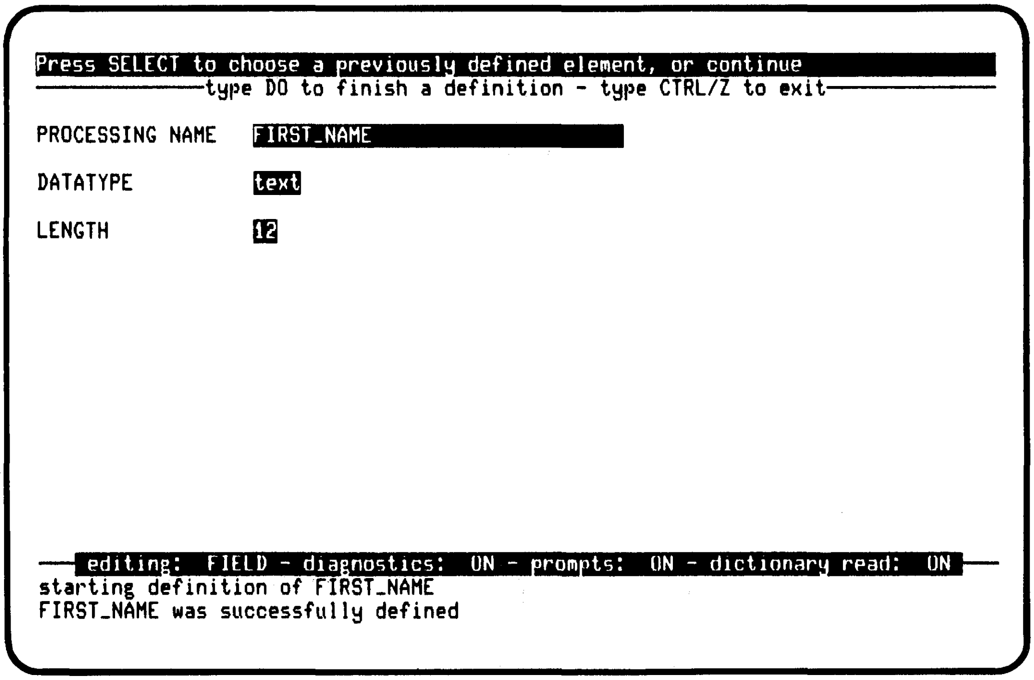


```
-------------type DO to finish a definition - type CTRL/Z to exit-------------
PROCESSING NAME   FIRST_NAME

DATATYPE          text

LENGTH            12










-- editing: FIELD - diagnostics: ON - prompts: ON - dictionary read: ON --
entity FIRST_NAME not found in dictionary
```

ZK-7470-HC

If you press RETURN, the menu of possible attributes and relationships is displayed again. At this point, you can select and enter more attributes. For this exercise, erase the menu by entering CTRL/Z. If you select an attribute by mistake, you can remove it from the display by pressing the REMOVE key.

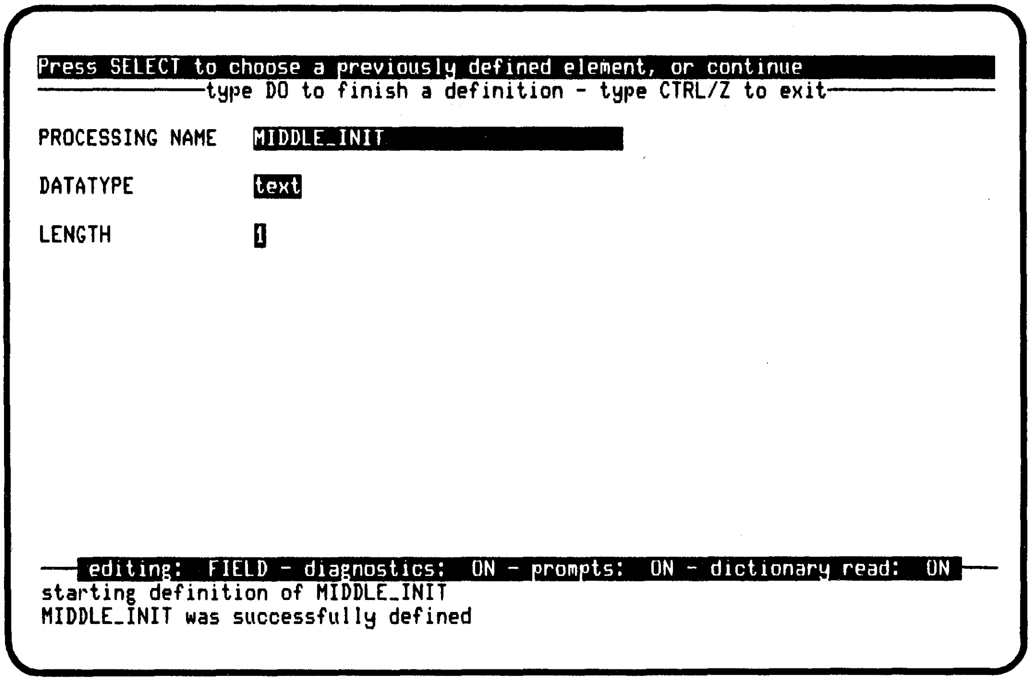
7. Press the DO key to enter the definition of the field FIRST_NAME. If you press the DO key before you select at least one attribute, the editor issues an error message. A success message is displayed at the bottom of your screen when CDO has completed the definition. The definition of the field FIRST_NAME is still displayed.



```
Press SELECT to choose a previously defined element, or continue
─────────────type DO to finish a definition - type CTRL/Z to exit─────────────

PROCESSING NAME   FIRST_NAME

DATATYPE          text

LENGTH            12










──editing:  FIELD - diagnostics:  ON - prompts:  ON - dictionary read:  ON──
starting definition of FIRST_NAME
FIRST_NAME was successfully defined
```

ZK-7472-HC

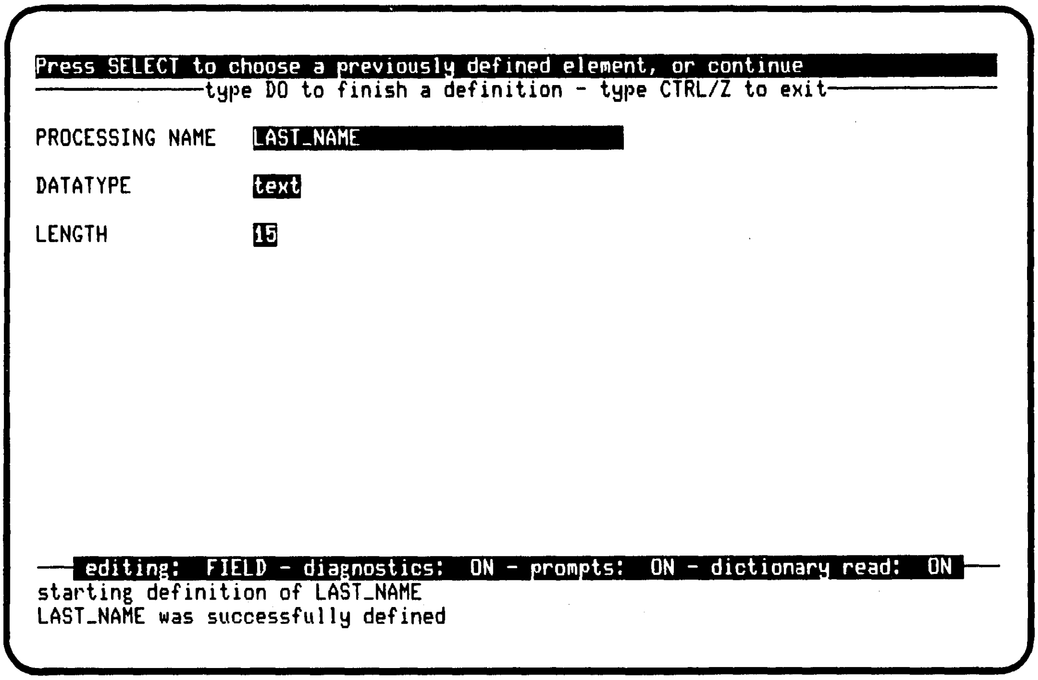
8. Begin a new definition by changing the processing name for the field definition. Delete FIRST_NAME with the DELETE key (the backspace key) and enter MIDDLE_INIT. Position the cursor on the 12 (using the arrow key) and change the text length to 1. The editor checks if MIDDLE_INIT is already defined. Press the DO key to enter the definition for the new field MIDDLE_INIT.



```
Press SELECT to choose a previously defined element, or continue
------------type DO to finish a definition - type CTRL/Z to exit------------

PROCESSING NAME  MIDDLE_INIT

DATATYPE         text

LENGTH           1








--- editing:  FIELD - diagnostics:  ON - prompts:  ON - dictionary read:  ON ---
starting definition of MIDDLE_INIT
MIDDLE_INIT was successfully defined
```

ZK-7471-HC

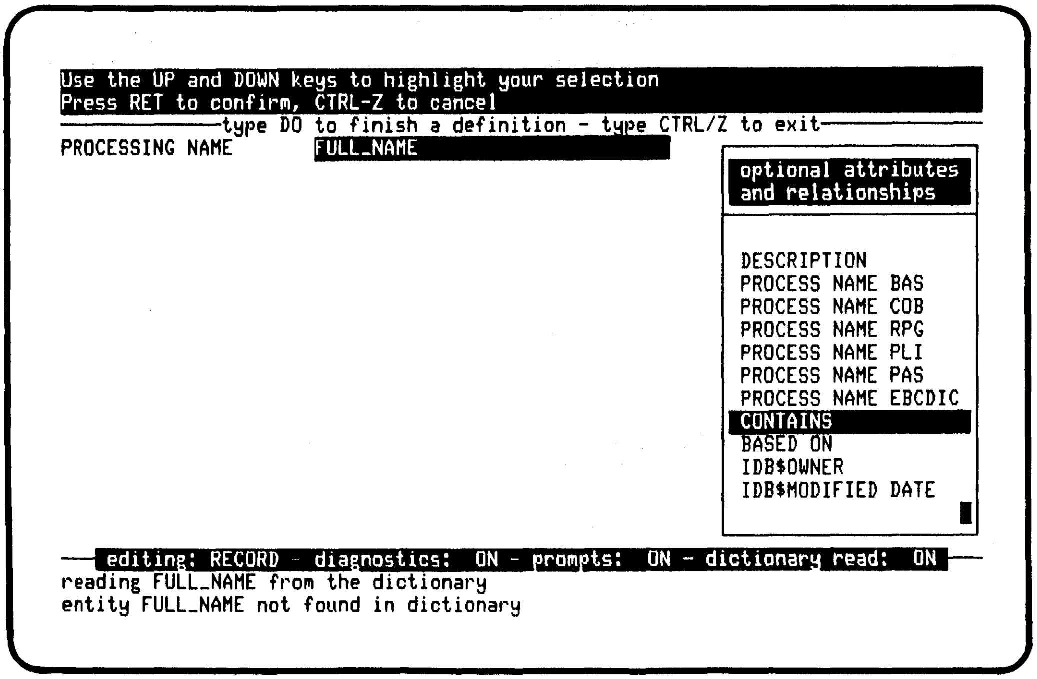
9. Begin a third field definition by changing the processing name to LAST_NAME. Change the text length to 15. You can change the data type and select other attributes at this point. Press the DO key to enter the definition of the field LAST_NAME.



```
Press SELECT to choose a previously defined element, or continue
-------------type DO to finish a definition - type CTRL/Z to exit-----------

PROCESSING NAME  LAST_NAME

DATATYPE         text

LENGTH           15








--- editing:  FIELD - diagnostics:  ON - prompts:  ON - dictionary read:  ON ---
starting definition of LAST_NAME
LAST_NAME was successfully defined
```

ZK-7473-HC

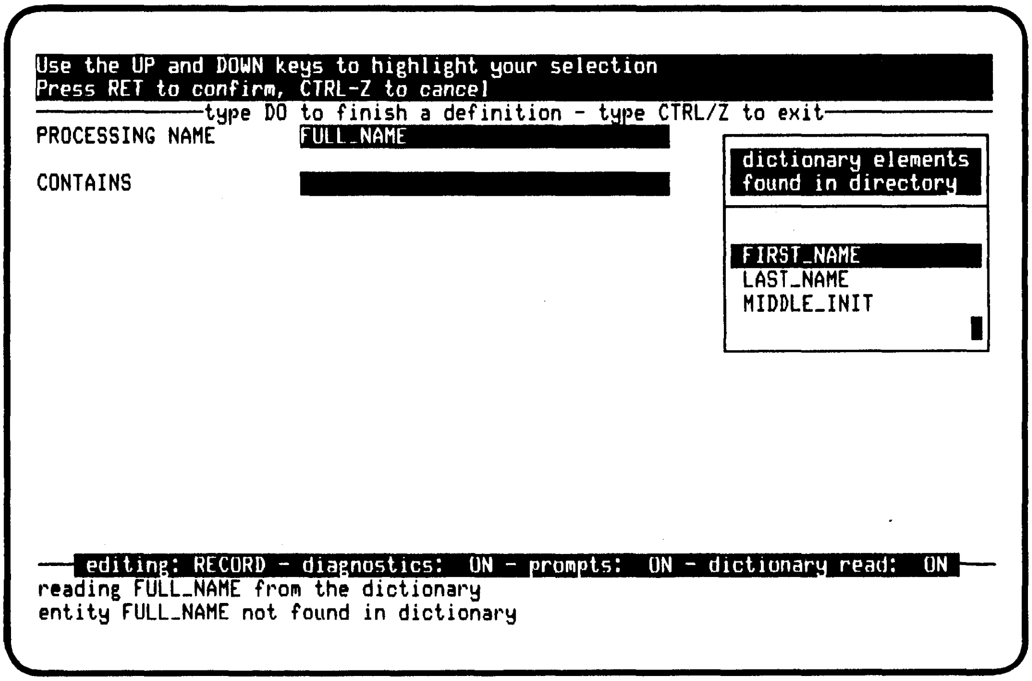
10. You are now going to create a record definition. Press PF1-KP8 to switch from defining a field to defining a record. Type FULL_NAME as the processing name for the record definition and press RETURN. CDO searches for a current record definition of that name. If CDO finds a record definition with that name, the definition is displayed on your screen; if the record does not already exist, you can begin to enter the new definition. A menu of possible record attributes is displayed on your screen. Move the cursor down to the CONTAINS attribute, as shown:



```
Use the UP and DOWN keys to highlight your selection
Press RET to confirm, CTRL-Z to cancel
─────────────type DO to finish a definition - type CTRL/Z to exit──────────
PROCESSING NAME     FULL_NAME
                                                     optional attributes
                                                     and relationships

                                                     DESCRIPTION
                                                     PROCESS NAME BAS
                                                     PROCESS NAME COB
                                                     PROCESS NAME RPG
                                                     PROCESS NAME PLI
                                                     PROCESS NAME PAS
                                                     PROCESS NAME EBCDIC
                                                     CONTAINS
                                                     BASED ON
                                                     IDB$OWNER
                                                     IDB$MODIFIED DATE

── editing: RECORD - diagnostics:  ON - prompts:  ON - dictionary read:  ON ──
reading FULL_NAME from the dictionary
entity FULL_NAME not found in dictionary
```

ZK-7474-HC

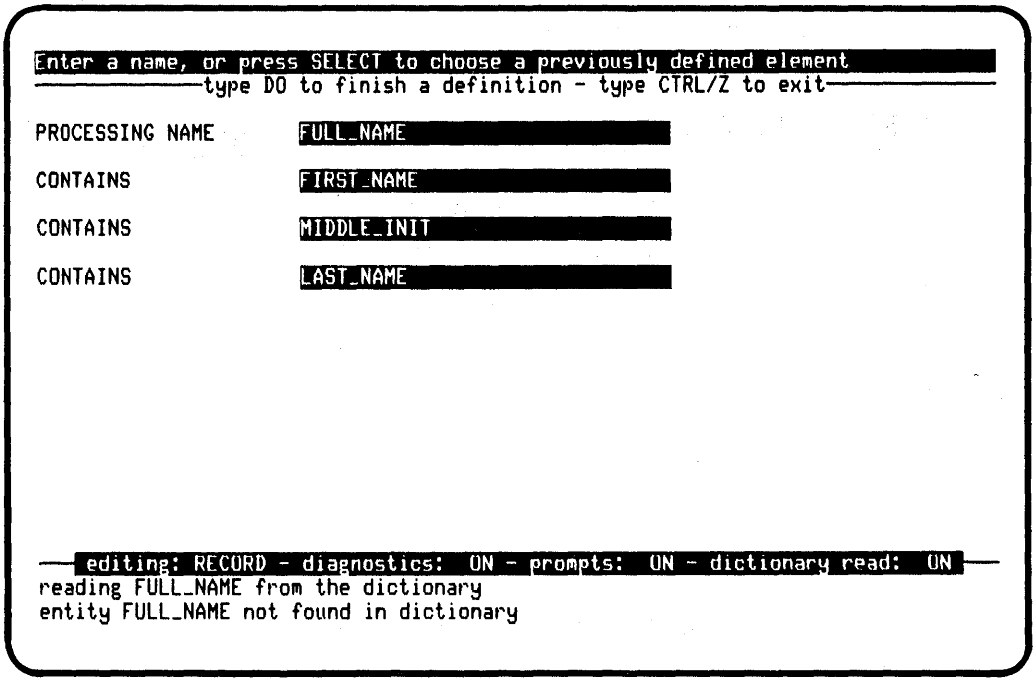
11. Select the CONTAINS attribute by pressing RETURN. You can now include fields and records that already exist in the new record definition. Press the SELECT key for a list of the current definitions in your dictionary directory. Use the arrow keys to highlight the field FIRST_NAME.



```
Use the UP and DOWN keys to highlight your selection
Press RET to confirm, CTRL-Z to cancel
─────────────type DO to finish a definition - type CTRL/Z to exit──────────
PROCESSING NAME      FULL_NAME
                                                  dictionary elements
CONTAINS                                          found in directory

                                                  FIRST_NAME
                                                  LAST_NAME
                                                  MIDDLE_INIT

── editing: RECORD - diagnostics:  ON - prompts:  ON - dictionary read:  ON ──
reading FULL_NAME from the dictionary
entity FULL_NAME not found in dictionary
```

ZK-7475-HC

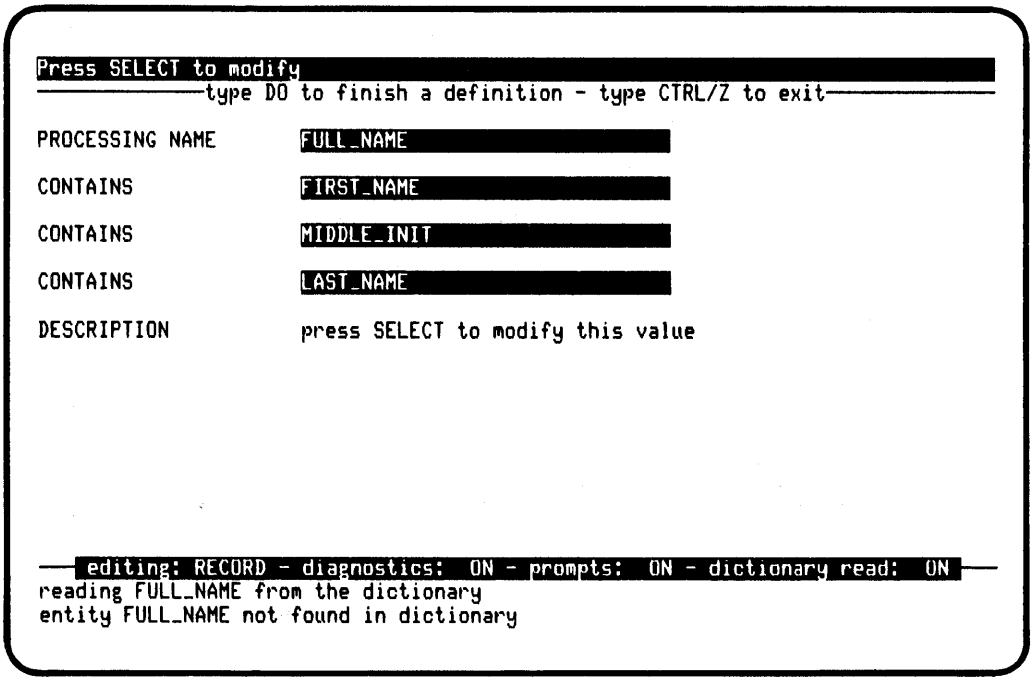
12. Press RETURN to include the field FIRST_NAME. After adding FIRST_NAME, press RETURN again to display the menu of record attributes. Repeat Steps 11 and 12 to select and include the fields MIDDLE_INIT and LAST_NAME.



```
Enter a name, or press SELECT to choose a previously defined element
--------------type DO to finish a definition - type CTRL/Z to exit--------------

PROCESSING NAME      FULL_NAME

CONTAINS             FIRST_NAME

CONTAINS             MIDDLE_INIT

CONTAINS             LAST_NAME







-- editing: RECORD - diagnostics:  ON - prompts:  ON - dictionary read:  ON --
reading FULL_NAME from the dictionary
entity FULL_NAME not found in dictionary
```

ZK-7476-HC

13. Press the INSERT key to view the menu of possible attributes again. Choose the DESCRIPTION attribute and press RETURN. Press the SELECT key to access the text editor. Enter some comments about the record definition, then press CTRL/Z and type EXIT to exit from the text editor. The record definition FULL_NAME is displayed on your screen again. The text you entered for the DESCRIPTION attribute is not displayed.



```
Press SELECT to modify
---------------type DO to finish a definition - type CTRL/Z to exit-----------

PROCESSING NAME        FULL_NAME

CONTAINS               FIRST_NAME

CONTAINS               MIDDLE_INIT

CONTAINS               LAST_NAME

DESCRIPTION            press SELECT to modify this value




---- editing: RECORD - diagnostics:  ON - prompts:  ON - dictionary read:  ON ---
reading FULL_NAME from the dictionary
entity FULL_NAME not found in dictionary
```

ZK-7477-HC

14. Press the DO key to enter the record definition for FULL_NAME.

15. Press CTRL/Z to exit from the editor.

16. Confirm that your definitions are contained in your examples directory by typing the following command at the CDO> prompt:

```
CDO> DIRECTORY *

Directory      DISK$01:[DICTIONARY_NAME]EDITOR_EXAMPLES

FIRST_NAME;1                                     FIELD
FULL_NAME;1                                      RECORD
LAST_NAME;1                                      FIELD
MIDDLE_INIT;1                                    FIELD
```

You can find more information about your definitions with the SHOW command, as described in Section 3.7.2.

3.4.4 CDO Editor Prompts

As you become more familiar with the CDO editor, you may no longer want the prompts displayed. Some of these prompts appear at the bottom of the sample screens in Section 3.4.3. To disable these prompts, press PF1-KP2.

The diagnostics prompt indicates whether or not the editor checks the validity of your entries. For example, the editor warns you if you try to enter a processing name of more than 31 characters. Similarly, when you specify a data type, you must specify a valid type from the supplied list. If, for example, you specify a data type that requires a signed word value for the scale, the editor issues an error message if you then supply a value that cannot be a signed word. You cannot enter hexadecimal or octal values within the editor. Diagnostics are ON by default. To disable diagnostics, press PF1-KP1.

The reading from the dictionary prompt indicates whether or not CDO searches through the dictionary for a definition with the specified processing name. For example, after entering the definition of a field, you can begin another definition by changing the processing name in the current editor display.

When reading from the dictionary is ON, CDO checks to see if a definition with this new processing name already exists; if it does, CDO places the definition in the editor display.

When reading from the dictionary is OFF, CDO does not search the dictionary for a definition with the same processing name and the screen display is not changed. Reading from the dictionary is ON by default.

To disable reading from the dictionary, press PF1-KP3. To resume reading from the dictionary after disabling, press PF1-KP3 again.

3.4.5 Editing Text Within the CDO Editor

Within the CDO editor, you can access VAXTPU to edit text in a definition. For example, to access the text editor to enter comments for a DESCRIPTION attribute, press the SELECT key while the cursor is placed on the DESCRIPTION attribute.

The CDO editor supplies the EVE interface for VAXTPU by default. You can choose another editing interface by defining an alternative section file for TPUSECINI in your login procedure. For example, after the following command is executed, the EDT interface is presented whenever VAXTPU is invoked:

```
$ DEFINE TPUSECINI  SYS$LIBRARY:EDTSECINI
```

3.4.6 Editing New Versions of Existing Definitions

When you enter a processing name for the editor, the editor searches through your dictionary for the named definition. If such a definition already exists, the editor displays the attributes and values associated with the previous definition. You can proceed to make the changes you require.

When you enter changes to a definition that was previously defined, you create a new version of the definition. The original version of the definition is *not* changed. (Entering definitions in the CDO editor is equivalent to using the DEFINE command, not the CHANGE command.) For example, if versions 1, 2, 3, and 4 of a field definition exist and you edit version 2, the new field definition becomes version 5.

CDO stores all versions of a definition unless you purge or delete them. An application can access any version of a definition provided that the application specifies the version number. When no version number is specified, CDO defaults to the highest version. For more information about changes and new versions, see Chapter 4.

3.4.7 Browsing Through Current Definitions

You can browse through your current definitions and examine them without leaving the editor by invoking the editor without specifying a definition name and pressing SELECT for a list of your current definitions. When you choose an item from this menu, the editor displays the selected definition on your editing screen. You can then select other definitions to view or change.

You can access a menu of previously defined definitions while you are editing another definition. Press the SELECT key while the cursor is positioned on an attribute that requires a definition name, such as the CONTAINS or BASED ON attributes, and the PROCESSING NAME. The items displayed in the menu are valid names of definitions already in your current dictionary directory. When you make your selection from the menu, the editor places the name you select in the attribute you are editing. In Section 3.4.3, Step 11 shows how you might complete a CONTAINS clause by selecting existing elements from a menu.

3.4.8 Exiting from the CDO Editor

To exit from the editor, press CTRL/Z. If you have not already pressed the DO key, the CDO editor prompts you to confirm whether or not you want to exit from the editor without entering your definition. Respond with a Y or N, depending on your choice. You can choose to quit from your editing session, or stay in the editing session and press the DO key to enter your definition before exiting.

3.5 Summary of CDO Environment Commands

There are two types of commands available in the CDO environment:

- Commands that allow you to set the environment characteristics to suit your needs. For example, you can log the output from your dictionary session in a file with the SET OUTPUT command. Most of these commands are discussed in this chapter.
- Commands that allow you to create and manipulate dictionary definitions and directories. For example, you can clear unused definitions with the PURGE and DELETE commands. The dictionary commands for creating and manipulating definitions are described in subsequent chapters of this manual.

For the most part, commands follow simple syntax rules. Note that the DEFINE, CHANGE, and DELETE commands require a terminating period. Table 3–3 summarizes the commands you can enter at the CDO> prompt.

Table 3–3: Summary of CDO Commands

Command	Purpose
AT (@ sign)	Executes a CDO environment command procedure.
ATTACH	Switches control from your current process to another process in your job.
CHANGE	Modifies the protection or definition of a dictionary definition. Requires a terminating period.
CLEAR	Deletes messages that have been sent to a dictionary definition.
CONVERT	Copies definitions from a DMU dictionary into CDO format.
COPY	Copies a named definition and its relationships.
DEFINE	Creates dictionaries, directories, access control lists, or dictionary definitions. Requires a terminating period.
DELETE	Deletes dictionaries, directories, access control lists, or dictionary definitions. Requires a terminating period.
DIRECTORY	Provides a list of definitions in a specified directory.
EDIT	Invokes the CDO screen editor and allows you to create field and record definitions.

Table 3–3: Summary of CDO Commands (Cont.)

Command	Purpose
ENTER	Creates a directory entry for specified fields within a converted record definition.
EXIT	Exits from the CDO environment and places your process at DCL level.
EXTRACT	Displays a specified definition in the format of the DEFINE command.
HELP	Invokes the online HELP facility and provides information about CDO commands.
MOVE	Moves a dictionary from one anchor to another, resolving all pointers to the old location.
PURGE	Deletes all but the highest-numbered versions of the specified definitions.
SET	Sets protection for dictionary definitions and allows you to select environment characteristics for your current session.
SHOW	Allows you to browse through your dictionary and display information about the characteristics of your current session.
SPAWN	Creates a subprocess and attaches to it.
VERIFY	Confirms the integrity of the dictionary and allows you to recover definitions after a dictionary session is aborted.

You can abbreviate a CDO command provided that the abbreviation is unique.

You do not need to use continuation marks to continue a command on the next line. If you press RETURN before your command is complete, CDO prompts you for the remaining input with a continuation prompt (cont>). The CDO> prompt is displayed again after the command is executed. CDO dynamically checks the command syntax as you enter partial command lines.

In the following example, CDO displays the continuation prompt after each line of command entry until the DEFINE FIELD command is completed:

```
CDO> DEFINE FIELD ASSET_ID
cont> DESCRIPTION IS
cont> /* Equipment asset number */
cont> DATATYPE IS ZONED NUMERIC
cont> SIZE IS 10.
CDO>
```

For complete information on the syntax of all CDO commands, see the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

3.6 Setting CDO Environment Characteristics

This section discusses the SET and DEFINE KEY commands, which allow you to select the characteristics for your session in the CDO environment. Section 3.6.5 includes a sample command procedure that uses these commands.

3.6.1 Sending Command Output to a File

Whenever you execute CDO commands, you can optionally send the output from the commands to a file. Command output is sent to the specified file and also displayed at your terminal unless another device is specified for the logical name SYS$OUTPUT.

To send output to a specified file, enter the command SET OUTPUT with the file specification at the CDO> prompt. The SET OUTPUT command remains in effect for other CDO commands that you execute during the same environment session. To stop sending command output to the specified file, reenter the command SET OUTPUT without any file specification, as shown:

```
CDO> SET OUTPUT DEFINITION.LOG
CDO> SHOW DEFAULT
CDD$DEFAULT = DISK$01:[CORPORATE.MIS]PERSONNEL
CDO>
     .
     .
     .
CDO> SET OUTPUT
CDO> EXIT
```

3.6.2 Setting the Default Directory

You can abbreviate references to dictionary definitions by setting a default dictionary directory. If you establish [CORPORATE.MIS]PERSONNEL as your default directory, for example, you can then omit that part of the name and refer to a definition in that directory by the definition name alone.

You can establish an initial default directory to take effect whenever you invoke CDO. Whatever you assign to the logical name CDD$DEFAULT becomes your initial default dictionary directory every time you invoke CDO. When you invoke CDO, CDD/Plus searches for a translation for the logical name CDD$DEFAULT. The directory associated with CDD$DEFAULT establishes your initial dictionary directory. The following DCL command defines CDD$DEFAULT to be an anchor (a VMS directory) plus a dictionary directory path. You should define your initial directory to be the dictionary area where you commonly work.

```
$ ASSIGN "DISK$01:[CORPORATE.MIS]PERSONNEL" CDD$DEFAULT
$ DICTIONARY OPERATOR
CDO> SHOW DEFAULT
DISK$01:[CORPORATE.MIS]PERSONNEL
```

If you work in the compatibility dictionary, you can use CDD$TOP in the definition of the logical name CDD$DEFAULT. The following example sets your initial dictionary area to be the CONTRACT directory in the compatibility dictionary:

```
$ ASSIGN "CDD$TOP.PERSONNEL.CONTRACT" CDD$DEFAULT
$
```

When no translation for CDD$DEFAULT exists, your initial dictionary directory is the VMS directory where your compatibility dictionary resides (CDD$TOP). You can override your initial default with the SET DEFAULT command.

You can change your dictionary directory at any time within the CDO environment with the SET DEFAULT command. With the SET DEFAULT command, you supply the full path to the dictionary area you plan to work in. You can specify the path in the following ways:

- A path originating with CDD$TOP
- A path originating with a VMS directory specification (the dictionary anchor)
- A logical name or search list

For example, after setting the default to DISK$01:[CORPORATE.MIS]PERSONNEL as shown below, you can create a field definition named JOB_TITLE to be contained in this directory without specifying the full path name for JOB_TITLE.

```
CDO> SET DEFAULT DISK$01:[CORPORATE.MIS]PERSONNEL
CDO> SHOW DEFAULT
DISK$01:[CORPORATE.MIS]PERSONNEL
CDO> DEFINE FIELD JOB_TITLE
cont> DATATYPE IS TEXT
cont> SIZE IS 15.
CDO> DIRECTORY
Directory DISK$01:[CORPORATE.MIS]PERSONNEL
JOB_TITLE;1                           FIELD
CDO>
```

Note

After a default directory has been established, it remains in effect until another default is established with a subsequent SET DEFAULT command.

You can change the default directory as often as you need to during a dictionary session. To change the default directory to DISK$01:[CORPORATE.MIS]SUPPORT, for example, simply specify this directory with the SET DEFAULT command. You can include the SET DEFAULT command in a CDO command procedure. For details on command procedures, see Section 3.8.

When you access CDO at the DCL prompt, you cannot override your default dictionary directory. Therefore, before you use a CDO foreign command with the DICTIONARY command, make sure that you have set your CDD$DEFAULT to the dictionary that you want to work in. You can use the ASSIGN command at the DCL prompt to set your default dictionary directory without entering CDO.

You can set up a logical name to be equivalent to one or more dictionary areas, then use the logical name in the SET DEFAULT command. In the following example, MY_DICT is defined as the logical name for the dictionary area DISK$01:[MCKAY.TESTS], then specified in the SET DEFAULT command.

```
$ DEFINE MY_DICT DISK$01:[MCKAY.TESTS]
$ DICTIONARY OPERATOR
CDO> SET DEFAULT MY_DICT
CDO> DIRECTORY ADDRESS_RECORD

Directory DISK$01:[MCKAY.TESTS]

ADDRESS_RECORD;2                        RECORD
ADDRESS_RECORD;1                        RECORD
CDO>
```

To override a default directory, you must specify fully qualified names. (For more information on fully qualified names, see Section 1.3.2.) You can specify another path explicitly, or use another logical name.

3.6.3 Using Search Lists

You can use access more than one physical dictionary by using search list logical names to identify physical dictionary areas that you want to treat as a single dictionary. You create a search list when you assign one or more dictionary areas to a logical name, using the DCL command ASSIGN or DEFINE. In the following example, a search list is established by defining the logical name MY_DICT as the four physical dictionaries illustrated in Figure 3-1.

```
$ DEFINE MY_DICT CDD$TOP.PERSONNEL,DISK$01:[CORPORATE.MIS]PERSONNEL,-
_$ CDD$TOP.CORPORATE,FARWAY::SYS$DISK:[MCKAY.DICTIONARY]
```

If you specify the logical name for your search list in the SET DEFAULT command, the first area specified in the search list becomes your default dictionary area. Commands that directly affect definitions, such as DEFINE and CHANGE, affect definitions in only the first dictionary area in a search list. Searching commands, such as DIRECTORY and SHOW, search through all areas in the search list, as shown in the following example:

```
$ !
$ !Define a logical name equivalent to several dictionary areas
$ !
$ DEFINE MY_DICT CDD$TOP.PERSONNEL,DISK$01:[CORPORATE.MIS]PERSONNEL,-
_$ CDD$TOP.CORPORATE,FARWAY::SYS$DISK:[MCKAY.DICTIONARY]
$ !
$ !Invoke CDO
$ !
$ DICTIONARY OPERATOR
CDO> !
CDO> !Set the default to your search list areas
CDO> !
CDO> SET DEFAULT MY_DICT
CDO> !
CDO> SHOW DEFAULT
MY_DICT
     = SYS$COMMON:[CDDPLUS]PERSONNEL
     = DISK$01:[CORPORATE.MIS]PERSONNEL
     = SYS$COMMON:[CDDPLUS]CORPORATE
     = FARWAY::SYS$DISK:[MCKAY.DICTIONARY]

CDO> DIRECTORY FIELD_TESTING

 Directory SYS$COMMON:[CDDPLUS]PERSONNEL
FIELD_TESTING;1                       FIELD

 Directory FARWAY::SYS$DISK:[MCKAY.DICTIONARY]
FIELD_TESTING;2                       FIELD
FIELD_TESTING;1                       FIELD
CDO>
```

You can associate a search list with the logical name CDD$DEFAULT. First assign a search list to a logical name, then assign the logical name to CDD$DEFAULT. Your initial default is the first dictionary area specified in the search list.

If you frequently access large portions of several different dictionaries, you can benefit from building a more complex group of logical names for dictionary areas. You can define a logical name to be equivalent to previously defined logical names, as shown in the following example. The subsequent DIRECTORY command searches through all the specified dictionary areas equivalent to the logical name CORPORATE.

```
$ DEFINE PERSONNEL RETIRED,CURRENT,INTRANSIT
$ DEFINE COMPANY PERSONNEL,FINANCE,PAYROLL
$ DEFINE CORPORATE COMPANY,REGION,MINE
$ DICTIONARY OPERATOR
    .
    .
    .
CDO> !
CDO> !List all fields in your widest search list area
CDO> !
CDO> DIRECTORY/TYPE=FIELD CORPORATE.BADGE_NO
    .
    .
    .
```

CDO searches through the dictionary areas for field definitions with the name BADGE_NO in the following order: RETIRED, CURRENT, INTRANSIT, FINANCE, PAYROLL, REGION, MINE. See Section 3.7.1 for more information on the DIRECTORY command.

You can use logical names as path names within VAX language statements that include definitions from the dictionary.

3.6.4 Defining Terminal Keys

You can save time entering frequently used commands by binding commands to terminal keys. To do this, use the DEFINE KEY command. You must supply the name of the key that you are defining, and the string that you want to be associated with the key. The definitions are valid until you exit CDO. (See the *VAX CDD/Plus Common Dictionary Operator Reference Manual* for valid key names.)

The following example uses the DEFINE KEY command to associate the key PF4 with the string SET DEFAULT [HEINES.CDDPLUS]. After the DEFINE KEY command is executed, you can enter SET DEFAULT [HEINES.CDDPLUS] at the CDO> prompt by simply pressing the PF4 key. The cursor remains at the end of the command line, allowing you to add directory names to your usual default, or to press RETURN to execute the command as it is.

```
CDO> DEFINE KEY PF4 "SET DEFAULT [HEINES.CDDPLUS]"
CDO> PF4
CDO> SET DEFAULT [HEINES.CDDPLUS]
```

To display the definition of a particular key in the CDO environment, enter the SHOW KEY command at the CDO> prompt. For example:

```
CDO> SHOW KEY PF4
DEFAULT keypad definitions:

SET DEFAULT [HEINES.CDDPLUS]              PF4
CDO>
```

The SHOW KEY/ALL command displays all of the key definitions you have set in the CDO environment.

The DEFINE command has many other uses that are discussed in later chapters of this manual and in the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

3.6.5 CDO Initialization Files

Rather than entering the same set of commands at the start of each CDO session, you can include the commands in an initialization file. A CDO **initialization file** is a command procedure that is executed when you invoke CDO. When you invoke CDO, it automatically searches your default directory for the file CDO$INIT.CDO and executes the file. (You can optionally define CDO$INIT.CDO to point to a location other than your default directory.)

You can write your own initialization file, name it CDO$INIT.CDO, and thereby set up your own environment characteristics at the start of each dictionary session. You can define a logical name for your initialization file by placing the following line in your LOGIN.COM file:

```
$ DEFINE CDO$INIT SYS$LOGIN:CDO$INIT.CDO
```

In the example above, you can replace SYS$LOGIN with any directory specification where you want to keep your initialization file. However, if you define CDO$INIT.CDO in your login file, you can use it no matter what directory you call CDO from.

The following sample initialization file sets up environment characteristics for a CDO session:

```
!
!CDO$INIT.CDO initialization file
!
!Subsequent commands executed in this procedure will be echoed at the terminal
!
SET VERIFY
!
!Define key to get back to top level quickly
!
DEFINE KEY PF4 "SET DEFAULT [CORPORATE.MIS]"
```

```
!
!Define key to quickly get to work area
!
DEFINE KEY PF3 "SET DEFAULT [CORPORATE.MIS]PERSONNEL"
!
!Select initial default to usual dictionary area
!
SET DEFAULT [CORPORATE.MIS]PERSONNEL.RETIRED
!
!Send output from subsequent commands to a file
!
SET OUTPUT OUTPUT.LOG
```

See Section 3.8 for more information about executing command procedures in CDO.

3.7 Manipulating Dictionary Definitions

Once you have definitions stored in the dictionary, you frequently need to view those definitions in various ways. The following sections discuss the DIRECTORY, SHOW FIELD, SHOW RECORD, and EXTRACT commands. These commands allow you to view the contents of your dictionary.

3.7.1 Listing Dictionary Definitions

The CDO command DIRECTORY produces a catalog of the dictionary definitions contained in the specified directory. You can use this command as an easy way to check on the contents and structure of a portion of your dictionary. In the following example, the DIRECTORY command shows a list of definitions contained in the directory [CORPORATE.MIS]. CDO displays the current directory and lists each definition, showing the definition name and its type.

```
CDO> SET DEFAULT DISK$01:[CORPORATE.MIS]
CDO> DIRECTORY
Directory DISK$01:[CORPORATE.MIS]
BENEFITS                                                DIRECTORY
CDD$PROTOCOLS                                           DIRECTORY
CONTRACT                                                DIRECTORY
PERSONNEL                                               DIRECTORY
RATE;2                                                  FIELD
RATE;1                                                  FIELD
TAXES                                                   DIRECTORY
CDO>
```

You can also use wildcards to list all definitions in the directory. For example, the following command produces a list of all the subdirectories and definitions contained within the CONTRACT and SALARIED subdirectories of the PERSONNEL directory. Note that all versions of the definitions are displayed.

```
CDO> SET DEFAULT [CORPORATE.MIS]PERSONNEL
CDO> DIRECTORY *...

Directory DISK$01:[CORPORATE.MIS]PERSONNEL.CONTRACT

BADGE_NO;1                                         FIELD
EMPLOYEE_REC;1                                     RECORD
FIRST_NAME;1                                       FIELD
FULL_NAME;1                                        RECORD
MIDDLE_INIT;1                                      FIELD
WAGE_CLASS;1                                       FIELD
YR_TO_DATE;1                                       FIELD

Directory DISK$01:[CORPORATE.MIS]PERSONNEL.SALARIED

BADGE_NO;1                                         FIELD
FIRST_NAME;1                                       FIELD
FIRST_NAME;2                                       FIELD
FULL_NAME;1                                        RECORD
MIDDLE_INIT;1                                      FIELD
WAGE_CLASS;1                                       FIELD
YR_TO_DATE;1                                       FIELD
CDO>
```

The optional /TYPE qualifier allows you to limit the type of definitions that the DIRECTORY command displays. The following command uses the /TYPE qualifier to display only the record definitions in the directory specified by the logical name CONTRACT_DIR.

```
CDO> DIRECTORY /TYPE=RECORD CONTRACT_DIR.*

Directory DISK$01:[CORPORATE.MIS]PERSONNEL.CONTRACT
EMPLOYEE_REC;1                                     RECORD
FULL_NAME;1                                        RECORD
CDO>
```

You can create CONTRACT_DIR in your initialization file, CDO$INIT.CDO, just as you would create other VMS system logical names in dedicated initialization files. For more information on initialization files, see Section 3.6.5.

As the example shows, you can use wildcards in the name of the specified entity. You cannot, however, use wildcards in the /TYPE=(protocol-name) clause. For example:

```
CDO> SET DEFAULT [PERSONNEL.CONTRACT]
CDO> DIR/TYPE=(REC*)
%CDO-E-ERRDIRE, error during directory
-CDO-E-BAD_NAME, The protocol name supplied contains illegal wildcard characters
```

The form of the preceding commands is DIRECTORY/BRIEF, which is the default. You can use the /FULL qualifier with the DIRECTORY command to obtain a list of directory definitions and all the related information that CDO stores for the definition, such as the creation date, modification date, protection provisions, and relative size of the definition. The DIRECTORY/FULL command also displays whether the specified definition is stored in CDO or DMU format.

```
CDO> SET DEFAULT [CORPORATE.MIS]PERSONNEL.CONTRACT
CDO> DIRECTORY/FULL

    Directory DISK$01:[CORPORATE.MIS]PERSONNEL.CONTRACT

RATE;1                                                 FIELD
Created: 6-MAR-1987 12:19:29.57       Revised: 6-MAR-1987 12:19:29.57
Size: 30 blocks                       Dictionary storage:  CDO format
Owner [13,10]
Access Control List:
(IDENTIFIER=[13,10],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+
    DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=PERSONNEL,ACCESS=READ+SHOW+DEFINE)
(IDENTIFIER=SECRETARIES,ACCESS=SHOW)

YR_TO_DATE;1                                             FIELD
Created: 7-JAN-1987 19:10:23.42   Revised: 28-JAN-1987 15:18:20:56
Size: 28 blocks                   Dictionary storage:   CDO format
Owner [13,62]
Access Control List:
(IDENTIFIER=[13,62],ACCESS=ALL)
(IDENTIFIER=PERSONNEL,ACCESS=READ+WRITE+CREATE)
(IDENTIFIER=[20,*],ACCESS=READ)
CDO>
```

Like the DIRECTORY in DCL, the DIRECTORY command in CDO accepts the /SINCE qualifier with the parameters TODAY, YESTERDAY, and TOMORROW. For example, the following CDO command includes the qualifier /SINCE=TODAY to list only the definitions created since 00:00 o'clock (midnight) on the current day, month, and year:

```
CDO> DIRECTORY/SINCE=TODAY *
Directory DISK$01:[CORPORATE.MIS]PERSONNEL.CONTRACT

RATE;2                                           FIELD
CDO>
```

Dictionary subdirectories are not subject to the /SINCE parameter, so they will appear in output regardless of when they were created.

3.7.2 Showing Dictionary Definitions

You can also display dictionary definitions with the SHOW FIELD, SHOW RECORD, and SHOW DATABASE commands. These commands display the attributes stored in the dictionary for a specified definition. For example, the following SHOW FIELD command displays the information stored in the dictionary for the field definition SUPERVISOR_NAME.

```
CDO> SHOW FIELD SUPERVISOR_NAME
Definition of field SUPERVISOR_NAME
|   Datatype             text size is 20 characters
CDO>
```

You can optionally use the /BRIEF qualifier to display the output shown above, which is the default SHOW command output. To display more information about a definition, you can use other optional qualifiers. The /FULL qualifier is shown in the following example:

```
CDO> SHOW FIELD /FULL SUPERVISOR_NAME
Definition of field SUPERVISOR_NAME
|   Edit_string           X(30)
|   Based on              LAST_NAME
|   |   Datatype              text size is 20 characters
CDO>
```

The output from /BRIEF or /FULL shows only the **user-specified** attributes, those attributes stored in the directory by the user who created or changed the field. To display more information about a definition, you can use the /ALL qualifier. The SHOW/ALL command displays the definition's **system-specified** attributes in addition to its user-specified attributes. System-specified attributes include the owner, creation date, modification date, protection provisions, history list, and relative size of the definition. For more information about user-specified attributes, see Section 4.1.2. You can use the /ALL qualifier with all of the SHOW commands except the usage tracking commands listed in Table 6-1. The following SHOW FIELD /ALL command displays both the user-specified and system-specified attributes of the field definition STATE.

```
CDO> SHOW FIELD /ALL STATE
Definition of field STATE
|   acl
(IDENTIFIER=[CDD,MCKAY],ACCESS=READ+WRITE+MODIFY+ERASE
           +SHOW+DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[*,*],ACCESS=READ+WRITE+MODIFY+ERASE
                    +SHOW+OPERATOR+ADMINISTRATOR)
|   Created time         30-APR-1987 15:48:19.04
|   Modified time        30-APR-1987 15:48:19.04
|   Owner                MCKAY
|   Datatype             text size is 2 characters
|   |   History entered by MCKAY ([CDD.MCKAY])
|   |   using CDO V1.0
|   |   |   CREATE definition on 30-APR-1987 15:47:41.24
CDO>
```

When you display a record definition with the SHOW RECORD command, CDO displays the included field definitions:

```
CDO> SHOW RECORD FULL_NAME
Definition of FULL_NAME
|   Contains field         FIRST_NAME
|   Contains field         MIDDLE_INIT
|   Contains field         LAST_NAME
CDO>
```

When you use the /FULL qualifier, as in the following SHOW RECORD command, CDO displays the included field definitions and attributes:

```
CDO> SHOW RECORD /FULL FULL_NAME
Definition of FULL_NAME
|   Contains field        FIRST_NAME
|   |   Datatype         text size is 15 characters
|   Contains field        MIDDLE_INIT
|   |   Datatype         text size is 1 characters
|   Contains field        LAST_NAME
|   |   Datatype         text size is 30 characters
CDO>
```

When you display a database definition with the SHOW DATABASE command, CDO displays the database name, file name, and fully qualified path name.

```
CDO> SHOW DATABASE/BRIEF DEPT5
Definition of database  DEPT5
|   database uses RDB database DEPT5
|   database in file DEPT5
|   |   fully qualified file SYS$COMMON:[MIS.DATABASES]DEPT5.RDB;1
```

To view the complete definition of an Rdb/VMS database, use the SHOW_USED_BY/FULL command, RDO (Relational Database Operator), or VAX SQL (Structured Query Language). See *VAX Rdb/VMS Reference Manual* or *VAX SQL Reference Manual* for information about displaying Rdb/VMS database definitions.

Note

The SHOW FIELD, SHOW RECORD, and SHOW DIRECTORY commands display processing names, not directory names. (The difference between the processing name and the directory name is discussed in Section 2.4.1.) Unless definitions were created using the CDD/Plus call interface, your field and record definitions have a processing name that is the same as the directory name that you use to refer to it. Therefore, in most cases, you need not be concerned with the processing names displayed by the SHOW FIELD and SHOW RECORD commands.

The processing name may be important to you if you use dictionary definitions in an Rdb/VMS database. For information about naming definitions for an Rdb/VMS database, see Chapter 7.

To see the history list for a definition, use the /AUDIT qualifier with the SHOW FIELD or SHOW RECORD command. Chapter 4 discusses how to enter descriptions and history lists for your definitions.

You can use the SHOW command for many purposes. Further examples of the SHOW commands are included throughout this manual; syntax information is contained in the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

3.7.3 Examining Current Definitions

It is easier to change or redefine a definition if you can examine the current definition. If you created a definition by selecting attributes in the CDO editor, the EXTRACT command recalls the full DEFINE command that the editor used to create the definition. In the following example, EXTRACT shows the DEFINE command that created the field definition FIRST_NAME.

```
CDO> EXTRACT FIELD FIRST_NAME
Define field SYS$DISK:[COMPAT_DICT]EDITOR_EXAMPLES.FIRST_NAME
   Datatype            text size is 12 characters

CDO>
```

You can optionally extract output to a file rather than to your terminal with the SET OUTPUT command. With the output from the EXTRACT command in a file, you can use one of the VMS text editors to change the command syntax, add new attributes, or make whatever changes you wish. You can then execute the new DEFINE FIELD command later at the CDO> prompt with the AT command (@). For example:

```
!
!Send output to a file
!
CDO> SET OUTPUT [WORKSPACE]FIRST_NAME.CDO
CDO> EXTRACT FIELD FIRST_NAME
!
!Set output to null to avoid locking file
!
CDO> SET OUTPUT
!
!Spawn to a subprocess to use a DCL editor
!
CDO> SPAWN
!
!Make changes to file in a text editor at DCL level
!
$ EDIT/TPU [WORKSPACE]FIRST_NAME.CDO
    .
    .
    .
!
!Return to your original process
!
$ ATTACH MCKAY
!
!Execute the modified command file
!
CDO> @[WORKSPACE]FIRST_NAME
```

The following chapter discusses how to create field and record definitions with the DEFINE command.

3.8 Executing CDO Environment Command Procedures

You can include frequently used commands in a file and execute the file at the CDO> prompt.

Command procedures should contain no prompts; place the CDO commands directly at the left margin of your file. CDO ignores characters after the exclamation point (!) on a command line; this allows you to include comments in your file.

Execute a command procedure in CDO with the AT sign command (@). During execution, CDO executes the commands in the procedure in sequential order. When no file extension is specified, CDO looks for a file with the file extension of .CDO.

You can optionally include the SET VERIFY command in a command procedure. The SET VERIFY command instructs CDO to display subsequent commands on your screen as they are executed. By default, command lines are not echoed (SET NOVERIFY). The following lines are contained in PERSONAL.CDO:

```
SET VERIFY
!
!Change default dictionary area
!
SET DEFAULT [DICTIONARY]PERSONAL
```

At the CDO> prompt, the following command executes all the commands in the procedure PERSONAL.CDO. Each command, except the SET VERIFY command, is echoed to the terminal. The full VMS file specification is included on the command line. Remember that the VMS directory where a CDO dictionary resides should contain no files other than those that CDD/Plus creates. You should not place a command procedure like PERSONAL.CDO in the VMS directory containing the CDO dictionary.

```
CDO> @SYS$DISK:[MYACCOUNT.COMS]PERSONAL
SET DEFAULT [DICTIONARY]PERSONAL
CDO>
```

You can define dictionary definitions in a command procedure. Instead of entering many definitions at the CDO> prompt or in the editor, you can include defining commands in a file and execute the file at the CDO> prompt. This method is convenient when you anticipate creating new versions of particular field or record definitions—for example, for testing purposes. Maintaining long definitions in command procedures can prevent repetition and syntax errors.

The following command procedure, DEFINE_ADDRESS.CDO, creates a record definition that includes five fields that are already defined. The record definition is created by executing DEFINE_ADDRESS.CDO at the CDO> prompt:

```
CDO> @SYS$COMMON:[MCKAY.COMFILES]DEFINE_ADDRESS
DEFINE RECORD ADDRESS.
       UNIT.
       STREET.
       HOME_TOWN.
       STATE.
       ZIP.
END ADDRESS RECORD.
```

The following file, CREATE_TESTS.CDO, executes several other CDO command procedures. The commands and output from the commands executed in CREATE_TESTS.CDO and any subsequent commands are logged in the file OUTPUT.LOG.

```
!
!Send output of commands in this procedure to OUTPUT.LOG
!
SET OUTPUT OUTPUT.LOG
!
!Each command will be listed after execution
!
SET VERIFY
!
!Set the default directory to be TESTS
!
SET DEFAULT [MAIN.FINANCE]TESTS
!
!Define the fields for testing
!
@DEFINE_TEST_FIELDS.CDO
!
!Define the records for testing
!
@DEFINE_TEST_RECORDS.CDO
!
!List contents of directory
!
DIRECTORY TEST*
!
!Run tests
!
@TEST_RUN.CDO
```

3.9 Checking Your Dictionary Version

You can use the CDO SHOW VERSION command to find out the version number of:

- CDD/Plus installed on your system
- CDO installed on your system
- Dictionaries you have accessed during the current session

Changes in the dictionary software and the disk structure (database schema) create new versions. Although dictionary users cannot cause such changes, you may need to give the version numbers when reporting dictionary problems to your system manager. The system manager can compare the version numbers to determine whether or not different dictionaries are at the same revision level.

In order to see the version number for a certain dictionary, you must have invoked that dictionary earlier in your session. To invoke a dictionary, issue a command that manipulates the elements within that dictionary, such as DEFINE or SHOW. You cannot invoke a dictionary using a SET DEFAULT command or a DIRECTORY command that specifies a dictionary directory that does not contain dictionary elements.

During a session in which you have invoked two dictionaries, SHOW VERSION produces output like the following:

```
CDO> SHOW VERSION
CDO V1.0
CDD/Plus V4.0
Dictionary DISK$01:[CORPORATE.MIS]CDD$DATABASE;1
        Major Version 15
        Minor Version 5
Dictionary SYS$COMMON:[CDDPLUS]CDD$DATABASE;1
        Major Version 15
        Minor Version 5
CDO>
```

A **minor** version of the dictionary is created when the new version of dictionary software will continue to run on old dictionary databases without change.

A **major** version of the dictionary is created when a rebuild of the dictionary is required before the new version of software will run on old dictionary databases. Minor versions of the dictionary are upward compatible; major versions are not.

During a session in which you do not invoke a dictionary, the SHOW VERSION command displays only the version of CDO, the version of CDD/Plus, and the fully qualified name of the dictionary.

Populating Your Dictionary 4

This chapter shows you how to create CDO field and record definitions and how to make changes to definitions.

4.1 Creating Dictionary Definitions

You can define and store many types of data definitions in a CDO dictionary. This chapter is limited to the most commonly used dictionary entities: field and record definitions. For information about creating database descriptions in the dictionary, see Chapter 7. For information about creating other dictionary entities, see the sections on the DEFINE GENERIC command in the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

You can create field and record definitions in the CDO utility in two ways:

- With the CDO screen editor
- With the DEFINE command at the CDO> prompt

CDD/Plus provides a convenient screen editor in the CDO utility. The CDO editor is menu driven and easy to use. The editor is described in Chapter 3. When you create definitions with the editor, it generates the appropriate defining commands for you. For the purpose of illustration, the definitions in this chapter are all created with the DEFINE command at the CDO> prompt.

All of the definitions listed in this chapter can be created in a command procedure with a text editor at DCL level. You can then execute the command procedure at the CDO> prompt with the AT command (@), as described in Chapter 3. CDO assumes the default file extension of .CDO and searches for the specified file in the default VMS dictionary directory, unless another directory is named.

4.1.1 Documenting Dictionary Definitions

You can document your definitions in two ways:

- The DESCRIPTION attribute allows you to include comments within a definition.
- The AUDIT attribute allows you to add an entry to the history list for a definition.

Both of these attributes are valid for the DEFINE and CHANGE commands. The text you supply for these attributes can spread over several lines provided that you delimit each line of text with the characters /* and */, as shown in the examples in these sections.

4.1.1.1 The DESCRIPTION Attribute —You can document a definition by enclosing comments in the DESCRIPTION attribute of a DEFINE or CHANGE command. Including the DESCRIPTION attribute in a definition is optional. However, if you include it, the DESCRIPTION attribute must be the first attribute listed in the DEFINE command.

```
CDO> DEFINE FIELD CDD$TOP.PERSONNEL.TEST
cont> DESCRIPTION /* This is a test */
cont>             /* Add more text like this */.
    .
    .
    .
```

You can display the DESCRIPTION text for an existing definition with the EXTRACT command or the SHOW command. The EXTRACT command displays a definition in the DEFINE command format.

4.1.1.2 The AUDIT Attribute —CDD/Plus maintains a history list for each CDO dictionary definition. You include a remark in the history list for a definition with the AUDIT attribute in the DEFINE command.

```
CDO> DEFINE FIELD CDD$TOP.PERSONNEL.TEST
cont> DESCRIPTION /* This is a test */
cont> AUDIT /* An example history entry */.
    .
    .
    .
```

Including the AUDIT attribute in a definition is optional. However, if you include it, the AUDIT attribute must be listed after the DESCRIPTION attribute in the DEFINE command. If no DESCRIPTION attribute is included, AUDIT must be the first attribute listed. If you do not include the AUDIT attribute, an empty remark is recorded in the history list for the definition.

When you compile a VAX language program that uses a dictionary definition, you can add an entry in the history list for that definition. For example, the following command adds the remark "Used for payroll run with PAYROLL_SALES2.COB" to the history list for the dictionary definition included in the COBOL program:

```
$ COBOL/AUDIT="Used for payroll run with PAYROLL_SALES2.COB"/LIST PAYROLL_SALES2
```

To display dictionary definitions, use the /AUDIT qualifier with the SHOW FIELD and SHOW RECORD commands. For example, the following command displays the history list associated with the field PAYROLL_SALES2. The /ALL qualifier to the SHOW command displays all the information stored for a definition, including the history list.

```
CDO> SHOW FIELD /AUDIT PAYROLL_SALES2
|   |  History entered by MCKAY ([CDD.MCKAY])
|   |          using CDO V1.0
|   |          to CREATE definition on 29-APR-1987 12:32:07.01
|   |          Explanation:
|   |              Used for payroll run with PAYROLL_SALES2.COB
|   |  History entered by MCKAY ([CDD.MCKAY])
|   |          using CDO V1.0
|   |          to MODIFY definition on 30-APR-1987 16:32:06.72
|   |          Explanation:
|   |               Change size to 12
CDO>
```

The following sections provide examples of creating dictionary definitions with the DEFINE command. Many of the command keywords are optional; for clarity, the examples in this manual show all keywords. See the *VAX CDD/Plus Common Dictionary Operator Reference Manual* for syntax diagrams and rules governing the DEFINE command.

4.1.2 Creating Field Definitions

A field definition is the smallest unit that you can store and access in the dictionary. Field definitions can include characteristics, or **field attributes**, ranging from data type information to validation criteria.

The following example shows how to define a simple field with the DEFINE FIELD command. When you enter partial command lines, as shown here, CDO continues to display the cont> prompt until you complete the command with a period.

```
CDO> DEFINE FIELD CDD$TOP.PERSONNEL.FIRST_NAME
cont> DESCRIPTION IS /*Up to 10 characters of first name.*/
cont> DATATYPE IS TEXT
cont> SIZE IS 10.
CDO>
```

If the default was previously set to be the path CDD$TOP.PERSONNEL, the field definition could have been referred to simply as FIRST_NAME, without the path name.

The DATATYPE attribute in the previous definition is an example of a field attribute that you can include in your definition. The SIZE clause is part of the DATATYPE attribute. Table 4-1 lists the possible user-specified field attributes. Complete details of these attributes can be found in the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

Table 4–1: User-Specified Field Attributes

Attribute	Purpose
ARRAY	Declares a field to be an array.
AUDIT	Creates an entry in the history list for a definition.
BASED ON	Permits one field definition to be based on another.
COMPUTED BY	Supplies expressions used to calculate the values of virtual fields.
DATATYPE [IS]	Defines the field data type and, optionally, the size, scale, and number of digits.
DESCRIPTION [IS]	Allows you to add comments about the field definition.

Table 4–1: User-Specified Field Attributes (Cont.)

Attribute	Purpose
[LANGUAGE] EDIT_STRING [IS]	Declares the format used to display the value of a field.[1]For more information, see the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*
INITIAL_VALUE [IS]	Sets a value for a field when it is first allocated.[1]
JUSTIFIED	Justifies character strings.
MISSING_VALUE [IS]	Specifies contents for a field that has never been assigned a meaningful value.[1]
NAME FOR	Declares a language specific name for a field; used by VAX BASIC, COBOL, RPG II, and PL/I.
OCCURS...TIMES	Declares a field to be a one-dimensional array.
QUERY_HEADER [IS]	Specifies a label for report headers.[1]
QUERY_NAME [IS]	Specifies an alternative name for the field.[1]
VALID_IF	Declares an input validation expression.[1]

[1]These attributes are currently product specific; however, they are stored in a generic format and can be shared among products.

The following example defines a field to be a two-dimensional array; the first dimension has a lower bound of 5 and an upper bound of 20, and the second dimension has a lower bound of 100 and an upper bound of 200. (You cannot currently specify an expression for an array bound.) By default, CDD/Plus arrays are row major; however, products that support column major arrays translate the order accordingly.

```
CDO> DEFINE FIELD CDD$TOP.PERSONNEL.MATRIX
cont> DESCRIPTION IS /* 2-dim array, row-major */
cont> DATATYPE IS LONGWORD
cont> SIZE IS 9 DIGITS
cont> ARRAY 5:20 100:200.
CDO>
```

The following field definition includes an INITIAL_VALUE attribute; this allows an initial value to be assigned by an application at run time. Note that some applications cannot support initial values. This field is defined as a one-dimensional array; the number of elements in the array is defined in the OCCURS attribute. An entry is added to the definition's history list with the AUDIT attribute, as shown.

```
CDO> DEFINE FIELD CDD$TOP.PERSONNEL.SIMPLE_LIST
cont> DESCRIPTION IS /* 1-dim array with initial value */
cont> AUDIT /* This is the initial definition entry */
cont> DATATYPE IS TEXT
cont> SIZE IS 1
cont> OCCURS 5 TIMES
cont> INITIAL_VALUE IS "X".
CDO>
```

Using the BASED ON attribute, you can easily create definitions that vary slightly from existing definitions. You can add more attributes to a definition that is based on another, or you can override attributes that were assigned to the original definition.

An important feature of the BASED ON attribute is that if the original definition is changed, the change is automatically reflected in definitions based on it.

In the next example, the field definition BUSINESS_ZIP is created based on the definition of ZIP_CODE. The attributes assigned to the field ZIP_CODE are automatically assigned to the field BUSINESS_ZIP using the BASED ON attribute. The length of the field has been increased, and an EDIT_STRING has been added. Since the field definition ZIP_CODE is in another directory, the full name is specified.

```
CDO> SET DEFAULT [CORPORATE.MIS]PERSONNEL.CONTRACT
CDO> DEFINE FIELD BUSINESS_ZIP
cont>   DESCRIPTION IS /* Based on field ZIP_CODE*/
cont>   BASED ON [CORPORATE.MIS]PERSONNEL.SALARIED.ZIP_CODE
cont>   DATATYPE IS TEXT
cont>   SIZE IS 9
cont>   EDIT_STRING IS 99999"-"9999.
CDO>
```

When you specify the name of a directory in a field or record definition, the directory must already exist; directories are not created implicitly. In the following example, the directory TAXES must be defined before the new field definition is created:

```
CDO> DEFINE DIRECTORY TAXES.
CDO> DEFINE FIELD TAXES.FICA_MAX
cont> DESCRIPTION IS /* Maximum FICA deduction */
cont> DATATYPE IS ZONED NUMERIC
cont> SIZE IS 8.
CDO>
```

The field definitions shown here display only a few of the possible attributes. Table 4-1 lists the available field attributes. For full descriptions of the field attributes, see the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

4.1.3 Creating Record Definitions

You create CDO record definitions with the bottom-up approach. However, for compatibility reasons, the top-down approach is also supported.

