Oracle CDD/Repository™

Using Oracle CDD/Repository on OpenVMS Systems

Release 7.0

Part No. A40174-1

ORACLE®

Using Oracle CDD/Repository on OpenVMS Systems

Release 7.0

Part No. A40174-1

Contents

2 Using the CDO Screen Editor

3 Managing a Repository

4 Enhancing Repository Performance

5 Managing Repository Elements

6 Implementing a Distributed Repository

7 Managing Repository Protection

8 Using Oracle CDD/Repository with Oracle Rdb

9 Managing RMS Files with CDO

Index

Examples

Figures

Tables

Send Us Your Comments

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

You can send comments to us in the following ways:

- Electronic mail — nedc_doc@us.oracle.com
- FAX — 603-897-3334 Attn: Oracle CDD/Repository Documentation
- Postal service

  Oracle Corporation
  Oracle CDD/Repository Documentation
  One Oracle Drive
  Nashua, NH 03062
  USA

If you like, you can use the following questionnaire to give us feedback. (Edit the online release notes file, extract a copy of this questionnaire, and send it to us.)

Name _____ Title _____

Company _____ Department _____

Mailing Address _____ Telephone Number _____

Book Title _____ Version Number _____

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?

- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available).

Preface

This manual provides information for repository users who store, maintain, and analyze repository elements in Oracle CDD/Repository using the Common Dictionary Operator (CDO) utility.

Document Structure

This document contains the following chapters:

- Chapter 1 provides an overview of Oracle CDD/Repository software concepts and terminology, naming conventions, and translation rules. It also describes how to start a repository session.
- Chapter 2 describes how to use the CDO screen editor.
- Chapter 3 describes the administrative tasks necessary to set up and manage a repository-based application development environment.
- Chapter 4 provides tuning concepts to enhance repository performance.
- Chapter 5 describes how to define, locate, convert, and modify Oracle CDD/Repository elements.
- Chapter 6 describes how to maintain and access a distributed repository.
- Chapter 7 describes the security and protection provisions in Oracle CDD/Repository.
- Chapter 8 describes how to use Oracle CDD/Repository with Oracle Rdb.
- Chapter 9 describes how to create and use Oracle CDD/Repository definitions that represent OpenVMS Record Management Services (OpenVMS RMS) database entities.

Related Documents

Documents related to Oracle CDD/Repository include the following:

- *Oracle CDD/Repository CDO Reference Manual*
- *Oracle CDD/Repository Architecture Manual*
- *Oracle CDD/Repository Callable Interface Manual*
- *Oracle CDD/Repository Information Model Volume I*
- *Oracle CDD/Repository Information Model Volume II*
- *Installing Oracle CDD/Repository on OpenVMS Systems*
- *Read Before Installing or Using Oracle CDD/Repository on OpenVMS Systems.*
- *Oracle CDD/Repository Release Notes*

Conventions

Table 1 shows the conventions used in this manual.

Table 1 Documentation Conventions

Convention	Description
n	A lowercase italic *n* indicates the generic use of a number. For example, 19*nn* indicates a 4-digit number in which the last 2 digits are unknown.
Return	A key name enclosed in a box indicates that you press that key.
	In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.
Oracle CDD/Repository	This manual uses the name Oracle CDD/Repository to refer to all versions of this product. Prior to Version 5.0, this product was known as CDD and CDD/Plus.
SQL	The SQL interface to Oracle Rdb is referred to as SQL in this manual. The SQL interface is the Oracle Rdb implementation of the SQL standard ASNI X3.135-1992, ISO 9075:1992, commonly referred to as the ANSI/ISO SQL standard or SQL92.

(continued on next page)

Table 1 (Cont.) Documentation Conventions

Convention	Description
$ SQL	In examples, interactive SQL is invoked by using the symbol SQL. It is assumed that this symbol has been defined to run the SQL$ image.
( )	In format descriptions, parentheses delimit the parameter or argument list.
[ ]	In format descriptions, brackets indicate optional elements. You can choose none, one, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification.)
“ ”	Quotation marks enclose system messages that are specified in text.
. . .	In format descriptions, horizontal ellipsis points indicate one of the following: • an item that is repeated • an omission, such as additional optional arguments • additional parameters, values, or other information that you can enter
. . .	Vertical ellipsis points indicate the omission of information from an example or command format. The information is omitted because it is not important to the topic being discussed.
italic type	Italic type emphasizes important information, indicates variables, and indicates complete titles of manuals.
UPPERCASE	Words in uppercase indicate a command, a property, a parameter, the name of a file, the name of a qualifier, the name of a file protection code, or an abbreviation for a system privilege.
$	A dollar sign ($) represents the OpenVMS DCL system prompt.
monospaced	This typeface is used in interactive examples and other screen displays.

Intended Audience

The audience for this manual consists of those users who access Oracle CDD/Repository through the CDO utility. These users include the following:

- The data administrator, or repository administrator, who is responsible for creating the repository contents, setting up the security provisions, and maintaining the repository structure.
- The database administrator, who is responsible for creating standard definitions that can be shared among databases and applications.
- Programming supervisors who are responsible for maintaining portions of the repository.
- Programmers who are responsible for maintaining portions of the repository and for writing applications that use the definitions stored in the repository.

The audience for this manual does not include users who access Oracle CDD/Repository through the DMU utility. These users should refer to the following manuals in the DMU Documentation Kit.

- *Oracle VAX Common Data Dictionary—Data Definition Language Reference Manual*
- *Oracle VAX Common Data Dictionary—Utilities Reference Manual*
- *Oracle VAX CDD/Plus User's Guide*

1

Introduction to Oracle CDD/Repository

This chapter provides the following information:

- an overview of Oracle CDD/Repository
- naming conventions and translation rules
- how to start a repository session and use the CDO utility
- how to use the repository template

1.1 Overview

The purpose of a repository is to provide shared access to metadata. Metadata describes data and how that data is used. Metadata includes the location, type, format, size, change history, and usage of the data. Oracle CDD/Repository lets you create, analyze, and administer metadata.

Oracle CDD/Repository is an active, distributed repository system. With it, you can organize, manage, control, and integrate tools for applications across an entire enterprise.

Oracle CDD/Repository is based on an object-oriented data model. With this model, you can represent complex software application components and their behavior in a straightforward and consistent manner. This allows you to create a controlled, disciplined software development environment.

1.1.1 Uses of Oracle CDD/Repository

You can use Oracle CDD/Repository in a variety of models to manage the following:

- system development
  - requirements—to identify the specific needs and scope of a project
  - specifications—to refine the requirements into a precise definition of data and process

  - design—to translate the specifications into a technical design for programs and data structures
  - implementation—to translate the design into actual programs and data declarations
  - testing—to ensure that programs work correctly and fulfill requirements
  - migration to production—to manage the promotion of the system component from a test status to production
  - maintenance—to modify the software to reflect new and revised requirements
- system management
  - project and system management—to coordinate and track the activities and roles of individuals throughout the development process
  - version control—to provide for the management of different versions or branches of the elements the repository controls
  - configuration management—to define and manage composite elements that comprise other elements; to store information about current activities and status in the form of a persistent process
- data administration and system architecture planning—to store data definitions and business models in the repository
- database administration
  - data security and integrity—to protect repository files from unauthorized users
  - extensible definitions—to create and store definitions unique to a particular application
  - Oracle CDD/Repository callable interface—to make direct calls to the Oracle CDD/Repository entry points from user programs

1.1.2 Object-Oriented Data Model

Oracle CDD/Repository describes repository contents in terms of elements, element types, and the element type hierarchy. By using the Common Dictionary Operator (CDO) utility, which is the command line interface to Oracle CDD/Repository, you can create, modify, manipulate, and view repository elements. The element type hierarchy organizes the metadata into a rigid structure.

Elements in a repository represent real-world objects such as files, data definitions, programs, reports, and procedures.

Element types are templates for elements or data structures.

For example, the BINARY element type defines what a binary file is, and the PARSER_TABLES.C element represents a specific binary file. The element PARSER_TABLES.C is an **instance** (occurrence) of the element type BINARY.

Properties (attributes) are the individual characteristics that differentiate various elements of the same element type. Properties can be user-specified, as in the element name, or system-specified, as in the creation and modification dates.

Methods (the element's behavior) serve to characterize elements further. An element receives a **message**, which requests an action but does not specify how the action occurs. The element responds to the message with a method that defines the specific action that makes sense for its type and properties.

The **type hierarchy** contains all the element types, positioned in an **inheritance** structure of **supertypes** and **subtypes**. A type's position in the hierarchy determines the properties and methods it inherits for its elements. A complete definition of an element type includes the definition of the supertype above it, the supertype's supertype, and on up the type hierarchy.

Each repository's type hierarchy is stored in the CDD$PROTOCOLS directory. CDO uses this information to interpret and manipulate the elements stored in that repository.

The object-oriented data model is useful in refining and extending the existing types, properties, and methods that are available to the repository. It also facilitates communication between tools, because the exact representation of an element does not need to be specified by an application requiring it.

1.1.3 Data Definitions and Relationships

A **field** definition is the smallest unit of metadata that can be created and accessed in the repository. Field definitions typically include information about the datatype, size, and other optional attributes of the field.

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 OpenVMS layered products. You only need to store one copy of a particular definition that is used by various sources.

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

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

Oracle CDD/Repository creates **relationships** when you connect two repository elements in the same way. For example, you can base the definition of a new field on a field definition that already exists in a repository. 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 the field and record definitions in CDO.

You can establish a relationship between two elements in different repositories that are distributed on different devices on a single node, on different nodes in a VMScluster environment, or on nodes connected by a local or wide area network. See Chapter 6 for more information about creating relationships between elements and distributing repositories.

1.2 CDO Naming Conventions

This section describes the CDO naming conventions for elements and directories.

1.2.1 Parts of an Element Name

Names for repository elements consist of an anchor, a path, and a version, as follows:

- A repository **anchor** specifies the OpenVMS directory where the repository hierarchy is located. It can consist of node, device, and directory components.

  The anchor can also be described in a logical name format. See Section 5.2.7 for information on using logical names.

- The **path** specifies the route to the desired repository element. The path is an optional list of directory names that are separated by periods or slashes and are terminated by an element name.