4.1.3.1 Bottom-Up Definitions – You create simple record definitions in the dictionary by including previously defined fields. You can then combine field and record definitions into complex record definitions that duplicate any data structure required by a VAX layered product. Alternatively, you can use the shareable CDO field definitions directly in a layered application to build up complex data structures.

The following record definition is created by listing the names of fields that have been previously defined in the same directory.

```
CDO> DEFINE RECORD FULL_NAME.
cont>    FIRST_NAME.
cont>    MIDDLE_INIT.
cont>    LAST_NAME.
cont> END FULL_NAME RECORD.
CDO>
```

The fields and structures included in a record definition must already be defined. You cannot implicitly define entities in CDO dictionaries by including them in other definitions.

You can nest record definitions. In the following example, the record definition FULL_NAME is included in EMPLOYEE_REC:

```
CDO> DEFINE RECORD EMPLOYEE_REC.
cont>    FULL_NAME.
cont>    DEPENDENTS;1.
cont>    WAGE_CLASS.
cont>    START_DATE.
cont> END EMPLOYEE_REC RECORD.
CDO>
```

When you do not specify the version of a definition in a command line, CDO defaults to the highest version that exists at the time of creation. For example, the field DEPENDENTS is specified in the definition of record EMPLOYEE_REC. If a new version of DEPENDENTS is created, you must redefine EMPLOYEE_REC to include the new version of DEPENDENTS. For more information, see Section 4.4.

The VARIANTS clause defines a set of two or more definitions that provide alternative descriptions for the same portion of a record definition. This allows you to map different data types to the same storage location. With VARIANT definitions, you have the option of specifying a tag expression whose value is used at run time to determine the current VARIANT. The following example shows the definition of the

EMPLOYEE_REC record definition with the RETIRED and DISMISSED variants for the field JOB_CODE:

```
CDO> DEFINE RECORD EMPLOYEE_REC.
cont>   FULL_NAME.
cont>   BADGE_NO.
cont>   DEPENDENTS.
cont>   WAGE_CLASS.
cont>   START_DATE.
cont>   JOB_CODE.
cont> VARIANTS.
cont>   VARIANT EXPRESSION IS JOB_CODE IN EMPLOYEE_REC="D".
cont>     DISMISSED.
cont>   END VARIANT.
cont>   VARIANT EXPRESSION IS JOB_CODE IN EMPLOYEE_REC="R".
cont>     RETIRED.
cont>   END VARIANT.
cont> END VARIANTS.
cont> END EMPLOYEE_REC RECORD.
CDO>
```

The previous example shows two variant definitions within one VARIANTS clause. To specify the DISMISSED definition, the user enters "D" in the field JOB_CODE. To specify the RETIRED definition, the user enters "R" in JOB_CODE. You can include as many variants as you want in a record definition and you can repeat the VARIANTS clause as required.

4.1.3.2 Top-Down Definitions – CDO allows you to create record definitions with a **top-down** approach. For example, you can create a record definition and add additional fields to it later.

You can also create arrays that depend on a field in the body of a record. To do this, you create structures. A **structure** is a named group within a record definition. The named structure allows you to refer to a field within the record definition and to use this entity as a dependency, even though it is not a separate dictionary definition. For example, the OCCURS clause in the following record definition of FAMILY depends on another field definition, NUM_OF_KIDS.

```
CDO> DEFINE RECORD FAMILY
cont>   DESCRIPTION IS /*Sample of record structures.*/.
cont>   NUM_OF_KIDS.
cont>   KIDS STRUCTURE
cont>     OCCURS 1 TO 5 TIMES DEPENDING ON NUM_OF_KIDS IN FAMILY.
cont>     NAME.
cont>     AGE.
cont>   END KIDS STRUCTURE.
cont>   BENEFITS STRUCTURE.
cont>     MED_INS.
cont>     DENTAL_INS.
cont>     WORK_STATUS.
cont>   END BENEFITS STRUCTURE.
cont> END FAMILY RECORD.
CDO>
```

Record definition structures can contain complex field and record definitions. You can use other structures to refer to fields within the record definition structure. If you define a record and do not name a structure, you cannot include dictionary entities that have not already been defined. You should use record structures only for compatibility reasons with CDDL definitions or for the particular language you use.

The record definitions shown in this chapter display only a few of the optional clauses and attributes; for complete details, see the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

4.1.4 Creating Relationships

Relationships are created implicitly by CDD/Plus when you explicitly connect CDO definitions in some way. For instance, a record definition relates several field definitions to the record when you include them in the record definition. Similarly, a field definition that specifies an array connects, or relates, the dimensions of the array.

A relationship exists between two CDO definitions. A dictionary definition is an **owner** of a relationship when it uses another definition or *depends on* another definition. A dictionary definition is a **member** of one relationship when it is used by another definition. A relationship has one owner and one member.

For example, the record definition FULL_NAME includes the field definitions FIRST_NAME, MIDDLE_INIT, and LAST_NAME, so FULL_NAME is the *owner* of three relationships. Each of the included fields is a *member* of a relationship with FULL_NAME.

In the following example, the field definition GIVEN_NAME is assigned the same attributes as FIRST_NAME using the BASED ON attribute. This definition implicitly relates the two fields. Any subsequent change to the definition of FIRST_NAME (the member) may affect GIVEN_NAME (the owner); however, a change to GIVEN_NAME will not affect FIRST_NAME.

```
CDO> SET DEFAULT [CORPORATE.MIS]PERSONNEL.CONTRACT
CDO> DEFINE FIELD GIVEN_NAME
cont>   BASED ON [CORPORATE.MIS]PERSONNEL.SALARIED.FIRST_NAME.
CDO>
```

Note

Relationships are created between specific versions of CDO dictionary definitions. When you do not specify the version of a member, CDD/Plus creates the relationship to the highest version of the member at the time the owner is defined. If you subsequently define a new version of an entity, the new version is not included in the previous relationship. The owner must be redefined to specify the new version of the member.

If you change the member of a relationship with the CHANGE command (and therefore do not create a new version), the changed member continues to be related to the owner of the relationship. For more information about changing relationships, see Section 4.4.3.

You can use the SHOW commands to review the relationships that your definitions create. The following SHOW USED_BY command lists the relationship owned by GIVEN_NAME, and the member of that relationship, FIRST_NAME.

```
CDO> SHOW USED_BY GIVEN_NAME
Members of DISK$01:[CORPORATE.MIS]PERSONNEL.GIVEN_NAME;1
| FIRST_NAME                          (Type : FIELD)
| |    via CDD$DATA_ELEMENT_BASED_ON
CDO>
```

The following SHOW FIELD command displays the actual attributes for GIVEN_NAME:

```
CDO> SHOW FIELD GIVEN_NAME
DEFINITION OF FIELD GIVEN_NAME
|     Datatype          text size is 10 characters
CDO>
```

Similarly, in the following example, the record definition FULL_NAME is defined as containing the fields FIRST_NAME, MIDDLE_INIT, and LAST_NAME. The subsequent SHOW RECORD command displays the relationships implicitly created in the definition of FULL_NAME:

```
CDO> DEFINE RECORD FULL_NAME.
cont>    FIRST_NAME.
cont>    MIDDLE_INIT.
cont>    LAST_NAME.
cont> END FULL_NAME RECORD.
```

```
CDO> SHOW RECORD FULL_NAME
Definition of FULL_NAME
| Contains field                  FIRST_NAME
| Contains field                  MIDDLE_INIT
| Contains field                  LAST_NAME
CDO>
```

The following SHOW USES command lists the definitions that use the field FIRST_NAME, and the relationships that the definitions own:

```
CDO> SHOW USES FIRST_NAME
Owners of DISK$01:[CORPORATE.MIS]PERSONNEL.FIRST_NAME;1
| FULL_NAME                         (Type : RECORD)
| |   via CDD$DATA_AGGREGATE_CONTAINS
| GIVEN_NAME                        (Type : FIELD)
| |   via CDD$DATA_ELEMENT_BASED_ON
CDO>
```

For more information about the SHOW commands and how to keep track of changes, see Chapter 6.

BASED ON and AGGREGATE CONTAINS are commonly created relationships. Table 4-2 lists other common relationships that you create implicitly when you create field and record definitions. CDD/Plus stores relationships in addition to those listed.

Table 4–2: CDD/Plus Relationships

Relationship	Description
CDD$DATA_AGGREGATE_CONTAINS	Created when a record definition includes field or record definitions
CDD$DATA_ELEMENT_BASED_ON	Created when one field definition is based on another
CDD$DATA_ELEMENT_COMPUTED_VALUE	Created when an expression in a field definition is dependent on the resolution of the expression at run time

You can relate a definition to a definition in another dictionary on the same node or on a different node in a network. For remote access, CDD/Plus must be installed on each node or VAXcluster.

For example, the field ID_NUM is based on a field in another dictionary on a network. The creator of the field ID_NUM must have SHOW access to the field SOC_SEC on the remote node.

```
CDO> DEFINE FIELD ID_NUM
cont> DESCRIPTION IS /*New ID number same as Social Security number*/
cont> BASED ON FARWAY::SYS$DISK:[TAX_DATA]SOC_SEC.
```

For faster performance, CDD/Plus creates a local copy of remote CDO definitions. When this DEFINE command is executed, CDD/Plus searches for a local copy of the remote field definition SOC_SEC. If no local copy already exists, CDD/Plus creates a local copy. Therefore, when a remote definition is first defined, the network link between the local and remote node must be viable. Section 4.4.3.3 discusses remote access further.

4.2 Supported CDD/Plus Data Types

The design of CDO record definition structures is such that the data structure of any VMS layered product can be duplicated in the dictionary and then used by the layered application. CDD/Plus supports most of the VMS data types.

Valid CDD/Plus data types include:

- character string
- fixed-point
- floating-point
- decimal string classes

Other valid CDD/Plus data types are listed in Table 4–7.

4.2.1 Character String Data Types

The character string data type TEXT represents text in strings of contiguous 8-bit bytes up to a maximum of 65,535 bytes.

4.2.2 Fixed-Point Data Types

Fixed-point data types represent scaled quantities in a binary format. They can be signed or unsigned.

Fixed-point numbers of the data type SIGNED are stored in two's complement form. Values range from $-2^{(n-1)}$ to $2^{(n-1)} - 1$, where n is equal to the number of bits in the data type. Fixed-point numbers of the data type UNSIGNED range from 0 to $2^n - 1$. Table 4–3 shows the fixed-point data types.

Table 4–3: Fixed-Point Data Types

Data Type	Length	Unsigned	Signed
BYTE	8 bits	0 to 255	−128 to 127
WORD	16 bits	0 to 65535	−32768 to 32767
LONGWORD	32 bits	0 to 4,294,967,295	$-2,147,483,648$ to 2,147,483,647
QUADWORD	64 bits	0 to $2^{64} - 1$	-2^{63} to $2^{63} - 1$
OCTAWORD	128 bits	0 to $2^{128} - 1$	-2^{127} to $2^{127} - 1$

4.2.3 Floating-Point Data Types

Floating-point data types represent approximations to quantities in a scientific notation consisting of a signed exponent and a mantissa. The floating-point data types are shown in Table 4–4.

Table 4–4: Floating-Point Data Types

Data Type	Length	Approximate Precision	Approximate Range
F_FLOATING	32 bits	7 decimal digits	$\pm 10^{-38}$ to 10^{38}
D_FLOATING	64 bits	16 decimal digits	$\pm 10^{-38}$ to 10^{38}
G_FLOATING	64 bits	15 decimal digits	$\pm 10^{-308}$ to 10^{308}
H_FLOATING	128 bits	33 decimal digits	$\pm 10^{-4932}$ to 10^{4932}

4.2.4 Complex Numbers

Complex numbers specify ordered pairs of floating-point data types representing the real and imaginary components of a number. Complex numbers are shown in Table 4-5.

Table 4–5: Complex Numbers

Data Type	Total Length	Approximate Precision of Each Part	Approximate Range of Each Part
F_FLOATING COMPLEX	64 bits	7 decimal digits	$\pm 10^{-38}$ to 10^{38}
D_FLOATING COMPLEX	128 bits	16 decimal digits	$\pm 10^{-38}$ to 10^{38}
G_FLOATING COMPLEX	128 bits	15 decimal digits	$\pm 10^{-308}$ to 10^{308}
H_FLOATING COMPLEX	256 bits	33 decimal digits	$\pm 10^{-4932}$ to 10^{4932}

4.2.5 Decimal String Data Types

The decimal string data types represent fixed scale quantities. They are efficient in applications that generate numerous reports and listings.

There are two classes of decimal string data types. Those in which each decimal digit occupies one 8-bit byte are called NUMERIC data types. In the more compact form called PACKED DECIMAL, two decimal digits occupy each byte. The decimal string data types are shown in Table 4-6.

Table 4–6: Decimal String Data Types

Data Type	Description
UNSIGNED NUMERIC	An unsigned numeric ASCII string.
LEFT SEPARATE NUMERIC	A signed numeric ASCII string; the leftmost byte contains the sign.
LEFT OVERPUNCHED NUMERIC	A signed numeric ASCII string; the sign and the leftmost digit occupy the same byte.
RIGHT SEPARATE NUMERIC	A signed numeric ASCII string; the rightmost byte contains the sign.
RIGHT OVERPUNCHED NUMERIC	A signed numeric ASCII string; the sign and the rightmost digit occupy the same byte.
ZONED NUMERIC	The VAX ZONED NUMERIC data type; this signed numeric ASCII string is similar to the RIGHT OVERPUNCHED NUMERIC, but the sign codes differ.
PACKED DECIMAL	A signed numeric ASCII string; two digits occupy each byte, and the low half of the last byte is reserved for the sign.

4.2.6 Other Data Types

CDD/Plus can also assign the data types listed in Table 4-7.

Table 4–7: Other Data Types

Data Type	Description
ALPHABETIC	Specifies a text string.
BIT	Specifies a string of contiguous bits.
DATE	Specifies a VMS standard 64-bit absolute date data type.
UNSPECIFIED	Sets aside a specified number of contiguous unsigned 8-bit bytes.
SEGMENTED STRING	Specifies a segmented string for DSRI compliant products.
VARYING STRING	Specifies a PL/I varying string class field.
POINTER	Specifies a field containing the address of another field or buffer.

4.2.7 VAX Information Architecture Support for CDD/Plus Data Types

Table 4–8 shows the VAX Information Architecture products support for the CDD/Plus data types. The following list provides a key to the symbols used in Table 4–8.

Key to Table 4–8

S	Indicates that the facility fully supports the data type.
W	Indicates that the facility translates the data type into one that is supported and issues diagnostics. In some cases, this support may not be usable.
X	Indicates a data type that DATATRIEVE can use but cannot define.
U	Indicates that the data type is unsupported and that the facility issues a fatal diagnostic.
E	Indicates that the data type is unsupported and that the facility issues a non-fatal error.
I	Indicates that the data type is ignored and that the facility issues no diagnostic messages.
R	Indicates that the facility can make use of an unsupported data type only to pass its address as a parameter.

Table 4–8: VAX Information Architecture Products Support for CDD/Plus Data Types

Data Type	DTR	DBMS	ACMS	Rdb /VMS	TDMS
UNSPECIFIED	S	S	S	U	S
SIGNED BYTE	S	S	R	U	S
UNSIGNED BYTE	X	S	R	U	S
SIGNED WORD	S	S	R	S	S
UNSIGNED WORD	X	S	R	U	S
SIGNED LONGWORD	S	S	S	S	S
UNSIGNED LONGWORD	X	S	R	U	S
SIGNED QUADWORD	S	S	R	S	S
UNSIGNED QUADWORD	X	S	R	U	S
SIGNED OCTAWORD	U	S	R	U	U
UNSIGNED OCTAWORD	U	S	R	U	U
F_FLOATING	S	S	R	S	S
F_FLOATING COMPLEX	U	S	R	U	U
D_FLOATING	S	S	R	U	S
D_FLOATING COMPLEX	U	S	R	U	U

(Continued on next page)

Table 4–8: VAX Information Architecture Products Support for CDD/Plus Data Types (Cont.)

Data Type	DTR	DBMS	ACMS	Rdb /VMS	TDMS
G_FLOATING	U	S	R	S	S
G_FLOATING COMPLEX	U	S	R	U	U
H_FLOATING	S	S	R	U	S
H_FLOATING COMPLEX	U	S	R	U	U
UNSIGNED NUMERIC	S	S	R	U	S
LEFT OVERPUNCHED NUMERIC	S	S	R	U	S
LEFT SEPARATE NUMERIC	S	S	R	U	S
RIGHT OVERPUNCHED NUMERIC	S	S	R	U	S
RIGHT SEPARATE NUMERIC	S	S	R	U	S
PACKED DECIMAL	S	S	R	U	S
ZONED NUMERIC	S	S	R	U	S
BIT	U	U	R	U	U
DATE	S	S	R	S	S
TEXT	S	S	S	S	S
VARYING STRING	U	U	U	S	S
POINTER	U	U	U	U	U
VIRTUAL FIELD	S	U	U	U	E
SEGMENTED STRING	U	U	U	S	U

4.2.8 VAX Language Support for CDD/Plus Data Types

CDD/Plus supports fixed-point, floating-point, character string, and decimal string data types. Not all of these types are supported by the VAX languages. The languages that currently support DMU dictionaries include:

- VAX BASIC
- VAX C
- VAX COBOL
- VAX DIBOL

- VAX FORTRAN
- VAX PASCAL
- VAX PL/I
- VAX RPG II

For more information about using CDD/Plus with VAX languages, see Chapter 1 and the documentation for the particular language. When you intend definitions to be shared by more than one language, be sure to use data types that are equally supported by each language. Table 4-9 lists the VAX COBOL, BASIC, DIBOL, and PL/I support for CDD/Plus data types; Table 4-10 lists the VAX PASCAL, FORTRAN, C, and RPG II support. The symbols used in Table 4-9 and Table 4-10 are the same as the symbols used in Table 4-8 except that the symbol X is not used.

Table 4–9: VAX BASIC, COBOL, DIBOL, and PL/I Support for CDD/Plus Data Types

Data Type	Language			
	COBOL	BASIC	DIBOL	PL/I
UNSPECIFIED	W	W	W	R
SIGNED BYTE	W	S	S	S
UNSIGNED BYTE	W	W	S	R
SIGNED WORD	S	S	S	S
UNSIGNED WORD	W	W	S	R
SIGNED LONGWORD	S	S	S	S
UNSIGNED LONGWORD	W	W	S	R
SIGNED QUADWORD	S	W	R	R
UNSIGNED QUADWORD	W	W	R	R
SIGNED OCTAWORD	W	W	R	R
UNSIGNED OCTAWORD	W	W	R	R
F_FLOATING	S	S	W	S
F_FLOATING COMPLEX	W	W	W	R
D_FLOATING	S	S	W	S

(Continued on next page)

Table 4–9: VAX BASIC, COBOL, DIBOL, and PL/I Support for CDD/Plus Data Types (Cont.)

Data Type	Language			
	COBOL	BASIC	DIBOL	PL/I
D_FLOATING COMPLEX	W	W	W	R
G_FLOATING	W	S	W	W
G_FLOATING COMPLEX	W	W	W	R
H_FLOATING	W	S	W	S
H_FLOATING COMPLEX	W	W	W	R
UNSIGNED NUMERIC	S	W	S	S
LEFT OVERPUNCHED NUMERIC	S	W	W	S
LEFT SEPARATE NUMERIC	S	W	W	S
RIGHT OVERPUNCHED NUMERIC	S	W	W	S
RIGHT SEPARATE NUMERIC	S	W	W	S
PACKED DECIMAL	S	S	S	S
ZONED NUMERIC	W	W	S	R
BIT	W	W	W	S
DATE	W	W	W	R
TEXT	S	S	S	S
VARYING STRING	W	W	W	S
POINTER	S	W	W	S
VIRTUAL FIELD	I	I	W	I
SEGMENTED STRING	W	U	U	U

Table 4–10: VAX FORTRAN, C, PASCAL, and RPG II Support for CDD /Plus Data Types

Data Type	Language			
	C	FORTRAN	PASCAL	RPG II
UNSPECIFIED	R	R	R	W
SIGNED BYTE	S	S	S	W
UNSIGNED BYTE	S	R	S	W
SIGNED WORD	S	S	S	S
UNSIGNED WORD	S	R	S	W
SIGNED LONGWORD	S	S	S	S
UNSIGNED LONGWORD	S	R	S	W
SIGNED QUADWORD	R	R	R	W
UNSIGNED QUADWORD	R	R	R	W
SIGNED OCTAWORD	R	R	R	W
UNSIGNED OCTAWORD	R	R	R	W
F_FLOATING	S	S	S	W
F_FLOATING COMPLEX	R	S	R	W
D_FLOATING	S	S	S	W
D_FLOATING COMPLEX	R	S	R	W
G_FLOATING	S	S	S	W
G_FLOATING COMPLEX	R	S	R	W
H_FLOATING	R	S	S	W
H_FLOATING COMPLEX	R	R	R	W
UNSIGNED NUMERIC	R	R	R	W
LEFT OVERPUNCHED NUMERIC	R	R	R	W
LEFT SEPARATE NUMERIC	R	R	R	W
RIGHT OVERPUNCHED NUMERIC	R	R	R	S

(Continued on next page)

Table 4–10: VAX FORTRAN, C, PASCAL, and RPG II Support for CDD/Plus Data Types (Cont.)

Data Type	Language			
	C	FORTRAN	PASCAL	RPG II
RIGHT SEPARATE NUMERIC	R	R	R	W
PACKED DECIMAL	R	R	R	S
ZONED NUMERIC	R	R	R	W
BIT	S[1]	R	S[1]	W
DATE	R	R	R	W
TEXT	S	S	S	S
VARYING STRING	R	R	S	W
POINTER	S	R	S	W
VIRTUAL FIELD	I	I	I	I
SEGMENTED STRING	U	U	U	U

[1]For bit fields that exceed 32 bits, see the individual language documentation.

4.3 Copying Dictionary Definitions

You can copy definitions when you need duplicates for production or testing. The COPY command allows you to copy a specified definition and the relationships it owns. You must supply the name of the definition to be copied and the name of the target.

For example, the following COPY command copies the record definition EMPLOYEE_REC from the SALARIED directory to the CONTRACT directory.

```
CDO> COPY CDD$TOP.PERSONNEL.SALARIED.EMPLOYEE_REC
cont> CDD$TOP.PERSONNEL.CONTRACT.EMPLOYEE_REC
CDO>
```

Wildcards are valid for both the source and the target. In the following example, the COPY command copies all the definitions from the PERSONNEL directory to the SALARIED directory.

```
CDO> DIRECTORY
 Directory DISK$01:[CORPORATE.MIS]PERSONNEL.SALARIED
ADDRESS_DATA_1;1                           FIELD
DEPARTMENTS;1                              RECORD
EMPLOYEE_REC;1                             RECORD
CDO>
CDO> COPY [CORPORATE.MIS]PERSONNEL.*
cont> [CORPORATE.MIS]PERSONNEL.SALARIED.*
CDO> DIRECTORY
 Directory DISK$01:[CORPORATE.MIS]PERSONNEL.SALARIED
ADDRESS_DATA_1;1                           FIELD
COLLEGE_NAME;1                             FIELD
DEPARTMENTS;1                              RECORD
DEPARTMENT_NAME;1                          FIELD
EMPLOYEE_REC;1                             RECORD
FIRST_NAME;1                               FIELD
FULL_NAME;1                                RECORD
LAST_NAME;1                                FIELD
STATUS_NAME;1                              FIELD
```

If a dictionary directory does not exist, you can implicitly create it when you copy dictionary definitions. For example, the following command copies all definitions in the directory SALARIED and the directories beneath it to the directory SALARIED.MAIN. The command implicitly recreates all directories beneath the directory SALARIED. However, the target directory, MAIN, must exist before you issue the COPY command.

```
CDO> COPY [CORPORATE.MIS]PERSONNEL.SALARIED...
cont> [CORPORATE.MIS]PERSONNEL.SALARIED.MAIN...
CDO>
```

When you copy a dictionary definition, no entry is added to the history list auditing the copy operation. However, the existing history list is copied to the new definition.

The COPY command can be used to copy a definition from one CDO dictionary to another. (To copy a DMU record definition to CDO format, use the CONVERT command, as described in Chapter 1.) The COPY command is valid across a network. When more than one version of the specified definition exists, CDD/Plus copies all versions unless you specify an alternative with the COPY command. When you use the COPY command to create another definition with the same name in the same directory, CDD/Plus creates a new version of the definition.

If a subsequent change is made to the original EMPLOYEE_REC in the directory SALARIED, this change is not reflected in the copied record definition in the directory SALARIED.MAIN.

4.3.1 Copying Relationships

The COPY command preserves all relationships. When both the owner and the member of a relationship are copied, the new relationships are between the new copies of both. Consider, for example, the relationship between GIVEN_NAME and FIRST_NAME. GIVEN_NAME is based on FIRST_NAME; therefore, GIVEN_NAME is the owner of the relationship and FIRST_NAME is the member. If FIRST_NAME and GIVEN_NAME are both copied, a new relationship is created between the new copies of both.

When an owner is copied but the member is not, the new relationship is from the new owner to the old member. For example, if GIVEN_NAME is copied but FIRST_NAME is not, the new copy of GIVEN_NAME has the original definition of FIRST_NAME for a member.

When a member is copied but the owner is not, no new relationship is created because no definitions own the copied member. For example, if FIRST_NAME is copied but GIVEN_NAME is not, no definition uses the copy of FIRST_NAME. The old relationship remains intact.

4.3.2 Copying a Complete Dictionary

To copy a complete dictionary, you must be aware of potential problems related to changing the anchor. External references to the dictionary must be changed to accommodate the new anchor. The MOVE DICTIONARY command changes the external references for you. For more information, see Chapter 6.

4.4 Changing Dictionary Definitions

There are two ways to change field and record definitions:

- You can create new versions of your definitions and phase in the change over time.
- You can change original definitions and cause immediate updates.

It is good practice to document all changes. You can include remarks with the DESCRIPTION and the AUDIT attributes for both the CHANGE and the DEFINE commands, as described in Section 4.1.1.

When you change a definition, dictionary definitions that do not automatically include the change are flagged with a message to warn the user about the change. When you create a new version with the DEFINE command, no dictionary definitions include the new version automatically; therefore, all dependent dictionary definitions are flagged with a message about the new version. When you change a dictionary definition with the CHANGE command, related record and field definitions automatically include the change, only external entities that do not include the change automatically (such as databases) are flagged with a message.

The following two sections discuss the differences between making changes with the CHANGE and DEFINE commands.

4.4.1 Creating New Versions of Definitions with the DEFINE Command

CDD/Plus allows you to store several versions of the same definition. You need DEFINE access rights to an existing definition to create new versions of it.

When you store several versions of the same definition, programs and other definitions can specify and access any of these versions. When you want to phase in a change over a period of time and continue to allow access to the original definition, you should create a new version of the definition with the DEFINE command rather than use the CHANGE command. Creating new versions, rather than changing the original definition, leaves an audit trail of changes.

When you define a record and include a field without specifying the field version number, CDD/Plus includes the highest version available at the time the record is defined. If you create a new version of the field definition with the DEFINE command, the record continues to include the previous version until you redefine the record. Relationships exist between specific versions of definitions and do not automatically change to accommodate new versions. For more information, see Section 4.4.3.

To redefine a record definition that uses an outdated version so that it will use the new version, you can use either of the following methods:

- Use the CDO editor
- Use the CDO EXTRACT command to display the record definition, then modify the definition using CHANGE or DEFINE commands

When you create a new version of a definition, all dictionary definitions that use the previous version of the definition are flagged with a message. For example, if you create a new version of the field definition BADGE_NO, the new version is not included in the record EMPLOYEE_REC and not automatically used by the Rdb/VMS database DEPT1. DEPT1 is flagged with a message that warns the user that a new version of BADGE_NO exists. Users can optionally take advantage of the new version of BADGE_NO. Consider also that EMPLOYEE_REC is used by another supporting product. If the product is attached to EMPLOYEE_REC, CDD/Plus attaches a message to that product's dictionary definition, to warn the user that a new version exists.

Chapter 6 includes more information about analyzing the impact of changes and tracking dictionary usage.

You need DEFINE access to create new versions of a definition. By default, only the original creator of a definition has the right to make new versions; however, the default protection can be changed by a user with CONTROL access. See Chapter 5 for more information about protection.

4.4.2 Changing Original Definitions with the CHANGE Command

The CHANGE command changes the original definition and does not create a new version of a definition. After you change a definition with the CHANGE command, you can no longer access the original definition at compilation time. If you change the original version of a field definition with the CHANGE command, a record that includes the field automatically includes the changes.

4.4.2.1 Receiving Messages About Changes – CDO sends messages when a dictionary definition changes or when a new version of a definition is created. A CDO **message** signals that a dictionary definition that is a member of a relationship has been changed. Messages can be sent from a member of a relationship to:

- The owner of the relationship
- The owner's ancestors
- Both the owner and the owner's ancestors

To find out which definitions will be flagged with a message as a result of a change in place, use the SHOW WHAT_IF command discussed in Chapter 6. Chapter 6 also provides information about accessing these messages from the CDO environment and from other products, as well as information about analyzing dictionary usage.

You can also determine which dictionary definitions receive messages by examining the CDD$MESSAGE_ACTION attribute. *VAX CDD/Plus Common Dictionary Operator Reference Manual* explains how to use the SHOW PROTOCOL command to examine the CDD$MESSAGE_ACTION attribute, and tells which CDD$MESSAGE_ACTION attributes cause messages.

When a field or record definition is changed with the CHANGE command, definitions that include it automatically include the change. Only dictionary definitions that do *not* automatically include the changed definition are flagged with a message.

For example, if the field definition BADGE_NO is changed with the CHANGE command, the record definition EMPLOYEE_REC automatically includes the change to BADGE_NO; therefore, no message is signaled on EMPLOYEE_REC. However, the copy of BADGE_NO contained in the database DEPT1 is now inconsistent with the BADGE_NO in the dictionary; so the database is flagged with a message about this.

```
CDO> CHANGE FIELD BADGE_NO
cont> DATATYPE IS TEXT
cont> SIZE IS 8.
%CDO-I-DBMBR, database SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1;1
may need to be INTEGRATED
CDO> SHOW MESSAGES DEPT1
SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1;1 is possibly invalid,
triggered by CDD$DATA_ELEMENT SYS$COMMON:[CDDPLUS]PERSONNEL.BADGE_NO;3
```

The message warns the database administrator that the definition of BADGE_NO is now inconsistent with the database copy and with the definition of BADGE_NO that is included in EMPLOYEE_REC.

With the CHANGE command, you can make changes to dictionary definitions and protection schemes. For details on how to change the protection scheme for dictionary definitions, see Chapter 5. For information about changing the instances of protocols, see the command in the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

The following sections provide examples of how to change field and record definitions with the CHANGE command and how relationships are affected by such changes. Remember that the CHANGE command, like DEFINE and DELETE, requires a terminating period.

You need CHANGE access rights to dictionary definitions in order to change them. By default, only the creator of a definition has the right to change it; however, the default protection can be changed by a user with CONTROL access.

To find out if you have CHANGE access rights, use the SHOW PROTECTION command or the SHOW PRIVILEGES command, as shown in the following example. The SHOW PROTECTION command displays the access control list for the specified definition; the SHOW PRIVILEGES command displays your current privileges for the definition. You would see the following SHOW PRIVILEGES output only if you were a member of group 20.

```
CDO> SHOW PROTECTION FOR FIELD RATE

Directory DISK$01:[CORPORATE.MIS]PERSONNEL.RATE
```

```
RATE;1
          (IDENTIFIER=[VDD,DICTIONARY],ACCESS=READ+
          WRITE+MODIFY+ERASE+SHOW+DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+
          ADMINISTRATOR)
          (IDENTIFIER=[VDD,DICTIONARY],ACCESS=READ+WRITE+MODIFY+ERASE
          +SHOW+DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
          (IDENTIFIER=[20,*],ACCESS=READ+WRITE+MODIFY+ERASE+
          SHOW+DEFINE+CHANGE+DELETE)
          (IDENTIFIER=[SECRETARIES,*],ACCESS=SHOW)
          (IDENTIFIER=[SALES,JONES],ACCESS=NONE)

CDO> SHOW PRIVILEGE FOR FIELD RATE

Directory DISK$01:[CORPORATE.MIS]PERSONNEL.RATE

RATE;1
          (IDENTIFIER=[20,*],ACCESS=READ+WRITE+MODIFY+ERASE+
          SHOW+DEFINE+CHANGE+DELETE)
CDO>
```

Chapter 5 discusses access rights in detail. The following sections show you how to make changes to field and record definitions with the CHANGE command.

4.4.2.2 Changing Field Definitions —In Section 4.1.2, the field SIMPLE_LIST was given an initial value. In the following example, the definition of SIMPLE_LIST is changed so that the definition no longer has an INITIAL VALUE attribute.

```
CDO> CHANGE FIELD SIMPLE_LIST
cont> AUDIT /* Removing initial value attribute */
cont>   NOINITIAL_VALUE.
CDO>
```

You can change attributes such as the data type with the CHANGE command. For example, the following command changes the data type of the previously defined field MATRIX from a longword to a floating-point value.

```
CDO> CHANGE FIELD MATRIX
cont>  AUDIT /* Change datatype to f_floating */
cont>  DATATYPE IS F_FLOATING.
CDO>
```

This command changes only the data type *of the defined field*, not the data type of the stored data.

The following example changes the array created in the field SIMPLE_LIST from a five-element array to a ten-element array:

```
CDO> CHANGE FIELD SIMPLE_LIST
cont>  AUDIT /* Changed from 5 to 10 array elements */
cont>   OCCURS 10 TIMES.
CDO>
```

When a field is included in a record definition, the record automatically includes the changed field definition if the change is made with the CHANGE command. Remember that this is not the case if new versions are created with the DEFINE command or with the CDO editor.

Caution

In languages that must be compiled, if you do not recompile a program after a dictionary definition has been changed, the program continues to use the executable code representing the original definition.

The CHANGE command allows you to change attributes such as missing values, special names for languages, query headers, and so on. For information on the field attributes that you can change, see the CHANGE FIELD command description in the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

4.4.2.3 Changing Record Definitions – Record definitions can also be modified with the CHANGE command. The following examples show some of the changes you can make to record definitions; for more information about the changes you can make, see the CHANGE RECORD command description in the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

The following example deletes a field from the record definition EMPLOYEE_REC. Other field or record definitions related to EMPLOYEE_REC remain unchanged.

```
CDO> CHANGE RECORD EMPLOYEE_REC
cont> AUDIT /* Removing DEPENDENTS field */.
cont>  DELETE DEPENDENTS.
cont> END EMPLOYEE_REC RECORD.
CDO>
```

To include an additional field in a record definition, use the CHANGE command with the DEFINE record attribute. The following example adds the field WAGE_STATUS to the record definition EMPLOYEE_REC. The included field becomes the last field in the record definition.

```
CDO> CHANGE RECORD EMPLOYEE_REC
cont> /* Adding new fields WAGE_STATUS and CLASS_CODE */.
cont>   DEFINE WAGE_STATUS.
cont>   END DEFINE.
cont>   DEFINE CLASS_CODE.
cont>   END DEFINE.
cont> END EMPLOYEE_REC RECORD.
CDO>
```

Although you use the keyword DEFINE to specify the new field WAGE_STATUS, this field must be already defined in the dictionary. You cannot implicitly create a field definition by specifying a new field in a CHANGE RECORD definition.

To make changes to the VARIANTS clause in a record definition, you must identify the variant by listing all the variants in the VARIANTS clause. The variant fields that you do not want to change need not be named, but all variant groups must be listed so that CDD/Plus can identify the variant you are changing by position. The following example defines the field RATE in the second variant in the VARIANTS clause of the record definition EMPLOYEE_REC; none of the other variants are changed.

```
CDO> CHANGE RECORD EMPLOYEE_REC.
cont>   VARIANTS.
cont>      VARIANT.
cont>      END VARIANT.
cont>      VARIANT.
cont>        DEFINE RATE.
cont>        END DEFINE.
cont>      END VARIANT.
cont>   END VARIANTS.
cont> END EMPLOYEE_REC RECORD.
CDO>
```

4.4.3 Changing Relationships

When two definitions are related, the relationship exists between the owner and the specified version of the member. When no version of the member is specified, the owner is related to the highest version of the member at the time the owner is defined.

The following two sections discuss the differences between changing members of relationships with the CHANGE command and the DEFINE command. Remember that when you redefine entities with the CDO editor, you define new versions as with the DEFINE command.

4.4.3.1 Changing a Member with the CHANGE Command

– Consider the record definition FULL_NAME. FULL_NAME includes the field definitions FIRST_NAME, MIDDLE_INIT, and LAST_NAME. FULL_NAME is the owner of three relationships, one relationship between each of the field definitions FULL_NAME includes. Each of the included fields are members of a relationship with FULL_NAME.

No versions were specified in the original definition of FULL_NAME:

```
CDO> DEFINE RECORD FULL_NAME.
cont>    FIRST_NAME.
cont>    MIDDLE_INIT.
cont>    LAST_NAME.
cont> END FULL_NAME RECORD.
```

Suppose that you want to change MIDDLE_INIT to increase the size to be three characters (to allow for the convention NMN for "no middle name"). If you change the original definition with the CHANGE command; therefore, you do not create a new version of the definition. The relationship continues to exist between FULL_NAME and the changed version of MIDDLE_INIT.

The size of MIDDLE_INIT is now increased by two additional characters. This change in size is incorporated automatically by all definitions that use MIDDLE_INIT. No message is attached to FULL_NAME or any other field or record definitions that automatically include the change.

When copies of dictionary definitions are maintained in a database, the database copies are not automatically updated after a change in the dictionary; therefore, database descriptions in the dictionary are flagged with a message about the inconsistency.

4.4.3.2 Changing a Member with the DEFINE Command

– Consider again the record definition FULL_NAME, which uses the field definitions FIRST_NAME, MIDDLE_INIT, and LAST_NAME. Because no versions were specified in the original definition of FULL_NAME, the relationships were established with the highest version of each of the members at the time FULL_NAME was defined.

Suppose that you decide to change MIDDLE_INIT to increase the size to three characters, but you choose to redefine the definition with the DEFINE command. The DEFINE command creates a new version of the definition, leaving the original version intact. The record FULL_NAME continues to be the owner of the relationship between FULL_NAME and MIDDLE_INIT;1 because version 1 was the highest version of MIDDLE_INIT when FULL_NAME was defined.

New versions of definitions are not automatically included in the definitions that use them. The newly created MIDDLE_INIT;2 is not related to FULL_NAME;1. The owner of a relationship must be changed or redefined to include the new version of the member.

If you redefine FULL_NAME with the DEFINE command and do not specify a version number for the members, FULL_NAME;2 will include the highest current version of MIDDLE_INIT, which is MIDDLE_INIT;2. Because new versions of definitions are not automatically reflected in the definitions that use them, a message that a new version exists is attached to each user of the previous version.

Relationships are often nested. For example, the record definition FULL_NAME is itself a member of a relationship with EMPLOYEE_NAME. You can list all dependent uses of a definition with either the /FULL or the /ALL qualifier to the SHOW USED_BY command, as discussed in Chapter 6.

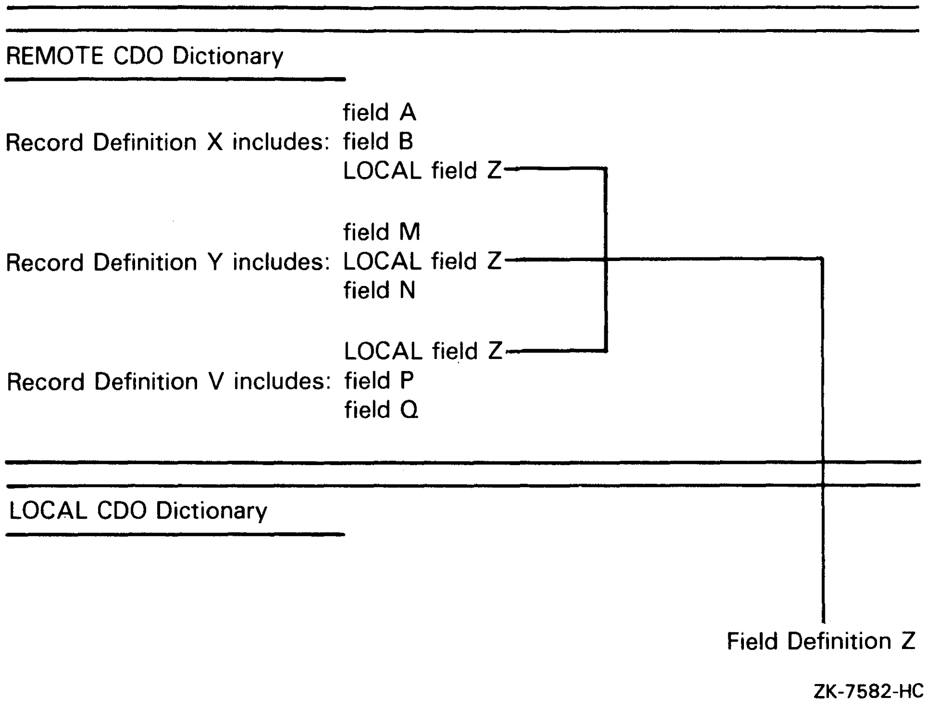
4.4.3.3 Changing Relationships in a Network —Figure 4-1 illustrates how CDD/Plus uses local copies for faster performance when dictionary definitions are related via a network.

When a local definition is changed with the CHANGE command, CDD/Plus automatically updates all remote copies. If all remote dictionaries containing copies of the definition are not available, the change is not allowed and CDD/Plus issues an error message. However, when a new version of the definition is created with the DEFINE command, the local copy is not changed. Instead, users of the definition are flagged with a message that the new version exists.

For example, if a user changes version 1 of field definition Z on the local node, CDD/Plus updates the changed definition on the remote node if the network link is viable. As the changed field is automatically included in the record definitions X, Y, and Z, these definitions are *not* flagged with a warning message about the change.

On the other hand, when a new version of a local definition is created with the DEFINE command, the copy on the remote node remains unchanged. For example, if a user creates version 2 of field definition Z on the local node, version 1 of field definition Z remains the only copy on the remote node. To update the relationship, you must redefine record definitions X, Y, and V on the remote node.

Figure 4–1: Creating Local Copies of Remote Dictionary Definitions



4.5 Deleting and Purging Dictionary Definitions

You need DELETE access to dictionary definitions to purge or delete them. By default, only the owner of a definition has the right to delete it; however, the default protection can be changed by a user with CONTROL access.

The ERASE access right is associated only with the protocol definition; only DELETE access allows you to delete or purge definitions.

To delete or purge definitions on a remote CDO dictionary, the network link between the local and the remote node must be viable.

4.5.1 Deleting Dictionary Definitions

When you are aware that a particular definition is not used at all, delete it with the DELETE command. For example, the following command deletes all versions of the field definition GIVEN_NAME from the dictionary.

```
CDO> DELETE FIELD GIVEN_NAME;*.
CDO>
```

You can supply several definition names with a single DELETE command, provided that the definitions are all the same type. For example, you can delete several fields or several records at the same time, but you cannot mix fields and records in the same DELETE command.

```
CDO> DELETE FIELD GIVEN_NAME,FIRST_NAME.
CDO>
```

You can delete definitions that are members of a relationship with the specified entity. For example, the following command uses the /DESCENDANTS qualifier to delete the record definition FAMILY and also delete all definitions that the record uses. If a member of a relationship to FAMILY is also a member of a relationship with another definition, CDD/Plus does not delete the descendant.

```
CDO> DELETE RECORD /DESCENDANTS /LOG FAMILY;*.
%CDO-I-ENTDELDESC, entity DISK$01:[CORPORATE.MIS]PERSONNEL.FAMILY;1 and its descendants
                   were deleted
CDO>
```

The default, DELETE/NODESCENDANTS, deletes only the specified definition.

You can delete a dictionary directory, provided that the directory is empty. The following command deletes the empty directory BENEFITS.

```
CDO> DELETE DIRECTORY [CORPORATE.MIS]BENEFITS.
CDO>
```

The DELETE DICTIONARY command allows you to delete a dictionary. To delete a complete dictionary, first make sure that the dictionary definitions are not used by entities external to the dictionary, such as databases or other dictionaries. If necessary, copy definitions to another dictionary and change the anchor and path to the definitions where appropriate.

You must have DELETE privilege for all of the definitions contained in the dictionary and CONTROL privilege to the dictionary files.

Caution

When CDD/Plus deletes a complete dictionary, all files in the anchor directory are deleted. DIGITAL recommends that you store only files created by CDD/Plus in a VMS directory that is dedicated to a dictionary. If you do store other files in the VMS directory, these files are deleted when you delete the dictionary.

The DELETE command also allows you to delete protection schemes (see Chapter 5). For a full list of qualifiers and other details on the DELETE command, see the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

4.5.2 Purging Dictionary Definitions

When you have created more than one version of a definition, you can purge earlier versions with the CDO command PURGE, which deletes all but the most recent version of the specified definition by default. You *cannot* purge a definition that is *a member of a relationship*, such as a field in a database.

The following command deletes all but the latest version of the field definition SALARY_MAX.

```
CDO> PURGE FIELD SALARY_MAX.
CDO>
```

The PURGE command provides two optional qualifiers: /KEEP and /[NO]DESCENDANTS. The /KEEP qualifier allows you to specify the number of versions to be kept after the purge. The /DESCENDANTS qualifier also purges members of relationships owned by the purged definition unless they are related to other definitions. By default, related definitions are not purged (/NODESCENDANTS).

The following command purges all but the highest two versions of the record definition EMPLOYEE_REC. All but the highest two versions of definitions that are related to EMPLOYEE_REC are also purged, unless these definitions are already related to others.

```
CDO> PURGE RECORD /KEEP=2/DESCENDANTS EMPLOYEE_REC.
CDO>
```

You can use wildcards with the PURGE command. The following command purges all of the fields that are not related to others in the current default directory.

```
CDO> PURGE FIELD *.
CDO>
```

Protecting Your Dictionary 5

This chapter shows you how to specify which users can access the definitions in your CDO dictionary. It explains users' access control rights and describes how to grant and deny these rights.

5.1 Controlling Dictionary Access

VMS security protects devices, directories, and files; individual definitions stored within the dictionary files are protected by CDO security provisions.

CDO dictionary security is based on the VMS style of security. Access to dictionary objects is controlled by access control lists that explicitly specify the access that a particular user is allowed.

5.1.1 User Identifiers

The basic component of the VMS protection scheme is an identifier. Identifiers can be User Identification Codes (UICs) or general identifiers:

- UICs are used to classify processes and groups of processes on VMS systems. The system manager assigns a UIC to each system user with the VMS Authorize Utility.

[group,member]	specifies a user's group and member
[group,*]	specifies any member of the named group
[*,member]	specifies a named member regardless of the group

- General identifiers can be system- or user-defined. They can identify users from different UIC groups, a single user, a group of users, a single UIC group, or a combination of these. For example, WRITERS might combine users from different UIC groups.

  System identifiers often correspond to an environment. For example, the identifier BATCH addresses all attempts at access made by batch jobs. Other system identifiers include: INTERACTIVE, NETWORK, LOCAL, REMOTE, and DIALUP.

The following are examples of valid identifiers:

```
[360,12]
[SALES,MARY]
[SALES,*]
[EXECUTIVES,MCKAY]
[BATCH]
```

To see identifiers for your current process, use the DCL command SHOW PROCESS, as in the following example:

```
$ SHOW PROCESS
 8-FEB-1988 16:26:38.09   RTA3:            User: LUFKIN
Pid: 20600A3F   Proc. name: _RTA3:         UIC: [CDD,LUFKIN]
Priority:   6   Default file spec: DISK$1:[LUFKIN]
Devices allocated: RTA3:
```

5.1.2 Access Control Entries

An access control list (ACL) allows you to define what types of access, if any, should be granted to the potential users of system entities. An ACL consists of a list of access control entries (ACEs), each specifying an identifier and the associated access rights. The order of the entries in the list allows VMS to determine access priorities; therefore, you should take care when ordering the entries in your ACLs.

To see the ACLs for one of your existing definitions, use the SHOW /ALL command, as in the following example:

```
CDO> SHOW PROTECTION FOR FIELD FIRST_NAME

 Directory SYS$COMMON:[CDDPLUS]PERSONNEL

FIRST_NAME;1
        (IDENTIFIER=[CDD,CDDPLUS],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+
        DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
        (IDENTIFIER=[*,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+OPERATOR+
        ADMINISTRATOR)
```

Table 5-1 lists the VMS access rights you can specify in an ACL for a file in a VMS directory.

Table 5–1: VMS Access Rights

Privilege	Access Permitted
READ	Grants the right to examine, print, or copy files
WRITE	Grants the right to modify files and create new files
EXECUTE	Grants the right to execute a file that contains an executable image; permits wildcard searches and matching, as with the DIRECTORY command
DELETE	Grants the right to delete a file
CONTROL	Grants the right to create, change, and delete ACLs

All rights can be denied with NONE; individual rights can be negated with the prefix NO—for example, NODELETE. Access rights are specified by separating each with a plus sign (+). For examples of commands that grant or deny access rights, see Section 5.3.2.

The entry that provides the greatest amount of file access for a user should be in the list. In the following example ACL, if one of the users has both the SECURITY and PERSONNEL identifiers, the user obtains the maximum access rights through the first match, which is the SECURITY identifier. The user with UIC [SALES,JONES] is prohibited from any access to the file. However, if Jones also happens to belong to one of the previously identified groups, he will acquire the access rights for that group.

```
CDO> SHOW PROTECTION FOR FIELD ID_NUMBER
 Directory SYS$COMMON:[CDDPLUS]PERSONNEL
FIRST_NAME;7
          (IDENTIFIER=SECURITY,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
          (IDENTIFIER=PERSONNEL,ACCESS=READ+WRITE+EXECUTE+DELETE)
          (IDENTIFIER=SECRETARIES,ACCESS=READ+WRITE)
          (IDENTIFIER=[20,*],ACCESS=READ)
          (IDENTIFIER=NETWORK,ACCESS=NONE)
          (IDENTIFIER=[SALES,JONES],ACCESS=NONE)
```

In this example, if Jones also belongs to the group SECRETARIES, then Jones is granted READ and WRITE access to the file because VMS processes the third ACE before the last ACE. This is probably an oversight on the part of the creator of the ACL. If the ACL creator wants to be certain that the user with UIC [SALES,JONES] cannot gain access to the file, the ACE that is at the bottom of this ACL should be moved to the top. For an individual to have fewer privileges than they have as a group member, the ACL creator must move their individual ACE above their group ACE.

Remember that users are often associated with more than one group or more than one identifier. A user is granted the privileges listed in the ACE that provides the *first* match for that user.

The following are two guidelines for ordering ACEs:

- An ACE that grants powerful privileges should be positioned at or near the top of the list of entries.
- If there are no conflicts in the user identifiers, place the more restrictive ACEs at or near the bottom of the list.

5.1.3 Determining Access to the Dictionary

This section describes how to protect VMS files such as the directory where you choose to create your CDO dictionary. To disallow a user from accessing the dictionary, access can be denied at the VMS directory level; however, you should fine tune this protection by protecting individual dictionary definitions, as described in Section 5.3.

When determining whether or not a user is allowed access to a particular system entity, VMS follows these steps:

1. If the system entity has an ACL associated with it, VMS uses this to determine whether or not access is permitted.
2. If the system entity does not have an ACL associated with it, VMS uses UIC-based protection to determine whether or not access is permitted. Access is granted or denied on the basis of the relationship between the entity owner's UIC and the user's UIC.

(This is a simplification of ACL processing by VMS. For more information, see the *Guide to VAX/VMS System Security.*)

The VMS protection provisions apply to the node, device, and directory or directories where your CDO dictionary resides (the anchor or the root of your compatibility dictionary). For users to access any dictionary elements, they must first be granted access to the node, device, and directory where the dictionary resides. You can control access to the dictionary anchors with the DCL command SET PROTECTION, or with the VMS ACL Editor.

The DCL command SET PROTECTION allows you to specify access rights to a file for four different categories of users:

SYSTEM (S)	All users with system privileges
OWNER (O)	Any user with the same user identification code as the creator of the file
GROUP (G)	All users who have the same group as the owner of the file
WORLD (W)	All users of the system

With the DCL command SET PROTECTION, you can specify four types of protection rights: READ, WRITE, EXECUTE, and DELETE. These rights allow the privileges shown in Table 5-1. In a SET PROTECTION command, the access rights can be abbreviated to the first letter of each right.

The following command specifies that all groups of users have READ and EXECUTE rights to the directory where our sample dictionary resides. SYSTEM, OWNER, and WORLD users are also granted WRITE privileges. Only SYSTEM and OWNER users are granted DELETE privileges.

```
$ SET PROTECTION = (S:RWED,O:RWED,G:RE,W:RWE) PERSONNEL.DIR
```

In addition to setting up protection schemes with the SET PROTECTION command, you can manipulate the entries directly by using the VMS ACL editor. The ACL editor is a screen-oriented editor that you can use to create and maintain ACLs.

To invoke the ACL editor, you enter the EDIT/ACL command at the DCL prompt. For example, to edit the protection scheme for the directory [CORPORATE.MIS], type the following command at the DCL prompt when [CORPORATE] is your process default directory:

```
$ EDIT/ACL MIS.DIR
```

The ACL editor displays the ACL for the specified file (if one exists already) and prompts you for changes. For more information about the ACL editor, see the appropriate material in the VMS documentation set.

To better understand VMS security procedures in general, and for information about other methods of creating or modifying ACLs, see the VMS documentation set. The following sections discuss how to protect individual definitions in a CDO dictionary.

5.2 CDO Dictionary Protection Provisions

Each dictionary definition has an ACL associated with it. Each entry in the ACL for the definition lists the access rights associated with an individual or group of users for that particular definition. The access rights that dictionary users can be granted include: SHOW, DEFINE, CHANGE, DELETE, and CONTROL. Table 5-2 lists the access allowed to a definition for each of these rights.

Table 5–2: CDO Functional Access Rights

Privilege	**Access Permitted**
[NO]SHOW	Read and copy definitions
[NO]DEFINE	Create new definitions and new versions of definitions
[NO]CHANGE	Change definitions
[NO]DELETE	Delete and purge definitions
[NO]CONTROL	Define, change, and delete ACLs
Reserved for future use	
[NO]OPERATOR	
[NO]ADMINISTRATOR	

Three additional terms make listing access rights easier: ALL, NOALL, and NONE. You can enable or disable all rights with ALL or NOALL, respectively. For example, to grant a user all access rights except CONTROL, you can specify ACCESS=ALL+NOCONTROL, rather than list each right separately. Similarly, ACCESS=NOALL+SHOW grants a user only SHOW access, while ACCESS=NONE denies all access to a user.

Before access to a dictionary definition is granted or denied, CDO checks the protection defined for the definition's protocol. For information about protocol protection, see the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

The access rights in a definition's protocol indicate which access rights can be confirmed for the definition. Protocol access rights for a definition include: READ, WRITE, MODIFY, and ERASE access.

If the definition's protocol includes READ, this means that SHOW access to the definition is confirmed. Similarly, if the protocol access right MODIFY is removed, CHANGE access to the definition cannot be confirmed; even if the definition's ACL shows that a user had CHANGE privilege, the definition cannot be changed unless the definition's protocol confirms the access.

Table 5-3 shows which access right must be present in a definition's protocol to confirm the related functional access to a definition. These rights are granted by default for all CDD/Plus-supplied protocols.

Table 5–3: Protocol Protection

Protocol Access Right	Purpose
READ	Required to confirm SHOW access to the definition
WRITE	Required to confirm DEFINE access to the definition
MODIFY	Required to confirm CHANGE access to the definition
ERASE	Required to confirm DELETE access to the definition

The protocols for field and record definitions include READ, WRITE, MODIFY, and ERASE access by default. This means that a user with CONTROL access can grant SHOW, DEFINE, CHANGE, and DELETE access to definitions and that these access rights will be confirmed.

The rights assigned to a definition by the definition's protocol can be ignored by most dictionary users unless a conflict of rights exists. A conflict can exist only if the default protocol protection is changed.

For example, if a user with CONTROL access to the definition's protocol definition eliminates the ERASE privilege *on the protocol* for that definition, CDO does not confirm any user's DELETE access. Such a conflict cannot arise unless the default rights are changed in the definition's protocol protection provisions.

5.2.1 Default Protection Provisions

By default, the creator of a CDO dictionary definition has full access to it, including CONTROL. All other users have only SHOW access. CDO dictionary definitions do not inherit access control lists from their parent directories. You can change the default protection, providing that you have CONTROL privileges.

When you first create a CDO definition, CDD/Plus attaches to it an ACL with two entries. Remember that the rights READ, WRITE, MODIFY, and ERASE listed in the default ACL entries of the definition's protocol confirm which functional access rights can be granted for the definition. The default ACL for each new dictionary definition consists of the following two entries.

1. The creator (owner) of the definition is given all access rights including CONTROL. For example:

```
(IDENTIFIER=[JONES],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+CHANGE
                              +DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
```

2. All other users are given only SHOW, ADMINISTRATOR, and OPERATOR access rights to the definition:

```
(IDENTIFIER=[*,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW
                              +ADMINISTRATOR+OPERATOR)
```

CDO dictionary directories cannot be protected. Directories are merely part of the hierarchy you create within a dictionary. Any user who has access to the dictionary can list directory contents. You can, however, protect all of the definitions contained in any directory (see Section 5.3.1).

Compatibility Note

When both DMU and CDO dictionaries are maintained, it is possible that some protection inconsistencies will exist unless you explicitly specify protection provisions as close as possible to your DMU protection.

When you create a definition in a DMU dictionary, it automatically inherits the protection of its parent. When definitions are created in CDO dictionaries, they are supplied with the default CDD/Plus protection provisions. Therefore, when a CDO dictionary item is created with the same directory name as a DMU definition, these items do not necessarily have the same ACLs attached to them.

For example, although the following record definitions share the same dictionary path, they do not have the same protection provisions. The first definition (a DMU record definition) inherited protection from the parent directory. The second definition (in the compatibility dictionary) is protected by CDO default provisions:

```
!
! Show protection for a record defined in DMU
!
CDO> SHOW PROTECTION FOR RECORD CDD$TOP.PERSONNEL.ADDRESS

Directory SYS$COMMON:[DMU_DIC]PERSONNEL

ADDRESS;1
   (IDENTIFIER=[CDD,5],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
                              CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   (IDENTIFIER=[12,5],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
   (IDENTIFIER=[12,9],ACCESS=SHOW+DEFINE)
   (IDENTIFIER=[12,*],ACCESS=SHOW)
```

```
!
! Show protection for a record defined in CDO
!
CDO> SHOW PROTECTION FOR RECORD CDD$TOP.PERSONNEL.ADDRESS_REC

   Directory SYS$COMMON:[CDDPLUS]PERSONNEL.ADDRESS_REC;1

   (IDENTIFIER=DBA,ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
                        CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   (IDENTIFIER=WORLD,ACCESS=READ+WRITE+MODIFY+ERASE+SHOW
                          +OPERATOR+ADMINISTRATOR)
CDO>
```

To avoid the problem of inconsistent protection schemes for definitions that exist in both a CDO and a DMU dictionary, you can define the protection scheme for the definition stored in your CDO dictionary to match the DMU protection as closely as possible. You do this with the DEFINE PROTECTION command. Chapter 2 includes a conversion table for protection provisions when DMU record definitions are converted to CDO format.

5.2.2 Listing Protection Provisions

To find out your access rights to a particular definition, use the SHOW PRIVILEGES command, as shown in the following example. The SHOW PRIVILEGES command displays your current privileges for the definition:

```
CDO> SHOW PRIVILEGES FOR FIELD HOME_PHONE

Directory SYS$COMMON:[CDDPLUS]PERSONNEL

HOME_PHONE;1
   (IDENTIFIER=[20,EVELYN],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+CHANGE)
CDO>
```

The SHOW PROTECTION command displays the complete ACL for the specified definition:

```
CDO> SHOW PROTECTION FOR FIELD CDD$TOP.PERSONNEL.HOME_PHONE

Directory SYS$COMMON:[CDDPLUS]PERSONNEL

HOME_PHONE;1
   (IDENTIFIER=[DBA],ACCESS=READ+WRITE+MODIFY+ERASE+CONTROL+SHOW+DEFINE+CHANGE)
   (IDENTIFIER=[20,EVELYN],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+CHANGE)
   (IDENTIFIER=[20,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE)
```

You cannot display the ACL for a dictionary definition on a remote node.

5.3 Protecting Dictionary Definitions

CDO protects individual CDO dictionary definitions with the default ACL. You need CONTROL access to a dictionary definition to be able to add or change entries in the ACL.

As previously discussed, an ACL consists of a list of entries each specifying access rights for one or more UICs. The entries are numbered and prioritized according to their position in the ACL; for example, the first ACE in the list is number 1, the second in the list is number 2, and so on. The relative position of an ACE is as significant in CDO as it is in VMS. CDO searches the entries of user identification criteria in order, and stops searching as soon as it finds a match.

You can add an entry to the history list for a definition with the AUDIT attribute in the DEFINE PROTECTION, CHANGE, and DELETE PROTECTION commands.

5.3.1 Adding New Access Control Entries

You can create new entries in the ACL for a dictionary definition with the DEFINE PROTECTION command in the CDO environment, providing that you have CONTROL access rights. This command defines an ACE for one or more UICs to be positioned in the ACL for a specified dictionary definition. You include the following parameters with the DEFINE PROTECTION command:

- The type of definition you are protecting
- The name of the dictionary definition you are protecting
- The position of the ACE you are specifying
- One or more identifiers for the users whose rights you are specifying
- The specific access rights you want to grant or deny

For example, the following DEFINE PROTECTION command:

1. Grants SHOW and DEFINE access to the record definition ADDRESS for the user with the identifier [SECRETARIES,SUSAN].
2. Inserts the new access entry as the second ACE, using the POSITION 2 qualifier.
3. Makes the original second ACE the new third ACE, since there were originally only two ACEs.

```
!Show previous protection
CDO> SHOW PROTECTION FOR RECORD ADDRESS

Directory SYS$COMMON:[CDDPLUS]PERSONNEL

ADDRESS;1
   (IDENTIFIER=[CDD,5],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
                          CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   (IDENTIFIER=[12,5],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
!Define a new ACE
CDO> DEFINE PROTECTION FOR RECORD ADDRESS
cont> POSITION 2 IDENTIFIER [SECRETARIES,SUSAN]
cont> ACCESS SHOW+DEFINE.
! New protection
CDO> SHOW PROTECTION FOR RECORD ADDRESS

Directory SYS$COMMON:[CDDPLUS]PERSONNEL

ADDRESS;1
   (IDENTIFIER=[CDD,5],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
                          CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   (IDENTIFIER=[SECRETARIES,SUSAN],ACCESS=SHOW+DEFINE)
   (IDENTIFIER=[12,5],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
CDO>
```

An optional AFTER clause allows you to specify which ACE the new entry is to be inserted after. In the AFTER clause, you specify the previous ACE with an identifier, not a position. When you specify an AFTER clause, you cannot also specify a POSITION clause. For example, the following command creates a new ACE for the field definition RATE and specifies that the new entry is to be placed after the ACE with the identifier [21,12]:

```
! Show current protection
CDO> SHOW PROTECTION FOR FIELD RATE

 Directory SYS$COMMON:[CDDPLUS]PERSONNEL

RATE;1
   (IDENTIFIER=[CDD,5],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
                          CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   (IDENTIFIER=[21,12],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
!Define a new ACE
CDO> DEFINE PROTECTION FOR FIELD RATE AFTER [21,12]
cont> IDENTIFIER [30,13]
cont> ACCESS NONE.
! New protection
CDO> SHOW PROTECTION FOR FIELD RATE

 Directory SYS$COMMON:[CDDPLUS]PERSONNEL

RATE;1
   (IDENTIFIER=[CDD,5],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
                          CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   (IDENTIFIER=[21,12],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
   (IDENTIFIER=[30,13],ACCESS=NONE)
CDO>
```

Compatibility Note

You cannot change the protection for a DMU definition from the CDO environment. However, you can display the protection scheme for DMU definitions from CDO with the SHOW PROTECTION command. CDO translates DMU access rights into the equivalent CDO rights for the display. For more information, see Section 2.2.2.

The following example defines an ACE for the dictionary definitions RATE and SALARY that is to be positioned after the entry associated with the group of users [EXECS,*]. CDO allows SHOW and DEFINE access only when the user [JOHN] accesses the dictionary in batch mode.

```
CDO> DEFINE PROTECTION FOR FIELD RATE,SALARY AFTER [EXECS,*]
cont> IDENTIFIER [JOHN]+[BATCH]
cont> ACCESS SHOW+DEFINE.
CDO>
```

Note

When you do not specify the position or the identifier for the new ACE, CDO places it in the first position by default. Because the first ACE is usually a powerful one, you should be careful that you do not create a new first ACE by mistake.