  For example, EMPLOYEES.CONTRACT.HOURLY_PAY represents the path that leads to the field HOURLY_PAY. Every name in the path, except the last name, is a repository directory name. The pathnames reflect the directory hierarchy in your repository.

  Note

  You need to specify the node name only if the repository is on a remote node or cluster.

  The following steps occur during pathname translation:

  1. If Oracle CDD/Repository can translate the first name in the supplied pathname, it will. For example, in the name FOO.BAR, "FOO" is translated. If the name contains a wildcard character, such as FO*.BAR, Oracle CDD/Repository cannot translate "FO".
  2. After the name translation, if the name contains an anchor (node name, device, directory or any combination of the three), the location defined by the CDD$DEFAULT logical name is not used. For example, if CDD$DEFAULT is defined as USR:[SALES.CDD], when Oracle CDD/Repository translates the name NODE2::TMP_DB, it does not prefix USR:[SALES.CDD] to the name because NODE2::TMP_DB contains a node name.

     However, if the supplied pathname is TMP_DB, because it does not contain any form of anchor, Oracle CDD/Repository translates the CDD$DEFAULT logical name and prefixes USR:[SALES.CDD] to it to form USR:[SALES.CDD]TMP_DB.

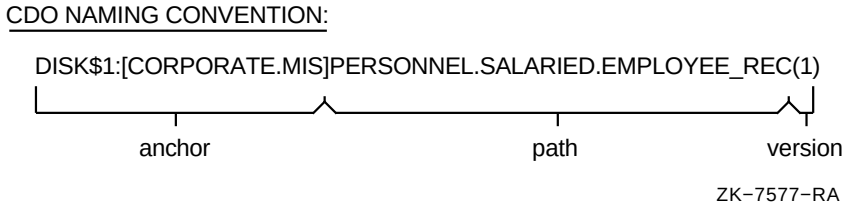
     If the CDD$DEFAULT logical name is not defined, the location defined by the CDD$COMPATIBILITY logical name is used.
  3. If any portion of the anchor (node name, device, or directory) is missing, your OpenVMS default device and directory are applied. In the NODE2::TMP_DB example, the device and directory are missing. Therefore, the OpenVMS default device and directory, USR:[SALES], are applied to form NODE2::USR:[SALES]TMP_DB.

In Version 4.*n*, the period (.) was used to separate the elements in a pathname. However, in Version 5.0 and later, the period can also be part of an element name. If you want to use the period as part of an element name, you must use the logical name CDD$SEPARATOR to define the slash character (/) to separate elements in the pathname. For example:

```
$ DEFINE CDD$SEPARATOR "/"
```

Figure 1–1 shows the format for element names.

Figure 1–1 Oracle CDD/Repository Element Names



- A **version** consists of a number, and an optional branch name and branch version number.

  A succession of versions of an element is called a **line of descent**; each version is a descendant of the previous version. An element can have several lines of descent. For example, if two people need to modify the same element simultaneously, the **main line of descent** must split to allow these parallel modifications. One of these paths branches from the main line of descent, while the other path continues the main line of descent.

  The main line of descent is indicated by a version that includes only a version number. For example, the second version of the element HOURLY_PAY in a main line of descent is HOURLY_PAY(2).

  A parallel **branch line** of descent is indicated by a version number (the version where branching began in the preceding line of descent) and its own branch name and branch version number. For example, you can create a succession of new branches, each originating from the other, as follows:

  ```
  HOURLY_PAY(2)
  HOURLY_PAY(2:BRANCH:1)
  HOURLY_PAY(2:BRANCH:1:ANOTHER_BRANCH:1)
  HOURLY_PAY(2:BRANCH:1:ANOTHER_BRANCH:1:YET_ANOTHER:1)
  ```

  You can create multiple versions and multiple branches of a repository element.

For more information about element name syntax, see Chapter 3 in the *Oracle CDD/Repository Architecture Manual*.

1.2.2 Directory and Processing Names

Repository **directories** are named sections of a repository that you use to organize repository elements, such as fields, records, and other directories. Before you define repository elements, create repository directories to group the related elements, keeping your planned structure in mind.

A repository directory is similar to an OpenVMS directory in terms of its hierarchical structure and purpose. Although a repository itself has no implicit hierarchical structure, repository directories allow you to group related elements and to use the resulting hierarchy in element names. The repository directories point to the elements you store in them; they are not repository elements themselves.

From the CDO environment, you can create repository directories in the compatibility repository or in a repository that you created. You create repository directories with the CDO DEFINE DIRECTORY command. See Section 1.3.8 for more information on creating repository directories.

When you create an element in CDO, you specify a repository directory and the name for the element; this is known as the **directory name**. The directory name acts as a pointer to the location of the actual element. Without a directory name, CDO cannot locate and perform operations on the elements in the repository.

A directory name identifies the element outside the repository system. CDO uses this name to automatically assign both a directory name and a processing name to the element.

An element's **processing name** is the name by which it is known to the repository system. Each element has only one processing name. Usually the directory name and the processing name are the same. However, you can assign several directory names to each element by using the CDO ENTER command. Refer to the *Oracle CDD/Repository CDO Reference Manual* for more information on the CDO ENTER command.

Without a directory name, the CDO DIRECTORY command cannot display an element, and you cannot include the element as part of other elements. For example, you would not be able to include a field without a directory name in an Oracle Rdb global field definition.

Elements that exist only as part of another element and that are never accessed independently do not have directory names. An element that is not part of another element (has no parents) and has no directory name is called an **orphan**. An orphan is created when corruption causes an element to lose its links; the element no longer has any relationships to the directory system or to other elements. Oracle CDD/Repository makes the element an orphan so that you can access it and reestablish its links, if necessary. To reestablish links to another object, you must redefine the relationship. To reestablish links to the directory system, use the CDO VERIFY command with the /ORPHANS qualifier.

1.3 Getting Started with Oracle CDD/Repository

This section shows you how to:

- use the CDO utility
- start and end a repository session
- use online help in CDO
- create a repository
- create a repository directory

1.3.1 Using the CDO Utility

The Common Dictionary Operator (CDO) utility is the command line user interface to Oracle CDD/Repository. You use CDO commands to do the following:

- set the environment characteristics to suit your needs
- create and manipulate repository elements and directories

The CDO commands are described in detail in the *Oracle CDD/Repository CDO Reference Manual* and in online help.

CDO Syntax Rules

Entering CDO commands is similar to entering OpenVMS DCL commands. CDO commands follow simple syntax rules: the MOVE, DEFINE (with the exception of the DEFINE KEY command), CHANGE, PURGE, and DELETE commands require a terminating period.

If you press the Return key before your command is complete, CDO prompts you for the remaining input with a continuation prompt (cont>). For example:

```
CDO> DEFINE REPOSITORY DISK1:[CORPORATE.MIS]
cont>.
```

CDO provides a spelling check feature on CDO command keywords. You can abbreviate a command keyword if the abbreviation is unique, and CDO processes the command. CDO also corrects the misspelling of a keyword and continues to process the command without generating an error if you:

- omit a character
- type an extra character
- transpose characters
- substitute an incorrect character for a correct one

1.3.2 Starting and Ending a Repository Session

To start a repository session, invoke the CDO utility by entering the REPOSITORY OPERATOR command at the OpenVMS DCL prompt ($). The system displays the CDO prompt, as follows:

```
$ REPOSITORY OPERATOR
CDO>
```

The DCL DICTIONARY OPERATOR command, which was used in earlier versions, can still be used to invoke CDO. The REPOSITORY OPERATOR and DICTIONARY OPERATOR commands are interchangeable.

To end a repository session and return to the OpenVMS DCL prompt, type EXIT or Ctrl/Z at the CDO prompt. For example:

```
CDO> EXIT
$
```

1.3.3 Using Online Help

To access online help for CDO commands and Oracle CDD/Repository concepts, enter the HELP command at the CDO prompt. The Help utility displays a list of available help topics.

```
CDO> HELP
```

The CDO Help utility also provides information on Oracle CDD/Repository error messages. To see the types of help that are available for error messages, type HELP ERRORS at the CDO prompt.

1.3.4 Preparing to Create a New Repository

Before you create a repository, confirm that you have enough free disk space. Each new repository requires 15,000 blocks.

Next, create an OpenVMS directory where the repository will reside. This directory is the repository anchor, and it should remain dedicated to your repository. *Do not store any other files in this directory.* The anchor should contain only files created by Oracle CDD/Repository, because if you decide to delete the repository later, Oracle CDD/Repository will delete all the files in this directory.

1.3.5 Creating a New Repository

Invoke CDO and enter the CDO DEFINE REPOSITORY command. Specify the OpenVMS directory where you want the repository to reside. If you specify a directory that does not exist, CDO creates one for you in your default directory and places the repository files there. For example:

```
$ REPOSITORY OPERATOR
CDO> DEFINE REPOSITORY DISK1:[CORPORATE.MIS].
```

You can create more than one repository on the same node and on the same disk. However, each repository must reside in a separate OpenVMS directory that has been created exclusively for the repository, and each repository should have a unique name.

Restrictions

Do not set default at the OpenVMS DCL level to a repository anchor and change the files in the anchor; Oracle CDD/Repository creates and maintains these files. You can set your process default to any directory, invoke CDO, and refer to your new repository from there by using the CDO SET DEFAULT command.

Do not create a repository in your top level directory [000000]. If you find that your repository or your configuration management subdirectories (CONTEXTS.DIR, DELTAFILES.DIR, and PARTITIONS.DIR) exceed eight nested subdirectories, the maximum number allowed on OpenVMS systems, use a concealed logical name to represent several directory levels. For example:

```
$  DEFINE AB$REPO/SYSTEM/TRANS=CONCEALED DISK$:[KIM.REP.AB.REPO.]
$  REPOSITORY OPERATOR
   .
   .
   .
CDO>  DEFINE REPOSITORY AB$REPO:[PART_NUMBERS].
```

1.3.6 Contents of a Repository

When you define a new repository, Oracle CDD/Repository automatically creates several files and places them in the anchor directory. *Do not delete these files; otherwise, you will corrupt your repository.*

- 00000000.30000000

  This root directory file is created for each repository that is defined. This file is always 0 blocks in size. This file is represented in the repository as the top level directory.

- 30000000CDD$PROTOCOLS.40000000

  This protocols file contains the sets of rules that Oracle CDD/Repository uses in defining types of elements and relations. This file is always 0 blocks in size. This file is represented in the repository as the CDD$PROTOCOLS subdirectory.

  The CDD$PROTOCOLS directory contains definitions that describe the types, properties, methods, and messages that you use in your data descriptions. These definitions are essential to the functioning of your repository; do not delete them.

- CDD$DATA.RDA, CDD$DATABASE.RDA, CDD$DATABASE.RDB, CDD$DATABASE.SNP, and CDD$DATA.SNP

  These repository database files are based on an Oracle Rdb database, where all the repository elements are stored.

- CDD$DIRECTORY.CDD

  This index file contains the directory structure for a repository.

- CONTEXTS.DIR

  This OpenVMS directory file is used for storing contexts in a repository. The directory is empty until you create contexts in the repository.

- DELTAFILES.DIR

  This OpenVMS directory file contains delta files for binary objects in the repository. The directory is empty until you create binary objects in the repository.

- PARTITIONS.DIR

  This OpenVMS directory file contains files related to partition objects in the repository. The directory is empty until you create partitions in the repository.

Although some of the files appear to be empty (that is, they show 0 blocks), they are referenced by the Oracle Rdb database. The repository depends on the integrity of these files, which must remain at version 1. Do not modify their contents.

The following is an example of the repository files in an anchor directory:

```
$ DIRECTORY/SIZE/DATE DISK1:[CORPORATE.MIS]

Directory DISK1:[CORPORATE.MIS]

00000000.30000000;1
                           0  14-AUG-1991 11:28:44.53
30000000CDD$PROTOCOLS.40000000;1
                           0  14-AUG-1991 11.28:44.02
CDD$DATA.RDA;1          4824  14-AUG-1991 11:28:36.60
CDD$DATA.SNP;1           408  14-AUG-1991 11:29:24.43
CDD$DATABASE.RDA;1      7616  14-AUG-1991 11:28:23.53
CDD$DATABASE.RDB;1       243  14-AUG-1991 11:28:21.79
CDD$DATABASE.SNP;1       972  14-AUG-1991 11:29:21.24
CDD$DIRECTORY.CDD;1
                           9   9-JUL-1991 13:54:32.36
CONTEXTS.DIR;1             1  14-AUG-1991 11:28:13.49
DELTAFILES.DIR;1           1  14-AUG-1991 11:28:12.93
PARTITIONS.DIR;1           1  14-AUG-1991 11:28:14.15

Total of 11 files, 14075 blocks.
```

1.3.7 Correcting Errors When Defining a Repository

This section describes some common errors that can occur when you define a repository.

The following errors occur when the anchor directory has invalid access control lists (ACLs):

```
%CDO-E-ERRDEFINE, error defining object
-RDB-E-NO_RECORD, access by dbkey failed because dbkey is no longer
associated with a record
-RDMS-F-NODBK, 0:8323074:1 does not point to a data record

%CDO-E-ERRDEFINE, error defining object
-CDD-F-ERRCREDIC, error creating dictionary
-RDB-F-BAD_SEGSTR_ID, invalid segmented string identifier
```

This error occurs due to a restriction in Oracle CDD/Repository. You must correct invalid ACLs on the anchor directory file. Examine the ACLs on the anchor directory file using the OpenVMS DCL DIRECTORY/SECURITY command. Determine which ACL is invalid and correct it using the OpenVMS DCL SET ACL command.

The following message occurs when you create a repository without using the logical name CDD$TEMPLATE and the repository is being created without the template.

```
CDD-I-CRECONT, define repository continuing without template
```

In this case, Oracle CDD/Repository creates the files for your repository from the beginning, and the CDO DEFINE REPOSITORY command will take several minutes. Using the template files improves performance and saves time when you create a repository. See Section 1.8 for more information about the repository template.

If you see messages indicating a problem with the version of Oracle Rdb, it could be that the repository template is at a different version of Oracle Rdb than the version of Oracle Rdb that is currently running on your system. This causes the DEFINE REPOSITORY operation to fail and produces the following error:

```
-CDD-F-NOATTACH, unable to attach to dictionary database
-RDB-F-WRONG_ODS, the on-disk structure of database filename is not
 supported by version of facility being used
-RDMS-F-ROOTMAJVER, database format 60.0 is not compatible with
 software version 41.0
CDO>
```

In this example, the repository template was created using Oracle Rdb Version 6.0, and the user is trying to use that template with Oracle Rdb Version 4.1. To use the Version 6.0 template, the version of Oracle Rdb must be set to Version 6.0 or higher.

Refer to *Installing Oracle CDD/Repository on OpenVMS Systems* for information on setting the version of Oracle Rdb, or see your system manager.

To verify that CDO created the repository, use the CDO DIRECTORY command:

```
CDO> DIRECTORY DISK1:[CORPORATE.MIS]*

Directory DISK$1:[CORPORATE.MIS]

CDD$PROTOCOLS                                   DIRECTORY
   .
   .
   .
CDO>
```

1.3.8 Creating a Repository Directory

In the following example, two repository directories are created: BUDGET and PERSONNEL. The names of these directories are appended to the repository anchor, which appears in brackets.

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

Note that the repository directory name is outside the brackets. If you place a repository directory name within the brackets, CDO looks for an OpenVMS directory by that name and generates an error message.

To list the contents of the directory, use the CDO DIRECTORY command with a wildcard character (*), which is the default. For example:

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

Directory DISK1:[CORPORATE.MIS]

BUDGET                                  DIRECTORY
CDD$PROTOCOLS                           DIRECTORY
EMPLOYEE_DATA(1)                        RECORD
EMPLOYEE_REC(1)                         RECORD
LAST_NAME(1)                            FIELD
NAME_STRING(1)                          FIELD
PART_NO(1)                              FIELD
PERSONNEL                               DIRECTORY
START_DATE(1)                           FIELD
   .
   .
   .
```

1.4 CDD$COMPATIBILITY Repository

In addition to CDO, Oracle CDD/Repository provides the Dictionary Management Utility (DMU), which was used with earlier versions of Oracle CDD/Repository. The **compatibility repository** is a special repository that coordinates elements that were created using DMU. Definitions in this repository and in the DMU dictionary, CDD.DIC, appear to be one, combined repository.

The compatibility repository provides an internal translation of the element's name from DMU format, so if an application references an element that was defined using DMU, it can be used in the CDO environment.

The **translation utility** translates definitions from CDO format into DMU format, and vice versa. You refer to DMU dictionary definitions by a pathname beginning with the origin CDD$TOP; you refer to CDO repository elements by a pathname beginning with an anchor. The translation utility translates the CDD$TOP logical name to be equivalent to the anchor of your compatibility repository.

The system logical name for the compatibility repository is CDD$COMPATIBILITY. The CDD$COMPATIBILITY logical name can be defined as only one repository (for one version of Oracle Rdb). You cannot have more than one compatibility repository; otherwise, you could have duplicate definitions.

If you invoke CDO and CDD$DEFAULT has not been defined, or if you have not set default to a specific anchor directory, Oracle CDD/Repository places you in the compatibility repository by default.

If you set default to or refer to an object in the compatibility repository, use the CDD$COMPATIBILITY logical name rather than the CDD$TOP logical name in the specified name. Using CDD$TOP causes a command to be performed only for the highest version of any object, rather than for all versions of that object.

1.5 Establishing Your Default Repository Directory

The **default repository directory** is the one in which you plan to work during the current session. You can define the logical name CDD$DEFAULT to point to a specific repository directory or search list, or you can set default to the directory within CDO.

If you issue the CDO SET DEFAULT command, Oracle CDD/Repository makes the latest directory specified in the command the current default repository. If the CDO SET DEFAULT command was not issued, the default repository is the location defined by the CDD$DEFAULT logical name. If you did not define CDD$DEFAULT, your default repository is the compatibility repository.

1.5.1 Using the CDD$DEFAULT Logical Name

You can use the CDD$DEFAULT logical name to specify the repository anchor, and optionally, an anchor subdirectory, to be your default repository directory when you invoke CDO.

The following command defines CDD$DEFAULT to be the anchor `DISK1:[CORPORATE.MIS]`, followed by the directory `PERSONNEL`, which was previously defined.

```
$ DEFINE CDD$DEFAULT "DISK1:[CORPORATE.MIS]PERSONNEL"
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> SHOW DEFAULT
DISK1:[CORPORATE.MIS]PERSONNEL
```

Restriction

Do not define CDD$DEFAULT as two OpenVMS logical names separated by a period. Oracle CDD/Repository translates the leftmost part of a string as a logical name and does not translate remaining substrings.

CDO cannot differentiate OpenVMS logical name syntax from Oracle CDD/Repository pathname syntax. OpenVMS accepts the STRING.STRING logical name as FILENAME.FILETYPE syntax. For example:

```
$  DEFINE CDD$DEV2 NODE$DKA300:[USER.CODE]
$  DEFINE MYDIR [USER.DICT]
$  DEFINE CDD$DEFAULT CDD$DEV2.MYDIR
$  SHOW LOGICAL CDD$DEFAULT
   "CDD$DEFAULT" = "CDD$DEV2.MYDIR" (LNM$PROCESS_TABLE)

$  REPOSITORY OPERATOR
   .
   .
   .
CDO>  SHOW DEFAULT
CDD$DEFAULT
    = NODE$DKA300:[USER.CODE]MYDIR
```

1.5.2 Setting a Default Directory Within CDO

To set a default directory within CDO, designate the directory with the CDO SET DEFAULT command, as in the following example:

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

All the elements that you create will be stored in the default directory until you specify a different default directory, or unless you explicitly specify another directory when you define the element.

Within CDO, you can set default to a directory any number of times.

After you establish a default directory, you can omit the directory name and refer to an element in that directory by its name alone. For example, after you set your default directory to `DISK1:[CORPORATE.MIS]PERSONNEL`, you can refer to the field `DISK1:[CORPORATE.MIS]PERSONNEL.LAST_NAME` as simply `LAST_NAME`.

When you issue a CDO command other than a CDO SET or SHOW DEFAULT command in a repository, Oracle CDD/Repository reads the protocols in the directory assigned to the CDD$PROTOCOLS logical name into memory.

The protocols for that repository are used for the rest of the CDO session; the CDO SET DEFAULT command does not change the CDD$PROTOCOLS directory assignment.

For example, suppose you have an extended repository, but you set default to a repository that does not contain these extensions (such as the compatibility repository) and you issue CDO commands. The protocols that are used during the CDO session will not contain your extensions, even if you later set default to the extended repository. You must exit CDO, set default to the extended repository, then issue the CDO commands.

If your default repository is defined as a search list, Oracle CDD/Repository creates elements and directories only in the first location in the search list. Oracle CDD/Repository does not try additional locations in the list if the first location is invalid.

1.5.3 Defining Special-Purpose Keys

You can save time entering frequently used commands by binding commands to keys. Use the CDO DEFINE KEY command to bind a specified key to the definition you want associated with the key. The definitions are valid until you exit CDO. For example:

```
CDO> DEFINE KEY PF4 "SET DEFAULT DISK1:[CORPORATE.MIS]"
```

It is not necessary to terminate the CDO DEFINE KEY command with a period.

1.6 Creating a CDO Initialization File

If you frequently start your repository sessions with the same series of commands, you can put them in an initialization file. An **initialization file** is a command procedure that automatically executes when you invoke CDO.

To create an initialization file, perform the following steps:

1. Use a text editor to create a file called CDO$INIT.CDO.
2. In the file, enter the CDO commands you would enter in an interactive session, but do not include the CDO prompt.

Whenever you invoke CDO, it searches your default directory for the initialization file CDO$INIT.CDO. By defining a logical name for CDO$INIT, you can invoke the file from other directories.

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 get to work area quickly.
!
DEFINE KEY PF3 "SET DEFAULT [CORPORATE.MIS]PERSONNEL"
!
!Select initial default to usual repository area.
!
SET DEFAULT [CORPORATE.MIS]PERSONNEL.RETIRED
!
!Send output from subsequent commands to a file.
!
SET OUTPUT OUTPUT.LOG
!
!Check the version of Oracle CDD/Repository that you are using.
SHOW VERSION
```

1.7 Using CDO Command Procedures

A **command procedure** is a file containing a fixed sequence of commands that you create, name, and store in an OpenVMS directory. You can run a command procedure at the CDO prompt to execute any series of frequently used commands. You can also run command procedures as part of an OpenVMS batch file.

To create a command procedure:

1. Set your OpenVMS working directory to be any directory that does not contain a repository. (A repository directory should contain only files created by Oracle CDD/Repository.)
2. At the DCL prompt ($), invoke a text editor.
3. Enter the CDO commands. Enter each new command at the left margin of your file, and do not include the CDO prompt.

To include comments in your command procedure, place an exclamation point (!) before each commented line.

4. Write the command procedure to a file. If you do not want to specify the file extension each time you run the procedure, use the .CDO default file extension.

You can include any CDO commands that you can use interactively (including the CDO DEFINE command) in a CDO command procedure and execute it at the CDO prompt. This method is convenient when you anticipate creating new versions of particular fields or records, such as for testing purposes.

Note

You cannot execute the CDO EDIT command in batch mode.

By maintaining long definitions in command procedures, you can avoid repetition and prevent syntax errors.

When you execute commands in files, Oracle CDD/Repository issues messages about syntax errors as the errors occur. When you execute a command procedure from CDO, CDO assumes the default file extension of .CDO and searches for the specified file in your current OpenVMS directory, unless you name another directory and file extension.

In the following example, the DEFINE_ADDRESS.CDO command procedure creates a record ADDRESS when executed at the CDO prompt. The record includes five fields that were previously defined.

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

In the next example, the CREATE_TESTS.CDO command procedure executes two other CDO command procedures; however, CREATE_TESTS.CDO does not include the contents of those procedures.

```
!
!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 DISK1:[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*
```

Using the CDO START_TRANSACTION and COMMIT Commands

Using the CDO START_TRANSACTION, CDO COMMIT, and CDO ROLLBACK commands will make larger command procedures run more efficiently. A single repository transaction is defined as an individual CDO command. However, a single transaction can also be defined as a series of individual CDO commands placed between CDO START_TRANSACTION and CDO COMMIT (or ROLLBACK) commands within a command procedure.

The CDO START_TRANSACTION command initiates a group of commands that Oracle CDD/Repository executes as a unit. The transaction ends with a COMMIT or ROLLBACK command.

Caution

While a transaction is open, locks are held against the repository database. Therefore, using the CDO START_TRANSACTION command can reduce the concurrency of the repository database.

The CDO COMMIT command causes all statements to execute. It ends a transaction and makes permanent any changes you made during that transaction. The CDO COMMIT command also releases all locks and closes

all open streams. It affects all databases participating in the currently open transaction.

The CDO ROLLBACK command also releases all locks, closes all open streams, and releases all readied tables. The CDO ROLLBACK command causes no statements to execute. It affects all databases participating in the currently open transaction.

To avoid Oracle CDD/Repository errors, CDO often checks for possible error conditions when you issue a CDO CHANGE or DELETE command. CDO checks that the object to be modified or deleted exists before performing an operation on the object. If the object exists, Oracle CDD/Repository makes the change. If the object does not exist, Oracle CDD/Repository returns an informational status to CDO, and CDO aborts the operation. Oracle CDD/Repository does not make the change. The operation of checking the existence of an object is successful even if the object does not exist. Therefore, Oracle CDD/Repository does not encounter any error conditions that would force a rollback of the transaction.

The general rules pertaining to a START_TRANSACTION and COMMIT session are as follows:

- If you receive an Oracle CDD/Repository error of -E- or -F- severity, such as CDD-E-NODNOTFND, you must roll back the transaction.
- If you receive a CDO error of -E- or -F- severity, such as CDO-E-NOTFOUND, you may continue to operate in the current transaction.

1.8 Using the Repository Template

Oracle CDD/Repository provides a repository template during the installation procedure. The repository template is a backup of an empty repository; it contains only the protocols (element, relation, and property types) that are supplied by Oracle CDD/Repository. The repository template supplied by Oracle CDD/Repository does not contain any extensions that you have made to your repository. However, you can customize the template to include those extensions. Section 1.8.1 describes how to customize the template.

Note

You cannot use the template itself as a repository.

When you use the CDO DEFINE REPOSITORY command, Oracle CDD/Repository:

1. restores the backup file in the directory assigned to the logical name CDD$TEMPLATEDB by using the RMU/RESTORE command
2. copies the files in the directory assigned to the CDD$TEMPLATE logical name to the new repository anchor by using the DCL COPY command
3. sets the internal repository name to the new repository anchor

The files in the template directories are supplied when Oracle CDD/Repository is installed, and the CDD$TEMPLATE and CDD$TEMPLATEDB logical names are defined during the installation.

1.8.1 Customizing the Repository Template

During the installation, Oracle CDD/Repository supplies a command procedure called SYS$LIBRARY:CDD_BUILD_TEMPLATE.COM that you can use to create a customized repository template.

You can then use the template to create a customized repository. If you extend your repository by defining new element types or by modifying existing types (using the Oracle CDD/Repository callable interface or the Oracle CDD/Administrator Extension utility), your modifications can be incorporated into a template. After you create a customized template, it can be used to automatically apply your extensions when you define a new repository.

To invoke the CDD_BUILD_TEMPLATE.COM procedure, enter the following command:

```
$ @SYS$LIBRARY:CDD_BUILD_TEMPLATE.COM device:[template] device:[templatedb]
```

where *device:[template]* represents the anchor of a new repository that you created containing your extensions. This repository will be used to create the template.

The location *device:[templatedb]* is the empty directory where the template database will reside.

When you want to use these customized templates, reassign CDD$TEMPLATE to be *device:[template]* and CDD$TEMPLATEDB to be *device:[templatedb]*. You can assign access to PROCESS, GROUP, or SYSTEM logicals.

If you no longer want to use the templates supplied by Oracle CDD/Repository and want to use only the customized template that you created, delete the original CDD$TEMPLATE and CDD$TEMPLATEDB directories. Modify the following lines in the SYS$STARTUP:CDDSTRTUP.COM procedure to point to the new location of the template.

```
$ DEFINE/NOLOG/SYSTEM/EXEC CDD$TEMPLATE device:[cdd$template]
$ DEFINE/NOLOG/SYSTEM/EXEC CDD$TEMPLATEDB device:[cdd$templatedb]
```

2
Using the CDO Screen Editor

This chapter introduces the CDO screen editor and shows you how to perform the following tasks using the editor:

- create new field and record definitions
- create new versions of previous definitions
- browse through your current definitions

Note

The CDO screen editor works exclusively for uncontrolled field and record definitions. See Section 5.2.1 for information on controlled and uncontrolled elements.

2.1 Overview of the CDO Screen Editor

The CDO screen editor is a flexible, menu-driven tool that lets you manipulate definitions by selecting items from menus and entering values from the keyboard. If you are entering common field and record definitions, you may prefer using the CDO screen editor instead of entering CDO command lines.