No position is included in the following command, so CDO creates a new first entry for the field definition YEAR_TO_DATE. As the default, ACL already had two entries (in positions one and two), the new ACE moves the original first entry into second position, and the original second entry into third position.

```
! Previous protection
CDO> SHOW PROTECTION FOR FIELD YEAR_TO_DATE

 Directory SYS$COMMON:[CDDPLUS]PERSONNEL

YEAR_TO_DATE;1
   (IDENTIFIER=[CDD,5],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
                           CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   (IDENTIFIER=[21,12],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
!Define the new ACE
CDO> DEFINE PROTECTION FOR FIELD YEAR_TO_DATE
cont> IDENTIFIER [12,*]
cont> ACCESS CONTROL.
! New protection
CDO> SHOW PROTECTION FOR FIELD YEAR_TO_DATE

 Directory SYS$COMMON:[CDDPLUS]PERSONNEL

YEAR_TO_DATE;1
   (IDENTIFIER=[12,*],ACCESS=CONTROL)
   (IDENTIFIER=[CDD,5],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
                           CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   (IDENTIFIER=[21,12],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
CDO>
```

When you specify a position number that is higher than the last ACE in the ACL, CDO places the new entry in the last position possible in the list. For example, the following command defines the tenth ACE for field VAC_DAYS in the BENEFITS directory. Since there are only two entries in the ACL for the specified field definition, CDO places the new ACE third.

```
CDO> DEFINE PROTECTION FOR FIELD BENEFITS.VAC_DAYS
cont> POSITION 10 IDENTIFIER [DOC,SUE] ACCESS ALL.
!Show new ACL
CDO> SHOW PROTECTION FOR FIELD BENEFITS.VAC_DAYS

 Directory SYS$COMMON:[CDDPLUS]PERSONNEL.BENEFITS

VAC_DAYS;1
   (IDENTIFIER=[CDD,5],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
                           CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   (IDENTIFIER=[21,12],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
   (IDENTIFIER=[DOC,SUE],ACCESS=ALL)
CDO>
```

You can use wildcards to define the protection for all of the definitions in a particular directory. The following command defines the fourth ACE for all versions of all record definitions in the BENEFITS directory:

```
CDO> DEFINE PROTECTION FOR RECORD BENEFITS.*;* POSITION 4
cont> IDENTIFIER [PERSONNEL,*] ACCESS SHOW.
CDO>
```

The previous command defines the protection for all *existing* definitions in the BENEFITS directory. CDO does not apply this as a default protection scheme for any new definitions created in this directory.

Note

If you define a new ACE with an identifier that matches the identifier in an existing ACE, CDO deletes the original ACE for that identifier and creates a new ACE with the rights and position you specify in the DEFINE PROTECTION command. To change the privileges for an identifier in an existing ACE, use the CHANGE PROTECTION command, as shown in Section 5.3.2.

5.3.2 Changing Access Control Entries

You can modify your protection scheme for particular definitions with the CHANGE PROTECTION command, providing that you have CONTROL access rights. The new access control rights you specify do not replace the old; instead, they are combined with the old set of rights to produce a new set.

The CHANGE PROTECTION command allows you to alter an ACE that already exists; only the DEFINE PROTECTION command allows you to create new ACL entries. With the CHANGE PROTECTION command, you can identify the ACE either by position or by identifier.

Caution

When neither the position nor an identifier is specified with the CHANGE PROTECTION command, CDO makes the change to the first ACE by default. It is always safer to identify the ACE you want to change than to risk altering the first ACE by mistake.

The following command changes the fourth ACE in the ACL list so that the user has no access to the field definition FIRST_NAME. A new ACE is not created, and the current fourth ACE is changed.

```
CDO> SHOW PROTECTION FOR FIELD FIRST_NAME
 Directory SYS$COMMON:[CDDPLUS]PERSONNEL
```

```
FIRST_NAME;1
   (IDENTIFIER=[CDD,5],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
                        CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   (IDENTIFIER=[21,12],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
   (IDENTIFIER=[21,19],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
   (IDENTIFIER=[21,*],ACCESS=SHOW)
!
!Change the fourth ACE
!
CDO> CHANGE PROTECTION FOR FIELD FIRST_NAME 4
cont> ACCESS NONE.
!
!List changed protection
!
CDO> SHOW PROTECTION FOR FIELD FIRST_NAME

 Directory SYS$COMMON:[CDDPLUS]PERSONNEL

FIRST_NAME;1
   (IDENTIFIER=[CDD,5],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
                        CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   (IDENTIFIER=[21,12],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
   (IDENTIFIER=[21,19],ACCESS=SHOW+DEFINE+CHANGE+DELETE)
   (IDENTIFIER=[21,*],ACCESS=NONE)
CDO>
```

The following command changes the ACE identified by [CDD,HALVORSON] so that the user has CONTROL access to the field definition DEPENDENTS. Unlike DEFINE PROTECTION, the CHANGE PROTECTION command does not use the IDENTIFIER keyword.

```
CDO> CHANGE PROTECTION FOR FIELD DEPENDENTS
cont> [CDD,HALVORSON]
cont> ACCESS CONTROL.
CDO>
```

The following command changes the third ACE so that the user only has SHOW access to the field definition CONTRACT.BADGE_NO:

```
CDO> CHANGE PROTECTION FOR FIELD CONTRACT.BADGE_NO 3
cont> ACCESS NOALL+SHOW.
CDO>
```

Remember that the order of the entries in the ACL determines priorities in the protection scheme. *Always take special care in ordering the entries within your ACLs.*

5.3.3 Deleting Protection Provisions

You can delete an ACE or a complete ACL with the DELETE PROTECTION command, providing that you have CONTROL access rights. To delete an entry from the ACL for a definition, identify the entry you want to delete by specifying the position or identifier for the ACE. The optional /LOG qualifier causes CDO to display a message indicating success when the ACE is deleted. For example:

```
CDO> DELETE PROTECTION /LOG FOR RECORD EMPLOYEE_REC 3.
%CDO-I-PROTDEL, specified protection on entity
SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_REC;1 deleted
CDO>
```

To delete every entry in an ACL, do not specify a position or identifier with the DELETE command. The following command deletes the complete ACL for EMPLOYEE_REC:

```
CDO> DELETE PROTECTION /LOG FOR RECORD EMPLOYEE_REC.
%CDO-I-PROTDEL, specified protection on entity
SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_REC;1 deleted
!
!List changed protection
!
CDO> SHOW PROTECTION FOR RECORD EMPLOYEE_REC

Directory SYS$COMMON:[CDDPLUS]PERSONNEL

EMPLOYEE_REC;1
CDO>
```

You should exercise caution when deleting complete ACLs. If the ACL for a dictionary definition is deleted, then that definition has *no* associated protection provisions and all users have full access to it.

5.3.4 Protecting Interrelated Definitions

When a user has conflicting access rights to definitions, CDO always selects the most restrictive protection provisions. For example, the field definition GIVEN_NAME is based on the definition of field FIRST_NAME. GIVEN_NAME is contained in the record FULL_NAME. If you have SHOW access to the record definition FULL_NAME, but no access to FIRST_NAME, CDO denies you SHOW access to FULL_NAME. CDO issues an error message if you attempt to view the attributes of the protected field definition with the SHOW command. You can, however, list the name of the protected field definition with the DIRECTORY command.

If a record definition contains four fields and you have access to the record but only three of the fields, CDO issues an error message in response to the SHOW RECORD command. You can, however, list the record contents with the DIRECTORY command.

Note

When you have no access rights, or access to only part of a definition, you should be careful when you access the definition from another product, such as DATATRIEVE or one of the VAX languages. For example, if you attempt to include a record definition in a language program when you have access to only four out of five field definitions, the included record definition will be incomplete.

5.3.5 Accessing Remote Dictionary Definitions

You should use a proxy account for remote access. When proxy accounts are used, the dictionary definition's ACL need not grant SHOW access to all users. For information about setting up proxy accounts, see the *Guide to VAX/VMS System Security.*

A remote CDO dictionary user with no proxy account is represented on the target node as the user DECNET with the UIC DECNET. You can limit remote access to dictionary definitions in two ways:

- If you specify ACCESS=NONE for the identifier DECNET, users on other nodes who do not have proxy accounts will be denied access.
- If you specify ACCESS=NONE for the identifier SYS$REMOTE, all users on other nodes are denied access to the dictionary definition over DECnet whether or not they have a proxy account.

You cannot display the entire ACL for a dictionary definition on a remote node: you can show the protection but not the privilege.

5.3.6 Protecting Your Dictionary from Corruption

The CDO security facilities are a safeguard against unauthorized access to the dictionary. These facilities are not guaranteed to prevent deliberate attempts to corrupt a dictionary.

For security reasons, you should be wary of granting DELETE or CONTROL access rights to dictionary users. CONTROL should be granted only to the data administrator, the system manager, and users with personal directories. Remember also that CDO and VMS file security provisions can be overridden by any user with VMS BYPASS or SYSPRV privileges. In general, you should grant users the minimum privileges they need to work in their portions of the dictionary.

CDO dictionaries are protected by Rdb/VMS locking mechanisms in addition to access control lists. CDO uses these mechanisms to protect your dictionary definitions from simultaneous updates. When a user is modifying a dictionary definition, another user cannot modify the same definition until the first user finishes, regardless of the user's access rights. Rdb/VMS also protects your dictionary in the event of a system failure during modification of the dictionary. Chapter 6 discusses dictionary integrity in greater detail.

Managing Dictionary Usage and Change 6

This chapter shows you how to monitor dictionary usage and control changes. Other management issues are discussed, such as verifying the status of the dictionary and improving dictionary performance.

6.1 Tracking Dictionary Usage

This section shows you how CDD/Plus keeps track of CDO dictionary usage. In the CDO environment, you can browse through your dictionary to monitor and analyze dictionary usage. Knowledge of how your dictionary is used can help you:

- Evaluate when a directory structure is overloaded
- Evaluate how far-reaching a change to a definition will be
- Decide when definitions become obsolete
- Decide when definitions should be purged
- Restructure your dictionary hierarchy

6.1.1 Using the SHOW Commands

CDD/Plus provides you with information about the relationships between the definitions you store in CDO dictionaries. Table 6-1 summarizes these commands. Examples of the commands are provided in the following sections. For details of command syntax and qualifiers, see *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

Table 6–1: Summary of Usage Tracking Commands

Command	Tracking Function
SHOW MESSAGES	Displays any of four kinds of messages attached to the specified definition. The message indicates that the specified element might be invalid because of a recent change to a related definition, that the specified definition *is* invalid because a related element was changed or deleted, that the specified definition might be invalid because a relationship whose owner is related to the element was changed, or that a new version of a related element exists. Supporting software products can read the message and generate a warning to the user.
SHOW UNUSED	Displays entity definitions that are not used by any other definition. This helps you decide when it is safe to purge or delete dictionary definitions.
SHOW USES	Displays all of the entity definitions that use the specified definition. This helps you to consider the impact of changing the definition by creating a new version.
SHOW USED_BY	Displays the definitions that are used by the specified definition.
SHOW WHAT_IF	Displays a listing of the dictionary definitions that would be flagged with a message after a definition is changed with the CHANGE command. This helps you to consider the impact of changing the original definition.

You can use optional qualifiers with the SHOW USES, SHOW UNUSED, SHOW USED_BY, and SHOW WHAT_IF commands to display various levels of usage:

- The /BRIEF qualifier allows you to display only those entities that are directly related to the specified entity.
- The /FULL qualifier allows you to display all database entities that are directly or indirectly related to the specified entity. Related entities include those that own the specified entity as well as those owned by that entity.

- The /ALL qualifier allows you to display all of the entities shown by the /FULL qualifier plus the history list and system information, such as create/modified time and protection, for the specified entity.
- The /TYPE qualifier allows you to display only entities of a specified protocol type (such as field or record). You can use /TYPE with any of the other three qualifiers to limit output.

The default is for CDO to display only one level of usage (/BRIEF). For full details of the optional qualifiers, see the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

CDO keeps track of all definitions and sources that use, or are used by, a particular definition. You can list these dependencies with the SHOW USES and SHOW USED_BY commands. In the following example, the SHOW USES command displays the definitions that *use* the record definition FULL_NAME; the SHOW USED_BY command displays all definitions that are *used by* the record definition FULL_NAME.

```
CDO> SHOW USES FULL_NAME
Owners of SYS$COMMON:[CDDPLUS]PERSONNEL.FULL_NAME;1
|   SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_NAME              (Type : RECORD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
|   SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_REC               (Type : RECORD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS

CDO> SHOW USED_BY FULL_NAME
Members of SYS$COMMON:[CDDPLUS]PERSONNEL.FULL_NAME;1
|   SYS$COMMON:[CDDPLUS]PERSONNEL.FIRST_NAME                 (Type : FIELD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
|   SYS$COMMON:[CDDPLUS]PERSONNEL.MIDDLE_INIT                (Type : FIELD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
|   SYS$COMMON:[CDDPLUS]PERSONNEL.LAST_NAME                  (Type : FIELD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS

CDO>
```

The SHOW USES, SHOW USED_BY, SHOW MESSAGES, and SHOW WHAT_IF commands display only definitions stored in CDD/Plus dictionary format.

6.1.2 Generating Messages by Changing Definitions

CDD/Plus provides an automatic message system that communicates information about changing definitions. Whenever you change a dictionary definition, you potentially affect any other definitions or sources that use the definition. You must decide if sources can use the original or the latest version of a definition. A change or a new version of a definition can potentially affect all definitions that use it.

To recap, CDD/Plus provides you with two ways to change CDO definitions:

- You can define a new version and allow users to accommodate the change over a period of time.
- You can change the original definition and cause users to automatically include the change at compilation time.

The changes you make with the CHANGE command:

- Are automatically included in dictionary definitions that use the changed definition.
- Are automatically included in all remote copies of the changed definition.
- Are not automatically copied to external copies of dictionary definitions, such as database copies. (You must explicitly integrate changed dictionary definitions into the database to resolve any inconsistencies (see Chapter 7).)
- Generate messages only to entities that represent objects with external dependencies.

New versions created with the DEFINE command:

- Are never automatically included in the definitions that use them.
- Generate messages to all users of the previous version of the definition.

Consider that a field definition RATE is used in record definition SALARY_REC and is also used in an Rdb/VMS database PERSONNEL. Consider also that SALARY_REC is included in the record EMPLOYEE_REC. When you change RATE with the CHANGE command, the database PERSONNEL is flagged with a message about the change when the database is next invoked. SALARY_REC and EMPLOYEE_REC both include the changed definition of RATE; therefore, no message is attached to those record definitions. However, if a new version of RATE is created with the DEFINE command, the new version is not automatically included in other definitions; therefore, all dependent dictionary definitions are flagged with a message.

Regardless of whether the changes were made with the DEFINE or the CHANGE command, external copies of the dictionary definitions in the PERSONNEL database are neither changed nor flagged with a message by the dictionary; only the dictionary definition of the database itself is flagged with a message about the inconsistency. For example, when an audit line is added to the existing definition of the field RATE, CDD/Plus notifies the user that the field is part of the database PERSONNEL, and that dictionary definition of RATE is now inconsistent with the definition of RATE in the external copy of the database PERSONNEL:

```
CDO> CHANGE FIELD RATE
cont> AUDIT IS /* INSERT NEW PAY RATE HERE */.
%CDO-I-DBMBR, database SYS$COMMON:[CDDPLUS]PERSONNEL;1
may need to be INTEGRATED
```

When you consider changing a definition, you need to know:

- which other entities use the definition
- which entities automatically include changes
- which entities will be flagged with a message if an inconsistency occurs

Any product that supports CDD/Plus can read messages that are attached to CDO definitions. For example, when you try to use a CDO definition from your Rdb/VMS database, Rdb/VMS generates a warning if it encounters a message attached to the definition. The message is a signal to you that the dictionary definition you are accessing has been altered in some way.

```
RDO> INVOKE DATABASE PATHNAME 'SYS$COMMON:[CDDPLUS]PERSONNEL'
%CDD-I-MESS, entity has messages
```

To find out which dictionary definitions would be flagged with a message if a definition were to be changed with the CHANGE command, use the /FULL qualifier with the SHOW WHAT_IF command. For example, if the field definition SIMPLE_LIST were to be changed with the CHANGE command, the records that use it would not be flagged with a message because the records automatically include the changed field; however, any Rdb/VMS database would be flagged with a message because the database is not automatically updated by the dictionary.

The following command displays a listing of the the dictionary definitions that would be flagged with a message if SIMPLE_LIST were to be changed with the CHANGE command. Remember that when a new version of a definition is created with the DEFINE command, *all* definitions that use the definition are flagged with a message; these definitions are not displayed by the SHOW WHAT_IF command.

```
CDO> SHOW WHAT_IF /FULL SIMPLE_LIST

Owners of SYS$COMMON:[CDDPLUS]SIMPLE_LIST;1

|   SYS$COMMON:[CDDPLUS]PERSONNEL                        (Type: CDD$DATABASE)
|   |   via CDD$DATABASE_SCHEMA
|   SYS$COMMON:[CDDPLUS]PERSONNEL   (Type: CDD$DATABASE)
|   |   via CDD$DATABASE_SCHEMA

CDO>
```

Entities that are not represented in the dictionary at all (such as a language program) can still use dictionary definitions. Copies of dictionary definitions in these external entities are neither changed nor flagged with a message by the dictionary; only entities that are represented in the dictionary can be changed or flagged with messages.

Note

Remember that messages are passive place markers; they do not automatically change, recompile, or update definitions. They indicate that *you* may need to take action to accommodate changes.

6.1.3 Accessing Messages about Changes

CDD/Plus messages communicate information about a change to a CDO dictionary definition to definitions that use it. Any product that supports CDD/Plus can read messages that are attached to CDO definitions. The message is a signal to you that a definition you access in the dictionary has been altered in some way. You may need to change your source program and perhaps recompile it. For example, when you next invoke the EMPLOYEES database, Rdb/VMS generates a warning that the database and dictionary are now inconsistent. To accommodate the inconsistency, you can issue the RDO statement INTEGRATE (see Chapter 7).

Compatibility Note

If you access changed definitions from a product that does not support the features of CDO dictionaries, you must use CDO to read or access the warning messages.

To read messages attached to a definition, you use the SHOW MESSAGES command. For example, to read any messages attached to the record definition ADDRESS_REC, specify the definition name on the SHOW MESSAGES command line as shown:

```
CDO> SHOW MESSAGES CDD$TOP.PERSONNEL.ADDRESS_REC

SYS$COMMON:[CDDPLUS]PERSONNEL.ADDRESS_REC;1 uses an entity
which has new versions, triggered by entity
SYS$COMMON:[CDDPLUS]PERSONNEL.ZIP_CODE;1

CDO>
```

CDD/Plus generates a message that a new version exists of ZIP_CODE;1 which is used by ADDRESS_REC.

After all the users have been warned of a change in the definition, the warning message may become superfluous. You can delete messages with the CLEAR MESSAGES command. For example, the following command clears the message attached to the record definition ADDRESS_REC.

```
CDO> CLEAR MESSAGES ADDRESS_REC
CDO>
```

You should not clear messages from a definition until you are certain that all sources have been altered to accommodate the changes.

If you have made a compatible change to a definition (such as merely adding text to the DESCRIPTION clause), you may decide to clear all messages without checking that all users have been updated. However, you should be aware that what seems like a compatible change to you may not be seen that way by another user. CDD/Plus considers all changes and new versions of CDO definitions to be incompatible changes with the exception of a change to a definition's history list. CDD/Plus, therefore, flags all dependent uses with messages when a new version is created.

6.1.4 Recompiling Application Programs

Messages are passive; it is up to you to change the source code, redefine the record, integrate the dictionary and database, or take other appropriate action.

The VAX languages include definitions from CDD/Plus at compile time. This means that any changes to definitions are not reflected in the programs unless they are recompiled. A change to a definition may require a related change in the program source code. When a source program is not recompiled, the executable code will continue to reflect the original definitions.

If you access an Rdb/VMS database with callable RDO, messages about new versions of definitions in the dictionary will be displayed by Rdb/VMS when you invoke the database. For more information, see Chapter 7.

Remember that if you access CDO definitions from a product that does not support CDD/Plus, warning messages will not be issued. If this is the case, it is up to the person who makes the change to the definition to isolate all programs that access the changed definition, and to notify the appropriate people to make any required changes to their sources.

When new versions have been accommodated by all users, dictionary definitions can be removed with the PURGE or DELETE commands.

6.2 Confirming Dictionary Integrity

It is good practice to monitor the condition of your dictionary periodically. The following sections discuss CDD/Plus journaling, making backup copies of your CDO dictionary, and using the CDO command VERIFY.

6.2.1 Journaling

CDD/Plus protects your dictionary automatically in the event of a system or network failure during modification of dictionary definitions. By default, CDD/Plus maintains journal files for all dictionary transactions until each transaction is complete. This feature allows CDD/Plus to recover when a dictionary session is aborted. When a dictionary session has completed without interruption, CDD/Plus deletes these journal files. When a session is aborted, CDD/Plus automatically uses the journal files to roll back to the start of the transaction that was interrupted.

No action is required in the event of a system failure; journaling and recovery is automatic.

6.2.2 Saving Copies of Your Dictionary

As a precaution, you should perform regular backups of each dictionary under your control with the VMS Backup Utility. The DCL command BACKUP saves the entire contents of the specified VMS directory that contains your dictionary. To back up a single physical dictionary, specify the anchor with the BACKUP command. Note that you cannot perform a backup from the CDO environment.

You should disallow all users from the dictionary before beginning the procedure. This is commonly accomplished by backing up the dictionary at night. You can also temporarily change the protection for the dictionary anchor to allow access only to the database administrator.

In the following example, the SAVE_SET qualifier is used with the DCL command BACKUP. The dictionary files in SYS$COMMON:[CDDPLUS] are written to a save set SAVEIT.BCK on the magnetic tape mounted on drive MTA0. The second command lists the files that were written into the save set.

```
$ BACKUP SYS$COMMON:[CDDPLUS] MTAO::SAVEIT.BCK/SAVE_SET
$ BACKUP/LIST MTAO::SAVEIT.BCK
```

When you manage more than one physical dictionary and when dictionary definitions are interdependent, back up all of your dictionaries as closely together as possible. This helps to eliminate inconsistencies between dictionaries. If you suspect a problem due to restoring backup copies of dictionaries, use the VERIFY command, as described in the following section.

Cross-dictionary relationships affect two dictionaries. For example, if a dictionary is backed up before a relationship is made to an external dictionary; the external dictionary would be aware of the relationship but the backed up copy would not be. After you restore a backup copy of a dictionary, you may need to verify that all relationships leaving or entering the dictionary are known by the related dictionaries. To do this, use the /EXTERNAL_REFERENCE qualifier to the VERIFY command. The VERIFY command is discussed in Section 6.2.3.

It is up to you to use the Backup Utility as often as you feel necessary. DIGITAL recommends that you back up your dictionary at least once per week. For more information about the BACKUP command, see the *VMS Backup Utility Manual.*

6.2.3 Verifying the Dictionary Condition

The VERIFY command scans your dictionary files to confirm that one or more dictionaries are structurally correct. The /FIX qualifier repairs any detected problems.

You should use the VERIFY command in the CDO environment in the following situations:

- When you restore backed up copies of dictionaries with interrelated definitions or a single dictionary with external relationships
- When you move a dictionary to a new location
- When an error message indicates the need for verification

To use the VERIFY command, you must have SHOW access to the dictionary and to any definitions that you are verifying. (To verify a cross-dictionary relationship, you must be able to read it.)

The VERIFY command can only confirm the integrity of a dictionary, or part of a dictionary, when no users are accessing the dictionary; therefore, you must prohibit all use of the dictionary before issuing the command.

Caution

When you use the VERIFY command, the dictionary is temporarily inaccessible to all other users. After execution, the dictionary becomes available again. This protects the dictionary from modifications that could potentially corrupt the dictionary again during a recovery.

The VERIFY command creates a report describing the condition of the dictionary and lists any corrupted definitions. For example, you may have restored the files on a different disk drive and thereby altered the dictionary anchor. The report displays any problems discovered in the specified dictionary. The following example uses the /LOG qualifier to show the output from the VERIFY command when no dictionary problems have been discovered.

```
CDO> VERIFY /LOG SYS$COMMON:[CDDPLUS]PERSONNEL
%CDD-I-NORECOVER, dictionary does not need recovery
CDO>
```

You can include a search list as a parameter to the VERIFY command. This allows you to obtain a report on all of the physical dictionaries that you manage. The following example uses a search list to verify the status of several dictionaries. A problem is found and repaired, as reported:

```
CDO> VERIFY /LOG /FIX MY_DICT
%CDO-F-ERRVERIFY, error verifying object
-CDO-I-FIXED, dictionary was corrupt, has been fixed
CDO>
```

If you used RDO or RMU to back up your dictionary database, you can use the VERIFY command with the /REBUILD_DIRECTORY qualifier to recover the directory system. You can also use VERIFY/REBUILD_DIRECTORY to recover a severely corrupted directory system that VERIFY/FIX/DIRECTORY could not fix. If you run the VERIFY command with /REBUILD_DIRECTORY and /LOG qualifiers, CDD/Plus provides numerous informational messages about the rebuilding process.

```
CDO>VERIFY/REBUILD_DIRECTORY/LOG SYS$COMMON:[CDDPLUS]PERSONNEL.
%CDD-I-NORECOVER, dictionary does not need recovery
%CDD-I-REBUILD, rebuilding directory structure for SYS$COMMON:[CDDPLUS]PERSONNEL.
%CDD-I-CREDIR, the backpointer 300000005000000A is being used to create
a new directory
%CDD-I-CREDIR, the backpointer 500000006000000B is being used to create
a new directory         .
                        .
                        .
%CDD-I-FIXBKPTR, the backpointer 3000000040000000CDD$ELEMENT_TYPE is
being used to generate a new directory name
%CDD-I-FIXBKPTR, the backpointer 3000000040000000CDD$ATTRIBUTE_TYPE is
being used to generate a new directory name
                        .
                        .
                        .
%CDD-I-FIXED, dictionary was corrupt, has been fixed
```

If you use the /NOLOG qualifier with the command above, the message display omits some of the informational messages, and begins with the %CDD-I-FIXBKPTR messages.

Caution

DIGITAL recommends that you back up your dictionary before using the /REBUILD_DIRECTORY option. You should use the /REBUILD_DIRECTORY option only when corruption is so severe that all other options fail.

Table 6-2 describes the functions of the various qualifiers to the VERIFY command.

Table 6–2: Qualifiers to the VERIFY Command

VERIFY Qualifier	Explanation
/ALL	Causes all verification options except /REBUILD_DIRECTORY to be performed[1]
/DIRECTORY	Confirms that every directory name corresponds to a dictionary definition; directory names that do not refer to actual definitions are deleted
/EXTERNAL_REFERENCE	Checks all relationships that leave or enter the dictionary and verifies that the other dictionary knows about them; use after a dictionary backup
/FIX	Repairs detected problems
/LOCATION	Determines the VMS directory that contains the dictionary and verifies or corrects all pointers to the dictionary from external sources; use after a dictionary is moved or copied (see Section 6.2.4)
/LOG	Displays all success, informational, and error messages
/ORPHANS	Confirms that all definitions are named or owned by other definitions; missing names are created and definitions with no name and no owner are named and placed in CDD$ORPHANS
/REBUILD_DIRECTORY	Causes all directory files for the specified anchor to be deleted and recreated; use to recover severely corrupted directory system after a back up with RDO or RMU

[1]The /ALL qualifier does not include /REBUILD_DIRECTORY because /REBUILD_DIRECTORY always causes the entire directory to be rebuilt: it does not only fix what is corrupt.

If you have interrelating definitions in two or more dictionaries, special action may be required after a dictionary has been backed up and recovered, or when a node has been unavailable for a long period of time.

When dictionary definitions use or are used by definitions on a remote node, it is possible that parts of the dictionary on the local node can be locked. For example, consider that record definition X on a dictionary in LOCAL1 uses field definition A on REMOTE1. A user changes field definition A on REMOTE1 with the CHANGE command (changes the original definition). The dictionary on REMOTE1 attempts to copy the new definition to LOCAL1. While the transaction is being committed, the system on REMOTE1 fails. Until the REMOTE1 system is running again, some definitions in the dictionary on LOCAL1 may be locked against change or use in other definitions.

6.2.4 Changing the Location of a Dictionary

You should not move a dictionary without forethought. Dictionary definitions that use or are used by definitions in other dictionaries must refer to them with a fully qualified name including node, disk, and VMS directory names. Therefore, when you move a dictionary, you potentially invalidate any references to the dictionary from other dictionaries. If you use concealed device logicals, you can avoid problems when moving disks.

To move a dictionary, use the MOVE DICTIONARY command in the CDO environment. The MOVE DICTIONARY command resolves all pointers to the old dictionary location so that the new location is indicated. The target location must be a VMS directory that contains no files.

Caution

When you use the MOVE DICTIONARY command, you must specify the entire name, *including device and directory pathname*, of both location and target files. If you specify only directory names, you could lose an entire VMS directory.

The following command moves the dictionary at SYS$COMMON:[CDDPLUS] to DISK$01:[CORPORATE.MIS]:

```
CDO> MOVE DICTIONARY SYS$COMMON:[CDDPLUS]
cont> TO DISK$01:[CORPORATE.MIS].
CDO>
```

If the dictionary you are moving contains definitions that are related to definitions in a remote dictionary, the MOVE DICTIONARY command can only resolve these references if the network link is viable at the time of execution. If the network link is not viable, the MOVE DICTIONARY command fails and you must retry the operation.

If you change the location of a dictionary without the MOVE DICTIONARY command, you must use the VERIFY command with the /FIX and /LOCATION qualifiers to resolve the pointers.

6.3 Enhancing Dictionary Performance

While dictionary performance is a prime concern, it should be considered along with other dictionary management issues, such as security and accessibility. Do you need contiguous space for files? Do you prefer a small number of elements in each directory? As the dictionary administrator, you must decide on the priorities so that your dictionary can best meet the needs of your own organization.

6.3.1 Removing Unused Definitions

Your dictionary will perform better if unused definitions are deleted from the dictionary. You can remove unused definitions from your dictionary with the PURGE and DELETE commands, as described in Chapter 4. Note that to delete or purge definitions on a remote CDO dictionary, the network link between the local and the remote node must be viable.

You can use the SHOW UNUSED command to isolate definitions that are not used by any other definition. For example, the following command displays all definitions stored in the BENEFITS directory that are not used by any other entities in the dictionary:

```
CDO> SHOW UNUSED [CDDPLUS]PERSONNEL.BENEFITS.*

SYS$COMMON:[CDDPLUS]PERSONNEL.BENEFITS.LAST_ZIP;1          (TYPE : FIELD)
SYS$COMMON:[CDDPLUS]PERSONNEL.BENEFITS.NEW_RATE;1          (TYPE : FIELD)
CDO>
```

You may also choose to delete or change definitions that require large amounts of space. You can find out the *relative* amount of storage that CDO definitions require with the DIRECTORY/FULL command. (CDD/Plus uses Rdb/VMS to compress CDO dictionary definitions.)

You need DELETE access to dictionary definitions to purge or delete them. By default, only the owner of a definition has the right to delete it; however, the default protection can be changed by a user with CONTROL access. Note that the ERASE access right is associated only with the protocol definition; only DELETE access allows you to delete or purge definitions.

6.3.2 Deleting a Dictionary

To delete a complete dictionary, first make sure that the dictionary definitions are not used by entities external to the dictionary.

When CDD/Plus deletes a complete dictionary, all files in the anchor directory are deleted. You should store only files created by CDD/Plus in a VMS directory that is dedicated to a dictionary. If you do store other files in the VMS directory, these files are deleted when you delete the dictionary.

If you delete the compatibility dictionary, you cannot read DMU definitions from CDO.

6.3.3 Structuring Your Dictionary

You may have special needs in your organization, such as security or maintenance requirements, that indicate a need for multiple dictionaries rather than one. CDD /Plus manages complex dictionary structures efficiently. However, a large number of dictionaries can be difficult to manage. An alternative is to create organizational or personal directories within a single dictionary.

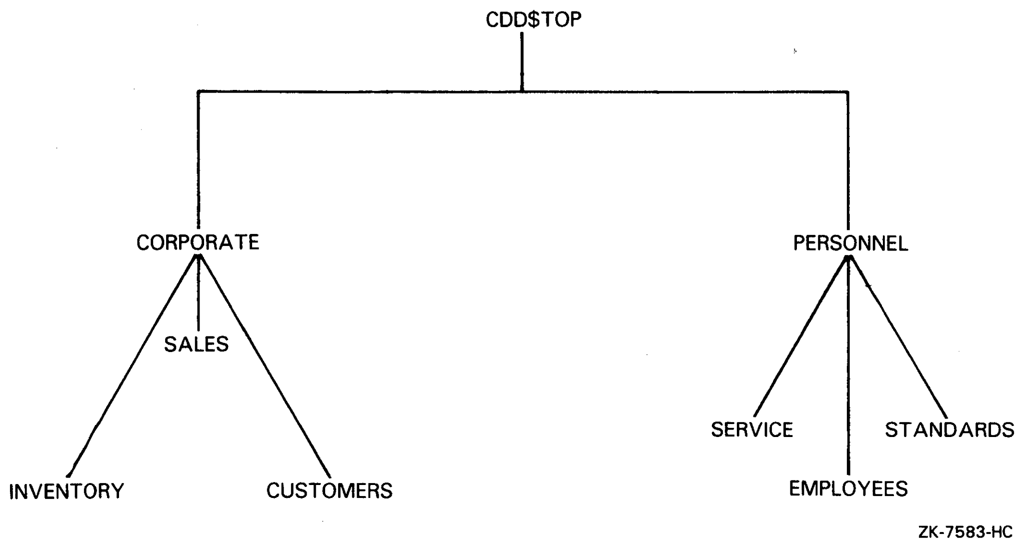
You should avoid creating a large number of directories immediately below the dictionary origin because this can cause unnecessary locking problems and may reduce performance. Rather, you should create a minimum of directories under the anchor directory and create personal or departmental directories at the next level. Figure 6-1 illustrates a hierarchy with several departmental directories two levels below the dictionary origin.

6.3.4 Improving Dictionary Performance over a Network

Maintaining several dictionaries that are distributed over a network offers several advantages in security and accessibility.

Because CDD/Plus uses local copies of CDO dictionary definitions, performance when accessing relationship members is not affected adversely by network issues. Transactions that perform retrieval operations over a network are fast. However, interrelating definitions spread over many remote dictionaries can affect the speed of transactions that involve changing, deleting, or creating new versions of related definitions. Change and delete operations are also dependent on a viable network link.

Figure 6–1: Placing Directories in Your Dictionary



Using VAX Rdb/VMS with VAX CDD/Plus 7

This chapter provides information about using CDD/Plus to create, change, and track the usage of dictionary definitions that may be shared by multiple Rdb/VMS databases. It discusses concurrent use of Rdb/VMS and CDD/Plus.

The features of CDD/Plus described in this chapter are available only for the definitions stored in CDO dictionary format. You can create dictionary definitions in a CDO dictionary when you use:

- CDO
- The CDD/Plus call interface
- RDO (Relational Database Operator)

Since dictionaries that are manipulated using the DMU Utility (DMU dictionaries) do not provide the features described in this chapter, information about converting definitions from DMU to CDO format is provided.

This chapter assumes that you are familiar with the dictionary concepts discussed in Chapter 1 and Chapter 3 of this manual and the basic concepts of Rdb/VMS.

7.1 Introduction to Using CDD/Plus with Rdb/VMS

Rdb/VMS is a relational database product that supports the features of CDO dictionaries. When you create CDO dictionary definitions for your database, other databases (and potentially other products) can share the same definitions. Storing definitions in one central repository helps to reduce redundancies. More importantly, sharing common dictionary definitions provides consistency of definitions across the databases and other software products as you develop an application.

Using CDD/Plus to store definitions in a CDO dictionary for your Rdb/VMS database allows you to:

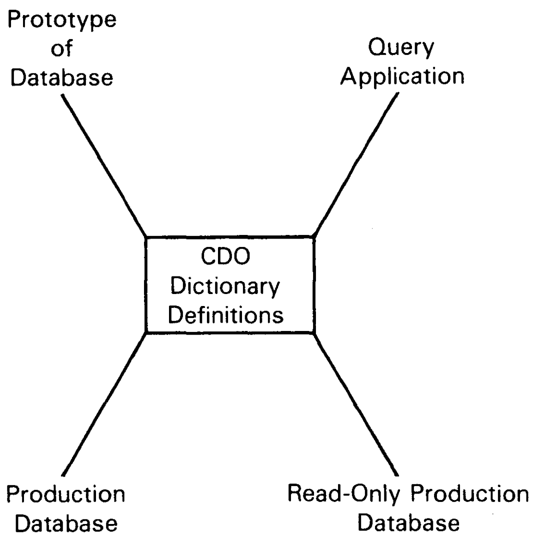
- Define fields and relations in your database based on dictionary definitions
- Share standard definitions among several databases
- Analyze the impact of changing shared definitions

In CDO dictionaries, CDD/Plus keeps track of all users of a particular dictionary definition. You can easily analyze the effects of changing a definition that is used throughout your application. Whenever a dictionary definition is changed, a message about the change is attached to the dictionary description of any Rdb/VMS databases that use the definition.

When you invoke an Rdb/VMS database, Rdb/VMS informs you if a message about a dictionary change is attached to the database. You can read the message from the CDO utility. Messages warn you that dictionary definitions may be inconsistent with database definitions. In this way, standard definitions can be created and maintained in a CDO dictionary, replicated in any number of Rdb/VMS databases, and potentially shared by other applications.

Figure 7-1 illustrates how CDO dictionary definitions can be shared by a prototype database, a production database, a read-only copy of the production database, and end-user applications.

Figure 7–1: Sharing Dictionary Definitions Among Database Products



ZK-7584-HC

7.1.1 Who Should Create Definitions in CDO?

You will benefit from creating and maintaining your data definitions in CDO if:

- You are developing an application where data definitions are shared among databases (and potentially other supporting products)
- You need to track the definitions that are shared among databases
- You require some measure of centralized control
- You anticipate that shared definitions will be changed

Creating definitions in the dictionary provides you with analysis tools and a method of controlling your shared definitions. However, not all applications benefit equally from these provisions. You need not create data definitions in the dictionary if:

- Your data definitions are specific to a single application
- No centralized control is required

If you are developing an application where data definitions are not shared among different storage sources and products, you may still choose to store definitions in the dictionary for completeness or for pieces tracking within that application.

Note

If you currently use Rdb/VMS with other products that do not support the features of CDO dictionaries, you should ensure that shareable definitions are created in your compatibility dictionary. For more details about supporting products and the compatibility dictionary, see Chapter 2.

7.1.2 Moving Between the RDO and CDO Utilities

You can create subprocesses and attach to them from both CDO and RDO. For example, from the CDO utility, you can use the SPAWN command to create a subprocess and invoke the RDO utility. In the subprocess from RDO, you can proceed to enter statements at the RDO> prompt. When you need to return to the CDO utility, you can use the $ATTACH statement at the RDO> prompt and specify the name of your original process. Remember to use the RDO statement COMMIT or ROLLBACK before leaving RDO so that you do not lock dictionary definitions.

You should be aware that RDO creates an additional subprocess each time you issue a $ATTACH statement. Therefore, you may quickly exceed your quota of subprocesses if you continue to issue the $ATTACH statement from RDO. A workaround exists to avoid running out of subprocesses. When you want to reattach to the process running the RDO statement $ATTACH, you must specify the second process name and not the first.

In the following example, the original process name is ACCOUNT. From CDO, a subprocess ACCOUNT_1 is spawned to invoke RDO. From RDO, the $ATTACH statement is used to attach to the original process ACCOUNT. However, to reattach to the process running RDO, the secondary process name, ACCOUNT_2, is specified.

```
CDO> !
CDO> !Spawn out of CDO and invoke RDO
CDO> !
CDO> SPAWN $RDO
    .
    .
    .
RDO> !
RDO> !Commit transactions before attaching to CDO process again
RDO> !
RDO> COMMIT
RDO> !
RDO> !Return to original process
RDO> !
RDO> $ATTACH ACCOUNT
CDO>
    .
    .
    .
CDO> !
CDO> !Reattach to the secondary subprocess created by the $ATTACH statement
CDO> !
CDO> ATTACH ACCOUNT_2
RDO>
    .
    .
    .
RDO> COMMIT
RDO> $ATTACH ACCOUNT
CDO>
    .
    .
    .
CDO> ATTACH ACCOUNT_2
RDO>
    .
    .
    .
```

The following sections discuss how to proceed when you have a large application that uses the dictionary to track definitions that are used in more than one storage location.

7.2 Creating a Database with Shareable CDO Definitions

When you define a new Rdb/VMS database, you can optionally specify a dictionary directory where a description of your database is stored. You can create more than one database in the same dictionary directory. This allows you to share definitions among several databases and thereby reduce redundancies. As no two objects in the same dictionary directory can have the same name (except different versions of the same definition), storing the definitions for several databases in the same CDD/Plus directory encourages sharing rather than duplication.

The following list shows the recommended procedure to follow when you intend to share dictionary definitions among two or more *new* databases, or more than one supporting product:

1. Define the fields and records that you expect to be shared from the CDO utility with the screen editor or with the DEFINE FIELD and DEFINE RECORD commands, (Section 7.2.1).
2. Define your database with the RDO statement DEFINE DATABASE and create a description of it in the dictionary (Section 7.2.3).
3. Include the shareable CDO definitions into the database with the RDO statement DEFINE...FROM PATHNAME (Section 7.3).
4. Complete your database design with RDO statements to create views, indices, and constraints (Section 7.3.4).
5. Load and test your database with data, as discussed in the Rdb/VMS documentation.
6. Analyze the impact of changes to definitions with the CDO pieces tracking commands (Section 7.4.3).
7. Make necessary changes to dictionary definitions with CDO commands or RDO statements (Section 7.4).
8. Integrate your changed dictionary definitions with the copies in the database (Section 7.4).
9. When necessary, repeat step 7, followed by step 8.

Setting up a database requires you to consider many issues that are beyond the scope of this manual. For more information about designing a database, see the VAX Information Architecture manual, *Introduction to Database Development*, and the *VAX Rdb/VMS Guide to Database Design and Definition.*

7.2.1 Defining Entities in CDO

For applications to share the same definitions, you should first create these definitions in a CDO dictionary. Dictionary definitions created in the CDO utility can be shared among the various applications that use them. You can create definitions in CDO with the screen editor or with the CDO command DEFINE. Section 3.4 describes the CDO editor; examples of the DEFINE command are provided in Chapter 4 and in this chapter.

The examples in this chapter assume that specified dictionary definitions are contained in the compatibility dictionary; therefore, CDD$TOP is used as the dictionary origin. If you do not use the compatibility dictionary, you should explicitly specify the anchor of the dictionary instead of using CDD$TOP.

When you invoke CDO, your default is set to the translation of the logical name CDD$DEFAULT. Unless you change your default, any subsequent data definitions will be created there. You can change your default while in the CDO utility with the CDO command SET DEFAULT. Chapter 3 explains how to use search lists with the SET DEFAULT command.

You can also change your default dictionary while you are within RDO with the SET DICTIONARY statement, as follows:

```
CDO> !
CDO> !Display dictionary default from CDO
CDO> !
CDO> SHOW DEFAULT
CDD$DEFAULT
= SYS$COMMON:[CDDPLUS]
CDO> !
CDO> !Exit CDO and invoke RDO
CDO> !
CDO> EXIT
$ RDO
RDO> !
RDO> !Display dictionary default from RDO
RDO> !
RDO> SHOW DICTIONARY
The current CDD dictionary is SYS$COMMON:[CDDPLUS]
RDO> !
RDO> !Change dictionary default
RDO> !
RDO> SET DICTIONARY SYS$COMMON:[CDDPLUS]PERSONNEL
RDO> !
RDO> !Display new dictionary default
RDO> !
RDO> SHOW DICTIONARY
The current CDD dictionary is  SYS$COMMON:[CDDPLUS]PERSONNEL
RDO> !
```

You can create shareable field and record definitions with the CDO editor. The editor is menu-driven and easy to use (see Section 3.4). You can also define shareable field definitions with the CDO command DEFINE FIELD. For example:

```
CDO> SET DEFAULT CDD$TOP.PERSONNEL
CDO> DEFINE FIELD EMPLOYEE_ID
cont>    DATATYPE IS TEXT
cont>    SIZE IS 7.
CDO> DEFINE FIELD ADDRESS_DATA_1
cont>    DATATYPE IS TEXT
cont>    SIZE IS 20.
```

You define the shared relations for Rdb/VMS databases and other products with the CDO command DEFINE RECORD. For example:

```
CDO> SET DEFAULT CDD$TOP.PERSONNEL
CDO> DEFINE RECORD NEW_HIRE.
cont>    EMPLOYEE_ID.
cont>    LAST_NAME.
cont>    FIRST_NAME.
cont>    ADDRESS_DATA_1.
cont>    ADDRESS_DATA_2.
cont>    CITY.
cont>    STATE.
cont>    POSTAL_CODE.
cont>    STATUS_CODE.
cont> END NEW_HIRE RECORD.
```

Only those fields and relations that you intend to be shared need be defined in CDO. Other fields and relations can be defined with RDO statements, as well as the views, constraints, and indices that are local to your database.

Before proceeding further, you should be aware of the differences between definitions that you create in CDO and those that you create in RDO.

When you create a definition in CDO, you specify the exact dictionary directory and the name for the definition; this is known as the directory name or pathname.

For different applications to share definitions, the definitions must be named and placed in a known CDO dictionary directory. The directory name acts as a pointer to the storage location of the actual entity definition. Applications access an entity definition by specifying the definition's directory name—for example, CDD$TOP.PERSONNEL.EMPLOYEE_ID. The name of the entity itself is known as the processing name.

When you define a database in RDO and specify a directory name, Rdb/VMS places copies of your database definitions in the dictionary as you create them.

If definitions are placed in the dictionary using the RDO statement DEFINE without the FROM PATHNAME keywords, the definitions are held within the database structure and are given only a processing name. (They are not given a directory name.) For example, although the database DEPT1 contains many definitions that are stored in the dictionary, these definitions are not listed among the contents of the dictionary directory at the given path:

```
CDO> DIRECTORY SYS$COMMON:[CDDPLUS]PERSONNEL.*

Directory SYS$COMMON:[CDDPLUS]PERSONNEL

DEPT1                         CDD$DATABASE
```

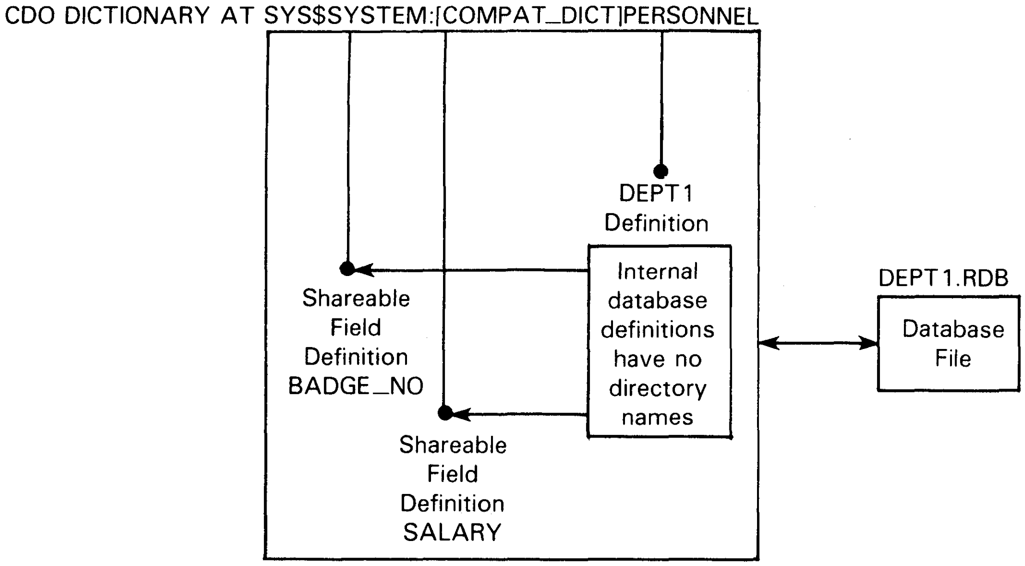
SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1 is the path name of the database; however, the dictionary definitions within the database DEPT1 have no directory name. Definitions created using RDO have no dictionary directory names and are not listed by the CDO command DIRECTORY.

When a definition is created using the CDO utility, the definition is given both a directory and a processing name. Therefore, when you intend to share definitions, create them with the CDO utility. After you create fields and records in CDO, you can include them in your databases using the RDO statements DEFINE FIELD and DEFINE RELATION with the FROM PATHNAME keywords, as described in Section 7.3.

If you create a definition in RDO, then later decide that you want it to be shareable, you can create a directory name for the definition with the ENTER command from CDO. The ENTER command is discussed in Section 7.6.

Figure 7-2 shows that dictionary copies of database definitions created with RDO statements are stored only within the database structure. Internal database definitions can be addressed only by traversing the database structure; whereas, the fields with directory names, such as BADGE_NO and SALARY, can be named, listed, and accessed directly by other applications.

Figure 7–2: Shareable CDO Definitions


CDO DICTIONARY AT SYS$SYSTEM:[COMPAT_DICT]PERSONNEL
DEPT1 Definition
Internal database definitions have no directory names
Shareable Field Definition BADGE_NO
Shareable Field Definition SALARY
DEPT1.RDB
Database File

ZK-7585-HC

Because definitions created inside an Rdb/VMS database have no directory name, they are not listed by a DIRECTORY command at the CDO> prompt; only the database description and the shareable fields are listed by the CDO command DIRECTORY. For example:

```
CDO> DIRECTORY CDD$TOP.PERSONNEL.*

Directory SYS$COMMON:[CDDPLUS]PERSONNEL

BADGE_NO                          FIELD
SALARY                            FIELD
DEPT1                             CDD$DATABASE
```

7.2.2 Displaying Entities and Attributes Used By Databases

To find out the names of the entities that a database uses, use the SHOW USED_BY /FULL command. In the following example, the /TYPE qualifier is used to limit the listing to field definitions used by the DEPT1 database:

```
CDO> SHOW USED_BY /TYPE=(FIELD) /FULL DEPT1

Members of SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1;1
|   SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_ID;1  (Type : FIELD)
|   |   via CDD$RDB_DATA_ELEMENT         (1 unshown intermediate node)
|   SYS$COMMON:[CDDPLUS]PERSONNEL.BADGE;1        (Type : FIELD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS (2 unshown intermediate nodes)
|   SYS$COMMON:[CDDPLUS]PERSONNEL.FIRST_NAME;1  (Type : FIELD)
|   |   via CDD$RDB_DATA_ELEMENT         (1 unshown intermediate node)
    .
    .
    .
```

The example has been truncated; the actual command output would also display system relations. The *unshown intermediate nodes* represent the number of intervening entities between the current entity and the entity named in the preceding indentation level. To display more information about an internal database entity, such as an index or constraint, use the SHOW GENERIC command. For details about SHOW GENERIC, see the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

If you use a product that only accesses DMU format dictionaries, you can list field and record definitions in a database using the DMU command LIST. The following LIST command displays the names of the relations in the DEPT1 database:

```
DMU> LIST CDD$TOP.DEPT1.RDB$RELATIONS.*
```

The definitions that are internal to the database structure only have processing names and, therefore, cannot be shared without traversing the database structures. However, from CDO, you can view the attributes of existing field, record and view definitions within the database structure with the SHOW FIELD and SHOW RECORD commands, provided that you know the processing name for the definitions.

The FROM DATABASE clause enables you to specify the database processing name. Note that wildcard characters are invalid with the SHOW...FROM DATABASE command; therefore, you can only display one field definition at a time with this command.

For example, if the field FIRST_NAME is defined for the database with RDO statements, a DIRECTORY command would not list the field definition. However, the SHOW FIELD command with the FROM DATABASE clause allows you to display the attributes for the field from CDO.

```
CDO> SHOW FIELD FIRST_NAME FROM DATABASE DEPT1

Definition of field FIRST_NAME
|   Datatype          text size is 12
```

You can display the attributes for known database relations with the SHOW RECORD command. For example, the following command displays the record EMPLOYEES in the database DEPT1:

```
CDO> SHOW RECORD EMPLOYEES FROM DATABASE DEPT1

Definition of record EMPLOYEES
|  Contains field     EMPLOYEE_ID
|  Contains field     LAST_NAME
|  Contains field     FIRST_NAME
|  Contains field     MIDDLE_INIT
|  Contains field     ADDRESS_DATA1
|  Contains field     ADDRESS_DATA2
|  Contains field     BIRTHDAY
```

You should be aware that CDO definitions are provided with a default access control list containing the following two entries:

1. The creator (owner) of the entity is given all access rights, including CONTROL. For example:

   ```
   (IDENTIFIER=[JONES],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+CHANGE
                                +DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
   ```

2. All other users are given only SHOW, ADMINISTRATOR, and OPERATOR access rights to the entity:

   ```
   (IDENTIFIER=[*,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW
                              +ADMINISTRATOR+OPERATOR)
   ```

For more information about protecting dictionary definitions, see Chapter 5.

7.2.3 Defining a New Database

In addition to shareable field and record definitions, the dictionary can contain a description, or definition of a database. To create an Rdb/VMS database description in the dictionary, use the RDO statement DEFINE DATABASE and specify the CDD/Plus dictionary path where you want the database definition to be located.

Note

Do not confuse the RDO statement DEFINE DATABASE with the CDO command DEFINE DATABASE. For CDD/Plus V4.0, the CDO DEFINE DATABASE command is currently restricted to creating an *RMS* database. See Section 8.3 for further information about DEFINE DATABASE.

In the following example, two database definitions—DEPT1 and DEPT2—are created within the directory at CDD$TOP.PERSONNEL using the RDO utility.

```
RDO> DEFINE DATABASE DEPT1
cont>  IN 'CDD$TOP.PERSONNEL'.
RDO> DEFINE DATABASE DEPT2
cont>  IN 'CDD$TOP.PERSONNEL'.
```

Note that the database definitions need not be in the same dictionary directory as your shareable field and record definitions. CDD/Plus provides shared access; therefore, you can include CDD/Plus definitions from other dictionary directories as well as from remote dictionaries.

Caution

Locking problems can occur when multiple users write to a dictionary. For example, when one user updates metadata using RDO, the database indices are locked against other users. Locking problems can also occur when users access multiple Rdb/VMS databases maintained in the same dictionary location. Shared dictionary definitions do not generate this problem and can be stored together in any dictionary location. To avoid these locking problems:

- Place databases in separate dictionaries whenever interactive RDO sessions will be used to update the metadata
- In RDO, use the COMMIT or ROLLBACK statement as often as possible to release locks

- Perform large updates, such as with the INTEGRATE and RESTORE statements, as after-hours batch jobs

---

CDO offers two ways to display the Rdb/VMS database definitions in your dictionary.

- Use the DIRECTORY command to list the Rdb/VMS database definition(s) in your default directory, as shown in Section 7.2.2.
- Use the SHOW DATABASE command to list the database name, filename, and fully qualified pathname of one or more Rdb/VMS database definitions.

The SHOW DATABASE command accepts the same optional qualifiers that can be used with SHOW FIELD and SHOW RECORD. In the following example, SHOW DATABASE is used with the /FULL qualifier to display the database name, file name, and fully qualified path name of the database definition.

```
CDO> SHOW DATABASE/FULL DEPT5
Definition of database  DEPT5
|   database uses RDB database DEPT5
|   database in file dept5
|   |   fully qualified file SYS$COMMON:[COMPAT_DICT.PERSONNEL]DEPT5.RDB;
```

For more information about SHOW DATABASE, see Section 3.7.2 or *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

7.2.4 Requiring the Dictionary for Database Definitions

In order to maintain consistent data definitions and to facilitate pieces tracking, a database administrator can create a database that requires the dictionary. When the keywords DICTIONARY IS REQUIRED are specified in the DEFINE DATABASE statement, subsequent invocations of the database that do not specify the dictionary path name will be invalid. This feature enables the database administrator to enforce dictionary usage.

If a database has been defined with DICTIONARY IS REQUIRED, you must specify the dictionary location to add or change definitions. You must use the PATHNAME keyword instead of the FILENAME keyword to invoke a database that requires the dictionary, as shown in the following example:

```
RDO> !
RDO> !Create a database, dictionary required
RDO> !
RDO> DEFINE DATABASE CANDIDATES DICTIONARY IS REQUIRED.
RDO> SHOW DATABASE
Database with db_handle CANDIDATES in file CANDIDATES
       CDD is being maintained.
RDO> FINISH
RDO> !
RDO> !Invoke by pathname and try to add a field -- Should succeed
RDO> !
RDO> DATABASE PATHNAME CDD$TOP.PERSONNEL.CANDIDATES
RDO> SHOW DATABASE
Database with filename CANDIDATES
       CDD is being maintained.
RDO> !
RDO> DEFINE FIELD COLLEGE
cont> DATATYPE IS TEXT SIZE IS 30.
RDO> ROLLBACK
RDO> FINISH
RDO> !
RDO> !Invoke by filename and try to add a field -- Should fail
RDO> !
RDO> DATABASE FILENAME CANDIDATES
RDO> SHOW DATABASE
Database with filename CANDIDATES
RDO> !
RDO> DEFINE FIELD COLLEGE
cont> DATATYPE IS TEXT SIZE IS 30.
%RDO-W-NOCDDUPDAT, database invoked by filename, the CDD will
not be updated
%RDO-E-NO_META_UPDATE, metadata update failed
-RDMS-F-CDDISREQD, CDD required for metadata updates is not
being maintained
RDO> ROLLBACK
RDO> FINISH
```

7.3 Including Dictionary Definitions in Your Database

If you have shareable field and record definitions already stored in one or more CDD/Plus dictionaries, you can include them in your databases.

To access dictionary definitions for your database, you must invoke the database and specify the dictionary location of the database description with the keyword PATHNAME. To include shareable definitions, specify the dictionary paths for the field and relation definitions with the keywords FROM PATHNAME in the RDO statements DEFINE FIELD and DEFINE RELATION.

In the following example, two databases are invoked specifying the dictionary path. Each database includes a record that was originally defined in CDO. Note that you must use an RDO FINISH command before invoking a different database.

```
RDO> INVOKE DATABASE PATHNAME
cont> 'CDD$TOP.PERSONNEL.DEPT1'
RDO> !
RDO> DEFINE RELATION EMPLOYEES
cont> FROM PATHNAME
cont> 'CDD$TOP.PERSONNEL.EMPLOYEES'
cont> END EMPLOYEES RELATION.
RDO> !
RDO> COMMIT
RDO> FINISH
RDO> INVOKE DATABASE PATHNAME
cont> 'CDD$TOP.PERSONNEL.DEPT2'
RDO> !
RDO> DEFINE RELATION FULL_NAME
cont> FROM PATHNAME 'CDD$TOP.PERSONNEL.FULL_NAME'
cont> END FULL_NAME RELATION.
RDO> COMMIT
RDO> FINISH
```

Field and record definitions in the dictionary are stored in a format that can be interpreted appropriately by different products. Rdb/VMS searches through all attributes in a dictionary definition. When Rdb/VMS encounters an attribute that is not valid for Rdb, it issues an error. For example, a data type of UNSIGNED QUADWORD is invalid for Rdb/VMS. Attributes such as NAME FOR COBOL are simply ignored by Rdb/VMS. For a complete list of valid Rdb/VMS attributes, see the chapter on field attributes in the manual *VAX Rdb/VMS Reference Manual.*

In your Rdb/VMS database, you can use dictionary definitions from another dictionary directory, from a different dictionary anchor, or from a CDD/Plus dictionary on another node. To include a dictionary definition from other than your default dictionary directory, specify the fully qualified name for the definition. For example:

```
RDO> DEFINE RELATION SALARY_HISTORY
cont>         FROM PATHNAME FARWAY::SYS$DISK:[MCKAY.DICTIONARY]EMPLOYEES.SALARY_HISTORY
cont>         END.
RDO> COMMIT
```

Note

When an RDO transaction terminates with an error, some operations dictionary may have completed successfully while at least one operation has failed. If such a transaction occurs when you are accessing a database invoked with the dictionary path name, this mixture results in the dictionary and the database becoming inconsistent. Therefore, when a transaction causes an error, you *must* roll back the database with the RDO statement ROLLBACK. The dictionary and database will return to their previously consistent state after you issue the ROLLBACK statement.

Definitions created in CDO are provided with the default protection provisions discussed in Section 7.2.1. Briefly, the definition owner is granted all rights including CONTROL and all other users are granted SHOW access. Users with CONTROL access can change and add new ACEs to the default dictionary ACL. Once you have included copies of the shareable dictionary definitions in your database, you can further fine tune the protection provisions with RDO statements, as necessary.

The FROM PATHNAME keywords in the RDO DEFINE FIELD and DEFINE RELATION statements allow you to specify dictionary definitions in only CDO format. To take advantage of pieces tracking, you must convert the DMU definitions to CDO format (see Section 7.6).

7.3.1 Naming Database Definitions

Database, field, and record names must be unique within a database: two entities with the same directory name cannot exist in the same subdirectory. Therefore, a new field or relation cannot be included from the dictionary into the database if another entity with the same name already exists. When you include a dictionary record into the database and one of the fields in the record is contained in the database already, RDO does not create a second copy of the field definition, provided that the two definitions are equal.

You can use either CDO utility or the RDO utility to find out if a name is already in use by a database. For information about using the RDO utility, see the *VAX Rdb/VMS Guide to Data Manipulation.* From CDO, specify the name you are concerned about with the SHOW FIELD or SHOW RECORD command and use the FROM DATABASE keywords. If the entity name you specify is not contained in the database, CDO issues an informational message.

In the following example, CDO displays that the field ID_NUMBER is in use by the database while the field ID_NO is not:

```
CDO> DIR

Directory SYS$COMMON:[CDDPLUS]PERSONNEL

DEPT2;1                                   CDD$DATABASE
ID_NUMBER;1                               FIELD
ID_NO;1                                   FIELD

CDO> SHOW FIELD ID_NUMBER FROM DATABASE DEPT2
Definition of field ID_NUMBER
|   Datatype is text     size is 7

CDO> SHOW FIELD ID_NO FROM DATABASE DEPT2
%CDO-E-NOTFOUND, entity ID_NO not found in dictionary
```

You cannot rename a dictionary definition as you include it into a database. However, you can create additional shareable definitions in the dictionary to resolve duplicate names. For example, suppose that you want to use the shareable dictionary field SOC_SEC_NUM in your database. The SHOW FIELD...FROM DATABASE command displays that a field with this name is already in use within the database. The attributes of the database SOC_SEC_NUM are different from the dictionary definition by the same name.

Suppose that you do want to use the shareable dictionary definition named SOC_SEC_NUM rather than the database field. You cannot include the dictionary field as it is because the name already exists in the database and you cannot rename the dictionary definition as you include it into the database. Instead, you can create another field, DRIVE_LIC, in the dictionary using the BASED ON attribute. (When naming definitions, you should consider your organization's naming standards.) Since the name DRIVE_LIC is not used in the database, you can include it instead of SOC_SEC_NUM. For example:

```
CDO> SHOW FIELD SOC_SEC_NUM
Definition of field SOC_SEC_NUM
|   Datatype is text     size is 11
|   Edit string is 999"-"99"-"9999.
CDO> !
CDO> SHOW FIELD SOC_SEC_NUM FROM DATABASE DEPT1
Definition of field SOC_SEC_NUM
|   Datatype is text     size is 11
CDO> !
CDO> !Test another name
CDO> !
CDO> SHOW FIELD DRIVE_LIC FROM DATABASE DEPT1
%CDO-E-NOTFOUND, entity DRIVE_LIC not found in database
SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1;1
CDO> !
CDO> !Define a new field in CDO
CDO> !
CDO> DEFINE FIELD DRIVE_LIC
cont> BASED ON SOC_SEC_NUM.
CDO> SPAWN $RDO
RDO> !
RDO> !Include the new field into the database
RDO> !
RDO> DEFINE FIELD DRIVE_LIC
cont> FROM PATHNAME 'CDD$TOP.PERSONNEL.DRIVE_LIC'.
RDO> COMMIT
```

If the dictionary field SOC_SEC_NUM was included in a dictionary record definition that you intend to include, you must make a new version or change the record definition to include the field DRIVE_LIC and use the changed record within your database. Otherwise, the database will continue to attempt to include the dictionary definition SOC_SEC_NUM.

Suppose that you also require the fields BADGE_NO and EMPLOYEE_ID with the same attributes as SOC_SEC_NUM. You can define the fields BADGE_NO and EMPLOYEE_ID based on DRIVE_LIC. All four dictionary definitions are shareable.

```
CDO> DEFINE FIELD BADGE_NO
cont> BASED ON DRIVE_LIC.
CDO> DEFINE FIELD EMPLOYEE_ID
cont> BASED ON SOC_SEC_NUM.
CDO> SPAWN $RDO
!
!Include the fields into the database
!
RDO> DEFINE FIELD BADGE_NO
cont> FROM PATHNAME 'CDD$TOP.PERSONNEL.BADGE_NO'.
RDO> DEFINE FIELD EMPLOYEE_ID
cont> FROM PATHNAME 'CDD$TOP.PERSONNEL.EMPLOYEE_ID'.
```

Alternatively, you can use the single field DRIVE_LIC in many relations that you create for your database from within RDO. However, these relations would not be directly related to the dictionary record definitions that use SOC_SEC_NUM. It is preferable to include dictionary record definitions where possible so that the dictionary can track the field and record usage within your database.

7.3.2 Duplicate Processing Names

Definitions created with CDO commands always have a directory name that is the same as the processing name. However, if you use definitions that have been created with direct calls to the CDD/Plus call interface, it is possible that a definition has a processing name that is not the same as its directory name. When a definition's directory name and processing name do not match, a naming problem can occur within the database.

When a field in the database has a processing name that matches the directory name of the field you are trying to include from the dictionary, RDO searches for an existing field with the directory name you specify. If a duplicate name is found, RDO compares the two definitions as follows:

If the two fields have the same processing name, then RDO considers them to be equal, no error is issued, and you can proceed with your definitions. If the two fields do not have the same processing name, then RDO considers them to be unequal, issues an error that the field already exists, and does not include the dictionary field. (It is possible that these two definitions are actually equal despite having different processing names.) To resolve this issue, you must rename the definition from the CDO utility, as described in Section 7.3.1.

7.3.3 Defining Relations Using Dictionary Definitions

When you define a relation in RDO, you can create the internal fields within the relation definition. However, when you define a record in the dictionary, the field definitions must already exist. You cannot implicitly define the fields as you include them in a dictionary record definition. This difference requires you to choose your method of including dictionary records with some care. Keep in mind the fact that you may already have included into your database one or more of the fields in the dictionary record definition.

You can define a relation for the database using dictionary definitions in the following ways:

- Include an existing dictionary record definition with the FROM PATHNAME keywords in the DEFINE RELATION statement. For example:

  ```
  RDO> DEFINE RELATION FULL_NAME FROM PATHNAME 'CDD$TOP.PERSONNEL'.
  ```

  RDO searches the database to see whether or not the fields for the relation are already included. If the fields are not already included in the database, RDO includes them for you.

  If RDO finds a field with the same name as a field required for the relation, it then checks to see if the fields are equal. If the fields are equal, then RDO proceeds to include the relation. If the fields by the same name are not equivalent, RDO issues an error message and the relation definition is not included. You must resolve any such naming conflict yourself before you re-try the operation. (See Section 7.3.2.)

- Include the existing shareable fields from the dictionary yourself and then include the dictionary record definition for the relation. For example:

  ```
  RDO> DEFINE FIELD FIRST_NAME FROM PATHNAME
  cont>     'CDD$TOP.PERSONNEL.FIRST_NAME'.
  RDO> DEFINE FIELD MIDDLE_INIT FROM PATHNAME
  cont>     'CDD$TOP.PERSONNEL.MIDDLE_INIT'.
  RDO> DEFINE FIELD LAST_NAME FROM PATHNAME
  cont>     'CDD$TOP.PERSONNEL.LAST_NAME'.
  RDO> !
  RDO> !Include the dictionary record
  RDO> !Fields are already in database
  RDO> !
  RDO> DEFINE RELATION FULL_NAME
  cont> FROM PATHNAME 'CDD$TOP.PERSONNEL.FULL_NAME'.
  ```

The FROM PATHNAME keywords are invalid within internal field definitions inside a relation definition. For example, the following RDO DEFINE statement is invalid and generates an error:

```
RDO> DEFINE RELATION FULL_NAME.
       FIRST_NAME FROM PATHNAME 'CDD$TOP.PERSONNEL.FIRST_NAME'.
                  ^
%RDO-W-LOOK_FOR_STT, syntax error, looking for:
%RDO-W-LOOK_FOR_CON,      BASED, VALID, period, DATATYPE,
%RDO-W-LOOK_FOR_CON,      COMPUTED, QUERY_NAME, EDIT_STRING,
%RDO-W-LOOK_FOR_CON,      QUERY_HEADER, DEFAULT_VALUE,
%RDO-W-LOOK_FOR_CON,      MISSING_VALUE
%RDO-F-LOOK_FOR_FIN,   found FROM instead
```

You cannot define a relation from a CDO record that contains other CDO records. Since RDO relations can contain only CDO fields, the RDO DEFINE statement in the following example is invalid and generates an error:

```
CDO> SHOW RECORD NEW_HIRE
Definition of record NEW_HIRE
|   Contains record         FULL_NAME
|   Contains field          ADDRESS_DATA_1
|   Contains field          ADDRESS_DATA_2
CDO> EXIT
$ !
$ RDO
RDO> INVOKE DATABASE PATHNAME 'CDD$TOP.PERSONNEL'
RDO> !
RDO> !Fields are already in database
RDO> !Record FULL_NAME is already in database as a relation
RDO> !
RDO> SHOW FIELDS
User Fields in Database with pathname  SYS$COMMON:[CDDPLUS]PERSONNEL;1
     ADDRESS_DATA_1                  text size is  25
     ADDRESS_DATA_2                  text size is  20
     EMPLOYEE_ID                     text size is  5
     FIRST_NAME                      text size is  10
     LAST_NAME                       text size is  14
     MIDDLE_INIT                     text size is  1
     SALARY                          signed longword scale  -2
RDO> SHOW RELATIONS
User Relations in Database with pathname  SYS$COMMON:[CDDPLUS]PERSONNEL;1
     CURRENT_INFO                    A view.
     FULL_NAME
     JOB_HISTORY
     SALARY_HISTORY
RDO> !
RDO> !Try to define a relation from a record that contains a record
RDO> !
RDO> DEFINE RELATION NEW_HIRE FROM PATHNAME
cont> 'CDD$TOP.PERSONNEL.NEW_HIRE'
%CDD-E-DTYPE_REQUIRED, field must have a datatype for inclusion
in an Rdb database
```

7.3.4 Completing Your Database Design with RDO Statements

Complete your database definition by defining the views, indices, and constraints with RDO statements. Currently, view, index, and constraint definitions can be stored in the dictionary, but only the view definitions are shareable. The definitions created in RDO are automatically stored in the dictionary at the path you specify, provided that you invoke the database with the PATHNAME keyword.

The following RDO statements create a sample index, view, and constraint for the DEPT1 database. You can find tutorial and reference information on these statements in the Rdb/VMS documentation.

```
RDO> DEFINE INDEX JOB_HISTORY_HASH
cont>   DESCRIPTION IS /* Hash index for job_history */
cont>   FOR JOB_HISTORY
cont>   DUPLICATES ARE ALLOWED
cont>   STORE USING EMPLOYEE_ID
cont>     WITHIN
cont>     EMPIDS_LOW WITH LIMIT OF "00200";
cont>     EMPIDS_MID WITH LIMIT OF "00400";
cont>     EMPIDS_OVER;
cont>   TYPE IS HASHED.
cont>   EMPLOYEE_ID.
cont> END JOB_HISTORY_HASH.
RDO> DEFINE VIEW CURRENT_JOB
cont> OF JH IN JOB_HISTORY.
cont>   CROSS E IN EMPLOYEES OVER EMPLOYEE_ID
cont>      WITH JH.JOB_END MISSING.
cont>       E.LAST_NAME.
cont>       E.FIRST_NAME.
cont>       E.EMPLOYEE_ID.
cont>       JH.JOB_CODE.
cont>       JH.DEPARTMENT_CODE.
cont>       JH.SUPERVISOR_ID.
cont>       JH.JOB_START.
cont> END VIEW.
RDO> DEFINE CONSTRAINT EMPLOYEE_ID_REQUIRED
cont>     FOR E IN EMPLOYEES
cont>     REQUIRE NOT E.EMPLOYEE_ID MISSING.
RDO> COMMIT.
RDO> FINISH
```

You cannot currently define shareable database indices, views, and constraints with CDO commands. (See Section 7.2.1 for information about displaying database views from CDO.)

7.3.5 Referencing Definitions from Programs

You can use RDO to test statements before adding them to programs. You do not need to give complete data descriptions in your program, because programs can locate dictionary definitions if you just reference the dictionary, using the syntax appropriate for the programming language.

Programs can simplify data entry. For example, if you want to store several records in the database, RDO would require that you enter the STORE statement for each record that you want to store. A program can contain a loop that repeatedly executes the STORE statement so that the end user only has to provide input data in order to store records.

A BASIC program can read records from CDO format dictionaries, and can read records with a STRUCTURE clause from DMU format dictionaries or the compatibility dictionary. The BASIC compiler can write a compiled module entity into a CDO dictionary. The **compiled module** entity (CDD$COMPILED_MODULE) represents the object file (.OBJ) that the compiler produces. (If you specify the /NOOBJECT qualifier on the command line, or if the program has compilation errors, a compiled module entity is not generated.) BASIC can then create dictionary relationships between this compiled module entity and the dictionary definitions that the BASIC program uses.

To reference a dictionary definition from a BASIC program, use the %INCLUDE %FROM %CDD directive. For example, the following BASIC program, called PAYROLL.BAS, references the dictionary definition EMPLOYEE_REC:

```
PROGRAM EMPLOYEE_PAYROLL
          .
          .
          .
%INCLUDE %FROM %CDD "SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_REC"
          .
          .
          .
END PROGRAM
```

In this example, the definition EMPLOYEE_REC is specified by dictionary pathname. You can optionally specify a field or relation by database pathname. For example:

- %INCLUDE %FROM %CDD "DEPT1.RDB$RELATIONS.EMPLOYEE_REC"
- %INCLUDE %FROM %CDD "DEPT1.RDB$FIELDS.NAME_FIELD"

If you want to create the relationship and the compiled module without copying the referenced object into the BASIC program, use the %REPORT directive. For example:

```
%REPORT %DEPENDENCY "SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_FORM" &
                    "CDD$COMPILED_DEPENDS_ON"
```

To create relationships from a VAX BASIC program, you must use the %INCLUDE or %REPORT directives in conjunction with the compilation qualifier /DEPENDENCY_DATA. For example:

```
$ BASIC/DEPENDENCY_DATA PAYROLL.BAS
```

As the result of this command, CDD/Plus:

1. Creates a compiled module entity called EMPLOYEE_PAYROLL in your current default dictionary directory
2. Copies the record SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_REC into the program
3. Creates a relationship between the compiled module EMPLOYEE_PAYROLL and the record EMPLOYEE_REC

If you do not specify /DEPENDENCY_DATA to the BASIC compiler, %INCLUDE directives can copy dictionary definitions, but no relationship will be created in the CDD. Programs can use dictionary objects without relationships, but relationships are needed for pieces tracking.

For more information about BASIC directives, see the *VAX BASIC User Manual.*

Rdb/VMS provides two precompilers to check and process data manipulation statements in programs: RDBPRE and RDML. These precompilers convert data manipulation language (DML) statements into a series of equivalent DIGITAL Standard Relational Interface (DSRI) calls to the database. You compile these calls with your application program.

The RDML preprocessor will operate with any DSRI compliant database management system. You can use the RDML/ADA, RDML/C, and RDML/PASCAL precompilers to process Ada, C, and PASCAL programs using statements very similar to RDO statements. For more information on RDML, see *RDML Reference Manual* and *VAX Rdb/VMS Guide to Programming.*

7.4 Changing and Integrating Definitions

You can change shareable definitions created in the CDO utility from either CDO or RDO. The CDO CHANGE command modifies the original dictionary definition. The RDO CHANGE FIELD or CHANGE RELATION statements modify the database copy of the definition.

Inconsistencies can develop between dictionary and database copies of the definitions if you change or add database definitions without changing the original data definition in the dictionary. CDD/Plus warns all users of the original definition with a message about the change. You can resolve these inconsistencies with the RDO statement INTEGRATE.

When the dictionary and database definitions are inconsistent, you must update the definitions with the RDO statement INTEGRATE. The INTEGRATE statement can update the database file to reflect the current state of the database description in the dictionary, *or* it can update the dictionary to reflect the current state of the definitions in the database file.

There are two formats for the INTEGRATE statement to accommodate these two activities. The following sections explain how and when to use each format.

Note

After an INTEGRATE operation is complete, you must specify the COMMIT or the ROLLBACK statement to save or abort the changes made by INTEGRATE.

7.4.1 Updating Dictionary Definitions with INTEGRATE IN

The INTEGRATE IN statement writes the latest definitions from the database file into the database description in the dictionary. The database file definitions replace the definitions in the dictionary. For example, the following statement writes the definitions in the database file with the database description DEPT1 into the dictionary at CDD$TOP.PERSONNEL:

```
RDO> INTEGRATE DATABASE DEPT1 IN PATHNAME 'CDD$TOP.PERSONNEL.DEPT1'
RDO> COMMIT
```

Any changes to definitions in the Rdb/VMS source database will now supersede the definitions in the dictionary description of the database. If any shareable dictionary definition changes, CDD/Plus attaches a message about the new definition version to all other dictionary definitions or databases that use the previous version. Users can adapt to the new version over time.

7.4.2 Updating Database Definitions with INTEGRATE FROM

The INTEGRATE FROM statement writes the latest changes of the definitions stored in the dictionary definition of the database into the actual database file.

```
RDO> START_TRANSACTION READ_WRITE
RDO> INTEGRATE DATABASE DEPT1 FROM PATHNAME 'CDD$TOP.PERSONNEL.DEPT1'
RDO> COMMIT
```

All definitions in the dictionary definition at CDD$TOP.PERSONNEL are written to the database file. If definitions exist in the dictionary that did not previously exist in the database file, these definitions are created in the database.

Caution

If there are definitions in the database file that do not also exist in the dictionary description of the database, these definitions are deleted during INTEGRATE FROM and data may be lost. Be sure to use the START_TRANSACTION statement prior to entering the INTEGRATE command so that you can roll back the database if necessary.

RDO automatically clears the messages attached to the database after the INTEGRATE statement is executed.

As the INTEGRATE statement executes, it displays on your screen the differences between the database and the dictionary and informs you if any deletions will occur during the integration. You are *not* prompted to confirm whether or not you want a particular definition to be deleted. In the following example, a new field has been added to the database file DEPT1 but not to the dictionary. When the database file is updated to integrate definitions from the dictionary, this definition is deleted. The user issues a ROLLBACK statement to restore the database to its condition before the field was deleted. Before integrating again, the user can define this field in the dictionary.

```
RDO> START_TRANSACTION READ_WRITE
RDO> INTEGRATE DATABASE DEPT1
RDO> FROM PATHNAME 'CDD$TOP.PERSONNEL.DEPT1'
%CDD-E-INT_DELETE, object NEW_FIELD will be deleted -- data may be lost
RDO> ROLLBACK
```

If you observe that definitions have been deleted when you want to keep them, you can use the RDO statement ROLLBACK to roll back the database to its state before you entered the INTEGRATE statement. Be sure to use the START_TRANSACTION statement prior to entering the INTEGRATE command.

The following sections describe how to change shareable dictionary definitions, how to evaluate the impact of these changes, and how to keep your dictionary and database definitions consistent after changes have been made.

7.4.3 Evaluating the Impact of Changes

From the CDO utility, the SHOW commands allow you to keep track of the dictionary entities that use a definition. These commands help you to analyze the effects of changing shareable dictionary definitions.

Suppose that you are considering a change to the definition EMPLOYEE_REC. To find out which definitions use EMPLOYEE_REC, issue the CDO command SHOW USES:

```
CDO> SHOW USES EMPLOYEE_REC
Owners of SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_REC;1
|   DEPT1                         (Type : CDD$RDB_DATABASE)
|   |   via CDD$DATA_AGGREGATE
|   DEPT2                         (Type : CDD$RDB_DATABASE)
|   |   via CDD$DATA_AGGREGATE
|   *** name is unspecified ***   (Type : CDD$DATA_INSTANCE)
|   |   via CDD$DATA_INSTANCE_PATH
```

CDO shows you that the databases DEPT1 and DEPT2 and an unspecified entity (a view) all use the record definition EMPLOYEE_REC. Therefore, if you change EMPLOYEE_REC, these entities will be affected in some way by the change.

To find out if any definitions will actually be inconsistent if you change a dictionary entity, use the SHOW WHAT_IF command. If you change the field EMPLOYEE_ID with the CHANGE command, the record definition EMPLOYEE_REC (which originally included the field EMPLOYEE_ID) automatically includes the changed field definition because the original definition has been changed. The definition in the database file will not be automatically updated; therefore, the database definition of EMPLOYEE_ID will be inconsistent with the definition in the dictionary. You can use the INTEGRATE FROM command to replace the definition in the database file with the dictionary definition.

```
!
!Display entities that will be inconsistent if EMPLOYEE_ID is changed
!
CDO> SHOW WHAT_IF /FULL EMPLOYEE_ID

SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1;1              CDD$DATABASE
SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT2;1              CDD$DATABASE
SYS$COMMON:[CDDPLUS]PERSONNEL.PT_TIME;1            CDD$DATABASE
```

If you change EMPLOYEE_ID, the database definitions listed by the SHOW WHAT_IF command are flagged with a message about the change. When the database is invoked, RDO informs you that a message is attached to the database in the dictionary. Messages are passive, *you* must take action to integrate the dictionary with the databases. Messages are illustrated in the following sections.

The following sections include examples of how to change shareable definitions and of using the various SHOW commands to manage the effects of these changes. The SHOW commands are also described in Chapter 6 of this manual and in the *VAX CDD/Plus Common Dictionary Operator Reference Manual.*

7.4.4 Changing Definitions from CDO

You can change dictionary definitions in two ways from the CDO utility:

- You can make an immediate change to an original dictionary definition with the CHANGE command
- You can create a new version of a dictionary definition and permit users to adapt to the change over time with the DEFINE command

Whether or not you use the CHANGE or the DEFINE command, if you change a definition, all databases that use the definition are flagged with a message about the possible inconsistency.

7.4.4.1 The Impact of Immediate Changes – If you make changes with the CHANGE command, the original definition is changed and no copy of the original is kept in the dictionary. The database is flagged with a message that the definitions in the dictionary and the database file are inconsistent. Once the dictionary and database are integrated, the database and dictionary definitions are consistent again.

The following example steps through the commands and displays that you can use before and after making an immediate change to a shared definition.

```
!
!Display entities that will be inconsistent if an immediate
!change is made to EMPLOYEE_ID
!
CDO> SHOW WHAT_IF /FULL EMPLOYEE_ID

SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1;1                 CDD$DATABASE
SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT2;1                 CDD$DATABASE
!
!Make the change
!
CDO> CHANGE FIELD EMPLOYEE_ID
cont> datatype is text size is 15.
```

```
!
!CDO issues warnings about possible inconsistencies
!
%CDO-I-DBMBR, database SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1;1
may need to be INTEGRATED
%CDO-I-DBMBR, database SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT2;1
may need to be INTEGRATED
!
!Read message about the change attached to the database
!
CDO> SHOW MESSAGES DEPT1
SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1;1 is possibly invalid,
triggered by entity SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_ID;1
!
!Spawn to a subprocess to integrate the database with new dictionary definitions
!
CDO> SPAWN $RDO
RDO> INVOKE DATABASE DEPT1
cont>  = PATHNAME 'SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1'
%RDO-I-CDOMESS, entity has messages
!
!Start a transaction so that you can roll back after the
!integrate operation, if necessary
!
RDO> START_TRANSACTION READ_WRITE
RDO> INTEGRATE DATABASE DEPT1
cont> FROM PATHNAME 'CDD$TOP.PERSONNEL.DEPT1'
    .
    .
    .
RDO> COMMIT
```

Remember that if the integrate operation deletes or creates definitions that you do not want, you can roll back the database to its state at the start of the transaction.

7.4.4.2 The Impact of Changes Over Time – To phase in a change more gradually, use the DEFINE command rather than the CHANGE command. The DEFINE command creates a new version of the definition and keeps the previous version in the dictionary. If you make new versions with the DEFINE command, entity definitions that use the previous version are flagged with a message that a new version exists. You may decide that you do not want to use a new version that exists in the dictionary. If you do not want to integrate the dictionary and database, you can clear the messages with the CDO command CLEAR MESSAGES. Other users can accommodate the new version at their own pace.

The following example creates a new version of the field EMPLOYEE_ID. Version 1 of EMPLOYEE_ID is used by the record definition EMPLOYEE_REC. When the record EMPLOYEE_REC was defined, EMPLOYEE_ID;1 was the highest version and, therefore, the relationship between EMPLOYEE_REC and EMPLOYEE_ID only exists between version 1 of each of these definitions. Version 2 of EMPLOYEE_ID is not automatically included in the definition of the record

EMPLOYEE_REC. To include the new version of the field EMPLOYEE_ID in EMPLOYEE_REC, you must change or redefine the record EMPLOYEE_REC.

```
!
!Create a new version of EMPLOYEE_ID
!
CDO> DEFINE FIELD EMPLOYEE_ID
cont>    DATATYPE IS TEXT
cont>    SIZE IS 7.
!
!New version of EMPLOYEE_ID is not used yet
!
CDO> SHOW USES /FULL EMPLOYEE_ID;2
%CDO-E-NOOWN, no owners found for entity
                             SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_ID;2
!
!Read any messages on EMPLOYEE_REC
!
CDO> SHOW MESSAGES EMPLOYEE_REC
SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_REC;1 uses an entity which has new
versions, triggered by entity SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_ID;1
!
!Create a new version of EMPLOYEE_REC to include version 2 of EMPLOYEE_ID
!A message about this new version is attached to all users of EMPLOYEE_REC;1
!
CDO> DEFINE RECORD EMPLOYEE_REC.
cont>    EMPLOYEE_ID.
cont>    FULL_NAME.
cont>    ADDRESS_DATA.
cont>    STATUS_CODE.
cont> END EMPLOYEE_REC RECORD.
!
!Confirm that version 2 of EMPLOYEE_ID is in use
!
CDO> SHOW USES EMPLOYEE_ID;2
Owners of SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_ID;2
SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_REC;2          (Type : RECORD)
|  via CDD$DATA_AGGREGATE_CONTAINS
!
!Spawn a subprocess and invoke RDO
!
CDO> SPAWN $RDO
```

```
    .
    .
    .
!
!Integrate the new dictionary version of EMPLOYEE_ID into DEPT1
!
RDO> INTEGRATE DATABASE DEPT1
cont> FROM PATHNAME 'CDD$TOP.PERSONNEL.DEPT1'
    .
    .
    .
RDO> COMMIT
!
!Go back to CDO utility
!
RDO> $ATTACH SMITH
!
!Confirm that DEPT1 now uses version 2 of EMPLOYEE_ID
!
CDO> SHOW USED_BY /TYPE=(FIELD) /FULL DEPT1
Members of SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1;1
|    SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_ID;2     (Type : FIELD)
!
!PARTS_INVENTORY still uses version 1 of EMPLOYEE_ID
!
CDO> SHOW USED_BY /TYPE=(FIELD) /FULL PARTS_INVENTORY
Members of SYS$COMMON:[CDDPLUS]PERSONNEL.PARTS_INVENTORY;1
|    SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_ID;1     (Type : FIELD)
```

Messages about the new versions are attached to all other users of the original versions of both EMPLOYEE_ID and EMPLOYEE_REC.

7.4.5 Changing Shareable Definitions from RDO

When you change a shareable CDD/Plus definition with the RDO statement CHANGE, the change is reflected in the database description in the dictionary. CDD/Plus then creates a new version of the dictionary definition. (The RDO statement CHANGE is equivalent to the CDO command DEFINE, not the CDO command CHANGE.) This allows other users of the changed dictionary definition to adapt to the change over time. Messages about the existing new version of the definition are attached to all users of the previous version, including all other databases. You must read the messages attached to an entity using the CDO command SHOW MESSAGES. (If the shareable definition you are changing is on a remote node, the network link must be viable; otherwise, the change is not permitted.)

In the following example, the database is invoked using the PATHNAME keyword and a shareable field definition is changed with the RDO statement CHANGE. Subsequent CDO commands show that a new version of the field definition has been created and that the appropriate messages have been generated.

```
CDO> SET DEFAULT CDD$TOP.PERSONNEL
CDO> DIRECTORY EMPLOYEE_ID
!
! Observe that only one version of EMPLOYEE_ID exists in dictionary
!
Directory SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_ID
EMPLOYEE_ID;1                            FIELD
!
!Show all users of EMPLOYEE_ID
!
CDO> SHOW USES /FULL EMPLOYEE_ID
Owners of SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_ID;1
SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_REC;1        (Type : RECORD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1;1   (Type : CDD$DATABASE)
|   |   via CDD$DATABASE_SCHEMA
SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT2;1   (Type : CDD$DATABASE)
|   |   via CDD$DATABASE_SCHEMA
!
!Spawn to a subprocess to work in RDO
!
CDO> SPAWN $RDO
!
!Invoke the database specifying the path name
!
RDO> INVOKE DATABASE DEPT1
cont>  = PATHNAME 'CDD$TOP.PERSONNEL.DEPT1'
!
!Establish your dictionary default for RDO
!
RDO> SET DICTIONARY CDD$TOP.PERSONNEL
!
!Change a field from RDO
!
RDO> CHANGE FIELD EMPLOYEE_ID
cont> DATATYPE IS TEXT SIZE IS 12.
!
!Return to the CDO process
!
RDO> $ATTACH HEINES
CDO> SHOW DEFAULT
 SYS$COMMON:[CDDPLUS]PERSONNEL
CDO> DIRECTORY EMPLOYEE_ID
!
! Confirm that a new version was created in dictionary
!
Directory SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_ID
EMPLOYEE_ID;2                            FIELD
EMPLOYEE_ID;1                            FIELD
```

```
!
! Confirm that there is a message about the new version
! attached to other users of EMPLOYEE_ID
!
CDO> SHOW MESSAGES EMPLOYEE_REC
SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_REC;1 uses an entity which has new
versions, triggered by entity SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_ID;1
```

Caution

If you invoke the database without using the PATHNAME keyword and subsequently change shareable definitions with RDO statements, a new version of the dictionary definition is *not* created. Therefore, your dictionary and database definitions will be inconsistent. It is up to you to issue the RDO statement INTEGRATE to update the dictionary copies of the database definitions. For example:

```
!
!Invoke the database without PATHNAME option
!
RDO> INVOKE DATABASE FILENAME DEPT1
!
!Change the relation EMPLOYEE_REC by adding
!globally defined field WAGE_CLASS
!
RDO> CHANGE RELATION EMPLOYEE_REC.
cont>     DEFINE WAGE_CLASS.
cont> END.
!
!Integrate the change in EMPLOYEE_REC into the dictionary
!
RDO> INTEGRATE DATABASE DEPT1
cont> IN PATHNAME 'CDD$TOP.PERSONNEL.DEPT1'
     .
     .
     .
RDO> COMMIT
```

Confirm that the new version of EMPLOYEE_REC is created in the dictionary with CDO commands as shown previously.

7.4.6 Deleting Shared Definitions

When a shared dictionary entity is no longer required by a database, you can delete the database copy with the RDO statement DELETE. The DELETE statement removes the link between the database and the shareable dictionary definition. The dictionary definition remains intact and can continue to be shared by other applications.

```
RDO> DELETE FIELD BIRTHDATE
RDO> $ DICTIONARY OPERATOR
    .
    .
    .
CDO>
!Confirm that the dictionary definition still exists
!
CDO> SET DEFAULT CDD$TOP.PERSONNEL
CDO> DIRECTORY BIRTHDATE

Directory SYS$COMMON:[CDDPLUS]PERSONNEL

BIRTHDATE;1                                 FIELD
```

The following example changes a database relation—ADDRESS_REC—by deleting the field PHONE from the relation. A new version of ADDRESS_REC is created in the dictionary and the appropriate messages are attached to users of ADDRESS_REC;1. However, the definition of PHONE remains intact in the dictionary so that other users can continue to access it.

```
RDO> CHANGE RELATION ADDRESS_REC.
cont> DELETE PHONE.
cont> END.
RDO> COMMIT
```

You can delete definitions with the CDO command DELETE. However, you cannot delete a dictionary definition that is used by another entity. For example, although PHONE is no longer used by DEPT1, you cannot delete the field from the dictionary because DEPT2 and the record EMPLOYEE_DATA use it:

```
CDO> SHOW USES PHONE
Owners of SYS$COMMON:[CDDPLUS]PERSONNEL.PHONE;1
|   EMPLOYEE_DATA;1                         (Type : RECORD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
|   DEPT2;1                                 (Type : CDD$DATABASE)
|   |   via CDD$DATABASE_SCHEMA

CDO> DELETE FIELD PHONE
%CDD-E-INUSE, Entity in use; not deleted
```

7.5 Restoring a Database That Uses Shareable Dictionary Definitions

You can copy your Rdb/VMS databases using either of two methods:

- RMU/BACKUP and RMU/RESTORE — for regular maintenance backups or disaster recovery.
- RDO IMPORT and EXPORT statements — for unloading and reloading databases, restructuring physical database files, or migrating a database from one DSRI (Digital Standard Relational Interface) system to another.

7.5.1 Backing Up and Restoring Databases

The RMU/BACKUP command creates a backup copy of an Rdb/VMS database and places it in a file. You can back up the entire database or you can request an incremental backup that backs up only the pages that have changed since the last full backup. In the event of subsequent damage to the database, you can specify backup files in an RMU/RESTORE command to restore the database to the condition it was in when you backed it up.

The RMU/RESTORE command recreates all the relationships between the database structure and shared definitions individually defined in CDO.

You can rename or move the files that comprise an Rdb/VMS database by using the RMU/BACKUP and RMU/RESTORE command combination. To move a *multifile* Rdb/VMS database, you *must* use the RMU/BACKUP and RMU/RESTORE commands or the RDO EXPORT and IMPORT statements. If you use the DCL COPY command with a multifile database, the resulting database will be unusable.

7.5.2 Restructuring and Reloading Databases

You can use the RDO EXPORT and IMPORT statements to:

- Migrate a database from one DSRI system to another; for example, from Rdb/ELN to Rdb/VMS
- Change parameters such as storage area definition or page size
- Reload a database (Leaving the storage area definitions the same but changing the device specifications)

For more details about RDO IMPORT and EXPORT statements, see *VAX Rdb/VMS Reference Manual.*

7.6 Converting Databases to CDO Format

You must convert database definitions that you created with versions of Rdb/VMS prior to Rdb/VMS V3.0, in order to use them with CDD/Plus.

CDD/Plus tracks the usage of definitions in CDO format only. You cannot take advantage of pieces tracking if definitions for your database are stored in DMU format. Definitions are created in DMU format by any software that does not support CDD/Plus.

To convert database definitions from DMU to CDO format, use the following procedure:

1. Use the RDO DELETE PATHNAME statement to delete the DMU definitions from the database structure.
2. Use the RDO INTEGRATE IN statement to copy the definitions to the CDO compatibility dictionary.
3. Create directory names for those converted field or record definitions that need to be shareable.

The converted definitions are placed in the compatibility dictionary with exactly the same path as before the conversion. Therefore, you can continue to use established applications for the database. When your converted definitions are stored in the compatibility dictionary, CDD/Plus tracks usage on the shared definitions, and these definitions can be read from all products that access CDD/Plus.

7.6.1 Copying Definitions Into the Compatibility Dictionary

In the following example, the DMU dictionary definitions for the DEPT1 database are converted to CDO format. The new definitions continue to be at the same path and are in the compatibility dictionary. The DMU format definitions are deleted prior to the conversion process, leaving a single copy of all converted definitions.

```
RDO> DELETE PATHNAME 'CDD$TOP.PERSONNEL.DEPT1'.
RDO> INTEGRATE DATABASE DEPT1
cont> IN PATHNAME 'CDD$TOP.PERSONNEL.DEPT1'
RDO> COMMIT
```

The definitions for the database definition DEPT1 located in the compatibility dictionary at CDD$TOP.PERSONNEL.DEPT1 are converted to CDO format but they continue to exist only within the database structure. These converted definitions have no directory name and are, therefore, not yet shareable. To be shared, a definition must have a full directory name.

7.6.2 Sharing Converted Definitions

You can create a directory name for a field or record definition with the CDO command ENTER. The ENTER ... FROM DATABASE command creates a directory name for a field or relation definition. After creating a directory name for a definition, that definition can be shared among databases and specified by other dictionary entities.

The following example illustrates use of the ENTER command. The example shows that converted fields are not listed with the other contents of a dictionary directory. However, after an individual definition is specified with the CDO command ENTER, the definition has a directory name and is listed by the DIRECTORY command. First, the database definitions in the dictionary are converted, using RDO commands:

```
RDO> DELETE PATHNAME 'CDD$TOP.PERSONNEL.DEPT1'.
RDO> INTEGRATE DATABASE DEPT1
cont> IN PATHNAME 'CDD$TOP.PERSONNEL.DEPT1'
RDO> COMMIT
```

To invoke CDO, you must spawn a subprocess. A CDO DIRECTORY command reveals that the converted field EMPLOYEE_ID is not listed.

```
RDO> $
$ DICTIONARY OPERATOR
CDO> SET DEFAULT CDD$TOP.PERSONNEL
CDO> DIRECTORY EMPLOYEE_ID

%CDO-E-NOTFOUND, entity EMPLOYEE_ID not found
```

To give the field a directory name, and make it shareable, use the ENTER command:

```
CDO> ENTER FIELD EMPLOYEE_ID
cont> FROM DATABASE DEPT1
CDO> DIRECTORY EMPLOYEE_ID

Directory SYS$COMMON:[CDDPLUS]PERSONNEL

EMPLOYEE_ID;1                                FIELD
```

The ENTER ... FROM DATABASE command can be used to create directory names for entities that are directly related to the named database. For example, consider that a database—DB1—includes a record—R1. Before you can create a directory name for any of the fields within R1, you must create a directory name for R1 with the command ENTER RECORD R1 FROM DATABASE DB1. Once R1 has a directory name, you can enter the command ENTER FIELD F1 FROM R1. Similarly, to create a directory name for a field within a STRUCTURE clause in R2, you must first create a directory name for R2, then specify the structure name in the ENTER FIELD ... FROM RECORD command, specifying the structure name rather than the record name. The ENTER command cannot create directory names for fields in a VARIANTS clause in a record definition.

You cannot use wildcard characters with this command; you must know and specify the processing name of the entity definition.

The RDO statement IMPORT automatically creates the dictionary definitions for a database in CDO format.

Once all the database definitions are in CDO format, you can track the usage of the various pieces of your application. For example, the following command displays all of the dictionary definitions that are used by the database DEPT1:

```
CDO> SHOW USED_BY /FULL DEPT1
Members of DEPT1;1
|   ADDRESS_REC;1                     (Type : RECORD)
|   |   via CDD$RDB_DATA_AGGREGATE
|   |   EMPLOYEE_ID;1                 (Type : FIELD)
|   |   |    via CDD$DATA_AGGREGATE_CONTAINS
|   |   STREET;1                      (Type : FIELD)
|   |   |    via CDD$DATA_AGGREGATE_CONTAINS
|   EMPLOYEE_REC;1                    (Type : RECORD)
|   |   via CDD$RDB_DATA_AGGREGATE
|   |   BADGE_NO;1                    (Type : FIELD)
|   |   |    via CDD$DATA_AGGREGATE_CONTAINS
   .
   .
   .
```

Note

If you use products that do not support the new features of CDO dictionaries, these products continue to have read access to your converted definitions but do not have write or delete access to CDO definitions. *Do not convert* definitions if you need to have write or delete access to them from such products.

7.7 Deleting Databases

You can delete Rdb/VMS databases with the RDO DELETE DATABASE statement, which has two qualifiers:

- PATHNAME — Deletes the database files and the dictionary definition for the database
- FILENAME — Deletes only the database files

The DELETE DATABASE statement deletes the physical database file (.RDB) and its snapshot file (.SNP). The PATHNAME qualifier deletes the CDD$DATABASE definition in addition to these files.

Issue the DELETE DATABASE statement before invoking the database or after issuing the FINISH statement, because you cannot delete a database when there are active users for that database. You must enclose the path name or file name for the database in single quotes.

When you use the PATHNAME qualifier, you can specify either:

- A **full** dictionary path name, such as CDD$TOP.PERSONNEL.DEPT1.
- A **relative** dictionary path name, such as DEPT1.

If you use a relative path name, be sure that the current CDD/Plus default directory is defined to be all of the path segments preceding the relative path name. The following example shows how to delete DEPT1, the only database in the current directory:

```
RDO> SET DICTIONARY CDD$TOP.PERSONNEL
RDO> DELETE DATABASE PATHNAME 'DEPT1'.
RDO> INVOKE DATABASE 'DEPT1'
%CDD-E-NOT_A_DB, dept1, is not the name of a CDD database
RDO> SHOW DATABASE ALL
The databases in the current CDD directory.
```

Because you have successfully deleted DEPT1, RDO shows no databases in the current CDD directory.

When you delete a database using the FILENAME qualifier, you can use either a full or partial file specification. The following command deletes the files DEPT1.RDB and DEPT1.SNP:

```
RDO> DELETE DATABASE FILENAME 'DEPT1'.
%RDO-W-NOCDDUPDAT, database invoked by filename, the CDD will not be updated
```

Because the dictionary is not updated when you delete by filename, the database definition remains in the dictionary directory.

```
RDO> SHOW DATABASE ALL
The databases in the current CDD directory.
Database with pathname  SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1;1
RDO> EXIT
$ DICTIONARY OPERATOR
Welcome to CDO V1.0
The CDD/Plus V4.0 User Interface
Type HELP for help
CDO> DIR
 Directory SYS$COMMON:[CDDPLUS]
DEPT1;1                                            CDD$DATABASE
```

You can delete the CDD$DATABASE definition by using the CDO DELETE GENERIC command, as shown:

```
CDO> DELETE GENERIC CDD$DATABASE DEPT1.
```

When you have successfully deleted the database definition, you will not be able to display it. In the example, the database definition was the only file contained in the specified directory.

Managing RMS Files with CDO 8

This chapter explains the relationship between logical database file definitions and the physical database files for VAX Record Management Services (RMS) databases. It shows you how to create and use database file definitions and physical RMS database files.

8.1 Using CDD/Plus with RMS: An Overview

If you use RMS services to create and access files and to process records, you can create CDD/Plus definitions for these entities. There are several advantages to representing RMS files in your dictionary. CDD/Plus can be used to:

- Store RMS database definitions for you to reference in programs.
- Help you standardize the definitions within an RMS database.
- Keep track of which entities are used by an RMS database.
- Generate messages to notify you when the definitions of entities used by an RMS database are changed.
- Provide an additional level of protection for database entities.
- Create RMS database entities that can be used by RALLY applications.

Each RMS file is an array of records of one particular type. Each record is an array of fields. An **RMS database** is an RMS file that is built according to a template—an entity of type CDD$RMS_DATABASE. CDD$RMS_DATABASE entities describe the logical definition of the file, which includes both a record definition and a file definition. In this document, entities of type CDD$RMS_DATABASE are called the **RMS database definitions**.

Table 8-1 lists the CDD/Plus commands used to manipulate RMS files.

Table 8–1: Summary of Commands For Manipulating RMS Files

Command	Function
DEFINE DATABASE	Creates the physical file and puts file entity into logical dictionary.
DEFINE RMS_DATABASE	Creates a file definition that includes descriptions of the RMS file and data structures within the file. Does not create the physical file.
SHOW DATABASE	Displays the file definition for the database, and the field and record definitions used by the database.
SHOW RMS_DATABASE	Displays a database file definition, its attributes, and the record used by the database.
SHOW MESSAGES	Displays any messages attached to the specified definition to indicate changes in that definition or a related one. See Chapter 6.
SHOW USED_BY	Displays the definitions that are used by the specified definition.
SHOW USES	Displays all of the entity definitions that use the specified definition.
CHANGE DATABASE	Changes dictionary attributes, moves the physical file, and updates the database name in the logical dictionary.
DELETE DATABASE	Deletes the physical file and the file entity in the logical dictionary.
DELETE RMS_DATABASE	Deletes the file definition only.

To define an RMS database in the dictionary, follow this two-step process:

1. Use the DEFINE RMS_DATABASE command to create a database definition.
2. Use the DEFINE DATABASE command to create a physical file based on the RMS database definition and an entry in the dictionary.

The following sections describe this process.

8.2 Creating RMS Database Definitions

Before you can define an RMS database, you must create an RMS database definition that describes the proposed database(s) in your dictionary. To create an RMS database definition, use the DEFINE RMS_DATABASE command. Command arguments allow you to specify the attributes and structure you want. All databases based on this definition share the specified attributes and structure.

As in other CDO operations, the following naming rules apply to the DEFINE RMS_DATABASE command:

- The record that you specify in the DEFINE RMS_DATABASE command must already exist in a dictionary.
- The name of the record that you specify in the command line cannot be the same as the database name.
- If you do not specify a full pathname for the database, CDD/Plus creates the RMS database definition in your current default directory.

The following example creates an RMS database definition that specifies two keys.

```
DEFINE RMS_DATABASE EQUIPMENT_RMS.
        RECORD PART_REC.
        FILE_DEFINITION
                MAX_RECORD_SIZE 41
                ORGANIZATION INDEXED.
        KEYS.
                KEY 0
                DUPLICATES
                SEGMENT MANUFACTURER IN TYPE IN PART_REC
                SEGMENT MODEL IN TYPE IN PART_REC.

                KEY 1
                CHANGES
                DUPLICATES
                SEGMENT MODEL IN TYPE IN PART_REC.
        END KEYS.
END.
```

Specifying File Attributes

Table 8-2 shows the three types of clauses used in specifying file attributes. *VAX CDD/Plus Common Dictionary Operator Reference Manual* describes the CDO syntax for each clause.

Table 8–2: DEFINE_RMS_DATABASE Attribute Clauses

Clause	Specifies
File-Definition	Defines file characteristics and certain run-time options.
Area-Definition	Controls file or area space allocation on disk devices to optimize performance.
Keys-Definition	Defines the characteristics of one or more key(s) in an indexed file.

Attribute clauses can enable you to fine-tune your RMS applications. For example, you can use a clause to pre-allocate the file in a situation where performance is important and size of the file is predetermined. Descriptions and valid values of attribute clause parameters appear in *VAX Record Management Services Reference Manual.*

8.3 Creating Physical Database Files

After completing your RMS database definition, you can use one or more DEFINE DATABASE commands to create specific physical RMS database file(s) based on that definition. A physical database file appears in the dictionary as the entity CDD$DATABASE.

The following examples show how a user might build an RMS database in the dictionary to maintain a list of current employees in three departments. The first step is to define the database objects that make up an employee record:

```
CDO> SET DEFAULT SYS$COMMON:[CDDPLUS]

CDO> DEFINE FIELD FIRST_NAME
cont> DATATYPE IS TEXT
cont> SIZE IS 20.
CDO> DEFINE FIELD LAST_NAME
cont> DATATYPE IS TEXT
cont> SIZE IS 30.
CDO> DEFINE FIELD ID_NUMBER
cont> DATATYPE IS UNSIGNED LONGWORD.
CDO> DEFINE RECORD EMPLOYEE_DATA.
cont>         LAST_NAME.
cont>         FIRST_NAME.
cont>         ID_NUMBER.
cont>   END.
CDO>
```

After the objects are defined in the dictionary, you can create an RMS database definition from the employee record. The following definition specifies two areas and two keys (ID_NUMBER and LAST_NAME):

```
CDO>  DEFINE RMS_DATABASE EMPLOYEE_INFORMATION
cont> AUDIT /* STORAGE FOR EMPLOYEE INFORMATION */.
cont>    RECORD EMPLOYEE_DATA.
cont>    FILE_DEFINITION
cont>        ORGANIZATION INDEXED
cont>        CHANNEL_ACCESS_MODE SUPER
cont>        CARRIAGE_CONTROL CARRIAGE_RETURN
cont>        ACCESS RECORD_IO
cont>        FILE_PROCESSING_OPTIONS BEST_TRY_CONTIGUOUS
cont>        FORMAT VARIABLE
cont>        SHARING GET, USER_INTERLOCK
cont>    AREAS.
cont>        AREA 0
cont>           ALLOCATE 1000
cont>           BUCKET_SIZE 10
cont>           EXTENSION 100
cont>           CONTIGUOUS.
cont>        AREA 1
cont>           ALLOCATE 1000
cont>           BUCKET_SIZE 1
cont>           EXTENSION 100
cont>           BEST_TRY_CONTIGUOUS.
cont>        AREA 2
cont>           ALLOCATE 1000
cont>           BUCKET_SIZE 1
cont>           EXTENSION 100
cont>           BEST_TRY_CONTIGUOUS.
cont>    END AREAS.
cont>    KEYS.
cont>        KEY 0
cont>           DATA_AREA 0
cont>           INDEX_AREA 2
cont>           LEVEL1_INDEX_AREA 1
cont>           SEGMENT ID_NUMBER IN EMPLOYEE_DATA.
cont>        KEY 1
cont>           DUPLICATES
cont>           DATA_AREA 1
cont>           INDEX_AREA 1
cont>           SEGMENT LAST_NAME IN EMPLOYEE_DATA.
cont>    END KEYS.
cont> END EMPLOYEE_INFORMATION RMS_DATABASE.
CDO>
```

Now you can use the DEFINE DATABASE command to create physical RMS databases based on the same RMS database definition. You must reference an RMS database definition in the USING clause: if a VAX Rdb/VMS database is referenced, an error will result.

Note

DIGITAL recommends that you create the physical RMS file(s) in a VMS directory other than your CDO anchor directory. If you create the RMS file in your anchor directory, CDD/Plus deletes it when you delete your dictionary.

The commands in the following example create three RMS databases that are located on different disks:

```
CDO> DEFINE DATABASE SALES_FILE USING
EMPLOYEE_INFORMATION ON DISK$02:[SALES]EMP.DAT.
%CDO-I-FILECRE, file DISK$02:[SALES]EMP.DAT; created
CDO> DEFINE DATABASE DEVELOPMENT_FILE USING
EMPLOYEE_INFORMATION ON DISK$03:[DEVELOPMENT]EMP.DAT.
%CDO-I-FILECRE, file DISK$03:[DEVELOPMENT]EMP.DAT; created
CDO> DEFINE DATABASE SUPPORT_FILE USING
EMPLOYEE_INFORMATION ON DISK$07:[SUPPORT]EMP.DAT.
%CDO-I-FILECRE, file DISK$07:[SUPPORT]EMP.DAT; created
```

When your definitions are completed, the RMS database definition appears in your dictionary directory as a CDD$RMS_DATABASE and the physical RMS file appears as a CDD$DATABASE entity. You can use a directory (DIR) command to make sure that you created the database.

```
CDO> DIR
 Directory SYS$COMMON:[CDDPLUS]
DEVELOPMENT_FILE;1                                 CDD$DATABASE
EMPLOYEE_DATA;1                                    RECORD
EMPLOYEE_INFORMATION;1                             CDD$RMS_DATABASE
FIRST_NAME;1                                       FIELD
LAST_NAME;1                                        FIELD
ID_NUMBER;1                                        FIELD
SALES_FILE;1                                       CDD$DATABASE
SUPPORT_FILE;1                                     CDD$DATABASE
CDO>
```

You can use the following commands to check that you created the physical RMS file in the VMS directory where you wanted it:

```
$ SET DEFAULT DISK$07:[SUPPORT]
$ DIR

Directory DISK$07:[SUPPORT]

EMP.DATA;1

Total of 1 files.
$ dir/full emp.data
```

```
Directory DISK$07:[SUPPORT]

EMP.DATA;1                    File ID:  (10657,17,0)
Size:         3006/3006       Owner:    [VDD,HALVORSON]
Created:  15-FEB-1988 15:42   Revised:  24-MAR-1988 15:10 (2)
Expires:   <None specified>   Backup:   10-APR-1988 19:10
File organization:  Indexed, Prolog: 3, Using 2 keys
                             In 3 areas
File attributes:    Allocation: 3006, Extend: 100, Maximum bucket size: 10
                    Global buffer count: 0, No version limit
Record format:      Variable length, maximum 19 bytes
Record attributes:  Carriage return carriage control
Journaling enabled: None
File protection:    System:RW, Owner:RWED, Group:RWED, World:RWED
Access Cntrl List:  None

Total of 1 file, 3006/3006 blocks.
```

You can use information from databases in a VAX RALLY application.

8.4 Using RMS File Definitions in Programs and Applications

VAX RALLY is a forms-based application generator. In its default form, the DCL command RALLY CREATE places an entity in the dictionary for each Application File (AFILE) and each Data Source Definition (DSD). These entities, RALLY$APPLICATION and RALLY$DATA_SOURCE_DEFINITION, are proxy objects. **Proxy objects** contain information about and point to an actual RALLY object, but the actual RALLY object is created and maintained in the AFILE. Since the dictionary entities keep track of the location and change history for the corresponding RALLY object, you can use CDO to display the location and creation history of RALLY applications.

RALLY builds DSDs based on the CDD/Plus pathname to an RMS file or record definition. You can use Data Source Definitions to connect the information in the RMS database to form/report fields or to variables in an AFILE. The *VAX RALLY Definition System User's Guide* describes RMS file access from RALLY, including dictionary requirements for such access.

8.4.1 Creating Relationships with REFERENCE FROM DICTIONARY

If the VAX language you use supports CDO dictionaries, you can insert a compiler directive anywhere in your program in order to create a compiled module. You can then create a relationship between the compiled module and an entity you name. For the proper syntax for compiler directives and creating a relationship, see the documentation for the language that you are using. You can reference any of the following data types in this command:

- Data aggregates
- Databases
- Subschemas
- Other source modules
- Other compiled modules

8.4.2 Tracking File Program Components

CDO keeps track of the relationships between the entities you define, create, or change. If you alter file metadata in an RMS database, CDO sends messages to any CDO database or any compiled module that owns that file, provided that module is represented in the dictionary.

You can use the CHECK_MESSAGES routine described in *VAX CDD/Plus Call Interface Manual* to see changes that may affect programs and applications. You can write a program that parses the message buffer and displays messages for you or queries you concerning inconsistencies noted by messages. CDO does not provide these services.

8.5 Showing Databases and Database Definitions

The SHOW commands listed in Table 8-1 let you check the contents of your database and your database definition.

The SHOW DATABASE command displays the database definition. SHOW DATABASE default output (/BRIEF) includes the database name and description, record name, file organization attributes, and fully qualified path name of an RMS database:

```
CDO> SET DEFAULT DISK$02:[SALES]
CDO> SHOW DATABASE SALES_FILE
Definition of database  SALES_FILE
|   database uses RMS database EMPLOYEE_INFORMATION
|   |   database uses record EMPLOYEE_DATA
|   |   file definition
|   |   |   channel mode protection supervisor
|   |   |   access for block or record IO
|   |   |   file contiguous best try
|   |   |   file organization index sequential
|   |   |   record carriage_control carriage_return
|   |   |   file record format variable
|   |   |   file sharing on get
|   |   |   file sharing on user interlock
|   |   |   key 0
|   |   |   |   key data area 0
|   |   |   |   key index area 2
|   |   |   |   segment ID_NUMBER IN EMPLOYEE_DATA
|   |   |   key 1
|   |   |   |   key duplicates
|   |   |   |   key data area 1
|   |   |   |   key index area 1
|   |   |   |   segment LAST_NAME IN EMPLOYEE_DATA
|   |   |   storage area 0
|   |   |   |   area allocation 1000
|   |   |   |   area bucket size 10
|   |   |   |   area contiguous
|   |   |   |   area extension 100
|   |   |   storage area 1
|   |   |   |   area allocation 1000
|   |   |   |   area bucket size 1
|   |   |   |   area contiguous best try
|   |   |   |   area extension 100
|   |   |   storage area 2
|   |   |   |   area allocation 1000
|   |   |   |   area bucket size 1
|   |   |   |   area contiguous best try
|   |   |   |   area extension 100
|   database in file DISK$02:[SALES]EMP.DAT
|   |   fully qualified file DISK$02:[SALES]EMP.DAT;
CDO>
```

The /FULL qualifier causes SHOW DATABASE to add the record definition attributes to the default display.

The SHOW RMS_DATABASE command displays the RMS database definition:

```
CDO> SHOW RMS_DATABASE EMPLOYEE_INFORMATION
Definition of RMS database  EMPLOYEE_INFORMATION
|   Description              'INFORMATION ON'
|                            'CURRENT EMPLOYEE'
|   database uses record EMPLOYEE_DATA
|   file definition
|   |   channel mode protection supervisor
|   |   access for block or record IO
|   |   file contiguous best try
|   |   file organization index sequential
|   |   record carriage_control carriage_return
|   |   file record format variable
|   |   file sharing on get
|   |   file sharing on user interlock
|   |   key 0
|   |   |   key data area 0
|   |   |   key index area 2
|   |   |   segment ID_NUMBER IN EMPLOYEE_DATA
|   |   key 1
|   |   |   key duplicates
|   |   |   key data area 1
|   |   |   key index area 1
|   |   |   segment LAST_NAME IN EMPLOYEE_DATA
|   |   storage area 0
|   |   |   area allocation 1000
|   |   |   area bucket size 10
|   |   |   area contiguous
|   |   |   area extension 100
|   |   storage area 1
|   |   |   area allocation 1000
|   |   |   area bucket size 1
|   |   |   area contiguous best try
|   |   |   area extension 100
|   |   storage area 2
|   |   |   area allocation 1000
|   |   |   area bucket size 1
|   |   |   area contiguous best try
|   |   |   area extension 100
CDO>
```

If you have more than one RMS database definition in your dictionary and do not specify one, CDD/Plus displays all of them.

8.6 Moving Databases with CHANGE DATABASE

When you want to move an RMS database to a new location, use the CHANGE DATABASE command. You can change your database in three ways:

- You can specify a new location with FILENAME attribute.
- You can enclose a comment with the DESCRIPTION attribute.
- You can add an entry to the history list for a definition with the AUDIT attribute.

The following example changes the location of the database SALES_FILE by specifying the new filename parameter after the optional word ON.

```
CDO> CHANGE DATABASE SALES_FILE;1 ON DISK$03:[MIS.MARKETING]EMP.DAT.
moving file DISK$02:[SALES]EMP.DAT; to DISK$03:[MIS.MARKETING]EMP.DAT;,
proceed? [Y/N] (N) Y
%CDO-I-FILECRE, file DISK$03:[MIS.MARKETING]EMP.DAT; created
%CDO-I-FILEDEL, file DISK$02:[SALES]EMP.DAT; deleted
CDO>
```

You can also use CHANGE DATABASE to add history information or change the description without changing the filename.

```
CDO> CHANGE DATABASE SALES_FILE;1
cont> DESCRIPTION IS/* Moved to new disk during departmental reorganization */
CDO>
```

8.7 Deleting Databases and Database Definitions

You can delete either the physical RMS database file or the file description. To delete the physical file as well as the entity that represents it in the database, use the DELETE DATABASE command. CDD/Plus prompts you for confirmation before deleting the database.

```
CDO>
CDO> DELETE DATABASE SUPPORT_FILE;2.
deleting file DISK$07:[SUPPORT]EMP.DAT;, proceed? [Y/N] (N)Y
CDO>
```

After all the databases that use it are deleted, you can use the DELETE RMS_DATABASE command to delete the RMS database definition in the dictionary. CDD/Plus deletes the definition without prompting you.

```
CDO> DELETE RMS_DATABASE EMPLOYEE_INFORMATION.
CDO>
```

If you want notification that the RMS database definition was deleted, you can use the /LOG qualifier with the DELETE RMS_DATABASE command, as follows:

```
CDO> DELETE RMS_DATABASE /LOG EMPLOYEE_INFORMATION.
%CDO-I-ENTDEL, entity SYS$COMMON:[CDDPLUS]EMPLOYEE_INFORMATION;2 deleted
```

User's Guide to DMU Format Dictionaries A

The CDD/Plus dictionary system supports dictionaries in two formats: the format of dictionaries manipulated with the CDO utility and the format of dictionaries manipulated with the DMU utility. The body of this manual provides tutorial material on creating and manipulating definitions in CDO dictionaries; this appendix provides tutorial material about dictionaries that are manipulated through the DMU utility.

Before using this appendix, you should read the description of the CDD/Plus compatibility scheme in Chapter 2. You should be aware of the access routes into CDD/Plus and which products can read, write, and delete definitions in DMU dictionaries. Depending on the products you use to access CDD/Plus, you may continue to require the information on DMU dictionaries that this appendix provides.

Since this appendix deals only with DMU format dictionaries, to avoid confusion, VAX Common Data Dictionary software is referred to as CDD, rather than CDD/Plus.

This appendix includes a separate index made up of entries referring only to subjects related to DMU format dictionaries. The main index at the end of this book includes no entries referring to this appendix.

A.1 DMU Dictionary Structure

You can think of the DMU Dictionary's hierarchical structure as a family tree defining parent-child relationships. Dictionary directories are the parents, and their children include other directories, as well as dictionary objects (see Figure A-1).

The rules that govern these hierarchical relationships are simple. Some of the rules help distinguish directories from objects:

- A dictionary object cannot have any children.
- A dictionary directory can have any number of children (within the limits of your system's resources).
- The children of a dictionary directory can be other directories, dictionary objects, or a combination of both dictionary directories and objects.
- It is possible to have multiple versions of a dictionary object; therefore, a dictionary object has a version number associated with it.
- It is not possible to have multiple versions of a dictionary directory; therefore, a dictionary directory does not have a version number associated with it.
- A dictionary directory cannot contain two directories with the same name.
- A dictionary directory can contain two objects with the same name *if they have different version numbers.*

Some of the rules apply to both directories and objects:

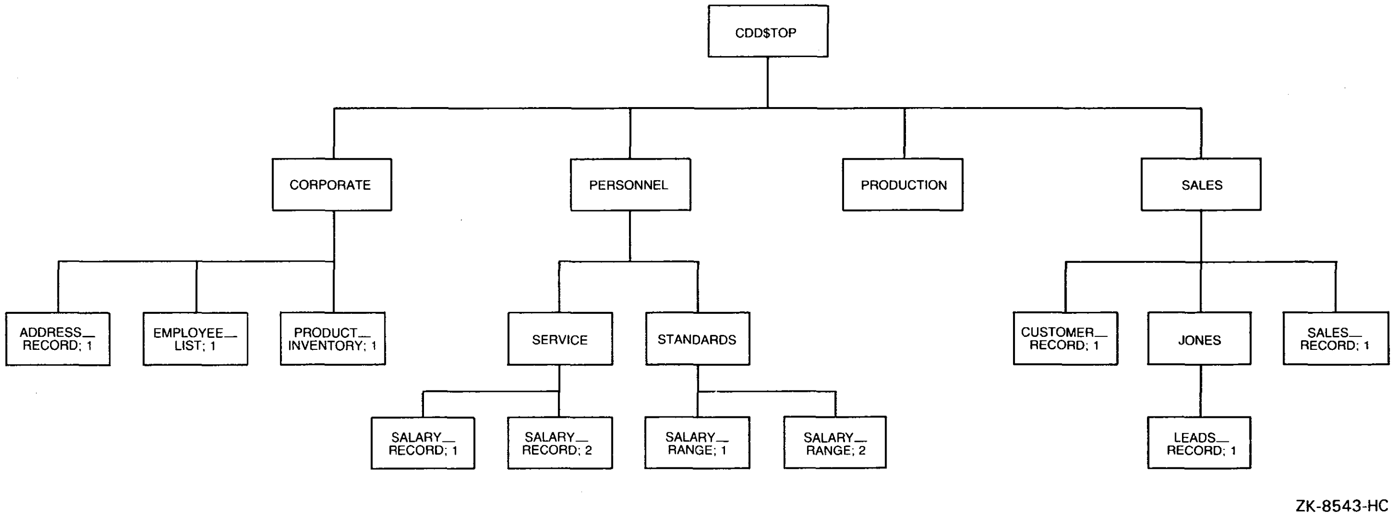
- When you create a dictionary directory or object, you assign it a name.
- A dictionary directory or dictionary object can have only one parent.

All of the dictionary directories that precede and are related to an object or directory are called its **ancestors**. All dictionary directories and objects of which a directory is an ancestor are called its **descendants**. The dictionary directory that serves as the origin of the directory hierarchy is CDD$TOP, the **root dictionary directory**. CDD$TOP is an ancestor of every other dictionary directory and object in the CDD.

A.1.1 Sample DMU Dictionary

The sample dictionary in Figure A-1 shows some of the relationships that can exist between dictionary directories and dictionary objects. All of the examples in this manual are drawn from this sample directory hierarchy and its associated data definitions.

Figure A–1: Sample DMU Dictionary Hierarchy



At the top of the sample dictionary is CDD$TOP, the root dictionary directory. The next level consists of four directories:

- CORPORATE
- PERSONNEL
- PRODUCTION
- SALES

Three of these directories have children:

- The CORPORATE directory has three children:
  - ADDRESS_RECORD;1
  - EMPLOYEE_LIST;1
  - PRODUCT_INVENTORY;1

  These are dictionary objects that contain record definitions for the product inventory, address record, and employee list.
- The SALES directory has three children:
  - CUSTOMER_RECORD;1
  - SALES_RECORD;1
  - JONES

  CUSTOMER_RECORD;1 and SALES_RECORD;1 are dictionary objects that contain record definitions.

  JONES is a dictionary directory with one child, the dictionary object LEADS_RECORD;1.

  SALES is an ancestor of LEADS_RECORD;1. LEADS_RECORD;1 is a descendant of SALES.
- The PERSONNEL directory has two children:
  - SERVICE
  - STANDARDS

  These are both dictionary directories.

  SERVICE has two children:
  - SALARY_RECORD;1

- SALARY_RECORD;2.

SALARY_RECORD;2 is an updated version of SALARY_RECORD;1.

STANDARDS also has two children:

- SALARY_RANGE;1
- SALARY_RANGE;2

SALARY_RANGE;1 is an updated version of SALARY_RANGE;2.

All the children of SERVICE and STANDARDS are descendants of PERSONNEL.

A.1.2 Subdictionary Directories in the DMU Dictionary

When you first install the VAX Common Data Dictionary, you create a dictionary file named CDD.DIC in which the directory hierarchy is physically stored. The DMU Dictionary then allows you to store portions of this single logical hierarchy in separate physical files called subdictionary files. The directories that point to separate subdictionary files are called subdictionary directories, or **subdictionaries**.

Except for their physical location, subdictionary directories are exactly like dictionary directories. Subdictionaries are part of the same logical hierarchy, and they perform the same functions, as dictionary directories. For more information about subdictionaries, see Section A.6.5 and Section A.9.1.5.

A.1.3 CDD Types in the DMU Dictionary

When you create dictionary directories and subdictionaries, and when you insert dictionary objects into the hierarchy, the CDD assigns each of them a dictionary **type**. Examples of CDD types include SUBDICTIONARY, CDD$RECORD, and DTR$DOMAIN.

The Dictionary Management Utility's LIST/BRIEF command displays the hierarchy and includes the type that the DMU Dictionary assigns to each subdictionary and dictionary object. LIST/BRIEF does not display a type for dictionary directories.

The following example shows how to use LIST/BRIEF to produce a listing of the sample dictionary. The listing shows PERSONNEL to be a subdictionary directory stored in the file DB3:[CASADAY.CDD]PERS.DIC.

```
DMU> LIST/BRIEF CDD$TOP>

   CDD$TOP
   |  CORPORATE
   |  |  ADDRESS_RECORD;1 <CDD$RECORD>
   |  |  EMPLOYEE_LIST;1 <CDD$RECORD>
   |  |  PRODUCT_INVENTORY;1 <CDD$RECORD>
   |  PERSONNEL <SUBDICTIONARY> : DB3:[CASADAY.CDD]PERS.DIC
   |  |  SERVICE _
   |  |  |  SALARY_RECORD;2 <CDD$RECORD>
   |  |  |  SALARY_RECORD;1 <CDD$RECORD>
   |  |  STANDARDS
   |  |  |  SALARY_RANGE;2 <CDD$RECORD>
   |  |  |  SALARY_RANGE;1 <CDD$RECORD>
   |  PRODUCTION
   |  SALES
   |  |  CUSTOMER_RECORD;1 <CDD$RECORD>
   |  |  JONES
   |  |  |  LEADS_RECORD;1 <CDD$RECORD>
   |  |  SALES_RECORD;1 <CDD$RECORD>
```

A.2 DMU Dictionary Paths

In the DMU Dictionary, when you want access to the CDD—for example, to create or modify the directory hierarchy, to copy a record definition into an application, or to ready a DATATRIEVE domain—you must be able to locate particular dictionary directories, subdictionaries, or objects. You do this by using **dictionary path names**. A path name is the string of given names linking CDD$TOP to the **given name** of the target dictionary directory, subdictionary, or object.

A.2.1 The Given Name in the DMU Dictionary

Every dictionary directory, subdictionary, and object in the CDD has a given name. Except for CDD$TOP, the given name of the root dictionary directory, you assign given names when you create each directory, subdictionary, and object.

Given names, however, are not necessarily unique designations. In fact, two directories or objects in the dictionary can have the same given name *as long as they do not share the same parent directory or subdictionary.* Objects with the same name can have the same parent directory or subdictionary if they are different versions of the same object, distinguished from each other by version numbers.

To uniquely identify a dictionary directory, subdictionary, or object, you must use its dictionary path name.

A.2.2 The Full DMU Dictionary Path Name

The full dictionary path name of a directory, subdictionary, or object consists of the concatenation of the given names of all its ancestors, beginning with CDD$TOP, and ending with its given name. Separate each given name from the next by a period.