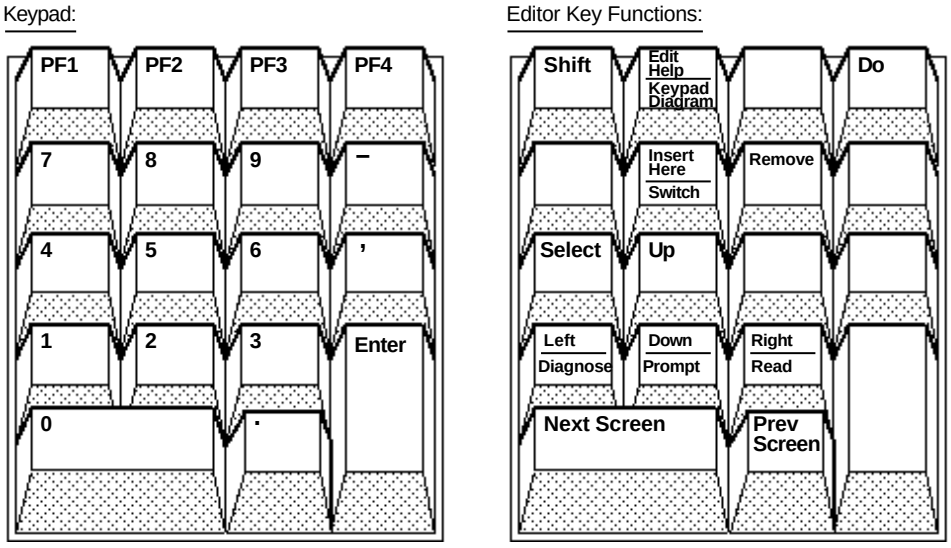
The following features make defining fields and records within the editor easy:

- convenient menus that provide available properties, relations, and allowed values
- browsing capability
- dynamic data type validation
- cursor movement between field properties
- keypad access to a text editor
- online help on keypad keys and field properties

2.2 CDO Screen Editor Second Function Keys

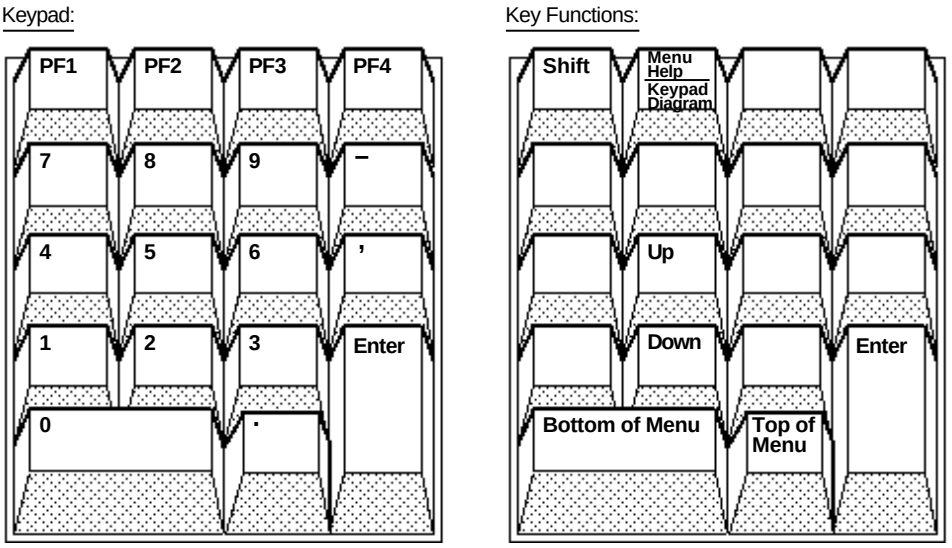
For faster editing, the CDO screen editor features keys that have been defined for editing functions. Figure 2–1 shows key functions on all keypads. Keys that show two commands have a second function. The second function is the lower notation on the key. To access the second function press PF1 (the editor's Shift key), then press the second function key.

Figure 2–1 CDO Screen Editor Key Functions



When the CDO screen editor displays a menu, some keys assume different functions to help you select menu items, as shown in Figure 2–2.

Figure 2–2 CDO Screen Editor Menu Keypad and Key Functions



ZK–7581–GE

You can also use the DCL line-editing keys, such as the Help and Do keys. For a full list of the DCL line-editing keys, see the OpenVMS documentation.

2.3 Accessing the CDO Screen Editor Help

The CDO screen editor provides online Help from the CDO prompt, as well as from within the editor.

- from the CDO prompt:
  - To display Help about the CDO screen editor, enter the `HELP EDIT` command. Select the `Interactive_Help` subtopic for a description of the editing function keys.
  - To display Help on properties and items in a CDO screen editor menu, enter the `HELP SPECIFY` command.
- during an editing session:
  - To display a keypad diagram of editing function keys, press the PF2 key or the Help key. You can then display help about a particular key function by pressing that key.
  - To display help on properties and items in a menu, use the arrow keys to select the menu item, and press PF1 followed by PF2.

2.4 Editing Text Within the CDO Screen Editor

By default, the CDO screen editor supplies the EVE editing interface for DECTPU that lets you enter text for an element's description.

You can choose another editing interface by defining an alternative section file for TPUSECINI in your login command procedure. For example, the following command defines EDT as the default editing interface whenever DECTPU is invoked:

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

2.5 Customizing Your CDO Screen Editor Display

Some of the CDO screen editor features can be customized. The status of these features is highlighted at the bottom of the CDO editor screen during an editing session. For example:

```
editing:  FIELD - diagnostics:  ON - prompts:  ON - dictionary read:  ON
```

CDO automatically displays the definition type (field or record) that you are editing. The Editing field cannot be customized. However, you may want to change the default values of the diagnostics, prompts, and dictionary read features as you become more familiar with using the CDO screen editor.

2.5.1 Diagnostics Feature

The diagnostics prompt indicates whether the editor validates your entries. For example, the editor warns you if you try to enter a processing name that contains more than 31 characters. Similarly, when you specify a data type, you must specify a valid type from the supplied list. See the *Oracle CDD/Repository Callable Interface Manual* for more information on validation.

Diagnostics are on by default. To disable diagnostics, press the PF1 key followed by KP1. To enable the diagnostics feature after it has been disabled, press the PF1 key followed by KP1.

2.5.2 Prompts Feature

By default, CDO displays prompts that describe the appropriate action for a particular field. The prompts are highlighted at the top of the editor screen during an editing session. Prompts are on by default.

Once you are familiar with the CDO screen editor, you may no longer want CDO to display the prompts. To disable them, press the PF1 key followed by KP2. To enable the prompts feature after it has been disabled, press the PF1 key followed by KP2.

2.5.3 Dictionary Read Feature

When you create a new definition using the CDO screen editor, the editor searches the repository for the named definition if the dictionary read feature is turned on.

If the definition does not exist in the repository, the editor creates it for you.

If the definition exists, the editor displays the properties and values associated with the definition with the following message:

definition-name was successfully read from the dictionary

When the dictionary read feature is turned off, CDO does not search the repository for a definition with the same processing name. The screen display is not changed. Reading from the repository is on by default.

To disable the dictionary read feature, press the PF1 key followed by KP3. To enable the dictionary read feature after it has been disabled, press the PF1 key followed by KP3.

2.6 Browsing Through Current Definitions

To browse through the current definitions in the repository directory, perform the following steps:

1. From the CDO prompt, enter the EDIT FIELD or the EDIT RECORD command without specifying a definition name.
2. Press the Select key to display a list of current definitions.
3. Use the arrow keys to choose an item from the list, then press Return.

   The editor displays the selected definition on your editing screen. You can then select other definitions to view or change.

You can also access a menu of current definitions while you are editing another definition. If a property requires a definition name, press the Select key followed by the Return key. The editor displays a menu of choices, such as CONTAINS, BASED ON, or PROCESSING NAME. When you select a name from the menu, the CDO editor places the name in the property you are editing.

2.7 Changing Existing Definitions

When you make changes to an existing definition, you create a new version of that definition; you do not change the original version.

Entering definitions using the CDO screen editor is equivalent to using the CDO DEFINE command, not the CDO CHANGE command. For example, if the following versions of a field exist and you edit version 2, the edited field definition becomes version 5.

```
FIRST_NAME(4)                                   FIELD
FIRST_NAME(3)                                   FIELD
FIRST_NAME(2)                                   FIELD
FIRST_NAME(1)                                   FIELD
```

CDO stores all versions of a definition unless you purge or delete them. To delete or purge a definition, you must use the CDO command line interface; you cannot use the screen editor for deleting and purging definitions. The CDO DELETE and PURGE commands are described in Chapter 5.

An application can access any version of a definition if the application specifies the version number. When a version number is not specified, CDO defaults to the highest version.

2.8 Exiting from the CDO Screen Editor

To exit from the CDO screen editor, press Ctrl/Z or F10.

If you have made changes to a definition, press the Do key to save the changes. If you do not press the Do key, the editor displays the following prompt:

```
Changes have not been saved by pressing DO.  Throw changes away (Y or N)?
```

You must confirm whether or not you want to exit without saving your edits.

- Answer Yes if you do *not* want to save your edits and you want to quit from your editing session.
- Answer No to resume the editing session. Press the Do key to enter your changes before exiting.

2.9 Sample CDO Editing Session

This sample session describes how to create three new field definitions and one record definition that contains the three fields.

1. Invoke the CDO utility:

   ```
   $ REPOSITORY OPERATOR
   CDO>
   ```

2. Create a repository directory called EDITOR_EXAMPLES that will group the definitions together. Then, set your default work area to the new directory.

   ```
   CDO> DEFINE DIRECTORY EDITOR_EXAMPLES.
   CDO> SET DEFAULT your$disk:[your_repository]EDITOR_EXAMPLES
   ```

3. Invoke the CDO screen editor to create a new field definition called FIRST_NAME. You can include the field or record definition name on the command line, or you can type the name once you are within the editor.

   ```
   CDO> EDIT FIELD FIRST_NAME
   ```

   Note

   Do not give a definition the same name as a previously defined logical name.

The editor displays the following screen:

The screen displays the MCS processingName field, which contains the name of the definition you are creating. In this example, the field contains the name FIRST_NAME.