For example, the full dictionary path name of the dictionary directory SERVICE in the sample dictionary (Figure A-1) is:

```
CDD$TOP.PERSONNEL.SERVICE
```

A.2.3 Version Numbers in Object Names in the DMU Dictionary

You can create multiple versions of dictionary objects. Nevertheless, every object must have a path name that uniquely identifies it. Therefore, each dictionary object is assigned a version number when it is created and when it is renamed. To identify a specific object, you must specify its version number. For example, the full dictionary path name of SALARY_RECORD;2 is:

```
CDD$TOP.PERSONNEL.SERVICE.SALARY_RECORD;2
```

The full dictionary path name of SALARY_RECORD;1 is:

```
CDD$TOP.PERSONNEL.SERVICE.SALARY_RECORD;1
```

There are a number of ways to specify the version of an object. For the most part, the rules for specifying versions of CDD objects follow the rules for specifying versions of VMS files, *except that the creation of multiple versions of CDD objects is not the default.* Table A-4 describes the various methods of specifying versions.

A.2.4 The Default DMU Dictionary Directory

You need not always use the full dictionary path name to identify directories and objects in the CDD. Instead, you can establish a **default directory** at a specified dictionary directory or subdictionary in the hierarchy. Then you have to use only that portion of the path name descended from the default to identify a directory or object. This shortened path name is called the **relative dictionary path name**.

You can establish your default directory in two ways:

- By using the Dictionary Management Utility's SET DEFAULT command (see Section A.5.3)
- By defining the logical name CDD$DEFAULT (see Section A.4.4)

For example, if you establish your default directory as CDD$TOP.SALES, then you can use either the full or the relative dictionary path name to identify the dictionary object LEADS_RECORD;1:

- Full path name: CDD$TOP.SALES.JONES.LEADS_RECORD;1
- Relative path name: JONES.LEADS_RECORD;1

A.3 CDD Utilities in the DMU Dictionary

CDD provides three utilities to help you create and maintain your DMU dictionary:

- The Dictionary Management Utility (DMU)
- The Dictionary Verify/Fix Utility (CDDV)
- The Dictionary Data Definition Language Utility (CDDL)

A.3.1 Dictionary Management Utility (DMU)

DMU commands allow you to create dictionary directories and subdictionaries, to manage the dictionary structure, and to control access to the dictionary. You can invoke the utility by typing:

```
$ RUN SYS$SYSTEM:DMU
```

DMU commands are briefly described in Table A–1. See Section A.5 for more information about using DMU commands. For a complete description of DMU commands, parameters, and qualifiers, see the *VAX Common Data Dictionary Utilities Reference Manual.*

Table A–1: The Dictionary Management Utility Commands

Command	Function
BACKUP	Makes a backup copy of the information in specified directories and objects in the CDD and stores it in an RMS file.
COPY	Duplicates portions of the CDD across directories.
CREATE	Creates new dictionary directories and subdictionaries.
DELETE	Deletes specified dictionary directories, subdictionaries, and objects.
DELETE/HISTORY	Purges history list entries.
DELETE/PROTECTION	Deletes portions of the access control list of a directory, subdictionary, or object.
EXIT	Returns control to DCL.
EXTRACT	Copies or generates the source text of a dictionary object and inserts it into an RMS file.
HELP	Displays documentary text about DMU.
LIST	Displays information about dictionary directories, subdictionaries, or objects and their history and access control lists.
MEMO	Adds an entry to the history list of a dictionary directory, subdictionary, or object.
PURGE	Deletes versions of dictionary objects, keeping a specified number of the highest versions.
RENAME	Allows you to change the name of a dictionary directory, subdictionary, or object, and to change the version number of an object.
RENAME/SUBDICTIONARY	Changes the VMS file specification to which a subdictionary directory points.
RESTORE	Copies portions of the dictionary from a backup file into the CDD.

(Continued on next page)

Table A–1: The Dictionary Management Utility Commands (Cont.)

Command	Function
SET ABORT	Halts utility command procedures when DMU or CDD signals an error.
SET DEFAULT	Allows you to establish a temporary default dictionary directory.
SET PROTECTION	Allows you to establish or modify access control lists.
SET PROTECTION/EDIT	Runs the access control list editor to modify access control list entries.
SHOW DEFAULT	Displays your current default dictionary directory.
SHOW PROTECTION	Displays your access privileges to a dictionary directory, subdictionary, or object.
SHOW VERSION	Displays the software version number.

A.3.2 Dictionary Verify/Fix Utility (CDDV)

Dictionary files can become corrupted (for example, by a hardware failure during an input/output operation). The Dictionary Verify/Fix Utility (CDDV) validates and repairs corrupt dictionary files and compresses valid dictionary files.

Users who own dictionary files can use CDDV to maintain those files. To use CDDV with files you do not own, you must have VMS SYSPRV or BYPASS privilege.

You can invoke the utility by typing:

```
$ RUN SYS$SYSTEM:CDDV
```

You can then issue CDDV commands, which are briefly described in Table A–2. See Section A.9.4 for more information about using CDDV. For a complete description of the CDDV commands, parameters, and qualifiers, see the *VAX Common Data Dictionary Utilities Reference Manual.*

Table A–2: The Dictionary Verify/Fix Utility Commands

Command	Function
COMPRESS	Copies and reorganizes a dictionary file, reducing its size.
EXIT	Returns control to DCL.
FIX	Scans a dictionary file, reports any inconsistencies, and makes repairs.
HELP	Displays documentary text about the CDDV.
SHOW VERSION	Displays the software version number.
VERIFY	Scans a dictionary file and reports any inconsistencies.

A.3.3 Dictionary Data Definition Language Utility (CDDL)

CDDL compiles record definitions contained in CDDL source files and stores them in the dictionary. CDDL allows you to:

- Create entirely new record definitions
- Replace already existing record definitions
- Create additional versions of existing record definitions without replacing old versions

You can invoke the utility by typing:

```
$ RUN SYS$SYSTEM:CDDL
```

CDDL then prompts you for the name of a CDDL source file. See Section A.7 and Section A.7.1 for more information about using CDDL. For a complete description of the CDDL commands, parameters, and qualifiers, see the *VAX Common Data Dictionary Data Definition Language Reference Manual.*

Table A–3: The Dictionary Data Definition Language Utility Commands

Command	Function
CDDL	Compiles a CDDL source file and stores the record definitions it contains in the dictionary.
CDDL/RECOMPILE	Recompiles record definitions from source text already in the dictionary.

A.4 Using DMU and CDDV in the DMU Dictionary

You use the Dictionary Management Utility (DMU) to create and maintain the DMU Dictionary's directory hierarchy, history lists, and access control lists. You can use the Dictionary Verify/Fix Utility (CDDV) to repair damaged dictionary files.

These sections show you how to invoke and use these utilities, how to specify dictionary path names, and how to define your default directory.

A.4.1 Issuing DMU and CDDV Commands

You can invoke the CDD utilities and issue utility commands in either of the following ways:

- By commands at the system (DCL) level
- Through a DCL command procedure

A.4.1.1 System Level Commands – To access the CDD utilities, enter one of the following DCL commands:

```
$ RUN SYS$SYSTEM:DMU
$ RUN SYS$SYSTEM:CDDV
```

The utility you are running then displays a prompt. The Dictionary Management Utility prompt is:

```
DMU>
```

The Dictionary Verify/Fix Utility prompt is:

```
CDDV>
```

To save keystrokes, you can define DMU and CDDV as global symbols in your login command file:

```
$ DMU:==$SYS$SYSTEM:DMU
$ CDDV:==$SYS$SYSTEM:CDDV
```

Then you can access the utilities simply by typing the symbol name:

```
$ DMU
$ CDDV
```

You can issue utility commands in interactive sessions within the utility or at the DCL level.

Within a utility program, you are prompted for commands until you enter EXIT. For example:

```
$ DMU
DMU> LIST/FULL CDD$TOP.PERSONNEL
     .
     .
     .
DMU> EXIT
$

$ CDDV
CDDV> VERIFY DB3:[CASADAY]PERS.DIC
     .
     .
     .
CDDV> EXIT
$
```

To enter utility commands at the DCL level, you must have defined DMU and CDDV as global symbols. Then you can invoke the utility and issue the utility command in one response to the DCL dollar sign prompt. For example:

```
$ DMU LIST/FULL CDD$TOP.PERSONNEL
$

$ CDDV VERIFY DB3:[CASADAY]PERS.DIC
$
```

A.4.1.2 DCL and Utility Command Procedures – You can run CDD utilities and issue utility commands from DCL command procedures. Include the command to invoke the utility followed by utility commands. You can then execute the procedure at DCL level by typing an at sign (@) followed by the file specification of the command procedure:

```
$ @file-specification
```

For example, you can store the following commands in a procedure named VERIFY.COM:

```
$ RUN SYS$SYSTEM:CDDV
VERIFY DB3:[CASADAY.CDD]PERS.DIC
EXIT
```

Executing the procedure then invokes CDDV and verifies the dictionary file PERS.DIC:

```
$ @VERIFY.COM
```

You can also create a command file that includes only utility commands. You execute a utility command file by:

1. Invoking the utility
2. Typing an at sign (@) followed by the file specification of the command file

For example, you can create a command procedure named LIST.COM containing the following DMU commands:

```
LIST/FULL CDD$TOP.PERSONNEL
EXIT
```

You can invoke DMU and execute the procedure to list information about PERSONNEL:

```
$ RUN SYS$SYSTEM:DMU
DMU> @LIST.COM
```

When you use DCL command procedures or utility command files, the following conditions apply:

- The at sign (@) must be the first character following the DCL or utility prompt.
- There must be no space between the at sign and the file specification.

- The file specification is a standard VMS file specification.
- The default file type is .COM.

A.4.1.3 Halting a DMU Command Procedure – All command procedures stop if a fatal error is signalled; however, a nonfatal error in a command procedure can cause commands after the error to be performed incorrectly. For example, consider the utility command procedure DELJONES.COM:

```
!PROCEDURE DELJONES
SET DEFAULT CDD$TOP.SOLES
DELETE/ALL JONES
EXIT
```

When DMU executes this command file in the sample dictionary, CDD tries to set the default directory to CDD$TOP.SOLES instead of CDD$TOP.SALES (see Figure A-1) because the second line of this procedure contains a spelling error. If there is no such directory, CDD issues an error message and the current default directory is not changed. If there is a directory JONES under the current default directory, the next command attempts to delete that directory and all its children. Thus, the spelling error in this procedure could cause the deletion of the wrong portion of the dictionary.

A DMU command, SET ABORT, halts a DMU command procedure and returns you to DMU command level if DMU or the CDD issues a nonfatal error. SET ABORT does not roll back any commands already executed (unless the error occurs in the middle of the execution of a COPY/STAGE or RESTORE/STAGE command).

Note

SET ABORT works only for DMU command procedures, that is, for command files executed from within DMU. It does *not* halt DCL or CDDV command procedures.

The following example illustrates the use of SET ABORT. If you add SET ABORT to DELJONES.COM, it looks like this:

```
!PROCEDURE DELJONES
SET ABORT
SET DEFAULT CDD$TOP.SOLES
DELETE/ALL JONES
EXIT
```

You can now execute the procedure:

```
DMU> @DELJONES
```

DMU issues the following error messages at the second line of the command procedure:

```
%DMU-E-CDDERROR, CDD error at node "CDD$TOP.SOLES"
-CDD-E-NODNOTFND, Directory or object not found
DMU>
```

As a result of these errors:

- SET ABORT halts the procedure.
- No commands after the errors execute.
- No directories or objects are deleted.
- DMU returns you to DMU command level.

A.4.2 Reading Command Lines in the DMU Dictionary

Both the Dictionary Management Utility and the Dictionary Verify/Fix Utility accept abbreviations and recognize the exclamation point (!) as a comment delimiter. You can continue a utility command on a second line by typing a hyphen (-). In addition, you can abort execution of DMU commands by pressing CTRL/C.

A.4.2.1 Abbreviations – You can use abbreviations for any of the DMU or CDDV command words. The only restriction is that you must specify enough characters to avoid ambiguity.

For example, the first two characters in RENAME are the same as the first two characters in RESTORE. Therefore, you need at least three characters for unique abbreviations: REN and RES. You can abbreviate LIST as L and VERIFY as V, however, because no other commands begin with the letters L and V.

A.4.2.2 Exclamation Point —DMU and CDDV interpret an exclamation point (!) occurring anywhere outside of a quoted string as a comment delimiter. In parsing a command line, the utilities ignore that part of the line following the exclamation point.

For example, the following input line validates the PERSONNEL subdictionary file.

```
CDDV> VERIFY DB3:[CASADAY.CDD]PERS.DIC ! Validate PERSONNEL
```

A.4.2.3 Hyphens in Utility Command Lines —If a utility command is too long to enter on a single line, you can use a hyphen (-) at the end of the first line to indicate that it is continued on the next line. Make sure the hyphen is the last character you type before you press the RETURN key. The DMU prompt for a continuation is DMU> . The CDDV prompt for a continuation is CDDV> . For example:

```
DMU> COPY/AUDIT="COPIED FROM CORPORATE DIRECTORY"/HISTORY-
DMU>_/PROTECTION CORPORATE.ADDRESS_RECORD;1 SALES.JONES
```

Despite this ability to continue command lines, you must limit utility commands to 255 characters.

Note

You can enter only one command per line. DMU no longer accepts multiple commands on a single line separated by a semicolon. You should change any DCL or utility command procedures that use multiple commands on a single line.

A.4.2.4 CTRL/C in DMU Commands —If you enter a CTRL/C while a DMU command is executing, DMU aborts the execution and returns you to DMU command level. If you enter CTRL/C in response to the DMU> prompt, DMU returns you to DCL command level.

```
DMU> LIST *
^C
%DMU-E-CTRLCAST, Execution terminated by operator
DMU>
    .
    .
    .
DMU> ^C
$
```

Note

CTRL/C does not roll back any part of the command that has already been executed before you pressed CTRL/C. If, for example, you entered a command to delete a number of objects, those objects that had been deleted before you pressed CTRL/C remain deleted. Always check the extent to which the command executed before being terminated by CTRL/C. The only exceptions to this rule are the COPY/STAGE and RESTORE/STAGE commands, which withhold committing any changes until all changes are made.

A.4.3 Specifying Paths in the DMU Dictionary

The following sections describe the rules for specifying CDD path names to identify dictionary directories, subdictionaries, and objects in the DMU Dictionary.

A.4.3.1 Path Name – A path name consists of a string of given names separated by periods. It uniquely identifies a dictionary directory, subdictionary, or object through its line of ancestry from CDD$TOP. Because it is possible to have multiple versions of dictionary objects, object names end with a semicolon and version number.

For example, you can specify the dictionary directory STANDARDS in the sample dictionary (Figure A–1) with the following path name:

```
CDD$TOP.PERSONNEL.STANDARDS
```

The following is not a legal path name because a path name cannot contain consecutive periods:

```
CDD$TOP..PERSONNEL
```

A.4.3.2 Given Name – A given name is a string of up to 31 characters. The legal characters in a given name are A-Z, 0-9, underscore (_), and dollar sign ($). The first character must be a letter from A-Z, and the last character cannot be _ or $. If you are using a terminal of the VT200 family, you can use 8-bit alphabetic characters in given names. Remember that other terminals cannot reproduce 8-bit characters. In a given name, DMU translates all lowercase letters to uppercase.

For example, SALES and CDD$TOP are legal given names. S{L?S, however, is not a legal given name because it contains the illegal characters { and ?.

A.4.3.3 Version Numbers in Object Names – The given name of an object also includes a version number, separated from the rest of the given name by a semicolon (;). This number can be an absolute version number, a relative version number, or an asterisk (*) used as a wildcard. You can also use just a semicolon or omit the semicolon and number.

For the most part, version numbers used with CDD objects in the DMU Dictionary follow the same rules as version numbers used with VMS files. Table A-4 lists the various ways of specifying version numbers, the results of such specification, and an example of each specification.

Table A–4: Specifying Version Numbers with CDD Objects in the DMU Dictionary

Specification	Result	Example
Absolute version number[1]	DMU operates on the object with the specified version number.	SALARY_RANGE;2
Relative version number[1]	DMU operates on the object a specified number of versions below the highest version.	SALARY_RANGE;-1
Wildcard version number[1]	DMU operates on all versions of the object.	SALARY_RANGE;*
Semicolon without a version number[1]	DMU operates on the highest version of the object.	SALARY_RANGE;
No semicolon or version number[2]	DMU operates on the highest version of the object.	SALARY_RANGE

[1]You cannot use this specification with PURGE.

[2]If you use this specification with LIST, DMU lists all the children in the directory.

The use of certain version specifications is restricted in various ways depending on the command used. See the DMU command specifications in the *VAX Common*

Data Dictionary Utilities Reference Manual for a description of the version specifications that can be used with each command.

If the logical name CDD$VERSION_LIMIT has been defined for your system, group, or process, the dictionary will store only the number of versions allowed by the quota CDD$VERSION_LIMIT specifies.

The following example limits the number of versions of any object to three. Then a user tries to place a fourth version of the object ADDRESS_RECORD in the dictionary and receives an error message:

```
$ DEFINE CDD$VERSION_LIMIT "3"
$ CDDL/VERSION ADDRESS.DDL
        0001    DEFINE RECORD
                1
%CDDL-E-CDDERROR, error encountered while creating record
%CDD-E-EXCVERLIM, exceeded version limit for object
        0009    END ADDRESS_RECORD.
                                  1
%CDDL-E-RECNOTCRE, error in record definition -- record not created

Informational: 0
Warnings:      0
Errors:        3
Fatal Errors:  0
```

Because the DMU Dictionary does not support multiple versions of directories and subdictionaries, you cannot use a semicolon or version number at the end of directory and subdictionary names.

A.4.3.4 Wildcard Characters —With some commands in the Dictionary Management Utility, you can specify a path by using its proper path name or by including wildcard characters in the path name.

Legal DMU wildcards are %, *, > , and @:

- The % replaces any single character in a given name. For example, SA%ES is equivalent to SALES.
- The * replaces any number of characters, and its use is legal even if there is no corresponding character to replace. So, for example, SA* and SALES* are both legal wildcard specifications, and they could both identify the directory SALES. The * can also replace a version number in a CDD object. Note that an * used by itself refers to all the children of a directory, including *all* versions of any object in that directory. Be careful about using the * with the DMU DELETE command because DMU then deletes all versions of any object in the default directory.

- The > as the last character in a path name indicates that you want to include all the descendants of the specified dictionary directory or subdictionary, including all versions of any objects. If you end a path name with .> , the wildcard indicates that only the descendants are to be processed. If you end a path name with > , the wildcard without the preceding period, the DMU processes the last specified dictionary directory or subdictionary as well as all of its descendants.

  In the sample dictionary (Figure A-1), for example, the following path specification identifies CUSTOMER_RECORD;1, SALES_RECORD;1, JONES, and LEADS_RECORD;1:

  ```
  CDD$TOP.SALES.>
  ```

  The following path specification identifies SALES as well as JONES, CUSTOMER_RECORD;1, SALES_RECORD;1, and LEADS_RECORD;1:

  ```
  CDD$TOP.SALES>
  ```

- The @ prefixed to the given name of a dictionary directory or subdictionary signifies that the directory or subdictionary and all of its named descendants are to be processed. In the sample dictionary, for example, the following path specification directs the DMU to process SALES, JONES, and LEADS_RECORD;1:

  ```
  CDD$TOP.@SALES.JONES.LEADS_RECORD;1
  ```

Table A-5 is a summary of the DMU wildcard characters.

Table A-5: Dictionary Management Utility Wildcard Characters

Character	Signifies	Restrictions
%	Any single character.	None.
*	Any number of characters.	None.
>	All descendants of last specified directory or subdictionary.	Can be used only as the last character in a path specification.
@	All specified given names that follow it.	Can be used only once; it must precede a given name.

The use of wildcards is restricted in various ways depending on the command used. See the DMU command specifications in the *VAX Common Data Dictionary*

Utilities Reference Manual for a description of the wildcards that can be used with each command.

A.4.3.5 Hyphens in Path Specifications —The hyphen (-) in place of a given name in any path name indicates a name one generation back. This substitution is valid only for the first given names in the specification. You can use multiple hyphens; however, once you have specified a given name, you cannot use any hyphens further down in the chain. The hyphens must be first in the string. If your default directory is CDD$TOP.SALES.JONES, for example, you can specify the CORPORATE directory by typing -.-.CORPORATE.

Note that -.SALES.-.PRODUCTION is not a legal usage because the sequence SALES.- is not allowed.

Note

When using a hyphen in place of a given name, make sure that the hyphen is not the last character in a command line. DMU interprets a hyphen that ends a line as a continuation character (see Section A.4.2.3). If no given name follows a hyphen, end the line by typing a space.

A.4.3.6 Specifying Passwords in Path Names —Within any type of path specification, each dictionary directory, subdictionary, and object can have a password associated with it. To use a password in a path specification, enclose the password in parentheses. Place it immediately after the given name of the directory or subdictionary, or after the version number of the object, with which it is associated. Do not type a space between the given name and the password. If you are using a > at the end of a path name, the wildcard follows any password associated with the last given name in the chain.

Passwords contain from 1 to 64 printable characters, including space and tab. DMU translates lowercase letters to uppercase. The only forbidden characters in a password are left parenthesis [(], right parenthesis [)], and period [.]. If you associate a password with an object name, the password must follow the version number.

The following are legal given names with passwords:

```
PERSONNEL(SEMI_SECRET)
SERVICE(SECRET)
LEADS_RECORD;1(EYESONLY)
```

The following is a legal path name with passwords:

```
CDD$TOP.PERSONNEL(SEMI_SECRET).SERVICE(SECRET).SALARY_RECORD;1
```

The following is not a legal given name and password because it contains illegal characters:

```
PRODUCTION(B(AD.CH)ARS)
```

A.4.3.7 Using Logical Names in Path Names – You can use logical names to save keystrokes if you work in several dictionary directories with long path names. With the DCL commands DEFINE and ASSIGN, you can define logical names for CDD path names you use often.

Use either of the following formats in response to the DCL dollar sign prompt:

```
DEFINE logical-name "_CDD$TOP . . . given-name"
ASSIGN "_CDD$TOP . . . given-name" logical-name
```

For example, you could give CDD$TOP.PRODUCTION the logical name PROD by typing one of the following commands or by including it in your login command file:

```
$ DEFINE PROD "_CDD$TOP.PRODUCTION"
$ ASSIGN "_CDD$TOP.PRODUCTION" PROD
```

Once you have defined logical names, you can use them in place of path names in utility command lines.

The CDD attempts to translate the first given name of any path specification as a logical name. For example, if you specify the path name SALES.JONES in the sample dictionary (Figure A-1), the CDD makes one attempt to translate SALES. If SALES is not defined as a logical name, the translation fails, and the CDD utility processes the directory CDD$TOP.SALES.JONES.

If, however, SALES is defined as a logical name, the translation succeeds, and the CDD utility attempts to process a path name beginning with the translation string of SALES. If, for example, SALES is defined as CDD$TOP.PRODUCTION, then the CDD processes the path name SALES.JONES as:

```
CDD$TOP.PRODUCTION.JONES
```

Note

The CDD makes no more than one logical translation per name. If your logical name translates to another logical name, the CDD does not make the second translation. Make sure any logical names that you define translate to actual path names in one step.

CDD$TOP is an exception to this rule. The name CDD$TOP is always translated once and preceded by an underscore (_). If a logical name translates to an absolute pathname, then the first name (CDD$TOP) will be translated again.

To prevent logical name translation, prefix the path specification with an underscore (_). The CDD makes no attempt to translate SALES if you specify the path name:

```
_SALES.JONES
```

Instead, the CDD processes CDD$TOP.SALES.JONES.

Some DMU commands limit the use of logical names. See the reference section on a specific command in the *VAX Common Data Dictionary Utilities Reference Manual* for more information on that command's use of logical names.

A.4.4 Defining Your Default Directory in the DMU Dictionary

Each time you invoke an image that uses the CDD, you are assigned to a default directory. To refer to a dictionary directory or object that is a child of your default directory, you need to type only the child's relative path name.

The CDD automatically assigns CDD$TOP as your default directory. You can, however, begin sessions in another dictionary directory. You need only define the logical name CDD$DEFAULT as the full dictionary path name of the dictionary directory you want assigned to you as a default.

You can use either of two DCL commands, DEFINE or ASSIGN, in response to the DCL dollar sign prompt to define your default directory. The name you use with DEFINE or ASSIGN must be an actual path name, not a logical name. CDD$DEFAULT is a logical name, and the CDD makes only one logical translation per name.

To avoid having to retype the command line each time you log into the system, you should insert the command into your login command file.

In the sample dictionary (Figure A-1), for example, you could define JONES as the default dictionary directory by typing one of the following commands or by entering it in your login command file.

```
$ DEFINE CDD$DEFAULT "CDD$TOP.SALES.JONES"
$ ASSIGN "CDD$TOP.SALES.JONES" CDD$DEFAULT
```

You can then refer to any descendant of your default directory by either its full dictionary path name or by its relative dictionary path name.

For example, if your default directory in the sample dictionary is JONES, you have two choices for specifying the dictionary path name of LEADS_RECORD;1. You can use the full path name:

```
CDD$TOP.SALES.JONES.LEADS_RECORD;1
```

Or you can use the relative path name:

```
LEADS_RECORD;1
```

If your default directory were SALES, then the relative dictionary path to LEADS_RECORD;1 would be:

```
JONES.LEADS_RECORD;1
```

A.5 Using DMU Commands in the DMU Dictionary

The purpose of this section is to give you an opportunity to try out some of the commands and common operations of the Dictionary Management Utility. Section A.6 contains information on using the DMU SET PROTECTION and SET PROTECTION/EDIT commands. Section A.8 contains information on using DMU commands in conjunction with the Data Definition Language Utility (CDDL). For a complete description of DMU commands and their qualifiers, see the *VAX Common Data Dictionary Utilities Reference Manual.*

A.5.1 Checking Your Privileges in the DMU Dictionary

When you enter DMU, CDD places you in your default directory. CDD determines your default directory by translating the logical name CDD$DEFAULT (see Section A.4.4). Once you have entered DMU, check your default directory with the SHOW DEFAULT command:

```
DMU> SHOW DEFAULT
```

SHOW DEFAULT displays a full path name. If CDD cannot place you in the directory you specified or translate CDD$DEFAULT for you, it places you in CDD$TOP, the only directory sure to exist in every CDD. If your default directory is CDD$TOP, you should read Section A.4.4 before you continue.

Every directory and object in the CDD has an access control list (ACL) associated with it. The **access control list** contains the protection information for each directory or object. When you enter a command, CDD checks the access control lists of all the affected directories and objects to determine if you have the privileges necessary to perform the operation. In this section, you need only know how to determine which privileges you have been assigned. Section A.6 describes how to create and modify an access control list.

To check your privileges at your default directory, type:

```
DMU> SHOW PROTECTION
```

Because you did not specify a path name, DMU displays your privileges in your default directory. If you have all privileges, the output of the SHOW PROTECTION command is:

```
Control (C)        -- may control access control list
Local Delete (D)   -- may delete subdictionary, directory or object
Global Delete (G)  -- may delete subdictionary or directory and its
                      children
History (H)        -- may add entries to history list
Pass Thru (P)      -- may pass thru subdictionary or directory
See (S)            -- may see (read) dictionary object
Update (U)         -- may update dictionary object
Extend (X)         -- may create dictionary children
Forward (F)        -- may create subdictionaries
DTR Read (R)       -- may ready DATATRIEVE domain for read
DTR Write (W)      -- may ready DATATRIEVE domain for write
DTR Extend (E)     -- may extend DATATRIEVE table or procedure
DTR Modify (M)     -- may ready DATATRIEVE domain for modify
```

If you have not been granted all privileges, SHOW PROTECTION displays only the privileges you have. To perform the operations in Section A.5.1 through Section A.5.7, you need EXTEND, HISTORY, LOCAL_DELETE, PASS_THRU, SEE, and UPDATE. To perform the operations in Section A.5.8, you also need CONTROL privilege.

Note

You may lack sufficient privilege to perform some of the tasks in these sections. See your system manager or data administrator to obtain additional privileges.

A.5.2 Copying a Directory in the DMU Dictionary

After checking your privileges, copy the contents of CDD$TOP.CDD$EXAMPLES to your default directory. CDD$TOP.CDD$EXAMPLES is created during the installation procedure and contains the sample dictionary (Figure A-1). There are two differences between the sample dictionary in Figure A-1 and the installed version of the sample dictionary in CDD$TOP.CDD$EXAMPLES:

- In the sample dictionary, CDD$TOP is the parent of directories CORPORATE, PERSONNEL, PRODUCTION, and SALES. In the installed version of the sample dictionary, CDD$TOP.CDD$EXAMPLES is the parent of CORPORATE, PERSONNEL, PRODUCTION, and SALES.
- The directory CDD$TOP.PERSONNEL in the sample dictionary is a subdictionary directory; the directory CDD$TOP.CDD$EXAMPLES.PERSONNEL is not. Section A.9.1.5.1 describes how to create a subdictionary directory.

COPY takes the contents of one directory and duplicates them in another directory, so you must also check your privileges at CDD$TOP.CDD$EXAMPLES to determine if you have sufficient privilege to copy its contents. Type:

```
DMU> SHOW PROTECTION CDD$TOP.CDD$EXAMPLES>
```

The wildcard character > at the end of the path name indicates that you want to see your privileges at CDD$TOP.CDD$EXAMPLES and at every directory and object under it. You need PASS_THRU and SEE at each of these directories and objects to perform the operations in this section.

Once you have determined that you have sufficient privileges to copy CDD$TOP.CDD$EXAMPLES to your default directory, type:

```
DMU> COPY/LOG CDD$TOP.CDD$EXAMPLES
```

When you do not specify a file to hold the output, the /LOG qualifier displays the results of the COPY operation on your terminal screen. The following list is the result of the COPY/LOG command above.

```
    "CDD$EXAMPLES" copied
    "CORPORATE" copied
    "ADDRESS_RECORD;1" copied
    "EMPLOYEE_LIST;1" copied
    "PRODUCT_INVENTORY;1" copied
    "PERSONNEL" copied
    "PRODUCTION" copied
    "SERVICE" copied
    "SALARY_RECORD;2" copied
    "SALARY_RECORD;1" copied
    "STANDARDS" copied
    "SALARY_RANGE;2" copied
    "SALARY_RANGE;1" copied
    "SALES" copied
    "CUSTOMER_RECORD;1" copied
    "JONES" copied
    "LEADS_RECORD;1" copied
    "SALES_RECORD;1" copied
 DMU>
```

Note that the COPY command copied all directories and objects under CDD$EXAMPLES even though you did not specify the wildcard > . The COPY command automatically copies all the descendants of any specified directory.

The contents of the directory CDD$TOP.CDD$EXAMPLES are now in your default directory. To check, type:

```
DMU> LIST _CDD$EXAMPLES>
```

The above command specifies a relative path name. The underscore (_) before the path name prevents CDD from attempting to translate it as a logical name. DMU looks in your default directory for a directory named CDD$EXAMPLES. The wildcard "> " tells DMU to list every directory and object under CDD$EXAMPLES. DMU displays the following hierarchy:

```
CDD$EXAMPLES
   |  CORPORATE
   |  |  ADDRESS_RECORD;1 <CDD$RECORD>
   |  |  EMPLOYEE_LIST;1 <CDD$RECORD>
   |  |  PRODUCT_INVENTORY;1 <CDD$RECORD>
   |  PERSONNEL
   |  |  SERVICE
   |  |  |  SALARY_RECORD;2 <CDD$RECORD>
   |  |  |  SALARY_RECORD;1 <CDD$RECORD>
   |  |  STANDARDS
   |  |  |  SALARY_RANGE;2 <CDD$RECORD>
   |  |  |  SALARY_RANGE;1 <CDD$RECORD>
   |  PRODUCTION
   |  SALES
   |  |  CUSTOMER_RECORD;1 <CDD$RECORD>
   |  |  JONES
   |  |  |  LEADS_RECORD;1 <CDD$RECORD>
   |  |  SALES_RECORD;1 <CDD$RECORD>
```

The form of the above display is the LIST/BRIEF form, which is the default for LIST. LIST/BRIEF displays the name of each specified directory and object and its type. (LIST/BRIEF does not display a type for directories.) Indentation shows the hierarchical relationships among the specified directories and objects. LIST/BRIEF is a useful way to check on the contents and structure of a portion of the dictionary.

A.5.3 Changing Default Directories in the DMU Dictionary

You will perform the next several operations on descendants of CDD$EXAMPLES. To save yourself the trouble of typing "CDD$EXAMPLES" in every path name, you can temporarily change your default directory to CDD$EXAMPLES by typing:

```
DMU> SET DEFAULT CDD$EXAMPLES
```

To ensure that you are in the correct directory, type:

```
DMU> SHOW DEFAULT
```

Your new default directory should be the CDD$EXAMPLES directory that is a child of your original default directory.

A.5.4 Creating History List Entries in the DMU Dictionary

The CDD's **history list** feature enables you to monitor the use of each dictionary directory, subdictionary, and object. This list of operations makes up an **audit trail** for each dictionary element.

Use of the history list feature is optional. History lists take up dictionary space, but they can provide you with a concise record of dictionary usage.

You can add your own text to the history list of a dictionary directory, subdictionary, or object to enhance or explain the information automatically stored by the CDD. For example, you can record the names of the programs that access a data definition in the definition's history list. With the LIST command, you can retrieve this information easily.

Whenever you perform any operation that modifies the dictionary, it is wise to create a history list entry noting the change. Such documentation makes it much easier to maintain the consistency of the dictionary. You must have HISTORY privilege at the specified directory or object to create a history list entry.

Some dictionary operations automatically create a history list entry, but you must use the /AUDIT qualifier to create history list entries with DMU commands. You can use the /AUDIT qualifier with the following DMU commands:

- BACKUP
- COPY
- CREATE
- DELETE/PROTECTION
- EXTRACT
- MEMO
- RENAME
- RENAME/SUBDICTIONARY
- RESTORE
- SET PROTECTION

You probably should have used /AUDIT to document the COPY operation in the previous section, but it is unlikely that you had HISTORY privilege at CDD$TOP.CDD$EXAMPLES. However, because you have HISTORY privilege in your default directory, you can now insert a history list entry into the history lists of the directories and objects under your default directory by using the MEMO command. The sole purpose of MEMO is to insert a history list entry into the history list of a directory or object.

Ordinarily, the history list entry automatically describes the operation that occurred. Because MEMO can be used to document any dictionary operation, however, you must include text describing the purpose of the entry.

For example, you could document the source of the ADDRESS_RECORD;1 copied object by typing:

```
DMU> MEMO/AUDIT="Copied from CDD$TOP.CDD$EXAMPLES.CORPORATE.-
DMU>_ADDRESS_RECORD;1"  CORPORATE.ADDRESS_RECORD;1
```

The hyphen (-) after CORPORATE allows you to continue the command on the next line. Make sure it is the last character typed before a carriage return.

CORPORATE.ADDRESS_RECORD;1 is a relative path name indicating that the object is a descendant of your default directory. The history list entry created by this command resembles the following one, except that where CASADAY's process characteristics appear, your process characteristics would appear.

```
Memo entered by CASADAY (UIC [30,10]) in process CASADAY
    using VAX CDD Dictionary Management Utility Version 3.00
    on  7-JAN-1984 16:22:51.39.
Explanation:
    Copied from CDD$TOP.CDD$EXAMPLES.CORPORATE.ADDRESS_RECORD;1
```

A.5.5 Listing the Contents of a DMU Dictionary Object

In Section A.5.2, you used the LIST/BRIEF command to display the name and type of CDD$EXAMPLES and its descendants and their hierarchical relationship to each other. You can use another form of the LIST command, LIST/FULL, to display the entire contents of a directory or object. This command is most useful when you want to see what a particular dictionary object contains. The following example lists the contents of CDD$EXAMPLES.CORPORATE.ADDRESS_RECORD;1:

```
DMU> LIST/FULL CORPORATE.ADDRESS_RECORD;1
```

In this case, LIST displays information similar to the following listing. Where CASADAY's process characteristics appear, your process characteristics appear in your listing.

```
CDD$EXAMPLES.CORPORATE.ADDRESS_RECORD;1 <CDD$RECORD>
    Created by VAX CDD Data Definition Language Version 3.00
        on  6-JAN-1984 15:13:28.29 using protocol version 4.
    Source:
        DEFINE RECORD _CDD$TOP.CDD$EXAMPLES.CORPORATE.ADDRESS_RECORD
            DESCRIPTION IS
                /* This record contains the standard format
                for addresses.  It provides the source from which all
                address fields in other record descriptions are copied. */.
            ADDRESS STRUCTURE.
                STREET              DATATYPE IS TEXT
                                    SIZE IS 30 CHARACTERS.
                CITY                DATATYPE IS TEXT
                                    SIZE IS 30 CHARACTERS.
                STATE               DATATYPE IS TEXT
                                    SIZE IS 2 CHARACTERS.
                ZIP_CODE STRUCTURE.
                    NEW             DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 4 DIGITS
                                    BLANK WHEN ZERO.
                    OLD             DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 5 DIGITS.
                END ZIP_CODE STRUCTURE.
            END ADDRESS STRUCTURE.
        END ADDRESS_RECORD.
    Description:
         This record contains the standard format
         for addresses.  It provides the source from which all
         address fields in other record descriptions are copied.
    Memo entered by CASADAY (UIC [30,10]) in process CASADAY
         using VAX CDD Dictionary Management Utility Version 3.0
         on  7-JAN-1984 16:22:51.39.
    Explanation:
        Copied from CDD$TOP.CDD$EXAMPLES.CORPORATE.ADDRESS_RECORD;1
```

In addition to the path name and CDD type of the specified directory or object, a LIST/FULL command displays the following kinds of information:

- For objects, creation information, including the facility or product (such as CDDL or DATATRIEVE) that created the object, the date and time when it was created, and the version of the protocol that was used.
- For objects, the source text of the object, if the source text is stored in the CDD. For more information about source text, see Section A.7 and Section A.8.
- For objects, the optional description clause contained in the source text. The description information is available only if the person creating the source code entered description information. For information about adding description information to record definitions, see Section A.7.3.
- The history list. Each history list entry contains:
  - The operation performed.
  - The user name and UIC of the person initiating the operation.

- The process in which the operation occurred.
- The facility or product (such as DMU or a VAX language compiler) that performed the operation.
- The date and time when the operation occurred.
- Explanatory text added when the history list entry was created. "Copied from CDD$TOP.CDD$EXAMPLES.CORPORATE.ADDRESS_RECORD", the phrase you added to the /AUDIT qualifier, appears in the history list entry.

LIST/FULL does not display protection information. To see the access control list for a directory or object, use the LIST/PROTECTION command. You must have CONTROL privilege to use LIST/PROTECTION.

Other qualifiers to the LIST command allow you to see just the parts of this information that you are interested in. For example, if you use LIST/AUDIT_TRAIL, DMU displays the name of the directory or object and its history list. For a complete description of the LIST command and all its qualifiers, see the *VAX Common Data Dictionary Utilities Reference Manual.*

A.5.6 Removing Obsolete DMU Dictionary Directories and Objects

Obsolete directories and objects clutter the dictionary (just as unused files can clutter a VMS directory). Keeping obsolete directories and objects can slow CDD performance, increase the size of the dictionary file, and make dictionary maintenance more difficult. You should remove unused portions of the dictionary when you are sure they are no longer needed.

For example, the directory PERSONNEL.SERVICE contains two versions of the object SALARY_RECORD: SALARY_RECORD;2 and SALARY_RECORD;1. You can use LIST/FULL to compare the contents of the two objects:

```
DMU> LIST/FULL PERSONNEL.SERVICE.SALARY_RECORD
```

```
CDD$EXAMPLES.PERSONNEL.SERVICE.SALARY_RECORD;2 <CDD$RECORD>
    Created by VAX CDD Data Definition Language Version 3.00
        on  6-JAN-1984 15:13:43.10 using protocol version 4.
    Source:
        DEFINE RECORD _CDD$TOP.CDD$EXAMPLES.PERSONNEL.SERVICE.SALARY_RECORD
            DESCRIPTION IS
                /* This is the record containing salary
                information for all employees.  It is sensitive, and access
                is carefully restricted. Direct deposit information added
                5-JAN-1984.*/.
            SALARY STRUCTURE.
                EMPLOYEE_ID         DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 9 DIGITS.
                PAY STRUCTURE.
                    JOB_CLASS       DATATYPE IS TEXT
                                    SIZE IS 3 CHARACTERS.
                    INCR_LEVEL      DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 1 DIGIT.
                    WEEKLY_SALARY   DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 6 DIGITS 2 FRACTIONS.
                    DIRECT_DEP      DATATYPE IS TEXT
                                    SIZE IS 1 CHARACTER
                                    VALID FOR DTR IF "DIRECT_DEP=Y OR
                                    DIRECT_DEP=N".
                END PAY STRUCTURE.
            END SALARY STRUCTURE.
        END SALARY_RECORD RECORD.
    Description:
         This is the record containing salary
         information for all employees.  It is sensitive, and access
         is carefully restricted. Direct deposit information added
         5-JAN-1984.
CDD$EXAMPLES.PERSONNEL.SERVICE.SALARY_RECORD;1 <CDD$RECORD>
    Created by VAX CDD Data Definition Language Version 3.00
        on  6-DEC-1984 15:13:41.72 using protocol version 4.
    Source:
        DEFINE RECORD _CDD$TOP.CDD$EXAMPLES.PERSONNEL.SERVICE.SALARY_RECORD
            DESCRIPTION IS
                /* This is the record containing salary
                information for all employees.  It is sensitive, and access
                is carefully restricted. */.
            SALARY STRUCTURE.
                EMPLOYEE_ID         DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 9 DIGITS.
                PAY STRUCTURE.
                    JOB_CLASS       DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 3 DIGITS.
                    INCR_LEVEL      DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 1 DIGIT.
                    WEEKLY_SALARY   DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 6 DIGITS 2 FRACTIONS.
                END PAY STRUCTURE.
            END SALARY STRUCTURE.
        END SALARY_RECORD RECORD.
    Description:
         This is the record containing salary
         information for all employees.  It is sensitive, and access
         is carefully restricted.
```

By comparing the DESCRIPTION clauses of both SALARY_RECORD definitions, you can see that SALARY_RECORD;2 is a revision of SALARY_RECORD;1, created to add direct deposit information. A comparison of the source text reveals that the data type of the field JOB_CLASS has been changed from UNSIGNED NUMERIC to TEXT.

Similarly, the directory PERSONNEL.STANDARDS contains two versions of the object SALARY_RANGE. A comparison of these two objects reveals that SALARY_RANGE;2 is an updated version of SALARY_RANGE;1.

Because both PERSONNEL.SERVICE.SALARY_RECORD;1 and PERSONNEL.STANDARDS.SALARY_RANGE;1 are obsolete, you should remove them from the dictionary. The following sections describe ways to remove unwanted directories and objects from the CDD.

A.5.6.1 Backing Up Portions of the DMU Dictionary —Because you are now going to delete objects from PERSONNEL, you should first back up that directory to protect yourself against errors. You can back up the PERSONNEL directory with the following command:

```
DMU> BACKUP/AUDIT="PERSONNEL directory backed up" PERSONNEL PERS.BAK
```

Like COPY, BACKUP automatically operates on the descendants of a specified directory. PERS.BAK is a VMS file created to contain the backed-up directories and objects. PERS.BAK is your protection against deleting needed objects by mistake. DMU places PERS.BAK in your default VMS directory unless you specify another VMS directory in the command line.

Because you used the /AUDIT qualifier, DMU creates a history list entry in the history lists of each backed-up directory or object noting the BACKUP operation.

You can include the history list of a directory or object in the backup file by using the /HISTORY qualifier, and you can include the access control list in the backup file by using the /PROTECTION qualifier. In the preceding example, the backed-up directories and objects do not contain access control lists or any history list entries, so it is not necessary to use these two qualifiers.

A.5.6.2 Purging DMU Dictionary Objects

A.5.6.2 Purging DMU Dictionary Objects – You can delete obsolete versions of objects by using the DELETE command (see Section A.5.6.3). However, when you have a number of obsolete versions of objects under the same directory, it is easier to use the PURGE command.

PURGE deletes earlier versions of dictionary objects, keeping a specified number of the highest versions. By default, PURGE keeps one version.

Because the SALARY_RANGE and SALARY_RECORD objects are descendants of the directory PERSONNEL, and because PERSONNEL contains no other CDD objects, you can purge them with the following command:

```
DMU> PURGE/LOG PERSONNEL>
    "SALARY_RECORD;1" deleted
    "SALARY_RANGE;1" deleted
DMU>
```

The wildcard > after PERSONNEL tells DMU to purge all objects under PERSONNEL. (PURGE has no effect on directories.) If you do not use the /KEEP qualifier to specify the number of versions you want to keep, PURGE keeps only the most recent version of an object. The /LOG qualifier displays the names of purged objects on your terminal unless you specify a file to hold the output.

A.5.6.3 Deleting DMU Dictionary Objects – The PURGE command operates only on dictionary objects and always leaves at least one version of the object in the CDD. To remove any unwanted directory or object, use the DELETE command.

DELETE allows you to remove individual directories and objects or to remove a directory and all its descendants. You must have GLOBAL_DELETE privilege to remove a directory with descendants, however. In a well-managed dictionary, very few users have GLOBAL_DELETE privilege.

You could remove the objects PERSONNEL.SERVICE.SALARY_RECORD;2 and PERSONEL.STANDARDS.SALARY_RANGE;2 with DELETE. You have backed up the PERSONNEL directory, so you have saved these two objects.

```
DMU> DELETE/LOG PERSONNEL.SERVICE.SALARY_RECORD
    "SALARY_RECORD;2" deleted
DMU> DELETE/LOG PERSONNEL.STANDARDS.SALARY_RANGE
    "SALARY_RANGE;2" deleted
DMU>
```

You did not need to specify the objects' versions because there is only one version of each. If there had been multiple versions, the above commands would have removed only the highest version of each object.

A.5.7 Restoring Portions of the DMU Dictionary

The RESTORE command inserts the contents of a backup file into the CDD. You can restore a portion of the dictionary to the directory from which it was backed up or to a different directory. If you restore the contents of a backup file to a different directory, the combined BACKUP and RESTORE operations work as if you had performed a COPY operation.

You can even restore an object to a directory that already contains an object with the same name if you use the /VERSION qualifier. If an object with the same name and version number exists, however, the RESTORE operation aborts. See Section A.5.8 for more information about using the /VERSION qualifier with DMU commands.

Earlier in this section, you backed up the PERSONNEL directory into the file PERS.BAK. Then you removed all the objects in the PERSONNEL directory. To display the current contents of PERSONNEL, type:

```
DMU> LIST PERSONNEL>
   PERSONNEL
   |  SERVICE
   |  STANDARDS
```

You can now restore PERS.BAK and retrieve the definitions you otherwise would have lost. It does not matter that PERS.BAK also contains the directories PERSONNEL, SERVICE, and STANDARDS, which have not been deleted. If a directory to be restored already exists, the RESTORE operation continues.

A.5.7.1 Checking the Contents of a Backup File – Before you restore the directory, you should check the contents of the backup file to ensure that you are restoring the objects you need. The BACKUP/LIST command displays the contents of a backup file. BACKUP/LIST:

- Allows you to check that you are restoring the correct backup file
- Reports whether or not the backup file contains history and access control list information
- Allows you to determine if you need to use the /VERSION qualifier to restore the objects in the backup file

To display the contents of PERS.BAK, type:

```
DMU> BACKUP/LIST PERS.BAK
The backup file contains no history or ACL list information.

   PERSONNEL
   |  SERVICE
   |  |  SALARY_RECORD;2 <CDD$RECORD>
   |  |  SALARY_RECORD;1 <CDD$RECORD>
   |  STANDARDS
   |  |  SALARY_RANGE;2 <CDD$RECORD>
   |  |  SALARY_RANGE;1 <CDD$RECORD>
DMU>
```

Because you did not specify a file to hold the output, BACKUP/LIST sends the output to your terminal. The first sentence of the listing tells you whether you included history lists or access control lists in the backup file. In this case, you did not, because none of the objects or directories had history lists or access control lists. The rest of the listing is the same as the output from the LIST/BRIEF command.

A.5.7.2 Restoring a Backup File —Now that you are certain that the backup file contains the objects you want to restore, you should restore it. Type:

```
DMU> RESTORE/LOG PERS.BAK
   "PERSONNEL" restored
   "SERVICE" restored
   "SALARY_RECORD;2" restored
   "SALARY_RECORD;1" restored
   "STANDARDS" restored
   "SALARY_RANGE;2" restored
   "SALARY_RANGE;1" restored
DMU>
```

Your CDD$EXAMPLES directory should look just as it did when you started. To check CDD$EXAMPLES, set your default directory back to your original default. You can move up one level in the hierarchy by using a minus sign (-) as the path name. *Be sure to type a space or tab after the minus sign, however, so that DMU does not interpret it as a continuation character.*

```
DMU> SET DEF -
DMU> LIST CDD$EXAMPLES>
   CDD$EXAMPLES
   |  CORPORATE
   |  |  ADDRESS_RECORD;1 <CDD$RECORD>
   |  |  EMPLOYEE_LIST;1 <CDD$RECORD>
   |  |  PRODUCT_INVENTORY;1 <CDD$RECORD>
   |  PERSONNEL
   |  |  SERVICE
   |  |  |  SALARY_RECORD;2 <CDD$RECORD>
   |  |  |  SALARY_RECORD;1 <CDD$RECORD>
   |  |  STANDARDS
   |  |  |  SALARY_RANGE;2 <CDD$RECORD>
   |  |  |  SALARY_RANGE;1 <CDD$RECORD>
   |  PRODUCTION
   |  SALES
   |  |  CUSTOMER_RECORD;1 <CDD$RECORD>
   |  |  JONES
   |  |  |  LEADS_RECORD;1 <CDD$RECORD>
   |  |  SALES_RECORD;1 <CDD$RECORD>
DMU>
```

A.5.8 Creating Multiple Versions of DMU Dictionary Objects

You can create multiple versions of DMU Dictionary objects with three DMU commands: COPY, RENAME, and RESTORE. *The creation of multiple versions of CDD objects is not the default, however.* Each of these commands requires you to use the /VERSION qualifier to ensure that you know that you are creating an additional version of an existing object.

The following sections show you how to create multiple versions of CDD objects with DMU RENAME and DMU COPY.

Note

To attempt these operations, you need CONTROL privilege in your default directory in addition to the privileges you needed in Section A.5.1 through Section A.5.7.

A.5.8.1 Using RENAME/VERSION to Change a Version Number – There are cases in which the CDD prohibits the creation of an additional version even if you use the /VERSION qualifier. For example, DMU does not allow you to create an object with the same full path name and version number as an existing object.

Suppose you enter the command:

```
DMU> COPY/VERSION CDD$TOP.CDD$EXAMPLES.PERSONNEL.SERVICE.SALARY_RECORD;2 -
DMU>_CDD$EXAMPLES.PERSONNEL.SERVICE
```

DMU sends you the following error message:

```
%DMU-E-OBJALREXI, object "SALARY_RECORD;2" already exists - not superseded
```

Your SERVICE directory already contains an object named SALARY_RECORD;2, and the COPY command cannot change the version number of the SALARY_RECORD;2 object it is copying. Therefore, you cannot copy an object named SALARY_RECORD;2 into your SERVICE directory.

You must use the DMU RENAME command to change the given name or the version number of one of the two SALARY_RECORD;2 objects to copy the object CDD$TOP.CDD$EXAMPLES.PERSONNEL.SERVICE.SALARY_RECORD;2 into your CDD$EXAMPLES.PERSONNEL.SERVICE directory. You probably lack the privileges to rename objects under CDD$TOP.CDD$EXAMPLES, however, so you should change the name or version number of the SALARY_RECORD;2 object under your default directory.

RENAME changes the name of directories and objects within the same directory. It does not change the location of the directory or object within the dictionary hierarchy.

You could change the given name of SALARY_RECORD;2. In this case, however, you probably would not want to change the given name of the object, because the directory would then contain two versions of the same definition under two different names.

The RENAME/VERSION command, on the other hand, allows you to change the version number of an object without changing the given name. Once you change the version number of SALARY_RECORD;2, you can copy CDD$TOP.CDD$EXAMPLES.PERSONNEL.SERVICE.SALARY_RECORD;2 into your CDD$EXAMPLES.PERSONNEL.SERVICE directory.

The following commands change the version number of SALARY_RECORD;2 and check the results of the rename operation.

```
DMU> SET DEFAULT CDD$EXAMPLES.PERSONNEL.SERVICE
DMU> RENAME/VERSION SALARY_RECORD;2 SALARY_RECORD;3
DMU> LIST>
    SALARY_RECORD;3 <CDD$RECORD>
    SALARY_RECORD;1 <CDD$RECORD>
DMU>
```

Note that the RENAME/VERSION command changes nothing about the object except its version number. SALARY_RECORD;3 has the same history and access control lists as SALARY_RECORD;2 did. With RENAME, the object being renamed always retains its access control list and history list.

A.5.8.2 Using COPY/VERSION – Now that you have changed the version number of CDD$EXAMPLES.PERSONNEL.SERVICE.SALARY_RECORD under your default directory, you can copy CDD$TOP.CDD$EXAMPLES.PERSONNEL.SERVICE.SALARY_RECORD;2 by typing:

```
DMU> COPY/VERSION CDD$TOP.CDD$EXAMPLES.PERSONNEL.SERVICE.SALARY_RECORD;2
%DMU-I-HIGHVER, higher version of "SALARY_RECORD;2 already exists
```

Because you did not specify the destination directory, DMU copies the object to CDD$EXAMPLES.PERSONNEL.SERVICE, the default directory you specified in the last section. The /VERSION qualifier allows you to copy SALARY_RECORD;2 into this directory, even though it already contains SALARY_RECORD;3 and SALARY_RECORD;1.

With the DMU COPY command, you can use the /HISTORY qualifier to copy the history list from the object in the source directory to the object in the destination directory. If you do not use the /HISTORY qualifier, the copied object inherits the history list of the highest existing object with the same name in the directory, in this case, SALARY_RECORD;3. Because you are more interested in the history list in your own default directory than in CDD$TOP.CDD$EXAMPLES, it makes sense in this case not to copy the history list from the source directory.

You can also copy the access control list from the source directory by using the /PROTECTION qualifier with COPY. To use the /PROTECTION qualifier with COPY/VERSION, however, you must have CONTROL privilege at:

- The source object
- The destination directory
- The highest existing version of the object in the destination directory (in this case, SALARY_RECORD;3)

If you do not use the /PROTECTION qualifier with COPY/VERSION, the object inherits the access control list of the highest existing version of the object in the destination directory. Because you are unlikely to have CONTROL privilege under CDD$TOP.CDD$EXAMPLES, it makes sense not to copy the access control list from the source directory. If you did, you would not be able to perform certain operations on the object, even though it is under your default directory.

DMU sends you an informational message informing you that the version of SALARY_RECORD that you copied is not the highest version in the destination directory. This fact is important because programs and products that use a definition without specifying the version number receive the definition with the highest version number. If you want the newly copied version to be the highest version, use RENAME/VERSION again to change its version number.

A.5.9 Exiting from DMU

To leave DMU and return to DCL level, enter:

```
DMU> EXIT
$
```

You can also leave DMU and return to DCL level by using CTRL/Z:

```
DMU> ^Z
$
```

A.6 Security and Protection for the DMU Dictionary

The DMU Dictionary provides security mechanisms to protect the dictionary against unauthorized use. You should use these CDD access control facilities, along with the VMS file security mechanisms, when you plan and implement the security strategy for your dictionary.

Note

The CDD security facilities are a safeguard against unauthorized access by nonmalicious users. The CDD cannot intercept and prevent deliberate attempts to corrupt the dictionary.

A.6.1 Access Control Lists and Access Control List Entries in the DMU Dictionary

The key to the CDD's system of protection is the **access control list** (ACL). An ACL controls access to each dictionary directory, subdictionary, and object in the DMU Dictionary. Specifically, access control lists determine whether or not an individual user or class of users can:

- Create, modify, or delete a dictionary directory, subdictionary, or object
- See the definition of a dictionary object and use it in an application
- See or modify the information in the history list of a dictionary directory, subdictionary, or object
- See or modify the access control list of a dictionary directory, subdictionary, or object
- Use the given name of a dictionary directory or subdictionary in the path name of another directory, subdictionary, or object

When you first install the CDD on your system, all users have all access privileges to CDD$TOP. In addition, the default access control list created with a directory or object grants privileges to the creator and denies privileges to no one. This means that each user has access to every descendant of CDD$TOP, *unless his or her access privileges are explicitly modified.*

In addition, CDD access privileges are inherited. Users inherit rights granted them at CDD$TOP at all of the children of CDD$TOP. This inheritance of privileges continues as users move from parent to child throughout the directory hierarchy.

Access control lists at each dictionary directory, subdictionary, or object in the hierarchy modify inheritance by specifically granting or denying privileges to users or groups of users. Therefore, you can specify which users can use different portions of the dictionary.

The CDD creates an access control list for every directory and object it creates, unless you specify the /NOACL qualifier. The access control list contains one or more **access control list entries**. Each access control list entry consists of three parts:

- A position number identifying a particular entry within an access control list
- User identification criteria identifying the users whose privileges are defined in the entry
- Privilege specifications modifying the access rights these users inherit

The following example of an access control list contains two entries. The first line of each entry contains the user identification criteria, and the second line contains the granted, denied, and banished privileges, which are specified by key letters (see Section A.6.1.2).

```
CDD$TOP
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX, Deny - none, Banish - none
  2:    [*,*]
        Grant - P, Deny - CDEFGHMRSUWX, Banish - none
```

A.6.1.1 User Identification Criteria —The CDD user identification criteria include:

- VMS user names
- VMS user identifiers
  - numeric user identification code (UIC)
  - alphanumeric user identification code (UIC)
  - rights identifier
- Terminal numbers and job classes
- Passwords

Whenever a user process attempts to access a dictionary directory, subdictionary, or object, the CDD searches the ACL entries for the *first* entry whose user identification criteria match the characteristics of the process.

To match, all the specified user identification criteria must be the same as the corresponding process characteristics. For example, if user name were the only identification criterion specified in an ACL entry, then any process logging in under that user name would match. If, on the other hand, both a user name and a terminal number were specified as user identification criteria, then processes using that user name would match only if they originated from the specified terminal.

The CDD begins searching for a match at the first entry in the access control list. If the user's process characteristics match the user identification criteria, the CDD modifies the privileges the user inherits according to the privilege specifications in the matching entry. Then the CDD discontinues its search.

If there is no match, the associated privilege specifications do not apply, and the CDD continues searching for a match at the next entry. If there is no match after all the access control list entries have been searched, the user process's inherited privileges remain unchanged.

A.6.1.1.1 VMS User Name —The CDD compares user names specified in the access control list entries to the user name of a process to determine if there is a match.

A.6.1.1.2 VMS User Identifier —VMS provides three alternative types of identifier to describe users:

- Numeric UIC

  A numeric user identification code (UIC) is a two-part number that identifies a user and determines his or her relationship to other users on the system. Each part of a UIC is an octal number of up to five digits. The two parts are separated by a comma, and the entire UIC is enclosed in either square brackets or angle brackets.

  The first part of a UIC identifies the group to which a particular user belongs. Group members thus share the digits in the first part of their UICs. The second part of the UIC identifies an individual user within the group.

  In CDD access control lists, you can identify users by numeric UIC in any of four possible ways:

  - By specifying all the digits of both parts of a UIC, you can identify one user.
  - By using the asterisk (*) as a wildcard in place of the first part of the UIC, you can identify users who share the second part. A UIC specification of [*,10], for example, matches users with UICs of [10,10], [20,10], and [30,10].
  - By using the asterisk (*) as a wildcard in place of the second part of the UIC, you can identify users who share the first part and so belong to the same group. A UIC specification of [30,*], for example, matches users with UICs of [30,10], [30,15], and [30,20].
  - By using asterisks in place of both parts of a UIC, you can identify all users regardless of UIC. A UIC specification of [*,*], therefore, matches all users on the system.

  If no UIC is specified in an access control list entry, the CDD supplies [*,*] as a default.

- Alphanumeric UIC

  An alphanumeric user identification code (UIC) consists of a single text string within brackets. Thus you can indicate user JONES by specifying a UIC [JONES] as an alternative to specifying a user name JONES.

- Rights identifier

  You can specify a rights identifier to indicate all the members of a group. Rights identifiers are defined in the rights database by the system manager. A rights identifier of SECRETARY, for example, matches all users defined in the rights database as owners of the right "SECRETARY".

A.6.1.1.3 Terminal Number or Job Class —You can specify the terminal line number or job class of a process as a user identification criterion. There are several options:

- You can identify users working from a particular terminal line by specifying the terminal number in the format TTcn, TXcn or WTcn - TXA4, for example.
- You can identify all users whose terminal lines are hard-wired to your local system by using the keyword LOCAL.
- You can identify all users whose processes are running on anything other than a hard-wired line by using the keyword NON_LOCAL. This specification includes all processes running in batch mode and all processes using dial-up lines, DECnet, and the Distributed Data Manipulation Facility to run DATATRIEVE from a remote node in a network.
- You can identify all batch processes by using the keyword BATCH.
- You can identify processes using the Distributed Data Manipulation Facility to run DATATRIEVE from a remote node in a network by specifying the keyword NETWORK.

A.6.1.1.4 Passwords —You can specify a password as an identification criterion in an access control list entry. To match an ACL entry containing a password, users must add the password, enclosed in parentheses, to the given name of the target directory, subdictionary, or object. (See Section A.4.3.6 to learn how to specify passwords in path names.)

For example, the password SECRET is associated with the directory SERVICE in the sample dictionary (Figure A-1), and only those users who know and specify the password are entitled to use SERVICE in a path name. To gain access to SALARY_RECORD;2, a child of SERVICE, a user would have to supply the following dictionary path name:

```
CDD$TOP.PERSONNEL.SERVICE(SECRET).SALARY_RECORD;2
```

A.6.1.2 Access Control Privileges

A.6.1.2 Access Control Privileges –With the DMU's SET PROTECTION and SET PROTECTION/EDIT commands, you can modify access control lists to grant, deny, or banish access privileges to users whose process characteristics match the corresponding user identification criteria. Therefore, at any dictionary directory, subdictionary, or object, you can use the access control lists to modify the privileges a user or group of users inherits by:

- Granting specified access privileges. These privileges are added to any the user may have inherited from the parent of the current dictionary directory, subdictionary, or object.
- Denying specified access privileges. If you deny a privilege, the user no longer inherits that privilege further down in the hierarchy. You can, however, grant the privilege again at a lower level.
- Banishing specified access privileges. Banishing a privilege denies it to a user at the current dictionary directory or subdictionary and at all of its descendants. Once you have banished a privilege for a user or class of users, you cannot grant the privilege again at a lower level.

Note

Banishment of privileges is irreversible at descendants of the directory where you used BANISH. If you want to retain the option of restoring a privilege later, use DENY.

You can also allow a user or group of users to inherit access privileges without modification. To allow unmodified inheritance, do not include an access control list entry that matches the process characteristics of that user or users.

You can grant, deny, or banish any of 13 access control privileges. Of these, 9 are specifically CDD privileges, and the remaining 4 are DATATRIEVE access rights (see the *VAX DATATRIEVE Reference Manual*). Table A-6 briefly describes each of these 13 privileges.

Table A–6: Access Control Privileges in the DMU Dictionary

Privilege	Description
CONTROL (C)	Allows you to read, modify, and delete access control list entries. You cannot deny yourself CONTROL privilege.
DTR_EXTEND/EXECUTE (E)	Allows you to ready a DATATRIEVE domain for EXTEND access, to access a DATATRIEVE table, and to execute a DATATRIEVE procedure.
DTR_MODIFY (M)	Allows you to ready a DATATRIEVE domain for READ and MODIFY access.
DTR_READ (R)	Allows you to ready a DATATRIEVE domain for READ access.
DTR_WRITE (W)	Allows you to ready a DATATRIEVE domain for READ, WRITE, MODIFY, and EXTEND access.
EXTEND (X)	Allows you to create children of dictionary directories and subdictionaries.
FORWARD (F)	Allows you to create subdictionary files.
GLOBAL_DELETE (G)	Allows you to delete dictionary directories and subdictionaries, including any children they may have, with a single command.
HISTORY (H)	Allows you to add entries to history lists.
LOCAL_DELETE (D)	Allows you to delete dictionary objects, as well as directories and subdictionaries with no children, to edit DATATRIEVE procedures, and to replace or recompile data definitions stored in the CDD.
PASS_THRU (P)	Allows you to use a dictionary directory, subdictionary, or object in a path name. You cannot deny yourself PASS_THRU privilege.
SEE (S)	Allows you to see the definition of a dictionary object.
UPDATE (U)	Allows you to update the definition of a dictionary object and to create new versions of an object.

When you create a dictionary directory, subdictionary, or object, the default access control list entry grants privileges to your VMS user name.

For dictionary or subdictionary directories:

- CONTROL
- LOCAL_DELETE
- HISTORY
- PASS_THRU
- SEE
- EXTEND

For dictionary objects:

- CONTROL
- LOCAL_DELETE
- DTR_EXTEND/EXECUTE
- HISTORY
- DTR_MODIFY
- DTR_READ
- SEE
- UPDATE
- DTR_WRITE

When you create a new version of an already existing dictionary object, the default access control list created with the new version may vary. For more information about access control lists and the creation of versions, see Section A.5.8 and Section A.8.3.

A.6.1.3 Suggestions for Using Access Control Lists —For many applications, users need only DTR_READ, PASS_THRU, and SEE to access the data definitions in the CDD. To safeguard the dictionary, consider restricting most users to these privileges. Other suggestions include:

- Restricting full access at CDD$TOP to the system manager or data administrator responsible for organizing and maintaining the directory hierarchy. Limit all other users to PASS_THRU.

- Including an access control list entry at the bottom of each access control list containing only the wildcard UIC [*,*] and denying all privileges except PASS_THRU. All users who do not match any other ACL entry match this entry. This suggestion is most useful at the highest levels of the dictionary.
- Distributing control over the next level in the hierarchy. If, for example, your dictionary is organized by department, give department managers the privileges they need to manage their portions of the hierarchy.
- Making full access more widespread at the second level below CDD$TOP, where some data definitions are stored and where some users have personal directories assigned to them. Grant DTR_READ, PASS_THRU, SEE, and HISTORY to those users who need only to access record definitions and record audit trail information in history list entries. For those users with personal directories, you can include CONTROL, LOCAL_DELETE, and EXTEND as well.

Be especially careful about granting GLOBAL_DELETE, CONTROL, and FORWARD:

- CONTROL allows the user who possesses it to modify the access control list. Therefore, CONTROL is equivalent to having all access privileges. At the top levels of the hierarchy, limit CONTROL to the system manager or data administrator.
- FORWARD allows the user to create subdictionary files. Subdictionaries can be more secure than dictionary directories, but they require more time for I/O operations, and they are charged against FILLM, your VMS open file limit, and also against the SYSGEN CHANNELCNT quota. Whether or not you choose to use subdictionaries, you should limit the ability to create them to the system manager or data administrator.
- GLOBAL_DELETE allows the user to delete a directory or subdictionary *and all of its descendants*. Deny GLOBAL_DELETE to all users except the system manager or data administrator. Remember, however, that users who modify DBMS schemas need GLOBAL_DELETE privilege to use DBO/MODIFY.

In general, you should grant users only those privileges they need to work in their portions of the CDD.

A.6.2 Sample Access Control Lists in the DMU Dictionary

The following examples, taken from the sample dictionary (Figure A-1), demonstrate the use of access control lists.

A.6.2.1 Accumulation of Privileges

A.6.2.1 Accumulation of Privileges – Along the path CDD$TOP.SALES.JONES.LEADS_RECORD;1, each of the four segments has an access control list associated with it. An examination of each ACL entry matching user JONES shows how privileges accumulate as they are granted, inherited, denied, and banished.

```
CDD$TOP
  1:    [*,*], Username: "JONES"
        Grant - P, Deny - CDEHMRSUWX, Banish - FG
```

At CDD$TOP, user JONES has PASS_THRU, which allows her to use CDD$TOP in a path name. She has also had two privileges banished, FORWARD and GLOBAL_DELETE.

```
CDD$TOP.SALES
  1:    [*,*], Username: "JONES"
        Grant - H, Deny - none, Banish - none
```

At the directory SALES, JONES is granted an additional privilege, HISTORY. Because she inherits PASS_THRU from CDD$TOP, JONES now has PASS_THRU and HISTORY.

```
CDD$TOP.SALES.JONES
  1:    [*,*], Username: "JONES"
        Grant - CFGSX, Deny - none, Banish - none
```

This access control list attempts to grant JONES five new privileges: CONTROL, FORWARD, GLOBAL_DELETE, SEE, and EXTEND. However, the ACL for CDD$TOP has already banished two of these, FORWARD and GLOBAL_DELETE, so they cannot be granted now. JONES, therefore, adds CONTROL, SEE, and EXTEND to her inherited privileges. She now has PASS_THRU, HISTORY, CONTROL, SEE, and EXTEND.

With CONTROL privilege at her directory, JONES can use the SET PROTECTION command to restrict access by other users.

```
CDD$TOP.SALES.JONES.LEADS_RECORD;1
  1:    [*,*], Username: "JONES"
        Grant - DEMRUW, Deny - none, Banish - none
```

The six new privileges JONES receives at dictionary object LEADS_RECORD;1 give her complete control over that record definition. She can use DATATRIEVE, she can update the record definition, and she can read and write access control list entries. She now has PASS_THRU, HISTORY, CONTROL, SEE, EXTEND, LOCAL_DELETE, DTR_EXTEND/EXECUTE, DTR_MODIFY, DTR_READ, UPDATE, and DTR_WRITE.

A.6.2.2 Combinations of ACL Entries

A.6.2.2 Combinations of ACL Entries – Along the path CDD$TOP.PERSONNEL.SERVICE.SALARY_RECORD;2, the access control lists demonstrate how combinations of user identification criteria at different levels of the CDD hierarchy control access to data descriptions. The user identification criteria in these ACL entries match the characteristics of four individual users: CASADAY, KELLERMAN, FOSTER, and JONES.

```
CDD$TOP
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX,  Deny - none, Banish - none
  2:    [*,*]
        Grant - P, Deny - CDEHMRSUWX, Banish - FG
```

CASADAY, the system manager, is responsible for organizing and maintaining the CDD, and so she retains all the access privileges granted by default. CASADAY inherits these privileges at each hierarchy level.

To protect the CDD against modification or redundancy, CASADAY grants only PASS_THRU to all other users, including KELLERMAN, FOSTER, and JONES. In addition, she banishes FORWARD and GLOBAL_DELETE to ensure that no other user can create subdictionary files or delete large portions of the dictionary.

```
CDD$TOP.PERSONNEL
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX, Deny - none, Banish - none
  2:    [*,*], Password: "SEMI_SECRET"
        Grant - HS, Deny - CDEFGMRUWX, Banish - none
  3:    [*,*]
        Grant - none, Deny - none, Banish - CDEFGHMPRSUWX
```

All the record definitions in the PERSONNEL subdictionary are sensitive, and so CASADAY has included the password "SEMI_SECRET" as a user identification criterion to restrict access. KELLERMAN, who is responsible for all the records in the Personnel Department, and FOSTER, an applications programmer who uses personnel record definitions, know the password and so have PASS_THRU (inherited from CDD$TOP), HISTORY, and SEE privileges to the subdictionary. JONES, who works in Sales, does not know the password and so has no access to PERSONNEL or any of its children.

The relative position of access control list entries is significant. The CDD stops searching the user identification criteria in the ACL entries as soon as it finds a match. In this example, if entries 2 and 3 were reversed, then only CASADAY would have any access privileges at PERSONNEL. All other processes would match the second entry, which banishes all privileges. Therefore, the CDD would discontinue the user identification search before reaching entry 3, the one granting access to those using the password.

```
CDD$TOP.PERSONNEL.SERVICE
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX, Deny - none, Banish - none
  2:    [*,*], Password: "SECRET"
        Grant - HS, Deny - DEMRUWX, Banish - C
  3:    [*,*]
        Grant - none, Deny - none, Banish - CDEFGHMPRSUWX
```

Confidential employee record definitions are stored in the SERVICE directory. CASADAY has added a new password, "SECRET," to limit the number of Personnel Department users with access to this directory. Authorized users like KELLERMAN and FOSTER now have access to this directory only when they include the appropriate passwords in the dictionary path name:

```
CDD$TOP.PERSONNEL(SEMI_SECRET).SERVICE(SECRET)
```

Failure to include either password would result in the banishment of all privileges, and users would be unable to proceed further, even if the next level granted all privileges to all users.

```
CDD$TOP.PERSONNEL.SERVICE.SALARY_RECORD;2
  1:    [*,*], Username: "CASADAY"
        Grant - CDEHMRSUW, Deny - none, Banish - none
  2:    [*,*], Username: "KELLERMAN"
        Grant - DEMRUW, Deny - none, Banish - none
  3:    [*,*], Username: "FOSTER"
        Grant - ER, Deny - none, Banish - none
```

Unlike CDD$TOP, PERSONNEL, or SERVICE, SALARY_RECORD;2 is a dictionary object that holds a record definition. This difference is reflected in the new default privileges that DMU displays for CASADAY.

ACL entry 2 grants KELLERMAN the privileges he needs to maintain the SALARY_RECORD;2 definition. FOSTER, whose process matches the third access control list entry, inherits PASS_THRU, HISTORY, and SEE from SERVICE, and he receives DTR_EXTEND and DTR_READ, which allow him to use DATATRIEVE with SALARY_RECORD;2.

A.6.3 Summary of ACL Results in the DMU Dictionary

Here is a summary of the effects access control lists can have on user access to the CDD:

- All users receive all privileges at CDD$TOP when you first install the CDD.
- Users inherit privileges from parent to child within the hierarchy until you grant, deny, or banish specific privileges in access control lists.
- Your access privileges to a dictionary directory, subdictionary, or object are those you inherit, modified by the current ACL entry.
- An ACL entry applies to a user only if all its user identification criteria match the characteristics of the user's process.
- The CDD begins its search of an access control list with entry 1 and ends its search as soon as it finds the first entry for which the characteristics of the user's process match all the user identification criteria.
- No one can restore a banished privilege to a user simply by granting it. If an access privilege is banished by an ACL entry, that privilege can be restored only if the banishment is deleted from the access control list.

A.6.4 Modifying Access Control Lists in the DMU Dictionary

If you have CONTROL privilege at a directory or object, you can modify its access control list. You can use the DMU SET PROTECTION command, as in the following example:

```
DMU> SET PROTECTION/POSITION=2/USERNAME=JONES/GRANT=<HS> CDD$TOP.SALES
DMU>
```

The above command creates an access control list entry at CDD$TOP.SALES for user JONES. The entry is the second entry in the ACL and grants JONES HISTORY and SEE privileges.

You can then use LIST/PROTECTION to see the access control list you have modified:

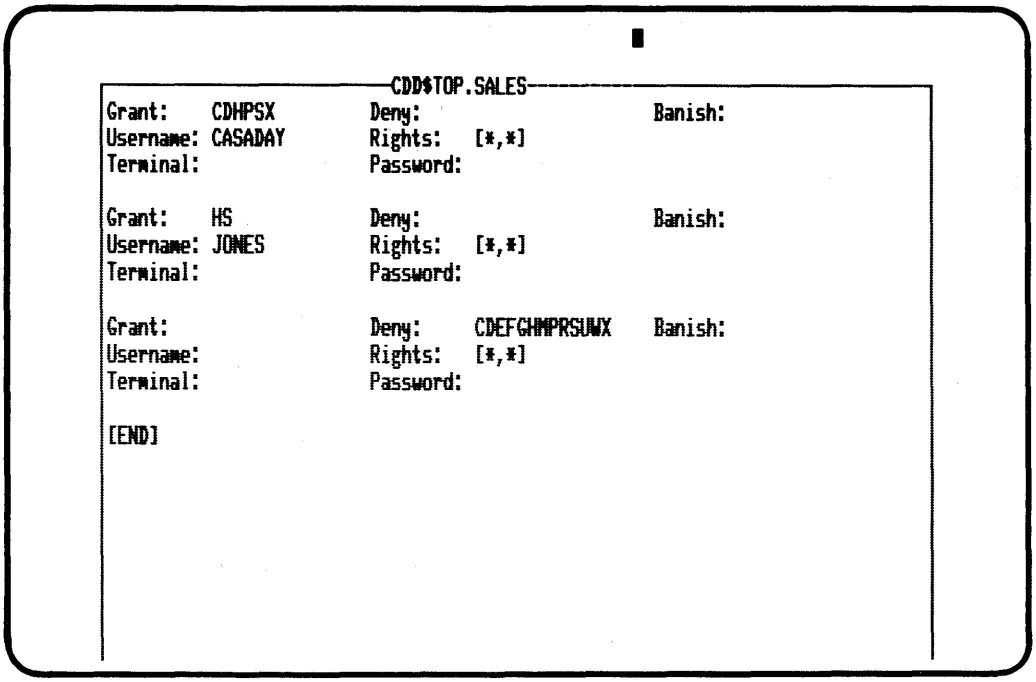
```
DMU> LIST/PROTECTION CDD$TOP.SALES

CDD$TOP.SALES
  1:    [*,*], Username: "CASADAY"
        Grant - CDEFGHMPRSUWX, Deny - none, Banish - none
  2:    [*,*], Username: "JONES"
        Grant - HS, Deny - CDEFGMPRUWX, Banish - none
DMU>
```

If you are using a VT52 terminal or a terminal of the VT100 or VT200 family, you can easily modify an access control list with the DMU SET PROTECTION /EDIT command. It allows you to edit access control lists with single-stroke keypad commands. SET PROTECTION/EDIT allows you to see the entire access control list as you are modifying it. In the sample dictionary (Figure A-1), for example, the system manager CASADAY has full access to the directory CDD$TOP.SALES. She can, therefore, modify the access control list. This section presents a sample VT100 session in which CASADAY uses the ACL editor to grant a new user named DENN access to CDD$TOP.SALES.

First, CASADAY invokes DMU and runs the access control list editor:

```
$ RUN SYS$SYSTEM:DMU
DMU> SET PROTECTION/EDIT CDD$TOP.SALES.
```

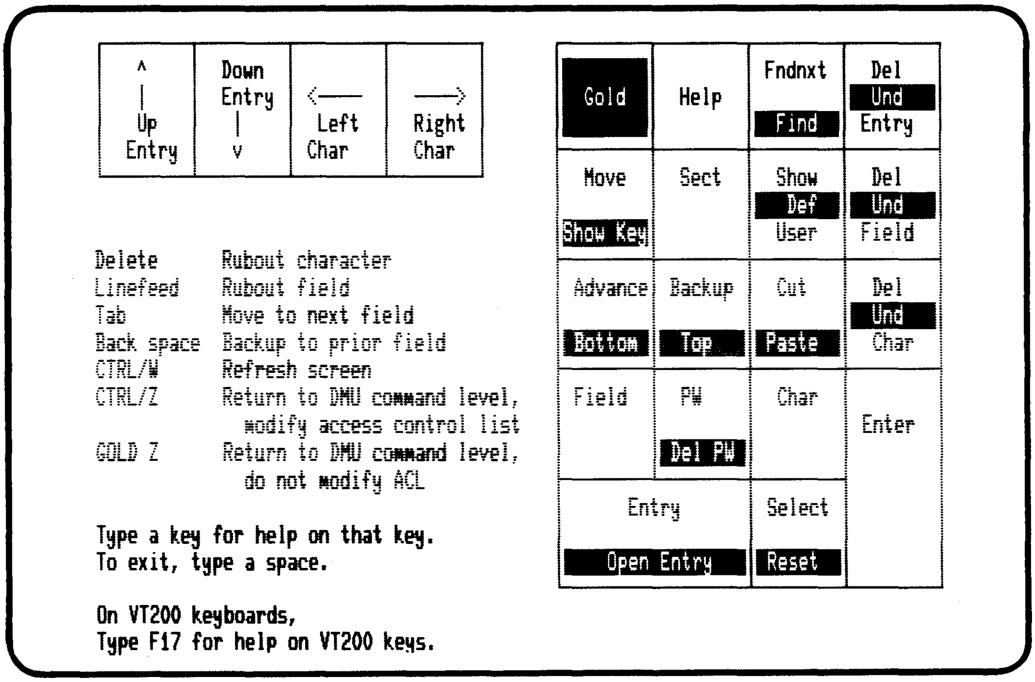
DMU then displays the current access control list:



```
                             CDD$TOP.SALES
Grant:    CDHPSX        Deny:                   Banish:
Username: CASADAY       Rights:   [*,*]
Terminal:               Password:

Grant:    HS            Deny:                   Banish:
Username: JONES         Rights:   [*,*]
Terminal:               Password:

Grant:                  Deny:     CDEFGHMPRSUWX Banish:
Username:               Rights:   [*,*]
Terminal:               Password:

[END]
```

ZK-8633-HC

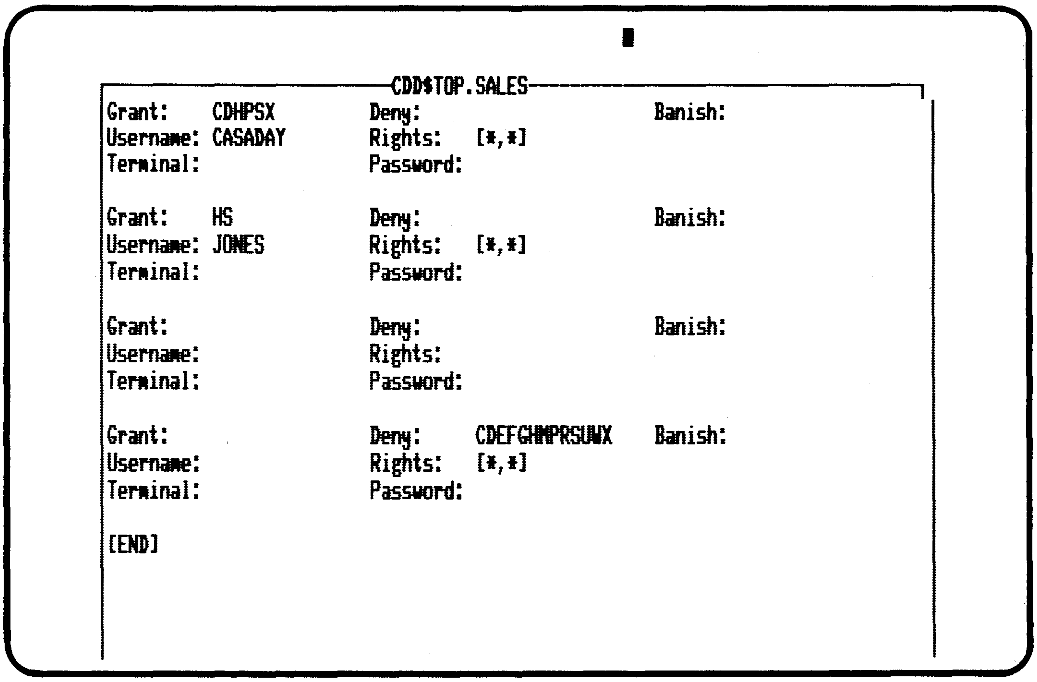
On the VT100, the keypad PF2 key is the HELP key for the access control list editor. CASADAY presses HELP, and her terminal displays the following:



```
Delete      Rubout character
Linefeed    Rubout field
Tab         Move to next field
Back space  Backup to prior field
CTRL/W      Refresh screen
CTRL/Z      Return to DMU command level,
              modify access control list
GOLD Z      Return to DMU command level,
              do not modify ACL

Type a key for help on that key.
To exit, type a space.

On VT200 keyboards,
Type F17 for help on VT200 keys.
```

ZK-8632-HC

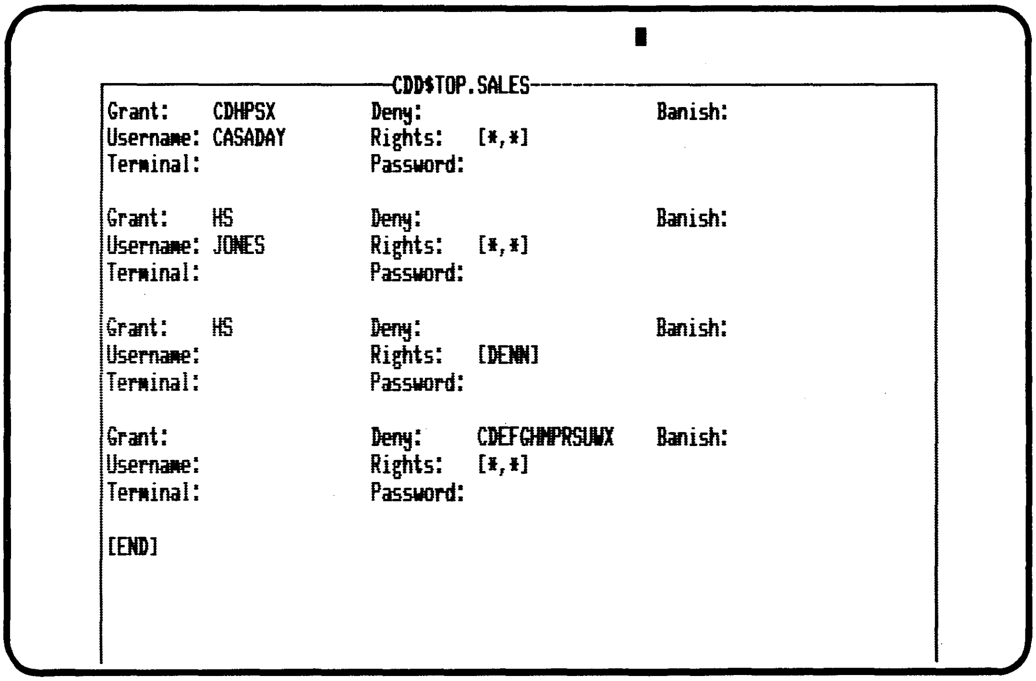
To return to editing, CASADAY presses the space bar. Because she wants to grant privileges to DENN at position 3, she presses the keypad ENTRY (0) key twice to move the cursor down to the third position, and then she opens a new entry at that position by pressing OPEN ENTRY (GOLD 0). She is now ready to add the new access control list entry:



```
                    CDD$TOP.SALES
Grant:    CDHPSX       Deny:                  Banish:
Username: CASADAY      Rights:   [*,*]
Terminal:              Password:

Grant:    HS           Deny:                  Banish:
Username: JONES        Rights:   [*,*]
Terminal:              Password:

Grant:                 Deny:                  Banish:
Username:              Rights:
Terminal:              Password:

Grant:                 Deny:    CDEFGHMPRSUWX Banish:
Username:              Rights:  [*,*]
Terminal:              Password:

[END]
```

ZK-8634-HC

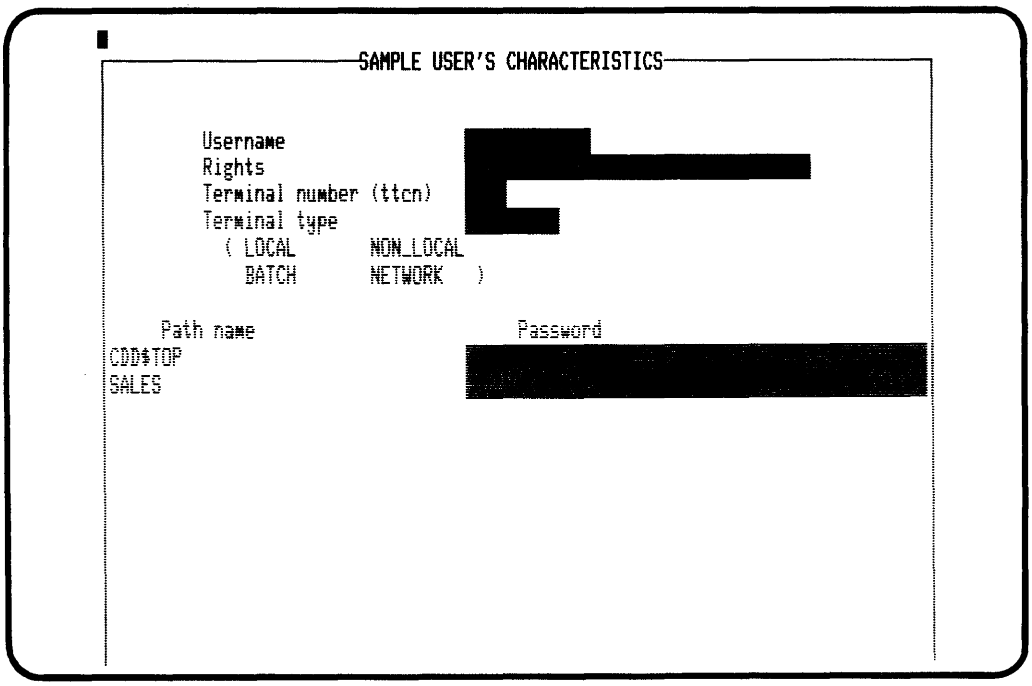
DENN inherits PASS_THRU from CDD$TOP, so he needs only HISTORY and SEE to work in the SALES directory. CASADAY, therefore, types the key letters H and S in the Grant field. She then uses the keypad FIELD key (1) to move the cursor to the Rights field, where she types DENN. Note that she could, alternatively, have identified him by typing DENN in the Username field.



```
                    ▮
---------------------CDD$TOP.SALES------------------------------
Grant:    CDHPSX        Deny:                     Banish:
Username: CASADAY       Rights:   [*,*]
Terminal:               Password:

Grant:    HS            Deny:                     Banish:
Username: JONES         Rights:   [*,*]
Terminal:               Password:

Grant:    HS            Deny:                     Banish:
Username:               Rights:   [DENN]
Terminal:               Password:

Grant:                  Deny:     CDEFGHMPRSUWX   Banish:
Username:               Rights:   [*,*]
Terminal:               Password:

[END]
```

ZK-8630-HC

Before committing the access control list change, CASADAY can test the new entry to be sure it functions correctly. She presses the DEFINE USER key (GOLD 9) to enter sample user characteristics, and the SAMPLE USER form appears on her terminal screen:



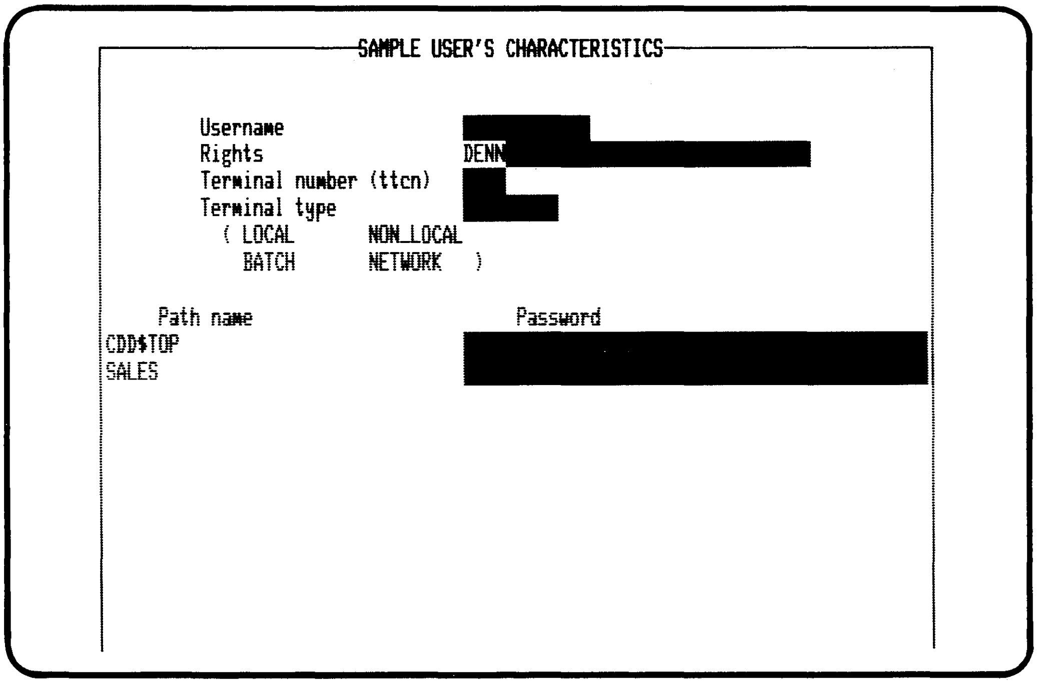
```
                    SAMPLE USER'S CHARACTERISTICS

      Username
      Rights
      Terminal number (ttcn)
      Terminal type
        ( LOCAL      NON_LOCAL
          BATCH      NETWORK   )

    Path name                     Password
CDD$TOP
SALES
```

ZK-8631-HC

The SAMPLE USER screen allows her to enter user identification criteria for any user. She can enter a user name, a rights identifier or UIC, a terminal number, a terminal type, and any necessary passwords. The SAMPLE USER screen contains a line for every directory from CDD$TOP to the directory or object whose ACL you are modifying. To associate a password with a directory or object, type the password on the correct line. The SAMPLE USER screen can display up to ten directory or object names at a time. If the path name is more than ten levels deep, you can scroll down with the down arrow key or up with the up arrow key.

In this case, CASADAY could enter a password for CDD$TOP or SALES.

Because no password is associated with either directory, CASADAY just types the alphanumeric UIC DENN in the Rights space.

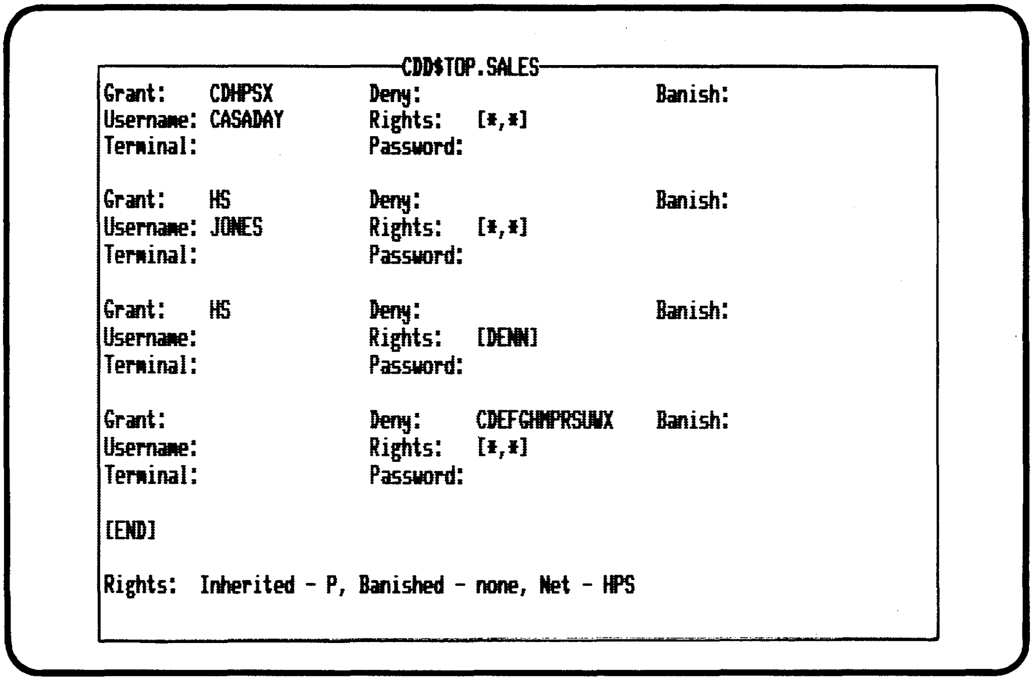


```
                    SAMPLE USER'S CHARACTERISTICS

       Username
       Rights                       DENN
       Terminal number (ttcn)
       Terminal type
         ( LOCAL        NON_LOCAL
           BATCH        NETWORK    )

    Path name                          Password
CDD$TOP
SALES
```

ZK-8629-HC

She enters the sample user characteristics by pressing ENTER. The ACL editor reappears on the screen.

To test the new entry, CASADAY presses the SHOW USER key (9):



```
                        CDD$TOP.SALES
Grant:    CDHPSX        Deny:                   Banish:
Username: CASADAY       Rights:   [*,*]
Terminal:               Password:

Grant:    HS            Deny:                   Banish:
Username: JONES         Rights:   [*,*]
Terminal:               Password:

Grant:    HS            Deny:                   Banish:
Username:               Rights:   [DENN]
Terminal:               Password:

Grant:                  Deny:     CDEFGHMPRSUWX Banish:
Username:               Rights:   [*,*]
Terminal:               Password:

[END]

Rights:  Inherited - P, Banished - none, Net - HPS
```

ZK-8627-HC

The line under the ACL display shows the privileges granted DENN at higher levels of the dictionary, any banished privileges, and his net privileges at CDD$TOP.SALES. He inherits PASS_THRU and has had no privileges banished. His inherited PASS_THRU privilege, combined with HISTORY and SEE privileges granted him at this directory, make his net privileges HPS.

CASADAY now presses CTRL/Z to commit the change and return to DMU command level. The LIST/PROTECTION command displays the new access control list:

```
DMU> LIST/PROTECTION CDD$TOP.SALES

CDD$TOP.SALES
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX, Deny - none, Banish - none
  2:    [*,*], Username: "JONES"
        Grant - HS, Deny - none, Banish - none
  3:    [DENN]
        Grant - HS, Deny - none, Banish - none
  4:    [*,*]
        Grant - none, Deny - CDEFGHMPRSUWX, Banish - none
```

When you use SET PROTECTION/EDIT, LIST/PROTECTION, and SET PROTECTION, remember:

- The relative position of entries is important. CASADAY had to be sure that the ACL entry for DENN came before the entry denying all privileges to all users.
- When you press the SHOW USER key (9), the ACL editor displays privileges granted at higher levels in the hierarchy, any banished privileges, and the net privileges a user has at the referenced directory or object. The LIST /PROTECTION command shows only the privileges a user has been granted, denied, or banished at the named directory or object, not the privileges inherited from higher directories.

If you are using a VT52 terminal or a terminal of the VT100 or VT200 family, you should use the ACL editor to modify access control lists because:

- The ACL editor allows you to use single-stroke keypad commands instead of entering commands from DMU command level.
- The ACL editor displays the whole access list for you.
- The ACL editor allows you to test new entries before you commit them. If any problems arise, you can abort the editing session by pressing GOLD Z.

Otherwise, use a combination of the DELETE/PROTECTION and SET PROTECTION commands to fine tune your dictionary's access control lists.

A.6.5 Using VMS File Protection in the DMU Dictionary

CDD access control lists provide protection against nonmalicious users, but they cannot provide complete protection for dictionary entries that contain sensitive material or must be kept secure. You can use VMS file protection to augment protection provided by access control lists.

The root dictionary directory is stored in a file with the name CDD.DIC. All the descendants of that directory are also stored in that file unless they are subdictionaries. Subdictionaries are stored in separate files that you specify in the DMU CREATE/SUBDICTIONARY command (see Section A.9.1.5.1).

A.6.5.1 The DCL SET PROTECTION Command – You can use the DCL SET PROTECTION command to set the protection of the root dictionary file and any subdictionary files. File protection provides another layer of protection in addition to the access control list. Rights denied through file protection cannot be gained through an access control list. For example, if you deny someone READ access to CDD.DIC, it does not matter which CDD access privileges you grant that person; that person cannot open the file.

For any file, four VMS file protection rights can be granted:

- READ (R): the right to examine, print, or copy a file
- WRITE (W): the right to modify a file
- EXECUTE (E): the right to execute a file that contains an executable image
- DELETE (D): the right to delete a file

Each of these rights can be granted to four different categories of users:

- SYSTEM (S): all users with system privileges
- OWNER (O): any user with the same user identification code as the creator of the file (for more information about the UIC, see Section A.6.1.1.2.)
- GROUP (G): all users who have the same group number (the first three digits of the UIC) as the owner of the file
- WORLD (W): all users of the system

The format of the DCL SET PROTECTION command is:

```
SET PROTECTION=(S:rights,O:rights,G:rights,W:rights) file-spec
```

The rights are any of the four protection rights you want to grant.

A.6.5.2 Granting Access to Certain Users – You have considerable flexibility in granting access to CDD.DIC. By default, all users are granted all rights except DELETE, as in the following command:

```
$ SET PROTECTION=(S:RWE,O:RWE,G:RWE,W:RWE) CDD.DIC
```

Note

Because CDD.DIC contains your root dictionary directory, it is inadvisable to grant anyone DELETE rights.

If, on the other hand, you want to grant WRITE and EXECUTE access only to yourself, type:

```
$ SET PROTECTION=(S:R,O:RWE,G:R,W:R) CDD.DIC
```

It is also possible to deny all access to anyone outside the group:

```
$ SET PROTECTION=(S,O:RWE,G:R,W) CDD.DIC
```

Choose the combination of rights that suits your needs.

A.6.5.3 Controlling Access to Subdictionaries — Certain directories and objects of the dictionary may need to be more secure than other directories and objects. PERSONNEL, a subdictionary directory in the sample dictionary, is an example. Such subdictionary directories are stored in files separate from the rest of the dictionary. You specify the location of such files by using the DMU CREATE/SUBDICTIONARY command. The PERSONNEL subdictionary, for instance, is stored in the file DB3:[CASADAY.CDD]PERS.DIC.

Because PERSONNEL is stored in a separate file, it can be protected differently from the root dictionary file, CDD.DIC. If extremely sensitive material is kept in that subdictionary file, you might grant access only to its owner and the system:

```
$ SET PROTECTION=(S:RWE,O:RWE,G,W) DB3:[CASADAY.CDD]PERS.DIC
```

Again, it is advisable not to grant DELETE rights to anyone.

If, however, members of the Personnel Department need to make changes to parts of that directory regularly, you can grant access to them, while denying access to other users:

```
$ SET PROTECTION=(S:RWE,O:RWE,G:RWE,W) DB3:[CASADAY.CDD]PERS.DIC
```

In some situations, security is the overriding concern in the use of the CDD. For example, several organizations may share a CDD through a time-sharing program. Under these circumstances, an organization may want assurances that their definitions are secure. You can store their definitions on a separate device and take it off line when it is not in use.

Putting certain directories and objects in subdictionaries allows you to tailor access to those sections, according to the needs of your organization.

For more information about creating and using subdictionaries, see Section A.9.1.5. For more information about the DCL SET PROTECTION command, see the related information in the VMS documentation set.

A.6.5.4 The DCL Access Control List Editor Utility –The DCL EDIT/ACL command invokes the VMS Access Control List (ACL) Editor Utility. You can use this utility to create or modify a VMS access control list for a specified root dictionary file or subdictionary file. EDIT/ACL allows you to edit VMS access control lists more quickly and easily than with the DCL SET PROTECTION command. The utility provides an easy way to see an entire access control list for a VMS file.

The format of the DCL EDIT/ACL command is:

```
EDIT/ACL file-specification
```

The ACL Editor prompts for each VMS access control entry. Each entry consists of two parts separated by a comma:

- An identifier. The identifier can be a numeric (group and member numbers) UIC, an alphanumeric UIC, or a rights identifier created in the rights database by the system manager. The identifiers BATCH, DIALUP, INTERACTIVE, LOCAL, NETWORK, and REMOTE exist by default on the system.
- Privileges associated with the identifier. In addition to READ, WRITE, EXECUTE, and DELETE privileges, you can specify CONTROL and NONE. CONTROL grants all the privileges of the file's owner.

The format of an ACL entry is:

```
(IDENTIFIER=[identifier], ACCESS=privilege[+privilege...])
```

The following example creates an VMS access control list for the subdictionary file CORP.DIC. CASADAY, the system manager, grants herself all five privileges. The second entry grants READ and WRITE access to any user on a local, hard-wired terminal whose group number is 210. CASADAY has created the identifier SECRETARY and assigned it in the rights database. Any user defined as SECRETARY in the rights database matches this entry. Any batch job or any user in group 210 whose terminal is not locally hard-wired receives only READ access. The sixth entry denies file access to any user whose terminal is not hard-wired. This entry is optional because CASADAY inludes the final entry as a safeguard to deny access to any user whose characteristics are not specified in a higher entry.

```
$ EDIT/ACL CORP.DIC
(IDENTIFIER=[CASADAY], ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
(IDENTIFIER=[210,*]+LOCAL, ACCESS=READ+WRITE)
(IDENTIFIER=SECRETARY, ACCESS=READ+WRITE)
(IDENTIFIER=BATCH, ACCESS=READ)
(IDENTIFIER=[210,*]+REMOTE, ACCESS=READ)
(IDENTIFIER=REMOTE, ACCESS=NONE)
(IDENTIFIER=[*,*], ACCESS=NONE)
```

In the following example, CASADAY uses the DCL DIRECTORY/ACL command to display the VMS access control list for the file CORP.DIC:

```
$ DIRECTORY/ACL CORP.DIC

Directory DB3:[CASADAY.CDD]

CORP.DIC;1
        (IDENTIFIER=[CASADAY], ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
        (IDENTIFIER=[210,*]+LOCAL, ACCESS=READ+WRITE)
        (IDENTIFIER=SECRETARY, ACCESS=READ+WRITE)
        (IDENTIFIER=BATCH, ACCESS=READ)
        (IDENTIFIER=[210,*]+REMOTE, ACCESS=READ)
        (IDENTIFIER=REMOTE, ACCESS=NONE)
        (IDENTIFIER=[*,*], ACCESS=NONE)
```

In the following example, CASADAY uses the keywords OPTIONS=PROTECTED to prevent deletion of the last entry if the remainder of the list is deleted. Then she can use the DCL SET FILE/ACL/DELETE command to delete all the unprotected entries. The DIRECTORY/ACL command shows that the last entry remains intact:

```
$ EDIT/ACL PERS.DIC
(IDENTIFIER=[CASADAY], ACCESS=WRITE+READ+EXECUTE+CONTROL+DELETE)
(IDENTIFIER=SECRETARY, ACCESS=READ+WRITE)
(IDENTIFIER=[*,*], OPTIONS=PROTECTED, ACCESS=NONE)
^Z
$ SET FILE/ACL/DELETE PERS.DIC
$ DIRECTORY/ACL PERS.DIC

Directory DB3:[CASADAY.CDD]
        (IDENTIFIER=[*,*], OPTIONS=PROTECTED, ACCESS=NONE)
```

When VMS receives a request for access to a file, it searches the access control list starting with the first entry and stops searching at the first entry matching the user's identifying characteristics.

You can use EDIT/ACL only on files that you own or that you can access with the VMS privilege BYPASS, SYSPRV, or GRPPRV.

EDIT/ACL creates an access control list if it does not already exist. You must specify the file type .DIC.

CTRL/Z terminates the editing session and returns you to DCL command level.

For more information about the DCL EDIT/ACL command, see the related information in the VMS documentation set.

A.6.6 Overriding Security in the DMU Dictionary

Neither VMS file protection nor the CDD security provisions apply to any process possessing the VMS BYPASS privilege.

A.7 Creating a CDDL Source File in the DMU Dictionary

The primary goal of the Data Definition Language Utility (CDDL) is to provide you with a mechanism for entering record definitions directly into the CDD. The actual procedure consists of two steps:

- Use a text editor to create a CDDL source file containing a record definition.
- Invoke the CDDL compiler to insert the record definition into the dictionary.

This section describes how to create a CDDL source file.

A.7.1 The CDDL Source File

In the source file, you specify the name and description of a record, the fields that comprise it, and the attributes of those fields. Field attributes include data type and alignment specifications, array or initial-value declarations, and a number of facility-specific attributes recognized only by a particular language, such as VAX COBOL, or language processor, such as DATATRIEVE.

The source file contains the record definitions to be placed in the dictionary. Source file names follow the rules for standard VMS file specifications. The default file type is .DDL.

The basic unit of a CDDL source file is the DEFINE statement, which names the record you are creating and includes documentary text with the DESCRIPTION IS clause. Following the DEFINE statement is a field description statement defining the record's field attributes. Subordinate field description statements can be embedded within the field description statement. Finally, the END statement completes the record definition. The general format of a source file is:

```
DEFINE RECORD path-name

    [DESCRIPTION [IS] /* text */].

     field-description-statement
    [ path-name  ]
END [ given-name ] [RECORD].
```

Example A-1 shows a simple CDDL source file named ADDRESS.DDL, which defines a dictionary object from the sample dictionary (Figure A-1). The object is a CDD record definition, and its dictionary path name is CDD$TOP.CORPORATE.ADDRESS_RECORD.

Example A-1: ADDRESS.DDL, Sample CDDL Source File

```
DEFINE RECORD CDD$TOP.CORPORATE.ADDRESS_RECORD
    DESCRIPTION IS
        /* This record contains the standard format
        for addresses.  It provides the source from which all
        address fields in other record descriptions are copied. */.
    ADDRESS STRUCTURE.
        STREET              DATATYPE IS TEXT
                            SIZE IS 30 CHARACTERS.
        CITY                DATATYPE IS TEXT
                            SIZE IS 30 CHARACTERS.
        STATE               DATATYPE IS TEXT
                            SIZE IS 2 CHARACTERS.
        ZIP_CODE STRUCTURE.
            NEW             DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 4 DIGITS
                            BLANK WHEN ZERO.
            OLD             DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 5 DIGITS.
        END ZIP_CODE STRUCTURE.
    END ADDRESS STRUCTURE.
END ADDRESS_RECORD.
```

A.7.2 Creating a Record Definition in the DMU Dictionary

A record definition begins with a DEFINE statement and ends with an END statement. A CDDL source file can contain any number of record definitions, provided each one begins with DEFINE and ends with END.

You name a record definition and specify its location in the CDD using the DEFINE statement. For example, in the source file ADDRESS.DDL:

- ADDRESS_RECORD is the given name of the record definition. You can specify the absolute version number of the definition. You need not specify a version number in the record definition. If you do not specify a version number, the CDDL compiler gives the record definition a version number when it inserts the record definition into the dictionary.

- CDD$TOP.CORPORATE is the directory into which CDDL places ADDRESS_RECORD. You can use a full path name or a relative path name. If you use a relative path name, CDDL appends the path name of the record definition to your default directory or to a path name specified with the /PATH qualifier (see Section A.8.1).

Within the DEFINE statement, you can document the nature and purpose of the definition by including a DESCRIPTION statement. The description must be enclosed in the text delimiters /* and */. CDDL stores the description in the record definition, and you can access it with the DMU LIST/ITEM=DESCRIPTION command.

End the DEFINE statement with a period.

The END statement contains the keyword END by itself or followed by the full path name of the definition, the given name, or just the keyword RECORD. If you specify a name in the END statement, it must match the name in the DEFINE statement, including any specified version number. The END statement must also terminate with a period.

A.7.3 Describing the Record Definition in the DMU Dictionary

You describe the contents of the record definition with field description statements. There are four types of field description statements.

A.7.3.1 Field Description Statements – A field description statement describing the contents of the record definition follows the DEFINE statement. You can choose from four types of field description statements:

- Elementary field description statements define fields that are not divided into subordinate fields. The fields CITY, STATE, NEW, and OLD in Example A-1 are examples of elementary fields.
- STRUCTURE field description statements define fields that are divided into one or more subordinate fields. The top-level field description statement for a record is ordinarily a STRUCTURE field description statement. For example, the ADDRESS field in Example A-1 is a STRUCTURE field divided into three elementary fields (STREET, CITY, and STATE) and one STRUCTURE field, ZIP_CODE. As this example demonstrates, you can nest STRUCTURE fields within other STRUCTURE fields.
- COPY field description statements insert the field descriptions of existing records into the descriptions of new records. As the DESCRIPTION IS clause of Example A-1 explains, CORPORATE.ADDRESS_RECORD is the source from which the address fields of all other records are copied. You could, therefore,

define an ADDRESS field in another record with the COPY field description statement:

```
ADDRESS    COPY FROM CDD$TOP.CORPORATE.ADDRESS_RECORD.
```

- VARIANTS field description statements define a set of two or more fields that provide alternative descriptions for the same portion of a record. The function of the VARIANTS field description is similar to that of the REDEFINES clause in VAX COBOL and DATATRIEVE. With VARIANT fields, you have the option of specifying a tag variable whose value is used at run time to determine the current VARIANT.

The following example shows a STRUCTURE field description statement with VARIANT fields and the tag variable option.

```
STOCK STRUCTURE.
    /* RECORD_IDENTIFIER determines field type:
         S --> In-stock record.
         B --> Back order record.
         O --> Out-of-stock record. */
    RECORD_IDENTIFIER         DATATYPE IS TEXT
                              SIZE IS 1 CHARACTER.
    VARIANTS OF RECORD_IDENTIFIER.
        VARIANT VALUE IS "S".
            IN_STOCK STRUCTURE.
                PRODUCT_NO    DATATYPE IS TEXT
                              SIZE IS 8 CHARACTERS.
                DATE_ORDERED  DATATYPE IS DATE.
                STATUS_CODE   DATATYPE IS BYTE.
                QUANTITY      DATATYPE IS LONGWORD
                              ALIGNED ON LONGWORD.
                LOCATION      ARRAY 1:4
                              DATATYPE IS TEXT
                              SIZE IS 30 CHARACTERS.
                UNIT_PRICE    DATATYPE IS LONGWORD SCALE -2.
            END IN_STOCK STRUCTURE.
        END VARIANT.
        VARIANT VALUE IS "B".
            BACK_ORDER STRUCTURE.
                PRODUCT_NO    DATATYPE IS TEXT
                              SIZE IS 8 CHARACTERS.
                DATE_ORDERED  DATATYPE IS DATE.
                STATUS_CODE   DATATYPE IS BYTE.
                QUANTITY      DATATYPE IS LONGWORD
                              ALIGNED ON LONGWORD.
                SUPPLIER      ARRAY 1:4
                              DATATYPE IS TEXT
                              SIZE IS 30 CHARACTERS.
                UNIT_PRICE    DATATYPE IS LONGWORD
                              SCALE -2.
            END BACK_ORDER STRUCTURE.
        END VARIANT.
        VARIANT VALUE IS "O".
            OUT_OF_STOCK STRUCTURE.
                PRODUCT_NO          DATATYPE IS TEXT
                                    SIZE IS 8 CHARACTERS.
                DATE_LAST_SOLD      DATATYPE IS DATE.
            END OUT_OF_STOCK STRUCTURE.
        END VARIANT.
    END VARIANTS.
END STOCK STRUCTURE.
```

End each field description statement with a period.

A.7.3.2 Documenting Field Description Statements – You can use the DESCRIPTION statement to document the entire record definition. There are also two ways to document individual fields:

- You can enclose a comment in the text delimiters /* and */. Place the comment immediately before the field description it documents. In the following example, the comment documents the STATE elementary field description.

  ```
  /*Use USPO standard abbreviations for state code.*/
  STATE               DATATYPE IS TEXT
  ```

- You can use the exclamation point (!) as a comment delimiter. CDDL ignores any text preceded by an exclamation point. Thus, you could also document the STATE field in this way:

  ```
  !Use USPO standard abbreviations for state code.
  STATE               DATATYPE IS TEXT
  ```

The preferred method of documenting field description statements is to use /* and */ as comment delimiters, because these comments are stored in the record definition. Comments delimited by an exclamation point are stored only in the source text. This difference becomes important, for example, when you use the /RECORD qualifier with the EXTRACT command (see Section A.8.3.1.2).

A.7.3.3 Field Attribute Clauses – In each field description statement, you use field attribute clauses to define the fields in a record. General field attributes provide unambiguous field descriptions recognized by every facility using the CDD. Table A-7 presents a brief description of each general field attribute clause.

Table A–7: General Field Attribute Clauses

Field Attribute Clause	Function
ALIGNED	Sets a field's starting boundary relative to the start of a record.
ARRAY, OCCURS, OCCURS...DEPENDING	Declares a field to be an array.
DATATYPE	Defines the type and size of a field. Data types are further explained in the *VAX Common Data Dictionary Data Definition Language Reference Manual.*
INITIAL_VALUE	Sets a value for a field when it is first allocated.

In addition to the general field attributes, there are facility-specific field attributes that are supported by some languages or language processors but not by others.

In the ZIP_CODE field of Example A-1, for example, the field NEW has the facility-specific field attribute BLANK WHEN ZERO. Only VAX COBOL recognizes this field attribute. Other language processors, such as DATATRIEVE, ignore the BLANK WHEN ZERO clause.

Table A-8 presents a brief description of each facility-specific field attribute clause.

Table A–8: Facility-Specific Field Attribute Clauses in the DMU Dictionary

Field Attribute Clause	Function
BLANK WHEN ZERO	Causes VAX COBOL to set an entire field to blanks when you assign that field a value of zero.
COMPUTED BY DATATRIEVE	Supplies expressions used to calculate the values of virtual fields.
CONDITION FOR COBOL	Associates condition names with specific values in a field.
DEFAULT_VALUE FOR DATATRIEVE	Sets a default value for a field used by DATATRIEVE.
EDIT_STRING FOR DATATRIEVE	Declares the format used to display the value of a field.
INDEXED FOR COBOL BY	Specifies index names for the INDEXED BY clause in VAX COBOL.
JUSTIFIED RIGHT	Causes the same nonstandard truncation as the VAX COBOL JUSTIFIED clause. Only VAX COBOL recognizes the JUSTIFIED RIGHT clause.
MISSING_VALUE FOR DATATRIEVE	Specifies contents for a field that has never been assigned a meaningful value.
NAME	Declares a VAX COBOL-specific, VAX BASIC-specific, or VAX PL/I-specific name for a field.
PICTURE	Declares a VAX COBOL-specific, DATATRIEVE-specific, or VAX PL/I-specific picture string for a field.
QUERY_HEADER FOR DATATRIEVE	Specifies a field label for report headers.
QUERY_NAME FOR DATATRIEVE	Specifies an alternate field name.
VALID FOR DATATRIEVE IF	Declares a DATATRIEVE validation expression.

A.7.3.4 CDDL Data Types —With CDDL, you can store record definitions using most VAX data types. Legal CDDL data types include fixed point, floating point, character string, and decimal string classes.

A.7.3.4.1 Character String Data Types —The character string data type represents text in strings of contiguous 8-bit bytes. The first character is stored in the first byte, the second character in the second byte, and so on, up to a maximum of 65,535 bytes.

A.7.3.4.2 Fixed Point Data Types —Fixed point data types represent scaled quantities in a binary format, and they can be SIGNED or UNSIGNED.

SIGNED fixed point numbers are stored in two's complement form. Values range from -2**(n-1) to 2**(n-1)-1, where n is equal to the number of bits in the data type. UNSIGNED fixed point numbers range from 0 to (2**n)-1, where n is equal to the number of bits in the data type.

The fixed point data types include the 8-bit BYTE, the 16-bit WORD, the 32-bit LONGWORD, the 64-bit QUADWORD, and the 128-bit OCTAWORD. Table A-9 shows the fixed point data types.

Table A–9: The Fixed-Point Data Types in the DMU Dictionary

Data Type	Length	Unsigned	Signed
BYTE	8 bits	0 to 255	−128 to 127
WORD	16 bits	0 to 65535	−32768 to 32767
LONGWORD	32 bits	0 to 4,294,967,295	−2,147,483,648 to 2,147,483,647
QUADWORD	64 bits	0 to $2^{64} - 1$	-2^{63} to $2^{63} - 1$
OCTAWORD	128 bits	0 to $2^{128} - 1$	-2^{127} to $2^{127} - 1$

A.7.3.4.3 Floating Point Data Types —Floating point data types represent approximations to quantities in a scientific notation consisting of a signed exponent and a fraction. The four floating point data types are:

- F_FLOATING (32 bits)
- D_FLOATING (64 bits)
- G_FLOATING (64 bits)
- H_FLOATING (128 bits)

In addition, you can use F_FLOATING COMPLEX, D_FLOATING COMPLEX, G_FLOATING COMPLEX, and H_FLOATING COMPLEX to specify ordered pairs of floating point data types representing the real and imaginary components of complex numbers. Table A-10 shows the floating point data types.

Table A–10: The Floating Point Data Types in the DMU Dictionary

Data Type	Length	Approximate Precision	Approximate Range
F_FLOATING	32 bits	7 decimal digits	$\pm 10^{-38}$ to 10^{38}
D_FLOATING	64 bits	16 decimal digits	$\pm 10^{-38}$ to 10^{38}
G_FLOATING	64 bits	15 decimal digits	$\pm 10^{-308}$ to 10^{308}
H_FLOATING	128 bits	33 decimal digits	$\pm 10^{-4932}$ to 10^{4932}

A.7.3.4.4 Decimal String Data Types —Decimal string data types represent fixed scaled quantities, and so they are efficient in applications that generate numerous reports and listings.

There are two classes of decimal string data types. Those in which each decimal digit occupies one 8-bit byte are called NUMERIC data types. In the more compact form, called PACKED DECIMAL, two decimal digits occupy each byte. Table A-11 shows the decimal string data types.

Table A–11: The Decimal String Data Types in the DMU Dictionary

Datatype	Description
UNSIGNED NUMERIC	An unsigned numeric ASCII string.
LEFT SEPARATE NUMERIC	A signed numeric ASCII string; the leftmost byte contains the sign.
LEFT OVERPUNCHED NUMERIC	A signed numeric ASCII string; the sign and the leftmost digit occupy the same byte.
RIGHT SEPARATE NUMERIC	A signed numeric ASCII string; the rightmost byte contains the sign.
RIGHT OVERPUNCHED NUMERIC	A signed numeric ASCII string; the sign and the rightmost digit occupy the same byte.
ZONED NUMERIC	The VAX ZONED NUMERIC data type; this signed numeric ASCII string is similar to the RIGHT OVERPUNCHED NUMERIC, but the sign codes differ.
PACKED DECIMAL	A signed numeric ASCII string; two digits occupy each byte, and the low half of the last byte is reserved for the sign.

A.7.3.4.5 Other Data Types — In addition to the data types described in the preceding sections, the CDDL can assign the following data types to fields within record definitions:

- BIT specifies a string of contiguous bits.
- DATE specifies a VMS standard 64-bit absolute date data type.
- UNSPECIFIED sets aside a specified number of contiguous unsigned 8-bit bytes.
- VARYING STRING specifies a PL/I varying string class field.
- POINTER specifies a field containing the address of another field or buffer.
- VIRTUAL FIELD specifies a DATATRIEVE virtual field data type. DATATRIEVE calculates the field's value at run time. No space is allocated for it in the record.

A.7.3.4.6 Language Support for CDDL Data Types —The languages that currently support the CDD include:

- VAX BASIC
- VAX C Version 2.0 and later
- VAX COBOL
- VAX DIBOL, Version 2.0 and later
- VAX FORTRAN, Version 4.0 and later
- VAX PASCAL, Version 3.0 and later
- VAX PL/I, Version 2.0 and later
- VAX RPG II, Version 2.0 and later

The VAX Information Architecture products that currently support the CDD include:

- ACMS
- DATATRIEVE
- DBMS
- Rdb/VMS
- TDMS

If you plan to share data definitions between two or more of these languages and products, you must be careful to use data types supported by each of them. Table A-12 and Table A-13 provide a cross reference of the CDDL data types and the languages that support them. Table A-14 provides a cross reference of CDDL data types and VAX Information Architecture products. In both tables:

S	Indicates that the facility fully supports the data type.
W	Indicates that the facility translates the data type into one that is supported and issues diagnostics.
X	Indicates a data type that DATATRIEVE can use but cannot define.
U	Indicates that the data type is unsupported and that the facility issues a fatal diagnostic.
E	Indicates that the data type is unsupported and that the facility issues a non-fatal error.
R	Indicates that the facility can make use of an unsupported data type only to pass its address as a parameter.

Table A–12: BASIC, COBOL, DIBOL, and PL/I Support for CDDL Data Types in the DMU Dictionary

Data Type	Language			
	COBOL	BASIC	DIBOL	PL/I
UNSPECIFIED	U	S	W	R
SIGNED BYTE	U	S	W	R
UNSIGNED BYTE	U	W	W	R
SIGNED WORD	S	S	W	S
UNSIGNED WORD	W	W	W	R
SIGNED LONGWORD	S	S	W	S
UNSIGNED LONGWORD	W	W	W	R
SIGNED QUADWORD	S	W	W	R
UNSIGNED QUADWORD	W	W	W	R
SIGNED OCTAWORD	U	W	W	R
UNSIGNED OCTAWORD	U	W	W	R
F_FLOATING	S	S	W	S
F_FLOATING COMPLEX	U	W	W	R
D_FLOATING	S	S	W	S
D_FLOATING COMPLEX	U	W	W	R
G_FLOATING	U	S	W	W
G_FLOATING COMPLEX	U	W	W	R
H_FLOATING	U	S	W	S
H_FLOATING COMPLEX	U	W	W	R
UNSIGNED NUMERIC	S	W	W	S
LEFT OVERPUNCHED NUMERIC	S	W	W	S
LEFT SEPARATE NUMERIC	S	W	W	S
RIGHT OVERPUNCHED NUMERIC	S	W	W	S

(Continued on next page)

Table A–12: BASIC, COBOL, DIBOL, and PL/I Support for CDDL Data Types in the DMU Dictionary (Cont.)

Data Type	Language			
	COBOL	BASIC	DIBOL	PL/I
RIGHT SEPARATE NUMERIC	S	W	W	S
PACKED DECIMAL	S	S	W	S
ZONED NUMERIC	U	W	S	R
BIT	U	W	W	S
DATE	W	W	W	R
TEXT	S	S	S	S
VARYING STRING	U	W	W	S
POINTER	U	U	W	S
VIRTUAL FIELD	W	W	W	E

Table A–13: FORTRAN, C, PASCAL, and RPG II Support for CDDL Data Types in the DMU Dictionary

Data Type	Language			
	C	FORTRAN	PASCAL	RPG II
UNSPECIFIED	W	R	S	W
SIGNED BYTE	S	S	S	W
UNSIGNED BYTE	S	R	S	W
SIGNED WORD	S	S	S	S
UNSIGNED WORD	S	R	S	W
SIGNED LONGWORD	S	S	S	S
UNSIGNED LONGWORD	S	R	S	W
SIGNED QUADWORD	R	R	R	W

Table A–13: FORTRAN, C, PASCAL, and RPG II Support for CDDL Data Types in the DMU Dictionary (Cont.)

Data Type	Language			
	C	FORTRAN	PASCAL	RPG II
UNSIGNED QUADWORD	R	R	R	W
SIGNED OCTAWORD	R	R	R	W
UNSIGNED OCTAWORD	R	R	R	W
F_FLOATING	S	S	S	W
F_FLOATING COMPLEX	R	S	R	W
D_FLOATING	S	S	S	W
D_FLOATING COMPLEX	R	S	R	W
G_FLOATING	S	S	S	W
G_FLOATING COMPLEX	R	S	R	W
H_FLOATING	R	S	R	W
H_FLOATING COMPLEX	R	R	R	W
UNSIGNED NUMERIC	R	R	R	W
LEFT OVERPUNCHED NUMERIC	R	R	R	W
LEFT SEPARATE NUMERIC	R	R	R	W
RIGHT OVERPUNCHED NUMERIC	R	R	R	S
RIGHT SEPARATE NUMERIC	R	R	R	W
PACKED DECIMAL	R	R	R	S
ZONED NUMERIC	R	R	R	W
BIT	S[1]	R	S	W
DATE	W	R	R	W
TEXT	S	S	S	S
VARYING STRING	W	R	S	W

[1]The bit field cannot exceed 32 bits and must be part of a structure field. VAX C does not support a separate bit data type.

(Continued on next page)

Table A–13: FORTRAN, C, PASCAL, and RPG II Support for CDDL Data Types in the DMU Dictionary (Cont.)

Data Type	Language			
	C	FORTRAN	PASCAL	RPG II
POINTER	S	R	S	W
VIRTUAL FIELD	E	E	E	W
TEXT	S	S	S	S
VARYING STRING	W	R	S	W
POINTER	S	R	S	W
VIRTUAL FIELD	E	E	E	W

Table A–14: VAX Information Architecture Products Support for CDDL Data Types in the DMU Dictionary

Data Type	DTR	DBMS	ACMS	Rdb /VMS	TDMS
UNSPECIFIED	S	S	S	U	S
SIGNED BYTE	S	S	S	U	S
UNSIGNED BYTE	X	S	S	U	S
SIGNED WORD	S	S	S	S	S
UNSIGNED WORD	X	S	S	U	S
SIGNED LONGWORD	S	S	S	S	S
UNSIGNED LONGWORD	X	S	S	U	S
SIGNED QUADWORD	S	S	S	S	S
UNSIGNED QUADWORD	X	S	S	U	S
SIGNED OCTAWORD	U	S	S	U	S
UNSIGNED OCTAWORD	U	S	S	U	S
F_FLOATING	S	S	S	S	S

Table A–14: VAX Information Architecture Products Support for CDDL Data Types in the DMU Dictionary (Cont.)

Data Type	DTR	DBMS	ACMS	Rdb /VMS	TDMS
F_FLOATING COMPLEX	U	S	S	U	U
D_FLOATING	S	S	S	U	S
D_FLOATING COMPLEX	U	S	S	U	U
G_FLOATING	U	S	S	S	S
G_FLOATING COMPLEX	U	S	S	U	U
H_FLOATING	U	S	S	U	S
H_FLOATING COMPLEX	U	S	S	U	U
UNSIGNED NUMERIC	S	S	S	U	S
LEFT OVERPUNCHED NUMERIC	S	S	S	U	S
LEFT SEPARATE NUMERIC	S	S	S	U	S
RIGHT OVERPUNCHED NUMERIC	S	S	S	U	S
RIGHT SEPARATE NUMERIC	S	S	S	U	S
PACKED DECIMAL	S	S	S	U	S
ZONED NUMERIC	S	S	S	U	S
BIT	U	U	S	U	U
DATE	S	S	S	S	S
TEXT	S	S	S	S	S
VARYING STRING	X	U	U	S	S
POINTER	U	U	U	U	U
VIRTUAL FIELD	S	U	U	U	E
SEGMENTED STRING	S	U	U	S	U

A.8 Compiling, Modifying, and Using CDDL Record Definitions in the DMU Dictionary

The Data Definition Language Utility (CDDL) is the CDD utility that compiles source files and inserts them into the CDD.

CDDL allows you to:

- Create a new record definition
- Replace an existing record definition with a modified record definition
- Modify an existing record definition and insert it into the CDD without replacing the old version
- Recompile definitions that use a modified record definition in a COPY field description statement

Once a definition is inserted into the CDD, VAX Information Architecture products and programs written in many VAX programming languages can copy the definition.

A.8.1 Compiling a New Record Definition in the DMU Dictionary

You use the CDDL command to compile the contents of a CDDL source file and place the record definitions it contains into the CDD. ADDRESS.DDL, for example, contains the source file for the CDD$TOP.CORPORATE.ADDRESS_RECORD record definition in the sample dictionary (Figure A-1). The following command compiles ADDRESS.DDL and inserts it into the dictionary:

```
$ CDDL/AUDIT ADDRESS.DDL
$
```

CDDL creates a default access control list containing one entry granting the creator the default privileges for a CDD object (see Section A.6.1.2). If you use the /NOACL qualifier, CDDL does not create an access control list.

You document the creation of a record definition by using the /AUDIT qualifier. The following history list entry was created when CASADAY compiled ADDRESS.DDL:

```
    First version created by CASADAY (UIC [30,10]) in process CASADAY
        using VAX CDD Data Definition Language Version 3.1
        on  2-JUL-1984 16:49:37.83.
```

CDDL automatically creates a listing file unless you specify /NOLISTING. ADDRESS.LIS, the listing file created when CASADAY compiled ADDRESS.DDL, contains the following information:

```
VAX CDD Data Definition Language Utility   Version 3.1
2-JUL-1984 16:05:59.97 Page   1
Command Line:  CDDL/AUDIT ADDRESS.DDL
Source File:  DB3:[CASADAY.CDD]ADDRESS.DDL;1
```

```
      DEFINE RECORD CDD$TOP.CORPORATE.ADDRESS_RECORD
        DESCRIPTION IS
          /* This record contains the standard format for
          addresses.  It provides the source from which
          all address fields in other record descriptions are
          copied.*/.
        ADDRESS STRUCTURE.
          STREET            DATATYPE IS TEXT
                            SIZE IS 30 CHARACTERS.
          CITY              DATATYPE IS TEXT
                            SIZE IS 30 CHARACTERS.
          STATE             DATATYPE IS TEXT
                            SIZE IS 2 CHARACTERS.
          ZIP_CODE          DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 5 DIGITS.
          END ADDRESS STRUCTURE.
      END ADDRESS_RECORD.
                  1
%CDDL-S-RECORDCRE, record "CDD$TOP.CORPORATE.ADDRESS_RECORD;1"
created in the CDD
```

The listing file contains:

- Creation information
- The command line used and the location of the source file
- The source text of the record definition
- A success message

ADDRESS.DDL contains a full path name in its DEFINE statement. Thus, the record definition can be placed in only one location in the CDD. If, however, you use a relative path name, CDDL determines the location of the record definition in one of two ways:

- If you specify a path name with the /PATH qualifier, the CDD appends the relative path name to the specified path name.
- If you do not specify a path name with the /PATH qualifier, the CDD appends the relative path name to your default directory.

Using a relative path name has several advantages:

- It is easy to move the definition to a new location in the CDD (or to a CDD on another system).
- You can place the definition under a directory that requires a password for access without having to include the password in the DEFINE statement. If you include the password in the DEFINE statement, anyone with SEE privilege can display the password by using the LIST/ITEM=SOURCE command. By specifying the

directory name and associated password in the /PATH qualifier, you need not include it in the source text.

For example, if the password SEMI_SECRET is associated with the directory PERSONNEL, and the password SECRET with the directory SERVICE, the full path name of the record definition SALARY_RECORD is CDD$TOP.PERSONNEL(SEMI_SECRET).SERVICE(SECRET).SALARY_RECORD. In order to keep the passwords out of the DEFINE statement of SALARY.DDL, you could use a relative path name in the source text.

```
!SALARY.DDL
DEFINE RECORD SALARY_RECORD
    DESCRIPTION IS
        /* This is the record containing salary
        information for all employees.  It is sensitive, and access
        is carefully restricted. Direct deposit information added
        5-JAN-1984.*/.
    SALARY STRUCTURE.
        EMPLOYEE_ID       DATATYPE IS UNSIGNED NUMERIC
                          SIZE IS 9 DIGITS.
        PAY STRUCTURE.
            JOB_CLASS     DATATYPE IS TEXT
                          SIZE IS 3 CHARACTERS.
            INCR_LEVEL    DATATYPE IS UNSIGNED NUMERIC
                          SIZE IS 1 DIGIT.
            WEEKLY_SALARY DATATYPE IS UNSIGNED NUMERIC
                          SIZE IS 6 DIGITS 2 FRACTIONS.
            DIRECT_DEP    DATATYPE IS TEXT
                          SIZE IS 1 CHARACTER
                          VALID FOR DTR IF "DIRECT_DEP=Y OR
                          DIRECT_DEP=N".
        END PAY STRUCTURE.
    END SALARY STRUCTURE.
END SALARY_RECORD RECORD.
```

You could then compile the record definition with this command:

```
$ CDDL/AUDIT/PATH=CDD$TOP.PERSONNEL(SEMI_SECRET).SERVICE(SECRET)-
$_SALARY.DDL
```

A.8.2 Using CDD Record Definitions in the DMU Dictionary

A number of VAX programming languages and VAX Information Architecture products can copy the definitions stored in the CDD. Section A.7.3.4.6 lists these languages and products. It also contains three tables listing CDDL data type compatibility.

Note

Not all languages accept all CDD data types, however, and there may be other minor compatibility problems. For problems other than data type compatibility, check the documentation for the language or product you are using.

A.8.2.1 Copying CDD Definitions into a Program — Each VAX programming language and VAX Information Architecture product has its own method of copying record definitions from the CDD. Consult the documentation for the language or product you are using to obtain this information. This section uses VAX BASIC as an example to show you how one language uses CDD record definitions.

FOSTER, an applications programmer, wants to copy the ADDRESS_RECORD definition into a BASIC program called MAILLIST.BAS. He includes the following declaration in MAILLIST.BAS:

```
%INCLUDE %FROM %CDD  'CDD$TOP.CORPORATE.ADDRESS_RECORD'
```

This line tells the BASIC compiler to copy the CDD definition CDD$TOP.CORPORATE.ADDRESS_RECORD into MAILLIST. FOSTER does not include a version number in the declaration because he wants to copy the highest existing version of the record definition. He must specify a full path name for ADDRESS_RECORD.

FOSTER compiles MAILLIST.BAS with the following command:

```
$ BASIC/LISTING/AUDIT  MAILLIST
```

The VAX BASIC compiler produces an object file, MAILLIST.OBJ, and a listing file, MAILLIST.LIS. MAILLIST.LIS contains the following record declaration, which shows how the BASIC compiler converts ADDRESS_RECORD into BASIC form.

```
     5  %INCLUDE %FROM %CDD 'CDD$TOP.CORPORATE.ADDRESS_RECORD'
C1   5    !   This record contains the standard format
C1   5    !   for addresses.  It provides the source from which all
C1   5    !   address fields in other record descriptions are copied.
C1   5    RECORD  ADDRESS                    ! UNSPECIFIED
C1   5      STRING  STREET  = 30             ! TEXT
C1   5      STRING  CITY  = 30               ! TEXT
C1   5      STRING  STATE  = 2               ! TEXT
C1   5      GROUP   ZIP_CODE                 ! UNSPECIFIED
C1   5        GROUP   NEW                    ! UNSIGNED NUMERIC
C1   5          STRING  STRING_VALUE  = 4
C1   5        END GROUP
C1   5        GROUP   OLD                    ! UNSIGNED NUMERIC
C1   5          STRING  STRING_VALUE  = 5
C1   5        END GROUP
C1   5      END GROUP
C1   5    END RECORD
.................................1

%BASIC-I-CDDSUBGRO, 1:        data type in CDD not supported,
                substituted group for: ADDRESS::ZIP_CODE::NEW.
%BASIC-I-CDDSUBGRO, 1:        data type in CDD not supported,
                substituted group for: ADDRESS::ZIP_CODE::OLD.
```

The "C1" designation at the beginning of each line is the way BASIC indicates that the definition has been copied directly from the CDD. Each line also contains a comment indicating the CDDL data type that produced the BASIC data type on the line. Note that the BASIC compiler also copied the DESCRIPTION clause from the CDD definition and included it in the listing.

As the two informational messages indicate, BASIC converted the two fields containing UNSIGNED NUMERIC data types, which BASIC does not support, to GROUP fields.

Once FOSTER links MAILLIST.OBJ, it runs like any other BASIC program.

A.8.2.2 Documenting the Use of Record Definitions – The /AUDIT qualifier allows you to insert an entry into the history list of any record definition. Most VAX programming languages and VAX Information Architecture products accept the /AUDIT qualifier.

When FOSTER compiles MAILLIST.BAS, for example, he uses the /AUDIT qualifier. In the history list of CDD$TOP.CORPORATE.ADDRESS_RECORD, BASIC produces the following entry:

```
Compiled by FOSTER (UIC [210,1]) in process FOSTER
   using VAX-11 BASIC V2.0 on 10-JAN-1984 11:02:55.98
   for program MAILLIST.
```

When you compile programs using CDD record definitions, however, you may be reluctant to use the /AUDIT qualifier until you are certain that the program runs as you expect.

If you compile the same program several times for debugging purposes, you can make several history list entries for the same program.

In order to avoid creating multiple history list entries for the same program, you can use the DMU MEMO/AUDIT command to insert a history list entry into the CDD. Section A.5.4 contains an example of the use of the MEMO/AUDIT command.

A.8.3 Modifying CDDL Source Files in the DMU Dictionary

Often, business decisions require that record definitions stored in the CDD be modified. To modify a CDD record definition, you:

- Create a modified CDDL source file.
- Compile the modified source file with the /REPLACE or /VERSION qualifier.
- Recompile any programs or applications that use the definition so that they reflect the change. Also recompile any other CDD record definitions that use that definition as a template record.

A.8.3.1 Obtaining the Source File – You can edit the original source file to obtain a source file containing the modifications you wish to make. If, however, the original source file is unavailable, you can:

- Create an entirely new source file. This represents a great deal of work when you want to modify only a part of the definition.
- Use the DMU EXTRACT command. DMU EXTRACT copies or generates CDDL source text from the CDD and puts it in a file that you can then edit.

A.8.3.1.1 Extracting CDDL Source Text – If the CDDL compiler creates a record definition, it stores the source text of that definition in the CDD. By default, EXTRACT copies that source text and places it in a VMS file. The following command extracts the source text stored with the object ADDRESS_RECORD;1 and places it in the file ADDRESS.LIS:

```
DMU> EXTRACT CDD$TOP.CORPORATE.ADDRESS_RECORD;1 ADDRESS.LIS
```

You can edit ADDRESS.LIS just as you would edit the original source file.

A.8.3.1.2 Extracting Record Definitions Without CDDL Source Text — Certain VAX Information Architecture products (DATATRIEVE, for example) create record definitions that do not contain CDDL source text. You can extract such record definitions from the CDD and generate CDDL source text for them if you use the /RECORD qualifier with DMU EXTRACT. EXTRACT/RECORD creates source text from record definitions stored in the CDD. For example, if you extract source text from the DATATRIEVE definition SKILL_REC;1 without using the /RECORD qualifier, the listing file contains DATATRIEVE source text:

```
DMU> EXTRACT SKILL_REC;1 SKILL.LIS
DMU> EXIT
$ TYPE SKILL.LIS
RECORD SKILL_REC USING
01 SKILL_REC.
   03 TASK_DESC  PIC X(25)
                 QUERY_HEADER "TASK"/"DESCRIPTION".
   03 JOB_CODE   PIC X(3)
                 QUERY_HEADER "TASK"/"JOB"/"CODE".
:
$
```

With the /RECORD qualifier, DMU creates CDDL source text from the information stored in the record definition:

```
DMU> EXTRACT/RECORD SKILL_REC;1 SKILL.LIS
DMU> EXIT
$ TYPE SKILL.LIS
DEFINE RECORD SKILL_REC.
  SKILL_REC STRUCTURE.
     TASK_DESC   DATATYPE IS TEXT
                 SIZE IS 25 CHARACTERS
                 PICTURE FOR DATATRIEVE IS "X(25)"
                 QUERY_HEADER FOR DATATRIEVE IS "TASK"
                                                "DESCRIPTION".
     JOB_CODE    DATATYPE IS TEXT
                 SIZE IS 3 CHARACTERS
                 PICTURE FOR DATATRIEVE IS "X(3)"
                 QUERY_HEADER FOR DATATRIEVE IS "TASK"
                                                "JOB"
                                                "CODE".
  END SKILL_REC STRUCTURE.
END SKILL_REC RECORD.
```

SKILL.LIS contains a CDDL description of the record definition created by DATATRIEVE. You can edit SKILL.LIS and recompile it using the CDDL compiler.

The /RECORD qualifier can be used only with record definitions, that is, with dictionary objects of the type CDD$RECORD.

A.8.3.2 Replacing a CDDL Definition – After you have modified a source file, you can compile it again and replace the original record definition. If you modify ADDRESS.DDL, for example, you can compile it again with the following command:

```
$ CDDL/REPLACE/AUDIT ADDRESS.DDL
```

The CDDL compiler then:

- Deletes the original version of ADDRESS_RECORD;1
- Compiles the revised ADDRESS.DDL as ADDRESS_RECORD;1
- Copies the access control list and history list of the original version to the new version, adding a history list entry describing the operation

It is important to use the /AUDIT qualifier in this case, so that the change in the definition is recorded in the history list. The history list entry alerts other users that the ADDRESS_RECORD;1 definition has been modified.

A.8.3.3 Creating an Additional Version of a Record Definition – You may want to save the original version of a record definition until you are sure that you are satisfied with the modified version. The /VERSION qualifier allows you to add a new version of the record definition to the CDD without deleting the old version:

```
$ CDDL/VERSION/AUDIT ADDRESS.DDL
```

The CDDL compiler then:

- Creates ADDRESS_RECORD;2.
- Copies the access control list and history list of ADDRESS_RECORD;1 to ADDRESS_RECORD;2. The compiler adds a history list entry documenting the operation to ADDRESS_RECORD;2's history list.

If you use the /NOACL qualifier, you can inhibit the creation of an access control list at ADDRESS_RECORD;2.

The higher version number indicates that someone has modified the record definition. Nevertheless, you should use the /AUDIT qualifier to document the change. The history list entry tells other users when the record definition was modified and who was responsible for the change. You can also add text explaining the nature and purpose of the change.

A.8.3.4 Recompiling Programs and Product Definitions – Programs written in VAX programming languages and most VAX Information Architecture products copy CDD record definitions at compile time. Therefore, after you modify a record definition, the programs and applications using the old definition become inconsistent with the definition in the dictionary. You should recompile the programs and product definitions that use that definition.

If you keep a complete and accurate history list, you can use the entries in it to determine which programs need to be recompiled. The following history list contains entries describing the use of a modified CDD$TOP.CORPORATE.ADDRESS_RECORD;1:

```
Record replaced by WOOLSON (UIC [30,25]) in process WOOLSON
    using VAX CDD Data Definition Language Version 3.1
    on  3-JUL-1984 10:32:52.73.
Compiled by FOSTER (UIC [210,1]) in process FOSTER
    using VAX-11 BASIC Version 2.0 on 10-JAN-1984 11:02:55.98
    for program MAILLIST.
Compiling by WOOLSON (UIC [30,25]) in process WOOLSON
    using VAX-11 COBOL V2.0-30 on 19-JUN-1983 10:36:37.17
    for program COB$DEVICE:[WOOLSON.CDD]CDDADDR.COB;1.
Explanation:
    To print address labels
Used as a generic record by FOSTER (UIC [210,1]) in process FOSTER
    using VAX-11 CDD Data Definition Language Version 2.3
    on 10-JUN-1983 12:31:56.23.
Explanation:
    CDD$TOP.CORPORATE.EMPLOYEE_LIST;1
Backed up by WOOLSON (UIC [30,25]) in process WOOLSON
    using VAX-11 CDD Dictionary Management Utility Version 2.3
    on 21-MAR-1983 21:14:24.53.
```

After WOOLSON modified the record definition, he could use the preceding history list to determine that he needed to recompile programs MAILLIST.BAS and CDDADDR.COB.

A.8.3.5 Recompiling CDD Record Definitions – Some record definitions contain COPY field description statements that copy other record definitions, called **template records**, at compile time (see Section A.7.3.1). CDD$TOP.CORPORATE.EMPLOYEE_LIST;1 is an example of such a record. A listing of its source file, EMPLOYEE.DDL, shows that the field ADDRESS copies the record definition CDD$TOP.CORPORATE.ADDRESS_RECORD:

```
DEFINE RECORD CDD$TOP.CORPORATE.EMPLOYEE_LIST
    DESCRIPTION IS
        /* This record contains the master list of all
        employees */.
    EMPLOYEE STRUCTURE.
        /* An employee's ID number is his
        or her social security number */
        ID                  DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 9 DIGITS.
        NAME STRUCTURE.
            LAST_NAME       DATATYPE IS TEXT
                            SIZE IS 15 CHARACTERS.
            FIRST_NAME      DATATYPE IS TEXT
                            SIZE IS 10 CHARACTERS.
            MIDDLE_INITIAL  DATATYPE IS TEXT
                            SIZE IS 1 CHARACTER.
        END NAME STRUCTURE.

        ADDRESS             COPY FROM
                            CDD$TOP.CORPORATE.ADDRESS_RECORD.
        DEPT_CODE           DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 3 DIGITS.
    END EMPLOYEE STRUCTURE.
END EMPLOYEE_LIST RECORD.
```

After WOOLSON modified CDD$TOP.CORPORATE.ADDRESS_RECORD;1, the definition stored at CDD$TOP.CORPORATE.EMPLOYEE_LIST;1 became inconsistent with the new definition, just as MAILLIST.BAS and COBADDR.COB did. You should recompile definitions containing COPY field description statements when the template record is modified.

A.8.3.5.1 Checking the Template Record's History List — The history list of CDD$TOP.CORPORATE.ADDRESS_RECORD;1 contains the following entry:

```
    Used as a generic record by FOSTER (UIC [210,1]) in process FOSTER
        using VAX-11 CDD Data Definition Language Version 2.3
        on 10-JUN-1983 12:31:56.23.
    Explanation:
        CDD$TOP.CORPORATE.EMPLOYEE_LIST;1
```

CDDL automatically makes an entry like the preceding one in the history list of a template record every time another CDD record definition copies it. When you modify a record definition, you should check its history list to determine if another record definition copies it.

A.8.3.5.2 Checking Template Records with DMU EXTRACT — CDDL automatically makes an entry in the history list of a template record. It does not, however, automatically make an entry in the history lists of the record definitions that copy the template record. If another user modifies a template record without making a history list entry, you may not know that a definition you are using is inconsistent with the current version of a template record it copies.

You can check a record definition containing a COPY field description statement to make sure that the information in the COPY field description is current. Using the /TEMPLATE qualifier with the DMU EXTRACT/RECORD command, you can compare the current definition in the template record with the definition that a record definition copied when it was compiled.

When you use the /TEMPLATE qualifier, DMU first creates source text from the definition in any template records copied by the specified definition. DMU then places the template record source text at the top of the output file. Finally, it creates source text from the specified definition, including the expanded COPY field description statement, and places that source text in the output file as well.

For example, to determine whether or not EMPLOYEE_LIST;1 contains the current version of ADDRESS_RECORD, you can type:

```
DMU> EXTRACT/RECORD/TEMPLATE CDD$TOP.CORPORATE.EMPLOYEE_LIST EMP.LIS
DMU> EXIT
```

DMU creates a file, EMP.LIS, to hold the template record and the definition that copies it:

```
$ TYPE EMP.LIS
DEFINE RECORD CDD$TOP.CORPORATE.ADDRESS_RECORD;1
DESCRIPTION IS
/* This record contains the standard format
        for addresses.  It provides the source from which all
        address fields in other record descriptions are copied. */.
    ADDRESS STRUCTURE.
        STREET      DATATYPE IS TEXT
                    SIZE IS 30 CHARACTERS.
        CITY        DATATYPE IS TEXT
                    SIZE IS 30 CHARACTERS.
        STATE       DATATYPE IS TEXT
                    SIZE IS 2 CHARACTERS.
        ZIP_CODE STRUCTURE.
            NEW         DATATYPE IS UNSIGNED NUMERIC
                        SIZE IS 4 DIGITS
                        BLANK WHEN ZERO.
            OLD         DATATYPE IS UNSIGNED NUMERIC
                        SIZE IS 5 DIGITS.
        END ZIP_CODE STRUCTURE.
    END ADDRESS STRUCTURE.
END ADDRESS_RECORD;1 RECORD.
```

```
DEFINE RECORD EMPLOYEE_LIST
DESCRIPTION IS
/* This record contains the master list of all
        employees */.
    EMPLOYEE STRUCTURE.
        /* An employee's ID number is his
                or her social security number */
        ID          DATATYPE IS UNSIGNED NUMERIC
                    SIZE IS 9 DIGITS.
        NAME STRUCTURE.
            LAST_NAME   DATATYPE IS TEXT
                        SIZE IS 15 CHARACTERS.
            FIRST_NAME  DATATYPE IS TEXT
                        SIZE IS 10 CHARACTERS.
            MIDDLE_INITIAL DATATYPE IS TEXT
                        SIZE IS 1 CHARACTER.
        END NAME STRUCTURE.
        ADDRESS     COPY FROM CDD$TOP.CORPORATE.ADDRESS_RECORD.
!       ADDRESS STRUCTURE.
!           STREET      DATATYPE IS TEXT
!                       SIZE IS 30 CHARACTERS.
!           CITY        DATATYPE IS TEXT
!                       SIZE IS 30 CHARACTERS.
!           STATE       DATATYPE IS TEXT
!                       SIZE IS 2 CHARACTERS.
!           ZIP_CODE STRUCTURE.
!               NEW         DATATYPE IS UNSIGNED NUMERIC
!                           SIZE IS 4 DIGITS
!                           BLANK WHEN ZERO.
!               OLD         DATATYPE IS UNSIGNED NUMERIC
!                           SIZE IS 5 DIGITS.
!           END ZIP_CODE STRUCTURE.
!       END ADDRESS STRUCTURE.
        DEPT_CODE   DATATYPE IS UNSIGNED NUMERIC
                    SIZE IS 3 DIGITS.
    END EMPLOYEE STRUCTURE.
END EMPLOYEE_LIST RECORD.
```

The first definition in EMP.LIS is the current version of ADDRESS_RECORD. The ADDRESS STRUCTURE field description is the definition of ADDRESS_RECORD that EMPLOYEE_LIST;1 copied when it was compiled. Note that every line of the expanded COPY field description is preceded by an exclamation point, a CDDL comment delimiter. If you used the CDDL compiler to replace or create an additional version of EMPLOYEE_LIST, the CDDL compiler would copy the definition at CDD$TOP.CORPORATE.ADDRESS_RECORD and ignore the expanded fields marked by exclamation points.

If you compare the definition of the template record and the expanded ADDRESS STRUCTURE field in EMPLOYEE_LIST, you can see that they contain the same description. In this case, you know that EMPLOYEE_LIST;1 is still current. If the definitions were different, you could recompile EMPLOYEE_LIST;1 with CDDL /RECOMPILE to incorporate the changes in ADDRESS_RECORD.

To use /TEMPLATE, you must also use /RECORD. Therefore, /TEMPLATE can be used only with dictionary objects of the type CDD$RECORD.

A.8.3.5.3 Using CDDL/RECOMPILE —The CDDL/RECOMPILE command allows you to update CDD definitions when a template record is modified. You need not create a new source file because the source text does not change. Instead of specifying a source file name, specify the path name of the record definition you want to update. For example, you could recompile EMPLOYEE_LIST;1 by typing:

```
$ CDDL/RECOMPILE/AUDIT CDD$TOP.CORPORATE.EMPLOYEE_LIST;1
```

The CDDL compiler then:

- Deletes the original EMPLOYEE_LIST;1
- Recompiles the definition, copying the modified ADDRESS_RECORD
- Copies the access control list and history list from the original EMPLOYEE_LIST;1, adding a history list entry documenting the operation

By default, CDDL/RECOMPILE replaces the original definition with the recompiled one. You can save the original definition for backup purposes by using the /VERSION qualifier with CDDL/RECOMPILE. You can create an additional version of EMPLOYEE_LIST by typing:

```
$ CDDL/RECOMPILE/VERSION/AUDIT CDD$TOP.CORPORATE.EMPLOYEE_LIST
```

The CDDL compiler then:

- Recompiles the definition, copying the modified ADDRESS_RECORD. It gives the new definition a version number one higher than the highest existing version number. In this case, the new definition would be CDD$TOP.CORPORATE.EMPLOYEE_LIST;2.
- Copies the access control list and history list from the original EMPLOYEE_LIST, adding a history list entry documenting the operation. You can keep the CDDL compiler from creating an access control list by using the /NOACL qualifier.

A.9 Organizing and Maintaining Your Dictionary in the DMU Dictionary

When you install the CDD on your system, only the following structures are created:

- CDD$TOP, the root dictionary directory
- CDD$TOP.CDD$EXAMPLES, the directory that contains the sample dictionary, and its descendants

You organize the rest of the dictionary so that you can tailor it to meet the needs of your organization. To derive the full benefit from the DMU dictionary, however, you must organize it carefully. The care you take in organizing your dictionary will be reflected in:

- Performance. You retrieve definitions from a well-organized dictionary faster than from a poorly-organized one. The increased speed will be most evident when you use information management products like DATATRIEVE and DBMS.
- Security. The CDD protection scheme allows you to use the CDD hierarchical structure to tailor access to directories and objects. A well-organized dictionary can take advantage of this feature. For more information about CDD security, see Section A.6.
- Ease of use. It is much easier to find the definitions you need if the dictionary is organized carefully.
- Accuracy. A well-organized dictionary is a safeguard against inconsistency and redundancy.
- Compact dictionary files. A well-organized dictionary is likely to be smaller than a poorly-organized one. Also, if your well-organized dictionary becomes too large, you can easily create subdictionaries out of several directories and reduce the size of your main dictionary file.
- Maintainability. A well-organized dictionary makes it easy to assess the impact of modifying definitions, reorganize parts of your dictionary, or move portions of your dictionary to other systems.

Before you populate your dictionary with record definitions, take some time to plan the organization. The benefits greatly outweigh the time spent.

A.9.1 Organizing Your DMU Dictionary Directory Hierarchy

There are a number of ways to set up the directory hierarchy:

- By organizational entity
- By application
- By individual user
- By a combination of criteria

A.9.1.1 By Organizational Entity —When you install the CDD on your system, you have one logical dictionary for your entire organization. You can, however, divide the dictionary by assigning directories separately to departments within the organization. Each department maintains its share of the CDD independently of the others.

Such a structure corresponds roughly to the first directory level below CDD$TOP in the sample dictionary (Figure A-1) and is most useful in situations where differences in data and security needs are sharply defined between divisions of the organization.

A.9.1.2 By Application —In an enterprise where divisions need shared access to most of the data descriptions, you can create a dictionary hierarchy organized according to application areas. You can set aside one directory in which to store those data definitions shared throughout the organization. You can then create application-specific directories for definitions less widely shared. In the sample dictionary (Figure A-1), the organization within the PERSONNEL subdictionary is an example of structuring a dictionary by application.

A.9.1.3 By Individual User —Some organizations have only a limited need to share many record definitions because individuals or small teams work separately on independent projects. You can reflect this structure in your directory hierarchy by assigning directories to individuals or project managers for their own use. User JONES, for example, has her own directory in the sample dictionary (Figure A-1), and in it she has stored a private record definition, LEADS_RECORD;1.

A.9.1.4 By a Combination of Criteria —The structural schemes described above represent different but not mutually exclusive approaches. For many organizations, a combination of approaches might offer the best solution. The overall structure of the sample dictionary (Figure A-1) represents such a combination.

Remember that the dictionary structure is only as useful as it is consistent with your organization's needs. In practical terms, you can organize your dictionary to mirror the current organization of your records and files or the needs of new applications. Whatever the case, *you must be aware of your data needs before you set up the directory hierarchy.*

The creation and maintenance of the CDD should be the responsibility of a system manager or data administrator who can work with top management to determine the organization's information requirements, and who understands the organization's data processing facility. If the CDD directory hierarchy or the security mechanisms are poorly planned, it will be difficult to protect the CDD against redundancy and inconsistency.

A.9.1.5 Using Subdictionaries —When you first install the VAX Common Data Dictionary, you create one dictionary file named CDD.DIC in which the directory hierarchy is physically stored. The CDD allows you, however, to store portions of this single logical hierarchy in separate physical files called subdictionary files. The directories that point to separate subdictionary files are called subdictionary directories, or **subdictionaries.**

Except for their physical location, subdictionary directories are exactly like dictionary directories. Subdictionaries are part of the same logical hierarchy and perform the same functions as dictionary directories. For most CDD users, the difference between dictionary directories and subdictionaries is invisible. You can use subdictionaries to:

- Provide extra security. Because the information contained in a subdictionary is stored in a separate file, you can augment dictionary security with VMS file protection (see Section A.6.5). You can even store the subdictionary file on another device and take it off line when it is not in use. This ability is most useful when:
  - You have two or more separate organizations using the same dictionary, as in a time-sharing system. Each organization can have its own subdictionary on its own device. Thus, one organization does not have any access to the other's definitions, even though they are using the same dictionary.
  - A particular department has sensitive material in its portion of the dictionary. In the sample dictionary in Figure A-1, the record definitions used by the Personnel Department are sensitive. Therefore, PERSONNEL has been created as a subdictionary directory stored in a separate file on a separate device.

- Reduce the size of your main dictionary file. If CDD.DIC, the main dictionary file, becomes too large, you can create subdictionaries to move some directories to another device.
- Facilitate transporting portions of the dictionary to another part of the same dictionary or to a dictionary on another system. You can change the location of a subdictionary within the CDD by deleting the existing subdictionary directory and creating another one elsewhere that points to the same file. You need not alter the file in any way. You can move a subdictionary to another system by copying the subdictionary file to the other system and creating a subdictionary directory on that system that points to the file. Again, you need not alter the file.
- Allow system managers to bill users for the amount of dictionary space their data descriptions use.

There are some limitations to consider when deciding whether or not to create a subdictionary:

- Subdictionaries require more time for I/O operations than dictionary directories. Thus, they can increase execution time if you use definitions from several subdictionaries at the same time.
- Subdictionaries are charged against FILLM, your process' VMS open file limit, and also against the SYSGEN CHANNELCNT quota.

A.9.1.5.1 Creating Subdictionaries – You create subdictionaries with the DMU CREATE/SUBDICTIONARY command. You must specify a VMS file to hold the contents of the subdictionary. The file specification must include a device and directory name for the file. You can use a system logical name for the device, directory, and file specification, but you cannot use a group or process logical name. CDD creates the file for you if it does not already exist.

The following command creates PERSONNEL as a subdictionary and creates the subdictionary file PERS.DIC:

```
DMU> CREATE/SUBDICTIONARY=DB3:[CASADAY.CDD]PERS.DIC PERSONNEL
```

If CASADAY created the system logical name CDD$DISK to stand for DB3:[CASADAY.C
she could create a subdictionary with the following command:

```
DMU> CREATE/SUBDICTIONARY=CDD$DISK:PERS.DIC PERSONNEL
```

Using a system logical name provides an easy way to move your subdictionary pointer. See Section A.9.1.5.3.

A.9.1.5.2 Deleting Subdictionaries —There are two ways to delete subdictionaries:

- If you use DELETE/SUBDICTIONARY, DMU deletes the subdictionary pointer and the directories and objects in the subdictionary file.
- If you use DELETE/NOSUBDICTIONARY (the default), DMU deletes only the subdictionary pointer.

In both cases, the file is not deleted.

A.9.1.5.3 Moving Subdictionary Files and Directories —Subdictionaries aid you in maintaining the CDD because they are easy to move. If you want to move the subdictionary file to a different device or VMS directory, you:

- Use DCL COPY or RENAME to move the subdictionary file to the new location.
- Use RENAME/SUBDICTIONARY to set the subdictionary directory to point to the new location; of course, if you used a system logical name when you created the subdictionary, you need only change the translation of the logical name.

On the other hand, you may want to leave the subdictionary file where it is and move the subdictionary directory. To do so:

- Use DELETE/ALL/NOSUBDICTIONARY to delete the subdictionary pointer.
- Use CREATE/SUBDICTIONARY to create a subdictionary directory in another location in the CDD pointing to the same subdictionary file.

It is important to delete the first subdictionary pointer. CDD allows only one subdictionary directory to point to a subdictionary file.

Note that the access control lists of the directories and objects in the subdictionary file remain intact. If the privileges they inherit in the new location are different from the ones they inherited in their old location, you may need to modify the access control lists (see Section A.6).

You can even move a subdictionary to another system:

- Use VMS COPY or BACKUP and RESTORE to copy the subdictionary file to the other system.
- Use CREATE/SUBDICTIONARY to create a subdictionary directory in the CDD of the other system pointing to the file.

A.9.2 Organizing Your DMU Dictionary to Enhance Performance

There are several ways to enhance the performance of dictionary operations. Follow these guidelines whenever they do not interfere with other organization goals, such as security.

A.9.2.1 Using the Directory Hierarchy —Dictionaries that use directories to group related objects perform better than dictionaries in which objects are stored, unorganized, near the top of the dictionary.

In general, a narrow dictionary is better than a broad dictionary; however, the benefits of a narrow dictionary are lost if related definitions are stored in widely separate portions of the hierarchy. For example, if JONES wants to access both LEADS_RECORD;1 and EMPLOYEE_LIST;1 from the sample dictionary (Figure A-1), CDD must:

1. Traverse the path from JONES' default directory to LEADS_RECORD;1
2. Travel up the hierarchy to CDD$TOP
3. Travel down a separate path, CDD$TOP.CORPORATE.EMPLOYEE_LIST;1, to access EMPLOYEE_LIST;1

Because the sample dictionary is well-organized, however, it is unlikely that JONES would want to access these two unrelated definitions. Use the directory hierarchy to group related definitions together.

A.9.2.2 Reducing DMU Dictionary Size —A small, compact dictionary performs better than a large dictionary. There are several ways to reduce the size of your dictionary.

- Delete unused dictionary directories and objects. Directories and objects used in obsolete applications needlessly clutter your dictionary and degrade performance. Periodically prune your dictionary of these unused directories and objects.
- Purge earlier versions of objects no longer needed for backup purposes. You may want to keep backup versions of dictionary objects until you are satisfied that the most recent version fulfills your needs. When you no longer need those earlier versions, use the DMU PURGE command to remove them from the dictionary. For more information about purging versions of dictionary objects, see Section A.5.6.2.
- Use CDDV COMPRESS. COMPRESS reduces the size of a dictionary file by eliminating free space created by dictionary deletions and by reorganizing dictionary pages. Periodically compress the main dictionary file and subdictionary files. For information about compressing dictionary files, see Section A.9.4.3.

A.9.2.3 Limiting the Creation of Subdictionaries – Subdictionaries enhance the security of portions of the CDD; however, they require more time for I/O operations than dictionary directories. Create only the number of subdictionaries necessary for security. Try to group related objects in the same subdictionary. Traversing from one subdictionary to another takes more time than traversing from one directory to another. For this reason, you should limit those users who can create subdictionaries to the data administrator and the system manager.

A.9.2.4 Preventing the Creation of History Lists – If dictionary performance is the overriding consideration in the organization of your system, you may want to deny users the privilege of creating history lists. Dictionaries without history lists are more compact than well-documented dictionaries. Of course, you can use the DMU DELETE/HISTORY command qualifiers to remove some of the entries in the history list of a directory or object. For more information about DELETE/HISTORY, see the *VAX Common Data Dictionary Utilities Reference Manual.*

Note

History lists are very valuable in maintaining the dictionary and assessing the impact of changing data definitions. Not using them makes it more difficult to avoid data definition redundancy and inconsistency.

A.9.3 Creating a Sample Hierarchy from a DMU Command Procedure in a DMU Dictionary

Execution of the following DMU command procedure, named CREATE.COM, creates the sample dictionary hierarchy (Figure A-1). Although you could enter these utility commands interactively, placing them in a DMU command procedure has two advantages:

- You can use a text editor to correct typing and syntax errors.
- You can use the DMU SET ABORT command to halt the procedure if CDD or DMU signals a non-fatal error.

```
! CREATE.COM
!
!
! This command procedure creates a sample VAX CDD dictionary
! hierarchy, including history lists and access control lists.
! This sample dictionary is the source of all of the examples
! in the VAX CDD documentation set.
!
!
! Use the SET ABORT command to halt the command procedure in case of
! a non-fatal error.
!
!
SET ABORT
!
!
! Use the SET PROTECTION command to establish access privileges
! at CDD$TOP.  The system manager retains full access privileges
! at CDD$TOP.
!
! Restrict the privileges other users will inherit.  Grant
! PASS_THRU, and banish FORWARD and GLOBAL_DELETE.
!
! A hyphen allows a long command to be continued on the next
! line.  Be certain that the hyphen is the last character on the line.
!
CREATE CDD$TOP
SET PROTECTION/POSITION=2/UIC=[*,*]/GRANT=<P>/DENY=<CDEHMRSUWX>-
/BANISH=<FG> CDD$TOP
!
!
! Next, use the CREATE command to create
! directories with the default access control list.  Use the /AUDIT
! qualifier to create a history list entry auditing the creation.
! Finally, use the SET PROTECTION command to modify the access
! privileges of other users on the system.
!
! Use full dictionary path names to eliminate ambiguity.
!
!
CREATE/AUDIT CDD$TOP.PRODUCTION
SET PROTECTION/POSITION=2/UIC=[*,*]/GRANT=<HS> CDD$TOP.PRODUCTION
!
!
! The /AUDIT qualifier also allows you the option of adding
! text enclosed in quotation marks to describe the directory
! or its contents.
!
!
CREATE/AUDIT="Storage of global definitions" CDD$TOP.CORPORATE
SET PROTECTION/POSITION=2/UIC=[*,*]/GRANT=<HS> CDD$TOP.CORPORATE
!
!
CREATE/AUDIT CDD$TOP.SALES
SET PROTECTION/POSITION=2/USER=JONES/GRANT=<HS> CDD$TOP.SALES
```

```
!
!
! Use the /SUBDICTIONARY qualifier to create a subdictionary
! directory.  The subdictionary file specification must contain
! both a device and a directory name in addition to the file name.
!
!
CREATE/AUDIT/SUBDICTIONARY=DBA5:[CASADAY.CDD]PERS.DIC CDD$TOP.PERSONNEL
SET PROTECTION/POSITION=2/PASSWORD="SEMI_SECRET"-
/GRANT=<HS> CDD$TOP.PERSONNEL
SET PROTECTION/POSITION=3/UIC=[*,*]/BANISH=ALL CDD$TOP.PERSONNEL
!
!
! After creating the first directory level, use the SET DEFAULT
! command to create the next level.
!
!
SET DEFAULT CDD$TOP.SALES
CREATE/AUDIT="Jones' personal directory" JONES
SET PROTECTION/POSITION=2/USER=JONES/GRANT=<CX> JONES
SET PROTECTION/POSITION=3/UIC=[*,*]/DENY=ALL JONES
!
!
SET DEFAULT CDD$TOP.PERSONNEL
CREATE/AUDIT="Department Standards" STANDARDS
!
!
CREATE/AUDIT="Sensitive Records" SERVICE
SET PROTECTION/POSITION=2/PASSWORD="SECRET"/GRANT=<HPS>-
/DENY=<DEMRUW>/BANISH=<C> SERVICE
SET PROTECTION/POSITION=3/UIC=[*,*]/BANISH=ALL SERVICE
!
SET DEFAULT CDD$TOP
LIST/FULL/PROTECTION >
EXIT
```

To execute the command, invoke DMU and type @CREATE at the DMU> prompt. After successful creation of the hierarchy, DMU lists the results on your terminal.

```
$ RUN SYS$SYSTEM:DMU
DMU> @CREATE

CDD$TOP.CORPORATE <DIRECTORY>
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX, Deny - none, Banish - none
  2:    [*,*]
        Grant - HS, Deny - none, Banish - none
    Created by CASADAY (UIC [30,10]) in process CASADAY
        using VAX CDD Dictionary Management Utility Version 3.00
        on 8-JAN-1984 21:12:03.22.
    Explanation:
        Storage of global definitions
```

```
CDD$TOP.PERSONNEL <SUBDICTIONARY> : DB3:[CASADAY.CDD]PERS.DIC
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX, Deny - none, Banish - none
  2:    [*,*], Password: "SEMI_SECRET"
        Grant - HS, Deny - none, Banish - none
  3:    [*,*]
        Grant - none, Deny - none, Banish - CDEFGHMPRSUWX
    Created by CASADAY (UIC [30,10]) in process CASADAY
        using VAX CDD Dictionary Management Utility Version 3.00
        on 8-JAN-1984 21:12:06.04.

CDD$TOP.PERSONNEL.SERVICE <DIRECTORY>
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX, Deny - none, Banish - none
  2:    [*,*], Password: "SECRET"
        Grant - HPS, Deny - DEMRUW, Banish - C
  3:    [*,*]
        Grant - none, Deny - none, Banish - CDEFGHMPRSUWX
    Created by CASADAY (UIC [30,10]) in process CASADAY
        using VAX CDD Dictionary Management Utility Version 3.00
        on 8-JAN-1984 21:12:09.85.
    Explanation:
        Sensitive Records

CDD$TOP.PERSONNEL.STANDARDS <DIRECTORY>
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX, Deny - none, Banish - none
    Created by CASADAY (UIC [30,10]) in process CASADAY
        using VAX CDD Dictionary Management Utility Version 3.00
        on 8-JAN-1984 21:12:09.65.
    Explanation:
        Department Standards

CDD$TOP.PRODUCTION <DIRECTORY>
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX, Deny - none, Banish - none
  2:    [*,*]
        Grant - HS, Deny - none, Banish - none
    Created by CASADAY (UIC [30,10]) in process CASADAY
        using VAX CDD Dictionary Management Utility Version 3.00
        on 8-JAN-1984 21:12:02.07.

CDD$TOP.SALES <DIRECTORY>
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX, Deny - none, Banish - none
  2:    [*,*], Username: "JONES"
        Grant - HS, Deny - none, Banish - none
    Created by CASADAY (UIC [30,10]) in process CASADAY
        using VAX CDD Dictionary Management Utility Version 3.00
        on 8-JAN-1984 21:12:04.44.
```

```
CDD$TOP.SALES.JONES <DIRECTORY>
  1:    [*,*], Username: "CASADAY"
        Grant - CDHPSX, Deny - none, Banish - none
  2:    [*,*], Username: "JONES"
        Grant - CX, Deny - none, Banish - none
  3:    [*,*]
        Grant - none, Deny - CDEFGHMPRSUWX, Banish - none
    Created by CASADAY (UIC [30,10]) in process CASADAY
        using VAX CDD Dictionary Management Utility Version 3.00
        on 8-JAN-1984 21:12:08.21.
    Explanation:
        Jones' personal directory
```

A.9.4 Maintaining DMU Dictionary Files

Use the CDD Verify/Fix Utility (CDDV) to maintain the main dictionary file and any subdictionary files. The owner of a dictionary file can use CDDV to maintain that file. If you do not own a dictionary file, you need VMS SYSPRV or BYPASS privilege to use CDDV with that dictionary file.

There are three major functions that CDDV performs:

- It verifies a dictionary file to determine if it is corrupt.
- It fixes corrupt dictionary files, when possible.
- It compresses dictionary files containing large amounts of free space.

When you use CDDV with the main dictionary file, nobody else can use the CDD. When you use CDDV with a subdictionary file, however, other users can use other portions of the CDD.

A.9.4.1 Verifying the Condition of a DMU Dictionary File – CDDV VERIFY scans a dictionary file for dictionary directories, objects, and history lists damaged by a hardware failure or some other cause. It writes a report describing the condition of the dictionary and reporting any corrupt directories, objects, or history lists it finds.

Verify the condition of your dictionary after hardware failures and before using CDDV COMPRESS or CDDV FIX. You should also verify your dictionary files periodically to monitor their condition.

CASADAY, the owner of PERS.DIC, a subdictionary file, could verify the condition of that file by typing:

```
$ RUN SYS$SYSTEM:CDDV
CDDV> VERIFY/COMPLETE/LISTING=PERS.LIS PERS.DIC
```

Because she uses the /COMPLETE qualifier, VERIFY creates a report listing:

- Corrupt directories that can be reconstructed
- Directories or objects whose corrupt parent directories can be reconstructed
- Uncorrupted directories whose corrupt parent directories cannot be reconstructed

The *VAX Common Data Dictionary Utilities Reference Manual* contains an example of a CDDV VERIFY listing of a corrupt file.

A.9.4.2 Fixing a DMU Dictionary File — When a dictionary file is corrupted, CDDV FIX:

- Reconstructs corrupt directories when possible, and restores their children
- Preserves uncorrupted directories and objects whose parents cannot be restored
- Deletes corrupted dictionary objects, history lists, and directories that cannot be repaired
- Rebuilds the free page list

For example, a hardware failure could corrupt the sample dictionary (Figure A-1). The following listing represents the state of the dictionary *before* the failure:

```
CDD$TOP
|  CORPORATE
|  |  ADDRESS_RECORD;1 <CDD$RECORD>
|  |  EMPLOYEE_LIST;1 <CDD$RECORD>
|  |  PRODUCT_INVENTORY;1 <CDD$RECORD>
|  PERSONNEL
|  |  SERVICE
|  |  |  SALARY_RECORD;2 <CDD$RECORD>
|  |  |  SALARY_RECORD;1 <CDD$RECORD>
|  |  STANDARDS
|  |  |  SALARY_RANGE;2 <CDD$RECORD>
|  |  |  SALARY_RANGE;1 <CDD$RECORD>
|  PRODUCTION
|  SALES
|  |  CUSTOMER_RECORD;1 <CDD$RECORD>
|  |  JONES
|  |  |  LEADS_RECORD;1 <CDD$RECORD>
|  |  SALES_RECORD;1 <CDD$RECORD>
```

If the hardware failure corrupts the two directories SALES and JONES and the object SALES_RECORD;1, FIX would do the following:

- Reconstruct SALES. SALES has an uncorrupted parent, CDD$TOP, and an uncorrupted child, CUSTOMER_RECORD;1. Therefore, FIX can reconstruct it from the pointers contained in its parent and its child.
- Create a directory, CDD$TOP.CDD$LOST_NODES, to hold LEADS_RECORD;1). LEADS_RECORD;1 is uncorrupted, but it cannot be restored to its former place in the dictionary because FIX cannot locate its parent, JONES. All uncorrupted directories and objects that cannot be restored are instead placed in CDD$TOP.CDD$LOST_NODES.
- Rebuild the free page list.

The following listing represents the sample dictionary *after* the FIX operation:

```
CDD$TOP
|  CDD$LOST_NODES
|  |  LEADS_RECORD;1 <CDD$RECORD>
|  CORPORATE
|  |  ADDRESS_RECORD;1 <CDD$RECORD>
|  |  EMPLOYEE_LIST;1 <CDD$RECORD>
|  |  PRODUCT_INVENTORY;1 <CDD$RECORD>
|  PERSONNEL
|  |  SERVICE
|  |  |  SALARY_RECORD;2 <CDD$RECORD>
|  |  |  SALARY_RECORD;1 <CDD$RECORD>
|  |  STANDARDS
|  |  |  SALARY_RANGE;2 <CDD$RECORD>
|  |  |  SALARY_RANGE;1 <CDD$RECORD>
|  PRODUCTION
|  SALES
|  |  CUSTOMER_RECORD;1 <CDD$RECORD>
```

Note that JONES and SALES_RECORD;1 are not reconstructed. JONES has an uncorrupted child, LEADS_RECORD;1, but its parent was corrupted. FIX cannot reconstruct a directory unless both its parent and a child are uncorrupted. Corrupted objects cannot be reconstructed. The *VAX Common Data Dictionary Utilities Reference Manual* contains sample CDDV FIX listings.

A.9.4.3 Compressing a DMU Dictionary File – When you delete a number of dictionary directories and objects, there may be a large amount of free space in a dictionary file. CDD uses this space before requesting additional space, but the dictionary file may contain unused space for a period of time. To free this space for use by other files, use the COMPRESS command.

COMPRESS also reorganizes dictionary pages to improve efficiency.

COMPRESS does not work if the file is corrupted. Before you compress a dictionary file, run CDDV VERIFY to make certain the file is not corrupted. If it is corrupted, use CDDV FIX before compressing the file.

CASADAY could compress PERS.DIC with the CDDV command:

```
CDDV> COMPRESS PERS.DIC PERS.DIC
```

This command compresses the file PERS.DIC and places the contents in a new file with the same name (but with a version number one higher). It is not necessary to use the same name for the input and output files, but it is better to keep the same name because:

- If you change the name of your subdictionary file, you must use the RENAME /SUBDICTIONARY command to set the subdictionary directory to point to it.
- After the COMPRESS operation, you can delete the old dictionary file simply by using the DCL PURGE command.

A compressed dictionary file often improves dictionary performance. You should compress your dictionary files periodically to maintain optimum dictionary performance.

A.10 Setting Up Your DMU Dictionary on a VAXcluster

This section discusses managing the dictionary in a VAXcluster environment. Material in this section applies only to systems that are connected in a VAXcluster.

VMS (Version 3.7 and later) permits you to connect an organization's individual computer systems into a cluster. Systems in a cluster can share processing resources as well as data stored on common disks.

If you are connecting your organization's systems in a cluster, you fall into one of three categories:

- You are installing your first dictionary.
- You currently have one existing dictionary.

- You currently have two or more dictionaries (each on a different system).

Whichever your category, you should plan to locate your root dictionary on a cluster disk that is always accessible to all CDD users from any system in the cluster. Placing CDD.DIC on a cluster disk and establishing a single logical dictionary:

- Allows a user access to the dictionary even if his particular system fails.
- Lets your organization share data definitions more widely. A single dictionary limits redundant data storage and helps ensure consistency of data. If you maintain separate dictionaries on different systems, you risk discrepancies in data definition and inconsistencies in updating data.
- Lets you maintain extra autonomy and security of any current system-specific dictionary by keeping it as a subdictionary.

A.10.1 Installing Your First DMU Dictionary

To install your first dictionary on a cluster-wide disk, see the *VAX Common Data Dictionary Installation Guide.*

Once you have physically established your root dictionary file on a shared disk, follow the recommendations in this manual on organizing and maintaining your single logical dictionary.

A.10.2 Moving a Single Existing DMU Dictionary to a Cluster Disk

If you have only one dictionary and want to move it to a cluster disk, follow the steps in this section. You may have placed the root dictionary on a cluster disk when you installed CDD Version 4.x and defined the logical name CDD$DICTIONARY in the CDDSTRTUP.COM prodedure. If you did not, you should now move the root dictionary to a disk that is always accessible by all systems in the cluster. You can use a system logical name for the device and directory, but you cannot use a group or process logical name. The default file name for the root dictionary file is CDD.DIC. Be sure to define CDD$DICTIONARY the same way on each system that accesses the CDD.

In the following example, CASADAY, the cluster manager, locates the dictionary on $2$DUA1:, a cluster disk:

```
$ DEFINE/SYSTEM CDD$DICTIONARY $2$DUA1:[CDD]CDD.DIC
```

Optionally, CASADAY could create a system logical name to stand for $2$DUA1:[CDD]. If she has created the logical name CDD$DISK for $2$DUA1:[CDD], for example, she can define the location of the root dictionary with the following command:

```
$ DEFINE/SYSTEM CDD$DICTIONARY CDD$DISK:CDD.DIC
```

Then she can copy the dictionary file to its new location on the shared disk. In the following example CASADAY copies the former root dictionary file from the disk DB2: and directory [CASADAY.CDD] into the directory [CDD] on $2$DUA1:, the cluster-wide disk:

```
$ COPY DB2:[CASADAY.CDD]CDD.DIC  $2$DUA1:[CDD]CDD.DIC
```

Remember that the new file created by DCL COPY has whatever default file protection is specified. You may need to explicitly change the protection.

A.10.3 Merging Two or More DMU Dictionaries onto a Cluster Disk

If you have more than one existing dictionary, you can merge them into a single dictionary in phases to minimize the impact on users:

1. Create a single logical dictionary from multiple dictionaries.
2. Reorganize the new single dictionary and eliminate redundant data definitions.
3. Change existing path names to reflect the changes in dictionary hierarchy.

Figure A-2 shows portions of the directory hierarchies of two dictionaries, one on system ALPHA and one on system OMEGA, before merging.

Figure A-2 shows the problem facing CASADAY, the cluster manager responsible for merging the two dictionaries existing on systems ALPHA and OMEGA. She cannot declare CDD$TOP the universal parent directory, because she would create duplicate path names CDD$TOP.CORPORATE.ADDRESS_RECORD. Therefore, she creates a new CDD$TOP as the parent of the ALPHA and OMEGA dictionaries. As Figure A-3 shows, the former CDD$TOPs on ALPHA and OMEGA thus become CDD$TOP.ALPHA and CDD$TOP.OMEGA respectively. She tells users to define the logical name CDD$TOP to translate to either CDD$TOP.ALPHA or CDD$TOP.OMEGA.

Figure A–2: Portions of Hierarchies of Two DMU Dictionaries

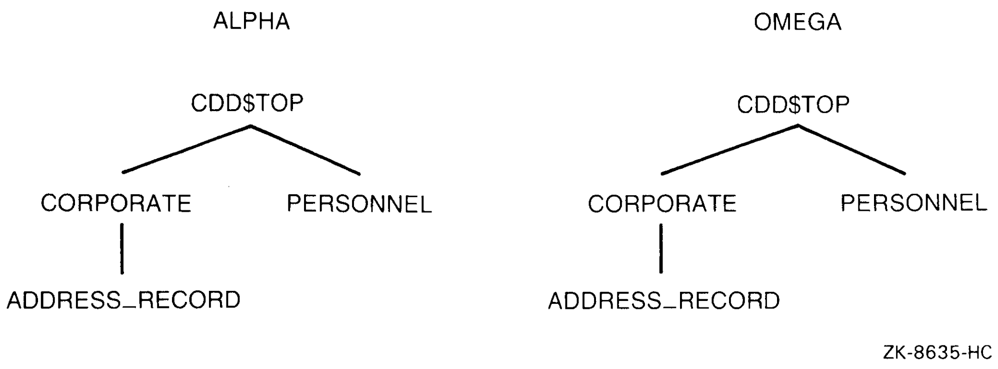
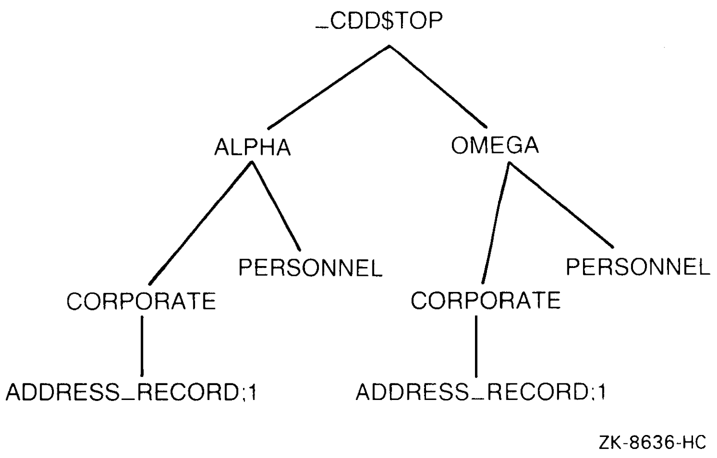


Figure A–3: DMU Dictionaries on ALPHA and OMEGA Merged



The underscore preceding the ultimate CDD$TOP prevents CDD from translating it as a logical name.

If she wants to retain the pre-cluster organization and keep the former ALPHA and OMEGA dictionaries separate, she can tell users to change their path names to reflect the new directories ALPHA and OMEGA.

After CASADAY has physically merged the dictionaries, however, the new cluster dictionary contains undesirable redundancies. She should now decide on a definitive ADDRESS_RECORD that all users of both former ALPHA and OMEGA dictionaries can share.

The following two sections describe in detail how to physically combine multiple dictionaries and how to logically reorganize your cluster-wide dictionary.

A.10.3.1 Creating a Single Logical DMU Dictionary – You create a new file for the root dictionary. During the transitional phase, the root dictionary will refer to your current dictionary files as subdictionaries.

The following steps merge dictionaries that exist on separate systems:

1. Boot the systems as for a VAXcluster, but use the same CDDSTRTUP.COM procedures as during the original installation of each dictionary; the dictionaries are still separate at this phase. During the next three steps, no one can use the CDD or products that require it.

2. Move each dictionary from its local device to a shared disk. Thus, access to the dictionary no longer depends on any particular system.

   In the following example, CASADAY creates the directory [CASADAY.CDD] on a cluster-wide disk, $2$DUA1:, and copies into it each existing dictionary from disk DB2:.

   ```
   $ CREATE/DIR $2$DUA1:[CASADAY.CDD]
   $ COPY DB2:[CASADAY.CDD]ALPHA.DIC $2$DUA1:[CASADAY.CDD]ALPHA.DIC
   $ COPY DB2:[CASADAY.CDD]OMEGA.DIC $2$DUA1:[CASADAY.CDD]OMEGA.DIC
   ```

   After checking that the dictionaries were copied successfully, she can delete the ones on DB2:.

3. Define the location of the new root dictionary, CDD$DICTIONARY, to be a file on a disk accessible to all the systems in the cluster. This location need not be the same as the disk or disks containing the system-specific dictionaries moved in Step 2.

   You can use a system logical name for the device and directory, but you cannot use a group or process logical name. The default file name for the root dictionary file is CDD.DIC. The following example locates the dictionary on $2$DUA1:, a cluster disk:

   ```
   $ DEFINE/SYSTEM CDD$DICTIONARY $2$DUA1:[CDD]CDD.DIC
   ```

   If CASADAY has created a system logical name CDD$DISK to stand for $2$DUA1:[CDD], she can define the location of the root dictionary with the following command:

   ```
   $ DEFINE/SYSTEM CDD$DICTIONARY CDD$DISK:CDD.DIC
   ```

Be sure to edit your CDDSTRTUP.COM file to reflect this new logical name definition.

When you invoke DMU, an empty root file will be created in the location CDD$DICTIONARY specifies.

4. Create a subdictionary directory pointing to each formerly separate dictionary.

   In the following example CASADAY invokes DMU. The file she has just created is empty until she creates a subdictionary directory pointing to each of two current dictionaries. This step merges the two dictionaries into a single logical dictionary.

   ```
   $ DMU
   DMU> LIST
   %DMU-W-NONODFND, no directories or objects found
   DMU> CREATE/SUBDICTIONARY=$2$DUA1:[CASADAY.CDD]ALPHA.DIC ALPHA
   DMU> CREATE/SUBDICTIONARY=$2$DUA1:[CASADAY.CDD]OMEGA.DIC OMEGA
   ```

5. Now you have changed the path names to existing objects; for example, the CDD$TOP of ALPHA is now CDD$TOP.ALPHA. How you choose to make the new path names consistent with those in the existing definitions depends on how you plan to organize your new merged dictionary. Possible options include:

   - You may decide to eliminate system-specific directory names from the new dictionary. If different departments of your organization will now share dictionary definitions, for example, ALPHA may be a meaningless entity. In this case you will probably be changing the structure of the former system-specific dictionaries. As you reorganize the dictionary, you must change affected path names.

   - You may decide to keep the system-specific names. If you want to keep OMEGA's definitions separate, for example, you can retain the directory OMEGA. In this case, you must change path names of objects under OMEGA to reflect this new directory name.

     Also, if any system-specific subdictionary will remain logically separate from the rest of the dictionary, a user who works in that subdictionary can now permanently adjust his login procedure to reflect his new default directory. The following command in user JONES' login command file establishes ALPHA as a directory in his default directory path name:

     ```
     $ ASSIGN "_CDD$TOP.ALPHA.JONES" CDD$DEFAULT
     ```

If your merged dictionary requires many changes, which it will take time to implement, you may choose to reorganize your dictionary gradually, a branch at a time. You can allow a user to use his own area of the dictionary while you are in the process of reorganizing other portions of the hierarchy. A user can temporarily define a process logical name to refer to the dictionary on his system. In the following example, user JONES defines _CDD$TOP.ALPHA to refer to the dictionary that exists on system ALPHA. The underscore prevents CDD from translating the root CDD$TOP as a logical. JONES can include this command in his login command file:

```
$ DEFINE CDD$TOP "_CDD$TOP.ALPHA"
```

When you are sure that all a user's path names under CDD$TOP are stable, he can delete the command. If the path name of his default directory has changed, he should modify any of his own definitions to reflect their new location. For example, he may need to change DBMS or DATATRIEVE definitions.

After you have finished redefining CDD$DICTIONARY and converting existing dictionaries to subdictionaries, users can access the CDD again.

A.10.3.2 Reorganizing and Eliminating Redundant Definitions – Although you now have one logical dictionary, it probably contains record definitions that are duplicate or overlapping. For example, if _CDD$TOP.ALPHA.CORPORATE.ADDRESS_RECORD is identical to _CDD$TOP.OMEGA.CORPORATE.ADDRESS_RECORD, one should be deleted. Similarly, if ADDRESS_RECORD on ALPHA is almost identical to ADDRESS_RECORD on OMEGA, you can establish either one (or create a new one) as the definitive ADDRESS_RECORD. On the other hand, if the two ADDRESS_RECORDs are totally different, you should rename one.

In addition, your merged dictionary may now include record definitions with *different* given names but similar source code. For example, record definitions named PART_REC and ITEM_REC from two former system dictionaries may store essentially the same definitions.

Sorting the definitions in your merged dictionary includes three steps:

- Planning the cluster dictionary's hierarchy
- Identifying redundant dictionary elements
- Moving portions of former system-specific dictionaries to appropriate directories of the cluster dictionary

A.10.3.2.1 Planning the Hierarchy of the Cluster Dictionary —Before you move portions of any system-specific subdictionary to the main dictionary file, plan an overall directory hierarchy that suits the needs of your organization. The more definitions you centralize and share widely, the more efficiently you can use your dictionary.

You may want to keep one or more of your former dictionaries logically separate, in its own branch of the hierarchy. For example, if ALPHA's data will be accessed only by its former users, CASADAY may leave its dictionary hierarchy untouched, as long as all its path names are unique.

In certain cases, you may even want to keep a former dictionary as a subdictionary. Subdictionaries require more time for I/O operations than dictionary directories, and they are charged against FILLM, your process' VMS open file limit. On the other hand, if a group of users works almost entirely in one area, keeping that area a subdictionary improves locking efficiency on a cluster. See Section A.9.1.5 for further recommendations about using subdictionaries.

The guidelines in Section A.9 for structuring your dictionary apply to cluster dictionaries as well as those on separate systems.

A.10.3.2.2 Identifying Redundant DMU Dictionary Elements —In the following example, CASADAY displays a list of all definitions of the type CDD$RECORD in the directories of ALPHA and OMEGA. Then she can inspect the file RECORDS.LIS for those that have the same given name or that she suspects contain duplicate or redundant definitions.

```
DMU> SET DEFAULT CDD$TOP
DMU> LIST/LISTING=RECORDS.LIS/TYPE=CDD$RECORD
```

The following example shows two separate portions of the listing CASADAY obtains. To see whether or not CDD$TOP.ALPHA.PRODUCTION.PARTS_REC and CDD$TOP.OMEGA.PRODUCTION.FACTORY.ITEM_REC are redundant, she then uses the LIST command to inspect the two source texts. She includes the /AUDIT_TRAIL qualifier to help her judge the relative impacts of eliminating either one:

```
   ALPHA
   |  PRODUCTION
   |  |  PARTS_RECORD;1 <CDD$RECORD>

   OMEGA
   |  PRODUCTION
   |  |  FACTORY
   |  |  |  ITEM_RECORD;1 <CDD$RECORD>
```

```
DMU> LIST/AUDIT_TRAIL/ITEM=SOURCE/LISTING=PARTS.LIS_
DMU>_CDD$TOP.ALPHA.PRODUCTION.PARTS_RECORD
DMU> LIST/AUDIT_TRAIL/ITEM=SOURCE/LISTING=ITEM.LIS_
DMU>_CDD$TOP.OMEGA.PRODUCTION.FACTORY.ITEM_RECORD
```

A.10.3.2.3 Moving Portions of Former Subdictionaries Within the Cluster Dictionary —You probably can use many existing definitions, but you may need to change their location to reflect any new combination of departments now sharing the cluster-wide dictionary. If more users will now share ADDRESS_RECORD, for example, locating it in an appropriate corporate directory simplifies access to it.

If you no longer need to keep a former system-specific dictionary as a separate branch of your cluster dictionary, you should eliminate the directory associated with it.

In the following example cluster manager CASADAY has decided to implement in the merged dictionary the same organizational hierarchy as in the old ALPHA system dictionary. She has already copied directories CDD$TOP.ALPHA.PERSONNEL, CDD$TOP.ALPHA.PRODUCTION, and CDD$TOP.ALPHA.SALES to CDD$TOP. She uses the LIST command to check these three directories and then copies the fourth and last directory, CDD$TOP.ALPHA.CORPORATE, to CDD$TOP. The /HISTORY and /PROTECTION qualifiers copy history lists and access control lists for each directory and object copied.

Because she has copied the other directories, she then deletes the subdictionary ALPHA.

```
DMU> SET DEFAULT CDD$TOP
DMU> LIST
   ALPHA <SUBDICTIONARY> : $2$DUA1[CASADAY.CDD]:ALPHA.DIC
   PERSONNEL
   PRODUCTION
   SALES
DMU> COPY/PROTECTION/HISTORY CDD$TOP.ALPHA.CORPORATE
DMU> LIST
   ALPHA <SUBDICTIONARY> : $2$DUA1:[CASADAY.CDD]ALPHA.DIC
   CORPORATE
   PERSONNEL
   PRODUCTION
   SALES
DMU> DELETE/ALL/SUBDICTIONARY CDD$TOP.ALPHA
DMU> LIST
   CORPORATE
   PERSONNEL
   PRODUCTION
   SALES
```

Copying a directory as recommended will convert any subdictionaries in the hierarchy below it to directories in the main dictionary file. You can change a directory back into a subdictionary with DMU CREATE/SUBDICTIONARY and COPY.

In the following example CASADAY recreates OMEGA as a subdictionary. Because the directory OMEGA already exists, she names the subdictionary directory OMEGA2. Then she uses a wildcard to copy all the directories and objects under OMEGA to OMEGA2. The example shows a portion of the list CASADAY obtains to check that the contents of OMEGA were copied successfully. When she has deleted the old directory and its contents, she can rename the subdictionary directory OMEGA2 back to OMEGA:

```
DMU> CREATE/SUBDICTIONARY=$2$DUA1:[CASADAY.CDD]OMEGA2.DIC OMEGA2
DMU> COPY OMEGA.* OMEGA2
DMU> LIST ALPHA2>
   OMEGA2 <SUBDICTIONARY> : $2$DUA1:[CASADAY.CDD]OMEGA2.DIC
   |  PRODUCTION
   |     PARTS_RECORD:1 <CDD$RECORD>

DMU> DELETE/ALL OMEGA
DMU> RENAME OMEGA2 OMEGA
```

You can locate a subdictionary on a different device from the root file, but only users with access to that device will be able to use it.

A.10.3.3 Modifying Former Absolute Path Names – When your dictionary structure is stable, you may need to modify many CDDL source files or programs that copy dictionary objects. If a file refers to a dictionary object by absolute path name or by a logical name that translates to an absolute path name, you must edit the file to reflect any changed location. Use the object's history list to discover the locations of source files, programs, and procedures that use the definition.

In addition, some VAX Information Architecture products store pointers to dictionary objects' locations by full path name, even if the user who defined the object entered only a relative path name. Therefore, after you copy an object elsewhere in the hierarchy, the dictionary tries to concatenate its former full path name to its new location, and fails to locate it. This section describes how to restore the correct path specification to DATATRIEVE and DBMS dictionary objects:

- DATATRIEVE Definitions

  Certain DATATRIEVE objects in the CDD, such as domains, refer in their definitions to associated objects:

  - A DATATRIEVE domain definition refers to a single associated record definition.
  - A DATATRIEVE domain table definition refers to a single associated domain definition.
  - A DATATRIEVE view domain definition refers to one or more associated domain definitions.

When DATATRIEVE processes these definitions, it translates the associated references to full CDD path names. Therefore, after you move a domain, domain table, or view domain definition, and its associated object, with DMU COPY, BACKUP, or RESTORE, you must reprocess these definitions in DATATRIEVE to make sure they refer to the correct path name for the associated object. To reprocess the definitions, use the DATATRIEVE REDEFINE command or the DATATRIEVE EXTRACT ALL command.

The DATATRIEVE REDEFINE command redefines definitions without deleting any history or access control lists. Follow one of two procedures, depending on the number of definitions you must redefine:

- If you have only a few definitions to redefine, follow these steps:

  1. Enter DATATRIEVE and use the SET DICTIONARY command to place yourself in the dictionary directory containing definitions to be redefined.

  2. Enter an EDIT path-name command for the domain or domain table to be redefined. DATATRIEVE EDIT automatically enters a REDEFINE command for the definition in your editing buffer.

  3. If SET NO EDIT_BACKUP is in effect during your session, remove the DELETE path-name command that appears at the top of your editing buffer. This step preserves access control and history lists.

  4. Exiting from the DATATRIEVE editor executes the REDEFINE command. REDEFINE redefines the definition to reflect its current path-name.

  5. Repeat steps 2 through 4 for each definition that needs to be redefined.

- If you have many definitions to redefine, you can use the DATATRIEVE EXTRACT ALL command, using these steps:

  1. Enter DATATRIEVE. Use EXTRACT ALL ON filename.COM to copy dictionary directory contents to a command file in your current VMS directory. For example, EXTRACT ALL ON TEMP.COM creates the VMS command file TEMP.COM.

  2. The command file contains a DELETE command, followed by a REDEFINE command, for each definition in your directory. You must exit from DATATRIEVE to edit the command file to remove all the DELETE commands. This step preserves access control and history lists.

  3. Reenter DATATRIEVE and use SET DICTIONARY to place yourself in the directory containing the definitions to be redefined.

4. Execute the command file. For example, the command @TEMP executes TEMP.COM.

- DBMS Definitions

  You can copy DBMS definitions to new locations using the INTEGRATE command. For information on this procedure, see the *VAX DBMS Database Administration Reference Manual.*

Index

This index uses the following symbols:

t	Entry occurrence in a table
f	Entry occurrence in a figure
e	Entry occurrence in an example

A

Abbreviations in utility commands, A-16
Access control list editor
  example, A-54 to A-62
  rationale, A-63
Access control list entry, A-43
Access control lists, A-26, A-43
  editing example, A-54 to A-62
  example, A-44, A-50 to A-53
  organization, A-49
  position of entries, A-52
  summary of results, A-54
Access privileges, A-47 to A-49
  banishment, A-47, A-51 to A-53
  checking with SHOW PROTECTION, A-26
  CONTROL (C), A-48t
  defaults, A-49
  denial, A-47, A-51 to A-53
  description, A-48t
  DTR_EXTEND/EXECUTE (E), A-47t
Access privileges (cont'd.)
  DTR_MODIFY (M), A-48t
  DTR_READ (R), A-48t
  DTR_WRITE (W), A-48t
  EXTEND (X), A-48t
  FORWARD (F), A-48t
  GLOBAL_DELETE (G), A-48t
  granting, A-47, A-51 to A-53
  HISTORY (H), A-48t
  inheritance, A-43
  LOCAL_DELETE (D), A-48t
  PASS_THRU (P), A-48t
  SEE (S), A-48t
  UPDATE (U), A-48t
ACL
  See Access control lists
ALIGNED field attribute, A-73t
Ancestor, A-2
ARRAY field attribute, A-73t
ASSIGN
  DCL command, A-24

Asterisk(*)
  in path names, A-20
  in UICs, A-45
  in version numbers, A-20
At sign(@)
  in path names, A-21
/AUDIT qualifier
  with CDDL, A-84
  with VAX Information Architecture products, A-88
  with VAX languages, A-88
Audit trails
  See History lists

B

BACKUP command, A-35
BACKUP/LIST command, A-37
BIT data type, A-77
BLANK WHEN ZERO field attribute, A-74t
BYPASS, A-68
BYTE data type, A-75t

C

CDD$DEFAULT, A-7, A-24 to A-25
CDD$LOST_NODES, A-109
CDD$TOP, A-2
CDD$VERSION_LIMIT, A-20
CDDL, A-11
  command descriptions, A-11t
  sample listing file, A-85
CDDL data types, A-74
  BIT, A-77
  BYTE, A-75t
  character string, A-75
  DATE, A-77
  decimal string, A-76
  D_FLOATING, A-76t
  fixed point, A-75
  floating point, A-75
  F_FLOATING, A-76t
  G_FLOATING, A-76t
  H_FLOATING, A-76t
  language support, A-78t
  LEFT OVERPUNCHED NUMERIC, A-77t
CDDL data types (cont'd.)
  LEFT SEPARATE NUMERIC, A-77t
  LONGWORD, A-75t
  OCTAWORD, A-75t
  PACKED DECIMAL, A-77t
  POINTER, A-77
  QUADWORD, A-75t
  RIGHT OVERPUNCHED NUMERIC, A-77t
  RIGHT SEPARATE NUMERIC, A-77t
  UNSIGNED NUMERIC, A-77t
  UNSPECIFIED, A-77
  VARYING STRING, A-77
  VAX Information Architecture support, A-81t
  VIRTUAL FIELD, A-77
  WORD, A-75t
  ZONED NUMERIC, A-77t
CDDL source files
  DEFINE statements, A-69
  documenting field description statements, A-72
  END statement, A-70
  sample, A-69
CDDV, A-10, A-107 to A-110
  abbreviations, A-16
  CDD$LOST_NODES, A-109
  command descriptions, A-9t
  comments in command lines, A-17
  COMPRESS command, A-110
  continuing commands on the next line, A-17
  FIX command, A-108
  invoking the utility, A-12 to A-13
  VERIFY command, A-107
Character string data types, A-75
Checking template records, A-94
Checking the contents of a backup file
  with BACKUP/LIST, A-37
Checking your privileges
  with SHOW PROTECTION, A-26
Cluster
  subdictionaries, A-117
Cluster disk
  moving existing dictionary to, A-111
Clusters, A-110
  merging multiple dictionaries on, A-114

Clusters (cont'd.)
  planning hierarchy of cluster dictionary, A-117
Compiling a record definition, A-84 to A-86
  sample listing file, A-85
  with a relative path name, A-85
COMPRESS command, A-110
COMPUTED BY DATATRIEVE field attribute, A-73t
CONDITION FOR COBOL field attribute, A-73t
CONTROL (C) privilege, A-48t, A-50
Control C
  See CTRL/C
Converting CDD record definitions in VAX BASIC, A-88
COPY command, A-27
  creating multiple versions of dictionary objects, A-41
COPY field description statements, A-70, A-92
Copying record definitions into programs, A-87
CREATE.COM
  DMU command procedure, A-103 to A-107
CREATE command, A-100
Creating a sample hierarchy, A-103 to A-107
Creating CDDL source text
  with DMU EXTRACT/RECORD, A-90
Creating multiple versions of dictionary objects, A-91
Creating the sample dictionary, A-103 to A-107
CTRL/C, A-17 to A-18

D

D_FLOATING data type, A-76t
Data Definition Language Utility
  See CDDL
DATATYPE field attribute, A-73t
Data types
  BIT, A-77
  BYTE, A-75t
  character string, A-75

Data types (cont'd.)
  DATE, A-77
  decimal string, A-76
  D_FLOATING, A-76t
  fixed point, A-75
  floating point, A-75
  F_FLOATING, A-76t
  G_FLOATING, A-76t
  H_FLOATING, A-76t
  language support, A-79t
  LEFT OVERPUNCHED NUMERIC, A-77t
  LEFT SEPARATE NUMERIC, A-77t
  LONGWORD, A-75t
  OCTAWORD, A-75t
  PACKED DECIMAL, A-77t
  POINTER, A-77
  QUADWORD, A-75t
  RIGHT OVERPUNCHED NUMERIC, A-77t
  RIGHT SEPARATE NUMERIC, A-77t
  UNSIGNED NUMERIC, A-77t
  UNSPECIFIED, A-77
  VARYING STRING, A-77
  VAX Information Architecture support, A-82t
  VIRTUAL FIELD, A-77
  WORD, A-75t
  ZONED NUMERIC, A-77t
DATE data type, A-77
DBMS
  modifying definitions, A-121
DCL
  ACL Editor Utility, A-66
  EDIT/ACL command, A-66
DCL ASSIGN command, A-24
DCL command procedures
  running CDD utilities from, A-14 to A-15
DCL DEFINE command, A-24
DCL SET PROTECTION command, A-64 to A-65
  categories of users, A-64
  rights, A-64
Decimal string data types, A-76
Default dictionary directory, A-7, A-24 to A-25
DEFAULT_VALUE field attribute, A-74t

DEFINE
  CDDL source file statement, A-68
  DCL command, A-24
DEFINE statements, A-69
DELETE command, A-36
  deleting subdictionaries, A-101
Descendant, A-2
Dictionary
  sample, A-2 to A-5
Dictionary directory, A-2
  default, A-7
  root, A-2
Dictionary Management Utility
  See DMU
Dictionary objects, A-2
Dictionary path names
  full, A-6 to A-7
  relative, A-7
Dictionary performance, A-102 to A-103
Dictionary types, A-5
Dictionary Verify/Fix Utility
  See CDDV
Directory
  default, A-7
  dictionary, A-2
  root dictionary, A-2
Directory hierarchy, A-2 to A-5
  organization, A-97 to A-99
DMU, A-8
  abbreviations, A-16
  BACKUP command, A-35
  BACKUP/LIST command, A-37
  command descriptions, A-9t
  comments in command lines, A-17
  continuing commands on the next line, A-17
  COPY command, A-27
    creating versions of dictionary objects, A-41
  CREATE command, A-100
  creating multiple versions of dictionary objects, A-39 to A-42
  DELETE command, A-36
    deleting subdictionaries, A-101
  EXTRACT/RECORD command, A-90
  EXTRACT/RECORD/TEMPLATE command, A-94
DMU (cont'd.)
  invoking the utility, A-12 to A-13
  LIST/BRIEF command, A-29
  LIST/FULL command, A-31
  PURGE command, A-36
  RENAME command, A-40
  RESTORE command, A-37
  SET DEFAULT command, A-29
  SHOW DEFAULT command, A-25
  SHOW PROTECTION command, A-26
  single utility command per line, A-17
DMU command procedures
  creating the sample dictionary, A-103 to A-107
  halting with SET ABORT, A-15 to A-16
Documenting dictionary directories and objects, A-30 to A-31
Documenting dictionary use, A-89
Documenting field description statements, A-72
DTR_EXTEND/EXECUTE (E) privilege, A-48t
DTR_MODIFY (M) privilege, A-48t
DTR_READ (R) privilege, A-48t
DTR_WRITE (W) privilege, A-48t

E

Editor
  See Access control list editor
EDIT_STRING field attribute, A-74t
Eightbit characters
  path names, A-19
Elementary field description statements, A-70
END statement, A-70
Exclamation point(!)
  in comments, A-72
  in utility command lines, A-17
EXTEND (X) privilege, A-48t
EXTRACT/RECORD command, A-90

F

F_FLOATING data type, A-76t
Facility-specific field attributes, A-73
Field attribute clauses, A-73

Field attribute clauses (cont'd.)
  ALIGNED, A-73t
  ARRAY, A-73t
  BLANK WHEN ZERO, A-74t
  COMPUTED BY DATATRIEVE, A-74t
  CONDITION FOR COBOL, A-74t
  DATATYPE, A-73t
  DEFAULT_VALUE, A-74t
  EDIT_STRING, A-74t
  facility-specific, A-73
  general, A-73t
  INDEXED BY, A-74t
  INITIAL_VALUE, A-73t
  JUSTIFIED RIGHT, A-74t
  MISSING_VALUE, A-74t
  NAME, A-74t
  OCCURS, A-73t
  OCCURS...DEPENDING, A-73t
  PICTURE, A-74t
  QUERY_HEADER, A-74t
  QUERY_NAME, A-74t
  VALID FOR DATATRIEVE IF, A-74t
Field attributes, A-68
Field description statements
  COPY, A-70
  elementary, A-70
  STRUCTURE, A-70
  VARIANTS, A-71
    sample, A-71
File protection
  categories of users, A-64
  using VMS, A-63 to A-68
  VMS rights, A-64
FIX command, A-108
Fixed point data types, A-75
Floating point data types, A-75
FORWARD (F) privilege, A-48t, A-50
Full dictionary path names, A-6 to A-7

G

G_FLOATING data type, A-76t
Given name, A-19
Given names, A-6
GLOBAL_DELETE (G) privilege, A-48t, A-50

H

H_FLOATING data type, A-76t
Halting DMU command procedures, A-15 to A-16
Hierarchy
  directory, A-2 to A-5
Hierarchy, directory
  organization, A-97 to A-99
HISTORY (H) privilege, A-48t
History lists, A-30 to A-31
Hyphen(-)
  as a continuation character, A-17
  in path specifications, A-22

I

INDEXED BY field attribute, A-74t
INITIAL_VALUE field attribute, A-73t

J

JUSTIFIED RIGHT field attribute, A-74t

L

Language support, A-79t
LEFT OVERPUNCHED NUMERIC data type, A-77t
LEFT SEPARATE NUMERIC data type, A-77t
LIST/BRIEF command, A-29
LIST/FULL command, A-31
LOCAL_DELETE (D) privilege, A-48t
Logical names
  in path names, A-23 to A-24
LONGWORD data type, A-75t

M

Maintaining dictionary files, A-107 to A-110
MISSING_VALUE field attribute, A-74t
Modifying a record definition, A-89 to A-96
  using DMU EXTRACT, A-89
Multiple versions of dictionary objects
  with DMU, A-39 to A-42

N

NAME field attribute, A-74t
Names
  source file, A-68
/NOACL qualifier
  with CDDL, A-84

O

Objects
  dictionary, A-2
OCCURS...DEPENDING field attribute, A-73t
OCCURS field attribute, A-73t
OCTAWORD data type, A-75t

P

PACKED DECIMAL data type, A-77t
Passwords
  in path names, A-22
PASS_THRU (P) privilege, A-48t
Path names, A-6 to A-8, A-18
  full, A-6 to A-7
  relative, A-7
/PATH qualifier
  with CDDL, A-85
Path specifications
  given name, A-19
  hyphen, A-22
  logical names, A-23 to A-24
  passwords, A-22
  path name, A-18
  wildcards, A-20 to A-22
  wildcard table, A-21t
Percent sign(%)
  in path names, A-20
Performance
  improving, A-102 to A-103
PICTURE field attribute, A-74t
POINTER data type, A-77
Privileges
  See Access privileges
Protection
  See Access control lists
PURGE command, A-36

Q

QUADWORD data type, A-75t
QUERY_HEADER field attribute, A-74t
QUERY_NAME field attribute, A-74t

R

/RECOMPILE qualifier
  with CDDL, A-96
Relative dictionary path names, A-7
Relative path names
  with CDDL, A-85
Removing obsolete directories and objects, A-33 to A-36
  with DELETE, A-36
  with DMU PURGE, A-36
RENAME command, A-40
/REPLACE qualifier
  with CDDL, A-91
Replacing a CDDL definition, A-91
RESTORE command, A-37
Restoring directories and objects, A-37 to A-39
Right angle bracket (>)
  in path names, A-21
RIGHT OVERPUNCHED NUMERIC data type, A-77t
Rights database, A-46
RIGHT SEPARATE NUMERIC data type, A-77t
Root dictionary directory, A-2

S

Sample CDDL listing file, A-85
Sample dictionary, A-2 to A-5
Security
  See Access control lists
SEE (S) privilege, A-48t
SET ABORT
  with DMU command procedures, A-15 to A-16
SET DEFAULT command, A-29
SHOW DEFAULT command, A-25
SHOW PROTECTION command, A-26
SIGNED NUMERIC data type
  See ZONED NUMERIC data type

SIGNED NUMERIC LEFT OVERPUNCHED data type
  See LEFT OVERPUNCHED NUMERIC data type
SIGNED NUMERIC LEFT SEPARATE data type
  See LEFT SEPARATE NUMERIC data type
SIGNED NUMERIC RIGHT OVERPUNCHED data type
  See RIGHT OVERPUNCHED NUMERIC data type
SIGNED NUMERIC RIGHT SEPARATE data type
  See RIGHT SEPARATE NUMERIC data type
Source file names, A-68
Source files
  general format, A-68
  sample, A-69
Specifying version numbers
  in object names, A-19t
STRUCTURE field description statements, A-70
Subdictionaries, A-99 to A-101
  creating, A-100
  deleting, A-101
  in cluster dictionary, A-117
  limitations, A-100
  moving, A-101
  rationale, A-99
  using system logical names, A-100
Subdictionary, A-5
  VMS file protection, A-65
/SUBDICTIONARY qualifier
  with CREATE, A-100
  with DELETE, A-101

T

/TEMPLATE qualifier
  with EXTRACT, A-94
Template records, A-92
Types, A-5

U

UNSIGNED NUMERIC data type, A-77t
UNSPECIFIED data type, A-77
UPDATE (U) privilege, A-48t
User identification criteria, A-44 to A-46
  alphanumeric UIC, A-45
  job class, A-46
  numeric UIC, A-45
  passwords, A-46
  rights identifier, A-45
  terminal number, A-46
  VMS user name, A-45
Using record definitions, A-87 to A-96
Utility command procedures
  running CDD utilities from, A-14 to A-16

V

VALID FOR DATATRIEVE IF field attribute, A-73t
VARIANTS field description statements, A-71
  sample, A-71
VARYING STRING data type, A-77
VERIFY command, A-107
Version numbers
  in dictionary object names, A-7
  specifying, A-19t
/VERSION qualifier, A-39 to A-42
  with CDDL, A-91
  with DMU COPY, A-41
  with DMU RENAME, A-40
Versions
  setting maximum number, A-20
VIRTUAL FIELD data type, A-77
VMS file protection, A-63 to A-68
  categories of users, A-64
  rights, A-64
  subdictionary, A-65

W

Wildcards in path names, A-20 to A-22
WORD data type, A-75t

Z

ZONED NUMERIC data type, A-77t

Glossary

access control list

A table listing the users that are allowed access to an object and the kind of access they are allowed. CDD/Plus maintains an access control list (ACL) for each object you store in the CDD/Plus dictionary system. In addition, CDD/Plus maintains an ACL for each directory contained in a DMU dictionary.

See also privilege

access control list entry (ACE)

An entry in an access control list (ACE). An ACE is a link between a user or group of users and the access privileges they have to a particular object.

anchor

A VMS directory containing all the files that describe a CDO dictionary and directory system.

array

An ordered list of elements of a given datatype. In CDD/Plus, you describe an array with the ARRAY attribute.

attribute

A characteristic of an element. An attribute contains a value for an element in the CDD/Plus dictionary system. Examples of attributes are length and datatype.

attribute protocol

A set of rules for defining and interpreting attributes. An example of an attribute protocol is CDD$DATA_ELEMENT_LENGTH.

buffer

A contiguous block of bytes that programs use to exchange information with the call interface. Buffers adhere to strict syntax rules, described in the *VAX CDD/Plus Call Interface Manual.*

buffer version

An internal number, comprising both a major and minor version number, CDD/Plus software clients use the number to identify the version of the syntax rules describing a buffer's format.

See also protocol version, object version, buffer

call interface

A set of software routines that directly manipulate the contents of the CDD /Plus dictionary system. CDD/Plus software clients, like CDO or RDO, call dictionary entry points to request specific dictionary operations. The Call Interface performs these operations, then returns control to the client.

CDD$COMPATIBILITY

A system logical name for the file specification of the compatibility dictionary. The translation of CDD$COMPATIBILITY must include a device and directory. This definition should never be changed.

CDDL

The CDD Data Definition Language Utility, the CDD/Plus utility that lets you insert record definitions into a DMU dictionary (CDD). You create the data descriptions in a CDDL source file that you compile with the CDDL compiler. You can also use CDDL to replace existing record definitions in a DMU dictionary.

See also DMU dictionary

CDDV

The CDD Verify/Fix Utility, the CDD/Plus utility that detects damaged DMU dictionary files and repairs them. CDDV also compresses the data in DMU dictionary files for more efficient storage.

See also DMU dictionary

CDO

The Common Dictionary Operator utility, the CDD/Plus user interface that lets you create and manage data definitions in CDO dictionaries, and read definitions from DMU dictionaries.

See also DMU dictionary

CDO dictionary

A dictionary created in a format that enables it to store not only definitions but information about how the definitions are related. A CDO dictionary is created on your system during CDD/Plus installation. You can create additional CDO dictionaries with the CDO utility.

See also DMU dictionary

client

A language processor, database management system, utility, or program that requests dictionary operations from CDD/Plus by using the call interface. For example, CDO is a client of CDD/Plus because CDO uses the call interface to read and write objects in the dictionary. Programs you write using the call interface are also dictionary clients.

command file

An RMS file of commands you can execute with the at-sign (@) operator. You can save useful sequences of CDO commands in a file and execute the file from the CDO prompt. The default file extension for files containing CDO commands is .CDO.

compatibility dictionary

A special CDO dictionary you use to coordinate your CDO-format definitions with DMU-format definitions. Each system can have only one compatibility dictionary.

See also CDD$COMPATIBILITY

computed-by clause

See virtual field and virtual record

conditional expression

An expression comprising value expressions, relational operators, and logical operators. The value of a conditional expression is either TRUE, FALSE, or MISSING.

data aggregate

A collection of data elements describing a record, Rdb/VMS relation, or a structure field.

data element

A CDD/Plus field contained in a CDO dictionary.

database

A collection of related information stored in one or more files. A database organizes related information to promote fast storage and retrieval.

datatype

An attribute assigned to a field definition determining the kind of data the field can contain.

DEC multi-national character set (DMCS)

A computer character set and collating sequence. Each character in the DMCS corresponds to a unique 8-bit byte.

default directory

When you use DCL, default directory refers to the VMS directory in which your process is currently operating. When you use CDO, default directory refers to the dictionary directory in which you are currently operating.

See also anchor, dictionary directory

dictionary client

Any program that uses the CDD/Plus call interface.

See also language processor, client

dictionary definition

An instance of an entity, a relationship, or an attribute protocol.

See also entity, instance, protocol

dictionary directory

A place in the dictionary hierarchy under which you store logically related dictionary objects or other dictionary directories. Dictionary directories are like VMS directories; they let you organize dictionary objects just as VMS directories let you organize files.

See also anchor

directory entry

The portion of a CDO data definition that allows you to refer to it by path name. When you create an object, you provide both a name and a description of it. CDD/Plus inserts the name in the directory system; it records the name in a structure called a directory entry. In contrast, CDD/Plus inserts the actual description of the object in the dictionary system in a structure called a dictionary entry.

DMU

The Dictionary Management Utilty, the CDD/Plus utility that lets you create and maintain the dictionary directory hierarchy and associated access control and history lists for non-CDO dictionaries.

See also CDDL, CDO dictionary

DMU dictionary

A dictionary containing data definitions that can be written to and manipulated by DMU and CDDL. All dictionaries created prior to CDD/Plus V4.0 are DMU dictionaries. If you do not already have one, a DMU dictionary is created during CDD/Plus V4.0 installation. You must use CDDL to insert record definitions into DMU dictionaries, because DMU dictionaries and CDO dictionaries have different storage formats.

See also CDO dictionary

edit string

A character or string of characters controlling how a dictionary client displays data in a field.

element

A relationship or an entity definition. An attribute is not an element.

entity

Any collection of data in an application system that can be treated as a unit. Examples of entities are the personnel database, the record describing employee JONES, or JONES' badge number.

See also entity type, entity protocol

entity protocol

A set of rules for defining and interpreting dictionary entities. Examples of entity protocols are CDD$DATA_ELEMENT and CDD$DATA_AGGREGATE.

See also entity and entity type

entity type

A description of a unit of data or processing in an application system. Examples of entity types are the EMPLOYEE_REC data aggregate and the ZIP_CODE data element.

See also entity protocol, entity

entry point

A software routine dictionary clients call to perform a specific operation on the contents of the CDD/Plus dictionary system files.

See also client

expression

An expression consists of operands separated by arithmetic, string, relational, or logical operators.

See also missing value, initial value, virtual field

fetch stream

A set of dictionary elements you process consecutively through a program that calls the CDD/Plus call interface.

field

The smallest meaningful unit of data in a business application. Examples of fields are employee MACWIRE's badge number, address, or zip code.

field description

See data element

generic entity

Any entity type in the CDD/Plus dictionary system. You can refer to any entity definition as a generic entity, regardless of the protocol under which it was originally created.

initial value

A value dictionary clients insert into a field when allocating storage for it.

initialization file

A file of CDO commands that CDD/Plus executes each time you enter the CDO utility. You identify the initialization file with the logical name CDO$INIT.

See also command file

instance

A specific occurrence of a protocol in the dictionary. For example, the field LAST_NAME is an instance of the field protocol.

integrity

The correctness of information in a database. There are three general types of integrity control:

- Integrity constraints ensure the database information remains correct when users try to modify it incorrectly.
- Concurrency control lets only one user at a time update a file while allowing many users simultaneous access to the database.
- Recovery restores a database to the state it was in before a system failure.

journaling

The process of recording onto a recoverable resource information about operations on a database. The type of information recorded depends on the type of journal being created.

language processor

A compiler or interpreter that uses the call interface to read or write definitions in the dictionary.

See also client

logical operator

An operator that links two conditional expressions into another, more complex conditional expression. CDD/Plus logical operators are AND, OR, and NOT.

See also conditional expression

major version

A number that designates the particular variation of on-disk structure (database schema) that CDD/Plus uses for dictionary storage. If the major version changes, the new version will be incompatible with existing dictionaries. Users must convert existing dictionaries in order to use them with the new version.

See also minor version

member

The subordinate participant in a relationship. For example, a field is subordinate to a record. Thus, when CDD/Plus stores a relationship between a field description (data element) and record description (data aggregate), the field description is the member of the relationship.

See also owner

message

An encoded report of the status of an element of the CDD/Plus dictionary system. When an element in the dictionary changes, CDD/Plus sends a message to related elements; the message contains a warning indicating that a portion of the dictionary has recently changed and that the change may affect the element receiving the message.

minor version

A number that designates the particular variation of dictionary software used by CDD/Plus. If the minor version changes, the minor version number of the software must be equal to or less than the minor version on the disk in order to be compatible with existing dictionaries. A minor version results when you change a protocol.

See also major version

missing value

An expression clients evaluate at run time when a field has no value. CDD/Plus clients use the result of the missing value expression in reports and screen displays.

object

A data definition in the CDD/Plus dictionary system.

See also element

overlay

One of a set of logical views of a single physical portion of a record description.

owner

The principal participant in a relationship. For example, a field is subordinate to a record. Thus, when CDD/Plus stores a relationship between a field description (data element) and record description (data aggregate), the record description is the owner of the relationship.

See also member

pathname

A string that identifies an object or directory in the CDD/Plus dictionary system. An object or directory's path name is the concatenation of the given names of all its ancestors in the hierarchy, starting with the root directory, and ending with the object's given name. (Separate each given name in the path name with a period [.].) For objects stored in CDO dictionaries, the root directory is a VMS anchor specification. For objects stored in DMU dictionaries, the root directory is CDD$TOP.

privilege

The ability to access a database, data definition, or other resource for a certain purpose.

See also security and access control list

processing name

The name of an object in the CDD/Plus dictionary system. In most cases, the processing name is the same as the given name of the object.

See also pathname and directory entry

protocol

A set of rules for storing definitions in the dictionary.

See also entity protocol, relationship protocol, attribute protocol

protocol version

An internal number CDD/Plus uses to keep track of changes to the strict conventions clients follow when storing definitions in the dictionary.

See also buffer version

query header

A string that calling programs use to label a field in screen displays and reports.

query name

A string dictionary clients use as an alternate name for a field. A query name is used merely for convenience; typically it is an abbreviation of the field's processing name.

RDO

The Relational Data Operator Utility, an interactive utility for maintaining databases, creating and modifying database elements, and storing and manipulating data.

record

A body of information within a database.

record definition

A description of a record's structure that includes its name and the fields it includes.

record selection expression

A phrase that defines specific conditions that individual records must meet before a language processor or database system includes them in a record stream.

relational operator

An operator that compares two value expressions, producing a value of TRUE or FALSE.

relationship

An element that logically links two other elements. For example, the CDD$DATA_AGGREGATE_CONTAINS relationship links a record definition with a field definition. Each relationship links only one owner and one member.

See also owner, member

relationship protocol

A set of rules for defining and interpreting relationships. An example of a relationship protocol is CDD$DATA_VALUE_DEPENDS_ON.

See also entity, entity type

security

The protection of information stored in a database or dictionary against unauthorized use.

See also privilege, access control list

structure field

A field comprising a number of subordinate fields. CDD/Plus clients typically use data aggregates as record or relation definitions. Clients can, however, use a data aggregate as a complex portion of a larger record or relation description. In such a case, the data aggregate describes a structure field.

See also data aggregate

subscript

An integer indicating the position of an element in an array.

See also array

tag

A longword value the CDD/Plus call interface and its clients use to identify attributes, relationships, and entity in buffers.

transaction

An exchange of information between a database user and a database. The operations in a transaction are treated as a unit; either all of them are completed at once or none of them is completed. A transaction is a complete process from input to output, regardless of the number of events in between. For DIGITAL database management products, a transaction groups a series of statements that perform an operation on a database.

validation

A rule or constraint that ensures that CDD/Plus users and clients insert only valid values in the dictionary system.

value expression

A field name or absolute value, or a string of symbols that evaluate to a field name or absolute value.

variant

See overlay

version

A number CDD/Plus uses to differentiate among various instances of a given dictionary definition. When you first create a definition, CDD/Plus assigns it a version number of 1. CDD/Plus assigns successively higher version numbers to subsequent versions of the dictionary definition.

See also protocol version, buffer version, major version, minor version

virtual field

A field whose value depends entirely on other fields within the record; a virtual field requires no physical storage. Users define virtual fields with a computed-by expression, which describes the formula CDD/Plus clients use to generate the value of the virtual field at run time.

virtual record

A record comprising only fields stored in other records; a virtual record requires no physical storage. Users define virtual records by naming the individual field to be included from other records.

Index

This index uses the following symbols:

t	Entry occurrence in a table
f	Entry occurrence in a figure
e	Entry occurrence in an example

A

Access control
  compatibility, 2-11
  conflicting rights, 5-16
  deleting ACE, 5-15
  editor, 5-5
  protection, 5-1, 5-3t
  rights, 5-6t
Access routes
  dictionary definition, 2-4, 2-7
Account
  proxy, 5-16
ACL
  See Access control lists
AGGREGATE CONTAINS relationship, 4-10, 6-3
/ALL qualifier
  SHOW command (CDO), 4-3
  SHOW FIELD command (CDO), 3-41
  SHOW USES command (CDO), 6-2
  VERIFY command (CDO), 6-12t
ALPHABETIC data type
  DMU compatibility, 2-8t
Analysis
  of usage, 6-1
  what-if, 1-6, 6-2
Anchor
  changing, 6-9
  creating VMS directory for, 3-7
  default, 3-33
  default compatibility, 2-2
  fully translated, 1-7
  initial, 3-32
  name, 1-7
  protection, 3-7
Application
  recompiling, 6-7
ARRAY attribute, 4-4
Array bounds
  DMU compatibility, 2-8t
ASSIGN command (DCL), 3-34
AT command (@ sign), 4-1

@ (At sign)
  See At sign command (@)
At sign command (@), 3-30
ATTACH command (CDO), 3-30t, 7-4
ATTACH command (DCL), 3-6
ATTACH statement (RDO), 7-4
Attribute clauses
  in RMS database definitions, 8-4
Attributes
  ARRAY, 4-4
  AUDIT, 4-4
  BASED ON, 4-4
  COMPUTED BY, 4-4
  DATATYPE, 4-4
  DESCRIPTION, 4-4
  EDIT_STRING, 4-4
  field, 4-4t
  INITIAL_VALUE, 4-4
  JUSTIFIED, 4-4
  MISSING_VALUE, 4-4
  NAME FOR, 4-4
  OCCURS, 4-4
  QUERY_HEADER, 4-4
  QUERY_NAME, 4-4
  record, 4-9
  system-specified, 3-41
  user-specified, 3-41
  VALID_IF, 4-4
AUDIT attribute, 4-4, 4-5, 4-23, 4-24, 5-10
  DEFINE command (CDO), 4-2
/AUDIT qualifier, 2-20
  for language compilation, 4-3
  SHOW command (CDO), 4-3
Audit trails
  See History lists

B

Backing up
  dictionaries, 6-8
  Rdb/VMS databases, 7-35
BACKUP
  Utility, 6-8
BACKUP command (DCL), 6-8 to 6-9
BASED ON
  attribute, 4-4, 4-6
    DMU compatibility, 2-8t
BASED ON (cont'd.)
  relationship, 4-10
BASIC support, 7-23
Bottom-up definition, 4-7
/BRIEF qualifier
  SHOW USES command (CDO), 6-2
Browse
  in editor, 3-28
  with SHOW commands, 6-1, 6-2

C

CDD$COMPATIBILITY, 2-1, 2-2
  logical name, 2-3
CDD$DATABASE, 8-6
CDD$DEFAULT, 2-7, 3-8, 3-32 to 3-34, 3-35
CDD$ORPHANS, 6-12
CDD$RMS_DATABASE, 8-2
CDD$TOP, 2-2, 2-7
CDDL COPY clause
  DMU compatibility, 2-8t
CDO
  commands, 3-30t
  editor, 3-11 to 3-29
  file extension, 3-44, 4-1
  format definitions, 2-2 to 2-7
  interface, 1-4
  invoking, 3-5
  prompt, 3-5
CDO$INIT.CDO, 3-37
CDO commands
  ATTACH, 7-4
  CHANGE, 4-24, 4-26 to 4-32
  CHANGE DATABASE, 8-11
  CHANGE GENERIC, 4-27
  CHANGE PROTECTION, 5-14
  CLEAR MESSAGES, 6-7, 7-29
  DEFINE, 4-1 to 4-9, 4-25 to 4-26
  DEFINE DATABASE, 8-5
  DEFINE DICTIONARY, 3-8
  DEFINE DIRECTORY, 3-9
  DEFINE GENERIC, 4-1
  DEFINE PROTECTION, 5-10
  DEFINE RECORD, 4-7
  DEFINE RMS_DATABASE, 8-3
  DELETE DICTIONARY, 4-34
  DELETE GENERIC, 7-40

CDO commands (cont'd.)
  DELETE PROTECTION, 5-15
  DIRECTORY, 3-38, 7-10
  EDIT, 4-1
  EXIT, 3-6
  EXTRACT, 3-30t
  HELP, 3-30t
  MOVE DICTIONARY, 4-24
  PURGE, 3-30t, 4-35
  SET, 3-30t
  SET DEFAULT, 3-33
  SET OUTPUT, 3-32
  SET VERIFY, 3-45
  SHOW, 3-30t, 4-3, 4-27
  SHOW DATABASE, 3-42, 7-14, 8-9
  SHOW FIELD, 3-41, 4-10, 7-17
  SHOW MESSAGES, 6-2, 6-6
  SHOW PRIVILEGES, 4-27, 5-9
  SHOW PROTECTION, 4-27, 5-9
  SHOW PROTOCOL, 4-26
  SHOW RECORD, 2-20, 3-42, 4-10
  SHOW RMS_DATABASE, 8-10
  SHOW UNUSED, 6-2, 6-14
  SHOW USED_BY, 6-2
  SHOW USES, 6-2
  SHOW VERSION, 3-46
  SHOW WHAT_IF, 6-2, 6-5
  SPAWN, 3-6, 3-30t, 7-4
  VERIFY, 3-30t
CDO dictionary, 1-1
  See also Compatibility dictionary
  See also Dictionary
  See also User-created dictionary
Change
  auditing, 6-4
  definitions, 4-24 to 4-30
  messages, 6-6
  new versions, 4-25
  recording, 6-2 to 6-7
  relationships, 4-30 to 4-32
  to Rdb/VMS database, 7-25 to 7-34
  tracking, 6-1 to 6-8
CHANGE command (CDO), 3-30t, 4-24, 4-26 to 4-32
  effect on Rdb/VMS database, 7-28 to 7-29
CHANGE DATABASE command (CDO), 8-11e
CHANGE GENERIC command (CDO), 4-27
CHANGE privilege, 4-27, 5-6
CHANGE PROTECTION command (CDO), 5-14 to 5-15
CLEAR MESSAGES command (CDO), 3-30t, 6-7, 7-29
Command
  editing, 3-1
  executing procedures in environment, 3-44
  output to a file, 3-32
  procedure, 3-44, 4-1
  recalling in CDO environment, 3-1
  summary, 3-30 to 3-37
  usage tracking, 6-2
Comments in definitions, 4-2 to 4-3
Common Dictionary Operator
  See CDO
Compatibility
  access rights, 2-6t
  DMU record structures, 4-8
  protection, 2-13, 5-8
  Rdb/VMS, 7-4
  upward, 3-47
Compatibility dictionary, 1-1, 2-1, 2-10, 3-4f, 3-33
  See also CDD$COMPATIBILITY
  access by VAX products, 2-4f
  anchor, 2-3
  creating directories, 3-9
  default, 3-5, 3-8, 3-33
  definition of, 2-1
  initial default, 3-33
  in search list, 3-34
  locating, 2-3
  logical name, 2-1
  purpose, 2-1 to 2-2, 3-7
  tasks, 2-6t
Compiled module
  BASIC, 7-23
  relationship with CDD/Plus entity, 8-8
Compiling definitions, 4-1
Compressing definitions, 6-14
COMPUTED BY attribute, 4-4

COMPUTED BY attribute (cont'd.)
  DMU compatibility, 2-8t
Continuation prompt, 3-31
Control C
  See CTRL/C
CONTROL privilege, 5-7
Conversion from DMU format, 2-14
CONVERT command (CDO),
    2-14 to 2-20,3-30t
Convert Rdb/VMS database, 7-36
COPY command (CDO), 3-30t,
    4-22 to 4-24
CREATE command (DCL), 3-7
Creating dictionaries
  See DEFINE DICTIONARY (CDO)

D

Database
  See Rdb/VMS database
  See RMS Database
Database relation, 7-12
Database view, 7-22
Data definition
  See Definition
Data Definition Language Utility
  See CDDL
Data Source Definition, 8-7
Data type, 4-12 to 4-22
  complex numbers, 4-14t
  decimal string, 4-15t
  fixed-point, 4-13t
  floating-point, 4-14t
  language support, 4-18 to 4-22
  string, 4-12
DATATYPE attribute, 4-4
Data values
  storage of, 1-3
DCL commands
  ASSIGN, 3-34
  ATTACH, 3-6
  BACKUP, 6-8 to 6-9
  CREATE, 3-7
  DEFINE, 3-34
  DICTIONARY, 3-5
  RALLY CREATE, 8-7
  SET PROTECTION, 5-5
DECnet account, 5-16
Default
  anchor, 2-2
  dictionary, 2-1, 7-7
  directory, 3-8, 3-32
  privileges, 5-7
  protection, 5-7
DEFINE command
  effect on Rdb/VMS database,
    7-29 to 7-31
DEFINE command (CDO), 3-30t,
    4-1 to 4-9, 4-25 to 4-26,
    4-30 to 4-32
  AUDIT attribute, 4-2
DEFINE command (DCL), 3-34
DEFINE DATABASE command (CDO),
    8-5, 8-6e
DEFINE DATABASE statement (RDO)
  DICTIONARY IS REQUIRED clause,
    7-14
DEFINE DICTIONARY (CDO), 3-8
DEFINE DIRECTORY (CDO), 3-9
DEFINE FIELD command (CDO),
    4-4 to 4-6
DEFINE GENERIC command (CDO), 4-1
DEFINE KEY command (CDO),
    3-36 to 3-37
DEFINE privilege, 4-26, 5-6
DEFINE PROTECTION command (CDO),
    5-10
DEFINE RECORD command (CDO), 4-7
DEFINE RMS_DATABASE command
    (CDO), 8-3, 8-5e
Definition
  attributes, 4-4t
  bottom-up, 4-7
  CDO editor sample, 3-15
  changing, 4-1, 4-26
  changing original, 4-26
  commenting, 4-2 to 4-3
  compiling, 3-10 to 3-29, 4-1 to 4-9
  compressing, 6-14
  copies, 6-7
  copying, 4-22
  copy to language, 1-9
  deleting, 2-7, 6-14, 7-34
  displaying, 3-43
  distributed, 7-16

Definition (cont'd.)
 DMU, 2-7, 2-10, 2-11, 2-14
 editing, 4-1
 entering, 4-1
 entity, 3-11 to 3-28
 field, 1-5, 4-4
 implicit, 4-8
 listing in editor, 3-29
 name, 1-8
 Rdb/VMS database, 7-3 to 7-40
 record, 1-5, 4-7
 RMS database, 8-2
 size, 3-39
 storage formats, 1-2, 2-1
 top-down, 4-8
 version, 4-25, 4-35
DELETE command, 4-34 to 4-35
DELETE command (CDO), 3-30t
DELETE DATABASE statement (RDO), 7-38
DELETE DICTIONARY command (CDO), 4-34
DELETE GENERIC command (CDO), 7-40
DELETE PATHNAME statement (RDO), 7-36
DELETE privilege, 4-33, 5-6, 6-14
DELETE PROTECTION command (CDO), 5-15
DELETE statement (RDO), 7-34
/DESCENDANTS qualifier
 PURGE command (CDO), 4-35
DESCRIPTION attribute, 4-2, 4-4, 4-24
Diagnostics in editor, 3-28
Dictionary
 active, 1-3
 backing up, 6-8
 CDO, 1-1
 changing anchor, 6-13
 compatibility for Rdb/VMS, 7-4
 copying, 4-24
 creating, 3-7
 default directory for, 3-7
 deleting, 4-34, 6-15
 design, 3-2
 directory, 3-9
 distributed, 1-5, 3-5, 6-15
 DMU, 1-1, 2-7, 2-10, 2-11, 2-14
Dictionary (cont'd.)
 DMU protection, 5-8
 enforced usage, 1-3
 hierarchy, 3-2
 integrity, 1-7
 listing contents, 3-38 to 3-43
 location, 6-12
 locked, 6-12
 logical, 1-5, 3-5, 3-34
 moving, 4-24, 6-12, 6-13
 multiple, 6-9
 network, 1-5, 3-5
 origin, 1-7
 passive, 1-3
 performance, 6-14
 planning, 3-2
 protection, 1-7, 5-1 to 5-17
 recovery, 6-8
 remote, 6-12
 restoring, 6-9
 saving, 6-8
 structure, 3-2
 support, 1-2
 usage, 6-1 to 6-8
 User-created, 1-1
 version, 3-46
 with Rdb/VMS, 7-1 to 7-40
DICTIONARY command (CDO)
 OPERATOR keyword, 3-5
DICTIONARY command (DCL), 3-5
DICTIONARY IS REQUIRED clause
 DEFINE DATABASE statement (RDO), 7-14
Dictionary Management Utility
 See DMU
Dictionary usage
 benefits, 7-3 to 7-4
Dictionary Verify/Fix Utility
 See CDDV
Directory
 contents, 3-38 to 3-43
 default, 3-32
 defining, 3-9, 4-6
 definition, 1-5
 initial, 3-32
DIRECTORY command (CDO), 3-30t, 3-38, 6-14, 7-10

DIRECTORY command (CDO) (cont'd.)
/FULL qualifier, 3-39
/SINCE qualifier, 3-40
/TYPE qualifier, 3-39
Directory name, 2-17, 2-18, 7-8
/DIRECTORY qualifier
VERIFY command (CDO), 6-12t
Distributed dictionary access, 1-5, 3-5, 4-12, 4-32, 6-12
for Rdb/VMS, 7-16
protection, 5-9, 5-16
DMU, 2-7
compatibility, 1-2, 2-1 to 2-20
converting records, 2-14
definition, 2-7
format definitions, 2-2 to 2-7
history list, 2-6
protection, 2-11, 5-8
reading from CDO, 2-10, 2-11e
support, 1-2, 2-1 to 2-20
DMU dictionary, 1-1
See also Appendix A
Documentation
AUDIT attribute, 4-2
DESCRIPTION attribute, 4-2
for changes, 4-24

E

EDIT command (CDO), 3-30t, 4-1
Editor
See Access control list editor
creating new versions, 3-29
exit, 3-29
invoking, 3-15
key definitions, 3-11f
list of current definitions, 3-29
sample definitions, 3-15
screen mode, 3-11
SHIFT key, 3-11
validation, 3-28
EDIT_STRING attribute, 4-4
ENTER command (CDO), 3-30t
FROM DATABASE, 7-37 to 7-38
FROM RECORD, 2-18 to 2-20
Entity
defining, 4-1 to 4-12
has messages, 6-5
Entity (cont'd.)
listing, 3-38
processing name for, 2-18
user-defined, 4-1
Environment
characteristics, 3-32 to 3-38
entering, 3-5
exiting, 3-6
prompt, 3-5
Error
checking, 3-31
syntax, 4-1
Exclamation point, 3-44
EXIT command (CDO), 3-6, 3-30t
EXPORT statement (RDO), 7-35
Expressions
valid for DMU, 2-8t
/EXTERNAL_REFERENCE qualifier
VERIFY command (CDO), 6-9, 6-12t
EXTRACT command (CDO), 3-30t, 3-43 to 3-44

F

Field
attributes, 4-4t
changing, 4-28
defining, 4-4
defining with editor, 3-11
definition, 1-5
implicit definition, 4-7, 4-8
including in record definition, 4-7
order in records, 4-29
shareable, 7-8 to 7-12
variants, 4-7
File access control, 5-1
File allocation
specifying in RMS applications, 8-4
File definition
for RMS database, 8-2
File extension
default, 3-44
FILENAME keyword
INVOKE DATABASE statement (RDO), 7-14
File specification
Rdb/VMS database, 7-39

/FIX qualifier
  VERIFY command (CDO), 6-12t
Foreign command, 3-5
Formats
  for dictionary definition storage, 1-2, 2-1, 2-2
/FULL qualifier
  DIRECTORY command (CDO), 3-39, 6-14
  SHOW USED_BY command (CDO), 7-11
  SHOW USES command (CDO), 6-2, 7-30e
Fully qualified name, 1-8, 3-34, 6-13, 7-16, 8-8, 8-9e
Fully translated anchor, 1-7
Future of dictionary, 1-3

G

Generic entity definition, 4-1

H

Help
  on CDO commands, 3-10
  within CDO editor, 3-15
HELP command (CDO), 3-10, 3-30t
Hierarchy of dictionary, 3-2
History list, 2-6, 2-20, 4-2 to 4-3, 4-23, 4-24, 5-10

I

Identifiers, 5-1
IMPORT statement (RDO), 7-35, 7-38
%INCLUDE %FROM %CDD directive (BASIC), 7-23
Index
  for Rdb/VMS database, 7-22
Initialization file, 3-37
INITIAL_VALUE attribute, 4-4, 4-5
INTEGRATE message (CDO), 6-5
INTEGRATE statement (RDO), 6-6, 7-25 to 7-33, 7-36
Interrupt, 6-8
INVOKE DATABASE statement (RDO), 7-14
INVOKE DATABASE statement (RDO) (cont'd.)
  FILENAME keyword, 7-14
  PATHNAME keyword, 7-14

J

Journaling, 6-8
JUSTIFIED attribute, 4-4

K

/KEEP qualifier
  PURGE command (CDO), 4-35
Key definition, 3-36

L

Layered products, 1-2
Line editing, 3-1
Location
  changing, 6-12
/LOCATION qualifier
  VERIFY command (CDO), 6-12t
Locking mechanisms, 5-17
Log file, 3-32
Logical dictionary, 1-5, 3-2, 3-5, 6-9
  structure, 3-2
Logical name, 1-8, 3-34, 4-4
/LOG qualifier
  VERIFY command (CDO), 6-12t

M

Major version of dictionary, 3-47
Message
  about change, 4-24, 4-26 to 4-27, 4-31, 4-32, 6-2 to 6-7
  about new version, 4-25 to 4-26, 4-32
  attached to CDO definitions, 6-5
  clearing, 6-7, 7-26
  for Rdb/VMS database, 7-28
  reading, 6-6, 6-7
  reading from language, 6-7
Metadata, 1-1
Migration from DMU format, 2-14
Minor version of dictionary, 3-47
MISSING_VALUE attribute, 2-8t, 4-4

MOVE DICTIONARY command (CDO),
    4-24, 6-13
Moving RMS databases, 8-11
Multiple versions of dictionary objects

N

Name
  anchor, 1-7, 6-9
  compatibility, 2-2
  default dictionary, 1-7, 1-8
  definition, 1-7
  directory, 2-17, 7-8
  duplicate, 1-8
  for CDO definitions, 2-2
  for DMU definitions, 2-2
  path, 1-7
  processing, 2-17, 3-42, 7-8
  qualified, 1-7, 1-8
  unique, 1-8
  version, 1-8
NAME FOR attribute, 4-4
Name for definition, 1-8, 4-4
Network
  access, 1-5, 3-5
  failure, 6-8, 6-12
  performance, 6-15
  protection, 5-16
  relationships, 4-12, 4-32
Node
  distributed dictionary, 1-5, 3-5, 6-15
  distributed relationships, 4-12, 4-32
/NODESCENDANTS qualifier
  PURGE command (CDO), 4-35
Notice
  See Message

O

Object
  See Definition
Object file
  BASIC, 7-23
OCCURS attribute, 4-4
OCCURS expressions
  DMU compatibility, 2-8t
OPERATOR keyword
  DICTIONARY command (CDO), 3-5
/ORPHANS qualifier
  VERIFY command (CDO), 6-12t
Output to a file, 3-32
Overlay tags
  DMU compatibility, 2-8t

P

Path name
  See also Directory name
  CDD$TOP, 2-7
  deleting, 7-39
  qualified, 1-7, 1-8
  relative, 7-39
PATHNAME keyword
  INVOKE DATABASE statement
      (RDO), 7-14
Performance, 6-14 to 6-15
Period
  terminating commands, 3-30
Physical dictionary, 3-2
  See also Dictionary
Pieces tracking, 1-6, 6-1 to 6-8, 7-4, 8-8
  for Rdb/VMS database, 7-27 to 7-34
Privilege
  CHANGE, 4-27, 5-6
  DEFINE, 4-26, 5-6
  DELETE, 4-33, 5-6, 6-14
  SHOW, 5-6
Privileges
  See Access privileges
Processing name, 2-17, 3-42, 7-8
Process quotas, 3-7
Program
  recompiling, 6-7
Programming languages
  data types supported, 4-19
  dictionary support, 8-8
Prompt
  continuation, 3-31
  in environment, 3-5
Protection, 5-1 to 5-17
  See Access control lists
  access control, 5-2
  changing, 5-14
  changing definitions, 4-27
  compatibility, 2-13

Protection (cont'd.)
  compatibility with DMU, 5-8
  confirming, 5-6
  conflict, 5-16
  conversion, 2-12t, 2-20
  default, 4-26, 4-27, 4-33, 5-7, 6-14
  definition, 4-26
  delete definitions, 4-33
  deleting, 5-15, 6-14
  for Rdb/VMS definitions, 7-17
  protocol, 5-6
  rights, 5-3 to 5-5
  setting, 5-10
  VMS, 5-1 to 5-5
Protocol
  protection, 5-6
Proxy account, 5-16
Punctuation for syntax, 3-30
PURGE command (CDO), 3-30t, 4-35
  /DESCENDANTS qualifier, 4-35
  /KEEP qualifier, 4-35
  /NODESCENDANTS qualifier, 4-35

Q

QUERY_HEADER attribute, 4-4
QUERY_NAME attribute, 4-4
Quotas
  process, 3-7

R

RALLY CREATE command (DCL), 8-7
RALLY DSD
  See Data Source Definition
RALLY support, 8-7
Rdb/VMS
  displaying relation from CDO, 7-12
  displaying view from CDO, 7-22
  locking mechanisms, 5-17
Rdb/VMS database, 7-1 to 7-40
  changing definitions, 7-25 to 7-34
  clearing messages, 7-26
  compatibility, 7-4
  converting, 7-36
  defining, 7-5, 7-13
  defining relations, 7-20
  deleting, 7-38
Rdb/VMS database (cont'd.)
  deleting shared definitions, 7-34
  displaying definitions from CDO, 7-11
  embedded statements, 7-23
  internal definitions, 7-22
  internal names, 7-18 to 7-19
  message chain, 7-28
  multiple, 7-2f
  reading messages, 6-7
  requiring dictionary, 7-14
  restoring, 7-35
  showing internal entities, 7-11
  showing internal fields, 7-17
  support, 1-9, 6-7
RDML, 7-23
RDO, 6-7
RDO statements
  ATTACH, 7-4
  DEFINE DATABASE, 7-13
  DEFINE FIELD, 7-15
  DEFINE RELATION, 7-20
  DELETE, 7-34
  DELETE DATABASE, 7-38
  DELETE PATHNAME, 7-36
  EXPORT, 7-35
  IMPORT, 7-35, 7-38
  INTEGRATE, 6-6, 7-25 to 7-33, 7-36
  INVOKE DATABASE, 7-14, 7-15
  ROLLBACK, 7-4, 7-16, 7-26
  SET DICTIONARY, 7-7
  SPAWN, 7-4
  START_TRANSACTION, 7-26
/REBUILD_DIRECTORY qualifier
  VERIFY command (CDO), 6-11, 6-12t
Recall previous commands, 3-1
Record
  adding new fields, 4-29
  attributes, 4-9
  changing, 4-29
  defining, 4-7
  defining with editor, 3-11
  nested, 4-7
  shareable, 7-8 to 7-12
  structure, 4-8
  unstructured, 2-20
  variant fields, 4-7
Record definition
  for RMS database, 8-2

Relation
  displaying, 7-12
Relationship
  AGGREGATE CONTAINS, 4-10
  BASED ON, 4-6, 4-10
  changing, 4-30 to 4-32
  creating, 4-9 to 4-12
  defining, 1-6
Remote access, 3-7, 4-12, 4-32, 5-16, 6-12
%REPORT directive (BASIC), 7-24
Repository
  See Dictionary
Requiring dictionary in DEFINE DATABASE statement (RDO), 7-14
Resource limits
  for processes, 3-7
Restrictions
  DMU compatibility, 2-8t
Rights
  See Protection
RMS database, 8-1 to 8-12
  See also RMS files
  creating database definition, 8-3
  creating physical database file, 8-4
  defining, 8-2
  deleting, 8-11
  moving, 8-11
  pieces tracking, 8-8
  showing, 8-8
RMS database definition, 8-2
  deleting, 8-11
  showing, 8-8
RMS files, 8-1 to 8-12
  access from RALLY, 8-7
  CDO commands related to, 8-2
  where to create, 8-6
RMU/BACKUP command, 7-35
RMU/RESTORE command, 7-35
Roll back of Rdb/VMS database, 7-26
ROLLBACK statement (RDO), 7-4

S

Search list, 3-34 to 3-36, 6-10
Security
  See Access control lists
SET command (CDO), 3-30t, 3-32 to 3-36
SET DEFAULT command (CDO), 3-33
SET DICTIONARY statement (RDO), 7-7
SET OUTPUT command (CDO), 3-32
SET PROTECTION command (DCL), 5-5
SET VERIFY command (CDO), 3-45
SHOW command (CDO), 3-30t, 4-3, 4-27, 6-2t
  qualifiers, 6-2
SHOW DATABASE command (CDO), 3-42, 7-14, 8-9e
SHOW FIELD command
  FROM DATABASE clause, 7-18
SHOW FIELD command (CDO), 3-41, 4-10
  FROM DATABASE clause, 7-12e, 7-17
SHOW MESSAGES command (CDO), 6-2, 6-6, 7-30e
SHOW privilege, 5-6, 5-7
SHOW PRIVILEGES command (CDO), 4-27, 5-9
SHOW PROTECTION command (CDO), 4-27, 5-9
SHOW PROTOCOL command (CDO), 4-26
SHOW RECORD command (CDO), 2-11, 2-20, 3-42, 4-10
  /AUDIT qualifier, 2-20
SHOW RMS_DATABASE command (CDO), 8-10e
SHOW UNUSED command (CDO), 6-2, 6-14
SHOW USED_BY command (CDO), 4-10e, 6-2, 7-11e, 7-38e
SHOW USES command (CDO), 4-11e, 6-2, 7-27e, 7-30e, 7-34e
SHOW VERSION command (CDO), 3-46
SHOW WHAT_IF command (CDO), 6-2, 6-5, 7-27e
SIGNED NUMERIC data type
  See ZONED NUMERIC data type
SIGNED NUMERIC LEFT OVERPUNCHED data type
  See LEFT OVERPUNCHED NUMERIC data type

SIGNED NUMERIC LEFT SEPARATE data type
  See LEFT SEPARATE NUMERIC data type
SIGNED NUMERIC RIGHT OVERPUNCHED data type
  See RIGHT OVERPUNCHED NUMERIC data type
SIGNED NUMERIC RIGHT SEPARATE data type
  See RIGHT SEPARATE NUMERIC data type
Simultaneous updates, 5-17
/SINCE qualifier
  DIRECTORY command (CDO), 3-40
Size of definition, 6-14
Source
  isolating, 6-2
SPAWN command (CDO), 3-6, 3-30t, 7-4
SPAWN statement (RDO), 7-4
START_TRANSACTION statement (RDO), 7-26
Storage of definitions, 6-14
  formats, 1-2, 2-1
Structure
  dictionary, 3-2
  record, 4-8
Subprocess, 7-4
Support for dictionary
  BASIC, 7-23
  languages, 1-2
  programming languages, 1-7, 4-19, 7-11, 7-23, 8-8
  Rdb/VMS, 1-9
  VAX Information Architecture, 1-2
  VIDA, 1-9
Syntax
  terminating commands, 3-30
Syntax errors, 3-31
SYS$OUTPUT, 3-32
System failure, 6-8
System-specified attributes, 3-41

T

Terminating commands, 3-30
Top-down definition, 4-8
Tracking, 6-1 to 6-8
Tracking (cont'd.)
  See also Change, tracking
Translation utility, 2-2
/TYPE qualifier
  DIRECTORY command (CDO), 3-39
  SHOW USED_BY command (CDO), 7-11
  SHOW USES command (CDO), 6-2

U

UIC-based protection, 5-1
Update locking, 5-17
Usage tracking
  See Pieces tracking
User-created dictionary, 1-1
User-specified attributes, 3-41

V

Validation in editor, 3-28
VALID_IF attribute, 4-4
VALID_IF expressions
  DMU compatibility, 2-8t
VARIANTS clause, 4-7 to 4-8, 4-30
VAXcluster
  distributed dictionary, 1-5, 3-5
VAX Information Architecture products, 1-3, 4-16
VERIFY command (CDO), 3-30t, 6-9 to 6-13
Version
  definition, 4-25, 4-35
  dictionary, 3-46
  major, 3-47
  minor, 3-47
  name, 1-8
Versions
VIDA with IDMS/R support, 1-9
View
  displaying, 7-22
  for Rdb/VMS database, 7-22

W

What-if analysis, 1-6, 6-1, 6-2
Wildcard, 3-38, 4-35
  restrictions, 3-39
  search list, 3-34

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local DIGITAL subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local DIGITAL subsidiary or approved distributor
Internal[1]	————	SDC Order Processing - WMO/E15 *or* Software Distribution Center Digital Equipment Corporation Westminster, Massachusetts 01473

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

VAX CDD/Plus User's Guide
AA–KL46A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

____

What I like best about this manual is ______________________________

____

What I like least about this manual is ______________________________

____

I found the following errors in this manual:
Page Description

______ ______________________________

______ ______________________________

______ ______________________________

Additional comments or suggestions to improve this manual:

____

____

____

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________ Dept. ____________

Company ______________________ Date ____________

Mailing Address ______________________________

______________________ Phone ____________

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear - Fold Here

Cut Along Dotted Line

Reader's Comments

VAX CDD/Plus User's Guide
AA–KL46A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

____

What I like best about this manual is ______________________________

____

What I like least about this manual is ______________________________

____

I found the following errors in this manual:
Page Description

_____ ______________________________

_____ ______________________________

_____ ______________________________

Additional comments or suggestions to improve this manual:

____

____

____

I am using **Version** _____ of the software this manual describes.

Name/Title ______________________ Dept. __________

Company ______________________ Date __________

Mailing Address ______________________________

______________________ Phone __________

- **Do Not Tear - Fold Here and Tape** --

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— **Do Not Tear - Fold Here** --

Reader's Survey

VAX CDD/Plus User's Guide
AA–KL46A–TE

1. How useful are the following methods for finding information in this manual?

	Most	Very	Moderately	Not Very	Not at All
Table of contents	☐	☐	☐	☐	☐
Divider pages (if applicable)	☐	☐	☐	☐	☐
Index (circle: book or master)	☐	☐	☐	☐	☐
Other (specify) ________________	☐	☐	☐	☐	☐

2. What feature do you most want to see improved in this manual? Why?

____

3. How helpful are these sources when you use the software this manual describes?

	Most	Very	Moderately	Not Very	Not at All
Handbook or user's guide	☐	☐	☐	☐	☐
Introduction or overview	☐	☐	☐	☐	☐
Reference manual	☐	☐	☐	☐	☐
Quick reference guide	☐	☐	☐	☐	☐
Online help	☐	☐	☐	☐	☐
Online tutorial (if available)	☐	☐	☐	☐	☐
Other: colleague, telephone support services (specify) ________________	☐	☐	☐	☐	☐

4. What business tasks are you using the software described by this manual to solve (for example: billing, funds transfer, report writing)?

____

5. Please estimate, if you can, how long the following VAX Information Architecture products have been used at your site:

VAX ACMS ________	VAX CDD/Plus ________	VAX DATATRIEVE ________
VAX Data Distributor ________	VAX DBMS ________	VAX RALLY ________
VAX Rdb/VMS ________	VAX SQL ________	VAX TEAMDATA ________
VAX TDMS ________	VIDA with IDMS/R ________	

6. This release of VAX Information Architecture documentation uses a 7x9 format for quick reference guides. Do you prefer such books in a 7x9 or a 4x8 pocket guide format? ________________________________

Thank you for your assistance.

May we contact you at work for further information? ☐ Yes ☐ No

Name ______________________________ Dept. ______________

Company ______________________________ Date ______________

Mailing Address ______________________________________

______________________________ Phone ______________

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK02–2/N53
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

– Do Not Tear - Fold Here