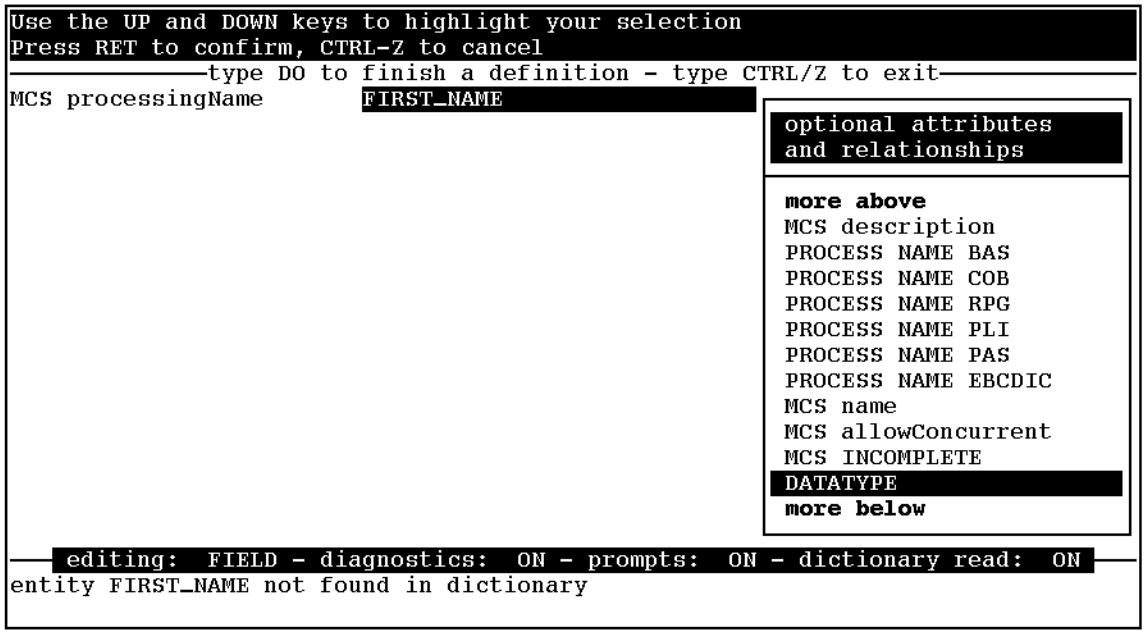
MCS processingName is the internal name used by Oracle CDD/Repository for the property (or attribute) of the element. The repository uses the processing name to identify the actual element.

Because this is a new definition, the following message at the bottom of the screen indicates that the editor did not find FIRST_NAME in the repository:

```
entity FIRST_NAME not found in dictionary
```

4. Press the Insert Here key to display the field property choices.

The editor displays a pop-up menu of optional attributes and relationships.

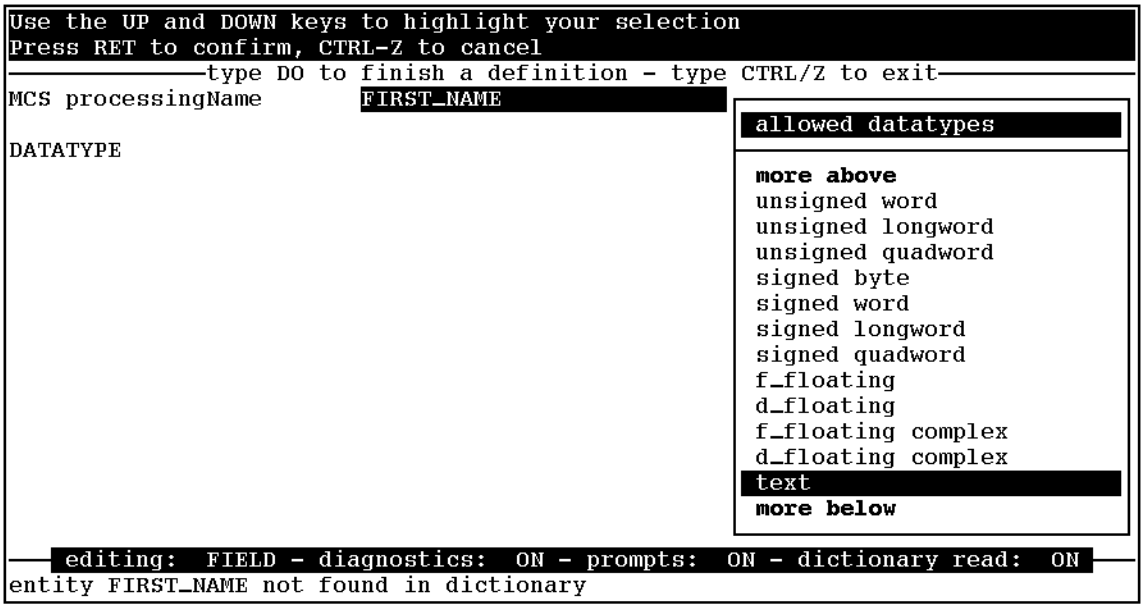


5. Use the up and down arrow keys to select a property from the menu. You must select at least one property.
6. For this example, press the down arrow key until the DATATYPE property is highlighted. Press the Return key to select DATATYPE.

The DATATYPE field appears on the screen.

7. Press the Select key to display the allowed data types.

8. Press the down arrow key until the cursor highlights the data type TEXT.



9. Press the Return key to select TEXT.

The editor inserts TEXT in the DATATYPE field on the screen. You must enter the length of the text for the FIRST_NAME field; therefore, the editor displays the LENGTH field on your screen.

10. Press the down arrow key to move the cursor to the LENGTH field. Enter the value 12. Your screen should look like the following:

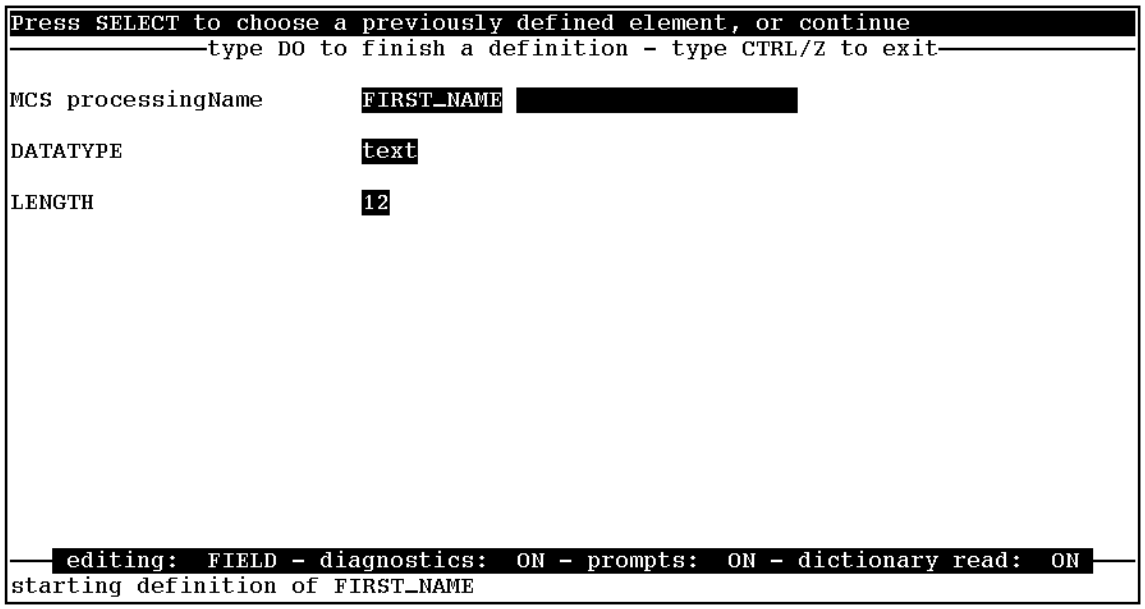
```
─────────────type DO to finish a definition - type CTRL/Z to exit─────────────

MCS processingName          FIRST_NAME

DATATYPE                    text

LENGTH                      12




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

11. Press the Do key to add the FIRST_NAME field definition to the repository directory.

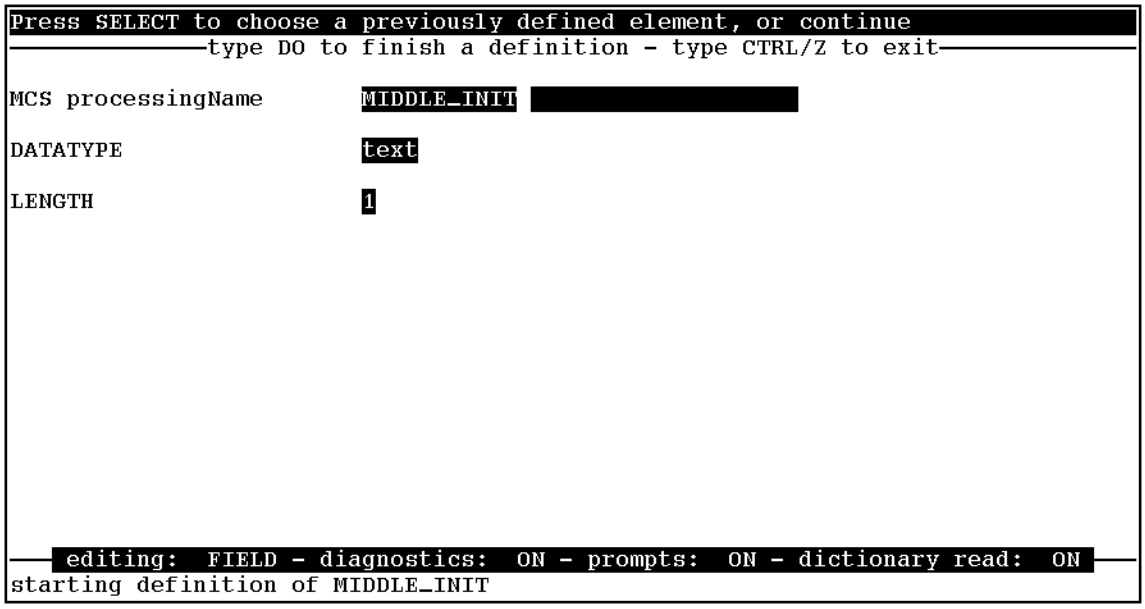
    When CDO adds the definition, the following message is displayed at the bottom of your screen:

    ```
    Starting definition of FIRST_NAME
    ```

    The screen continues to display the definition of the field FIRST_NAME, and is ready for you to enter another definition.

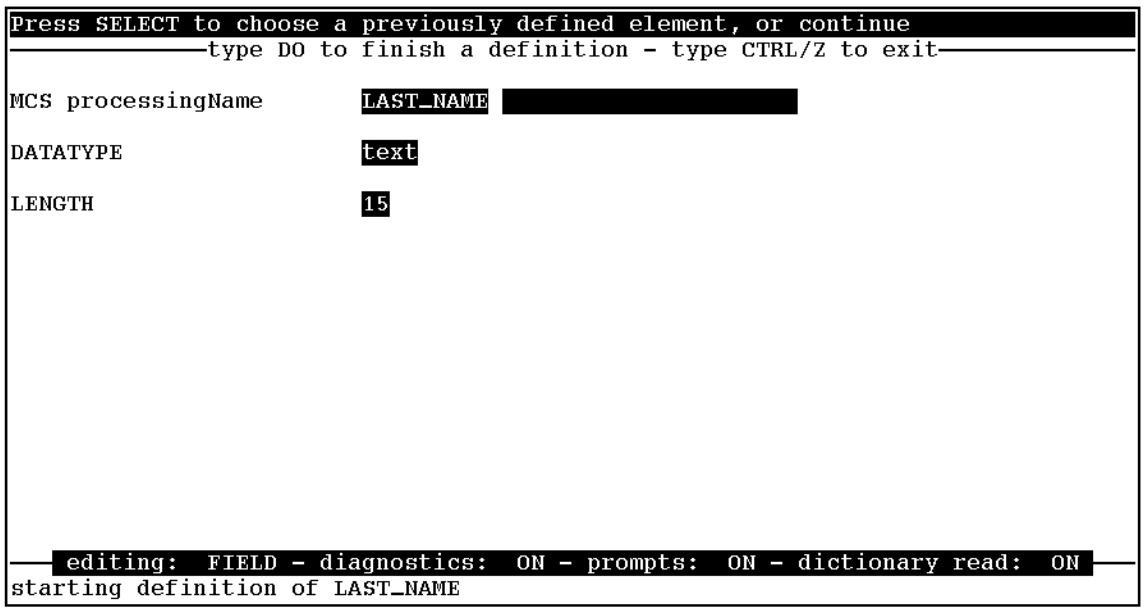


12. To add another definition, use the Delete key (the Backspace key) to delete FIRST_NAME from the MCS processingName field.
13. Type the field name MIDDLE_INIT.
14. Using the down arrow key, position the cursor on the value 12 in the LENGTH field. Change the text length to 1.
15. Press the Do key to enter the definition for the new field MIDDLE_INIT.



16. Add a third field definition by deleting the MIDDLE_INIT name from the MCS processingName field. Type the field name LAST_NAME. Change the text length to 15. You can change the data type and select other properties at this point; however, for this example, use the existing data type and property.

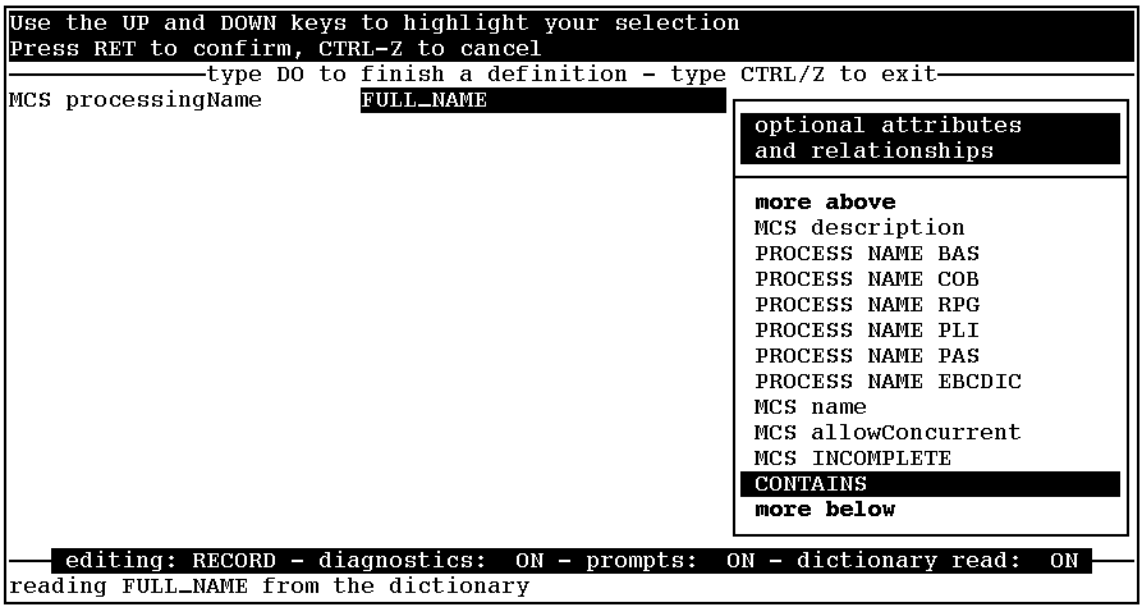
17. Press the Do key to enter the LAST_NAME field definition.



18. Next, create a record definition. Press the PF1 key followed by the KP8 key to switch the editing mode to define records. The PF1 KP8 key combination toggles between field definition and record definition mode.

19. Type the record definition name FULL_NAME in the MCS processingName field, and press the Return key.

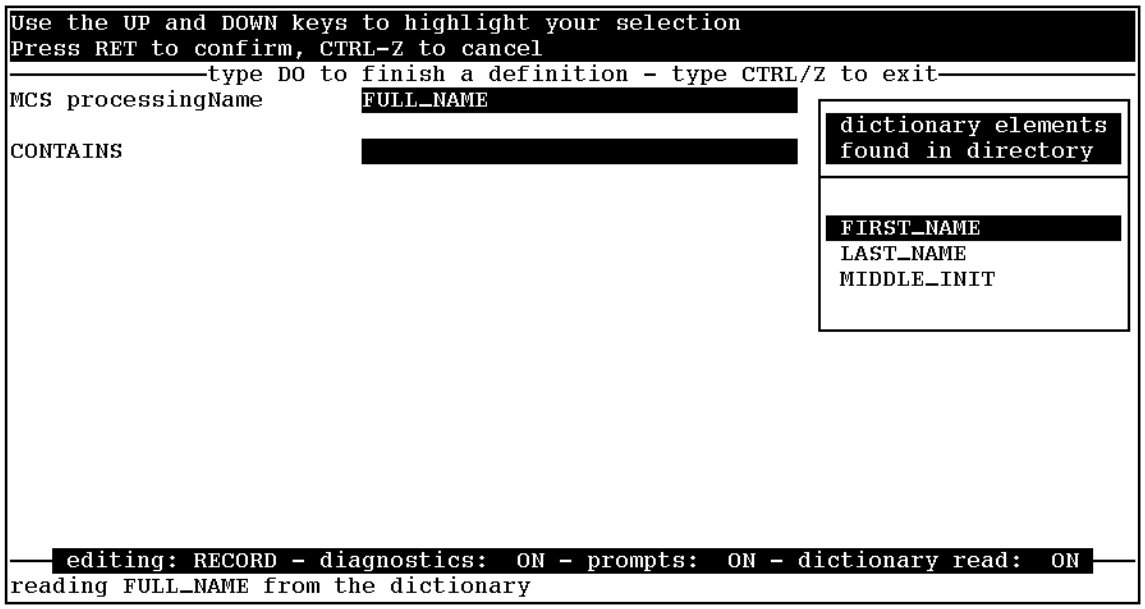
    CDO searches for an existing record definition by that name. If the record exists in the repository, the screen displays the definition. Because this is a new record, CDO displays a menu of record properties from which you can choose.



20. Use the arrow keys to highlight the CONTAINS property, and press the Return key.

21. CDO displays the CONTAINS property on the screen. You can now add fields to this record definition. Press the Select key to display a list of the current definitions in your repository directory.

22. Use the arrow keys to highlight the field FIRST_NAME, if it is not already highlighted.



23. Press the Return key to include the FIRST_NAME field in the record.

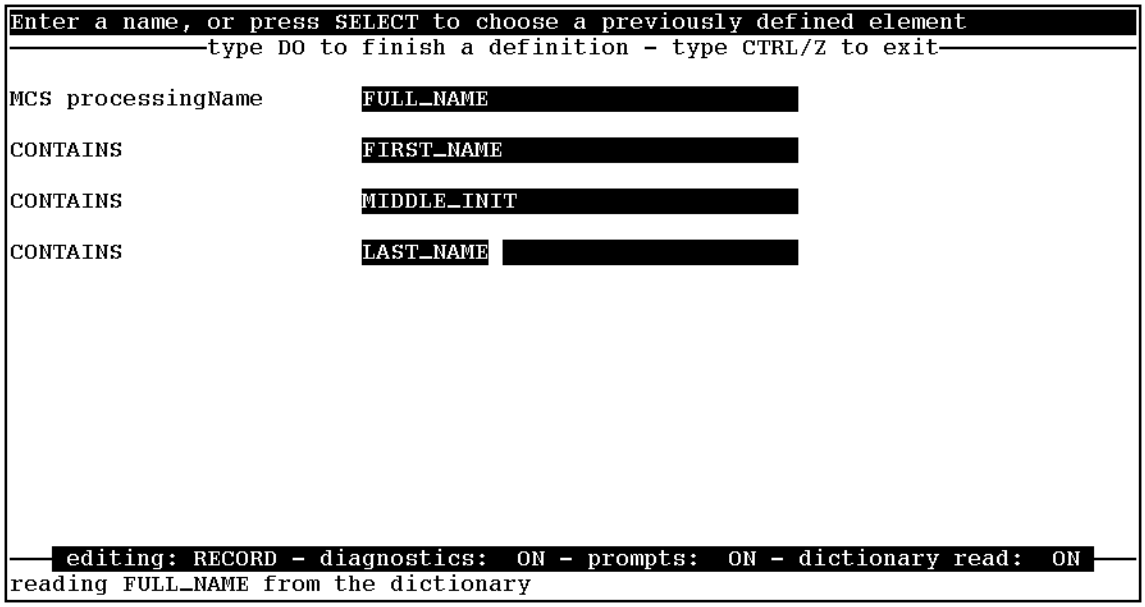
24. After adding FIRST_NAME, press Return again to display the menu of record properties. Select CONTAINS and press the Return key.

25. Press the Select key to display the list of existing field definitions. Select the MIDDLE_INIT field and press the Return key.

    After adding MIDDLE_INIT, press Return again to display the menu of record properties. Select CONTAINS and press the Return key.

26. Select the LAST_NAME field from the menu and press the Return key to add it to the record definition.

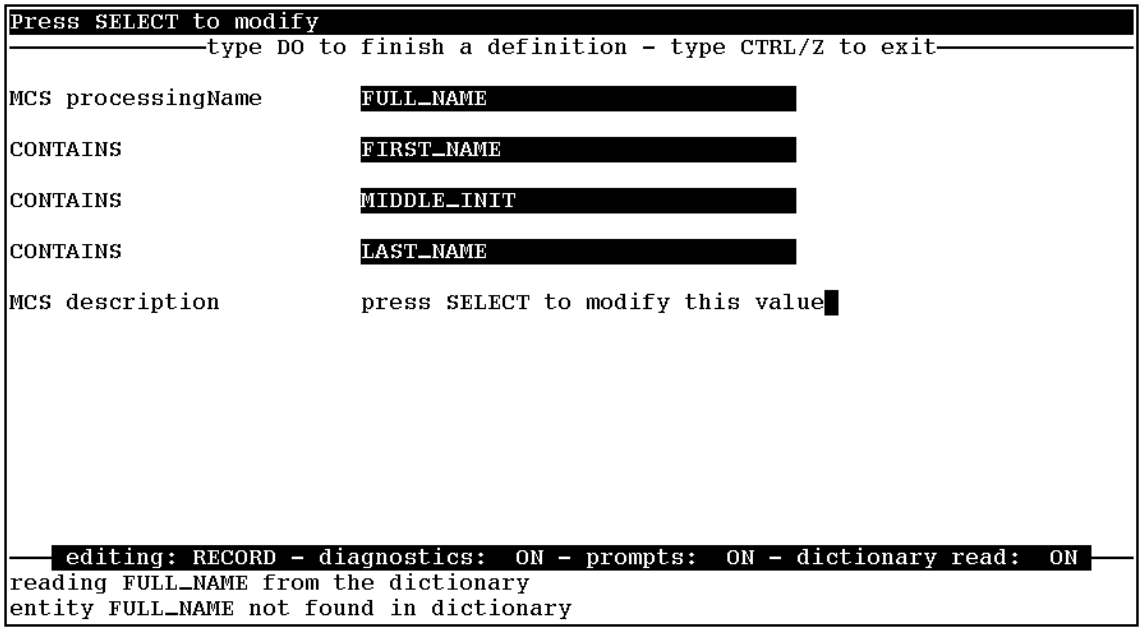
The screen will contain the following information:



27. Now, add a description for the record. Press the Insert Here key to display the menu of available properties.
28. Use the arrow keys to position the cursor on the `MCS description` property and press the Return key.
29. Press the Select key to access the text editor.

    Enter a description for the record definition. It is not necessary to add comment delimiters; CDO adds them for you. However, if the description for a record exceeds 1024 characters, use comment delimiters /* */ to separate the description; otherwise, it will be truncated.
30. Press Ctrl/Z to exit from the text editor.

The screen displays the record definition FULL_NAME. Notice that the comments you entered for the DESCRIPTION property do not appear on the screen; the description is only available through the text editor.



31. Press the Do key to enter the record definition for FULL_NAME in the repository. The editor displays the following success message:

```
FULL_NAME was successfully defined
```

32. Press Ctrl/Z to exit from the CDO screen editor.

33. To confirm that the EDITOR_EXAMPLES directory contains your definitions, enter the CDO DIRECTORY command, as follows:

```
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
```

34. To display more information about your definitions, use the CDO SHOW command. For example:

```
CDO> SHOW RECORD FULL_NAME
Definition of record FULL_NAME
|   Description             /* Full-time employee name */
|   Contains field          FIRST_NAME
|   Contains field          MIDDLE_INIT
|   Contains field          LAST_NAME

CDO>
```

35. Press Ctrl/Z to exit from CDO.

3
Managing a Repository

This chapter describes the administrative tasks necessary to set up and manage a repository-based application development environment. The following topics are discussed:

- backing up, verifying, and rebuilding repositories
- moving a repository
- deleting a repository
- upgrading a dictionary or repository

3.1 Backing Up Repositories

As part of repository maintenance, you should include a regular backup of the repository, in addition to the normal system backup procedure established at your site.

A full OpenVMS system backup and incremental backups do not capture metadata changes and are not adequate for repository backup. Your repository backup schedule should be based on the number of users and their requirements, and the amount and frequency of changes to the repository.

You perform a repository backup from the OpenVMS environment, not the CDO environment. Backing up a repository involves using the OpenVMS Backup utility (BACKUP) to back up the nondatabase files in the anchor directory, and then using the Oracle RMU Backup command to back up the database files.

An anchor specifies the OpenVMS directory where the repository is stored. The anchor can consist of node, device, and directory components. It can also be described in a logical name format, such as CDD$COMPATIBILITY. The anchor directory contains the CDD$DATABASE.RDB relational database file, as well as several files and subdirectories that make up the repository hierarchy.

The instructions in the following sections describe how you back up a repository by using both the OpenVMS BACKUP command and the Oracle RMU Backup commands. Both parts are critical for a valid backup. If you use only the OpenVMS BACKUP command for the anchor directory, including the database files, or only the Oracle RMU Backup command, the repository backup will not be valid.

Caution

Do not use the CDO VERIFY/REBUILD_DIRECTORY command with the Oracle RMU Backup and Oracle RMU Restore commands to perform a repository backup. The purpose of the VERIFY/REBUILD_DIRECTORY command is to rebuild the directory system in a repository. See Section 3.3.2 for information on rebuilding the directory system.

Do not use the REPOSITORY EXPORT command to back up your files. See Section 3.6 for information on using REPOSITORY EXPORT to upgrade a dictionary or repository.

3.1.1 Locating Repositories

To locate the repositories on your system, use the OpenVMS DCL DIRECTORY command and search each device for the CDD$DATABASE.RDB relational database file.

```
$  DIR disk:[000000...]CDD$DATABASE.RDB
```

If you have distributed repositories (linked repositories that share data), enter the following command to locate them:

- If you have Oracle CDD/Repository Version 5.*n* or higher installed, enter the CDO SHOW REPOSITORIES command:

  ```
  $ REPOSITORY OPERATOR
  CDO> SET DEFAULT disk:[anchor_dir]
  CDO> SHOW REPOSITORIES
  ```

- If you have Version 4.*n* installed, enter the CDO SHOW GENERIC command:

  ```
  CDO> SHOW GENERIC CDD$ANCHOR/FULL disk:[anchor_dir]CDD$PROTOCOLS.CDD$SELF
  CDO>
  ```

3.1.2 Before You Begin a Backup

When possible, back up the repository during a time when your system is least used. You can also temporarily change the protection on the repository anchor by using the CDO CHANGE PROTECTION command and allow access to only the database administrator.

To prohibit the use of the repository by other users, follow these steps:

1. Broadcast a message to notify users that you are preparing to back up the repository. This command requires OPER (operator) privilege.

   For example:

   ```
   $ REPLY/ALL/BELL "Repository backup - please exit the repository"
   ```

2. Issue the following command on each cluster for each repository you are about to back up to be sure that no one is accessing the repository or database:

   ```
   $ RMU/DUMP/USERS disk:[anchor_dir]CDD$DATABASE.RDB
   ```

   If you have linked repositories, prohibit the use of each repository to avoid inconsistencies between them.

3. Prevent users from starting up Oracle CDD/Repository by invoking the SYS$STARTUP:CDDSHUTDOWN.COM command procedure.

3.1.3 Regular Repository Backup

To perform a regular backup of your repository, follow these steps:

1. Back up the nondatabase components by using the OpenVMS BACKUP command. Do not create the backup file (.BCK) in the OpenVMS directory that contains the repository you are backing up. Either specify a different directory for the .BCK file, or set default to a different directory. This command requires SYSPRV to back up the repository files.

   ```
   $  BACKUP/VERIFY/EXCLUDE=(.RDA,.RDB,.SNP) -
   _$  disk:[anchor_dir...] disk:[different-dir]filename.BCK/SAVE
   ```

2. Back up your repository database using the Oracle RMU Backup command. Do not create the backup file (.RBF) in the OpenVMS directory that contains the repository you are backing up. Either specify a different directory, or set default to a different directory.

   ```
   $ RMU/BACKUP disk:[anchor_dir]CDD$DATABASE -
   _$ disk:[different-dir]filename.RBF
   ```

If you have linked repositories, perform backups at the same time to avoid inconsistencies between repositories.

Note

If you are backing up more than one repository, it is important to give the .BCK and .RBF backup files meaningful names.

If you are using Oracle CDD/Repository for configuration management, you may want to perform an image backup rather than a regular backup. An image backup backs up the entire contents of the device that contains your repository. A regular backup makes extra copies of any files (binaries) that are opened in contexts. As a result, more disk space is required during the restore operation. Refer to the OpenVMS documentation on the Backup utility for more information.

If all the components of the CDD$DATABASE files and the .RUJ files are not on the same disk as the image backup, you must also perform the Oracle RMU Backup operation.

3.1.4 Restoring a Repository from a Regular Backup

If you need to restore the repository from a backup, follow these steps:

1. Set default to an empty directory. Create a directory if one does not exist.

   ```
   $ SET DEFAULT disk:[new-anchor-dir]
   ```

   This will be your new anchor directory. The new anchor directory should not contain any files or subdirectories.

2. Restore the nondatabase components, as follows:

   ```
   $ BACKUP filename.BCK/SAVE_SET [...]/LOG/VERIFY
   ```

3. Restore the database, as follows:

   ```
   $ RMU/RESTORE/NOCDD/DIRECTORY=[]/NOAFTER_JOURNAL filename.RBF
   ```

4. Verify each repository. Refer to Section 3.3 for how to verify a repository.

If the repository you are restoring is linked to other repositories through external references, you must restore each repository. Be sure to restore the repositories from backups that were made at the same time, to prevent inconsistencies between repositories. Perform the verify operation *after* you have restored all the repositories.

3.2 Recovering from System Failure

Oracle CDD/Repository protects your repository automatically in the event of a system or network failure during modification of repository definitions. By default, Oracle CDD/Repository maintains OpenVMS RMS journal files (.CDDJNL) for all repository transactions until each transaction is complete. When a repository session completes without interruption, Oracle CDD/Repository deletes these journal files. When a session ends abnormally, Oracle CDD/Repository automatically uses the journal files to roll back to the start of the transaction that was interrupted.

Oracle CDD/Repository performs journal file recovery on the first command of a session and whenever you issue the CDO VERIFY command. Generally, you do not need to perform any maintenance or recovery operations on these journal files. However, if you have a problem accessing the repository, it can be caused by old, inactive journal files in the anchor directory. This is particularly true if you enter the first command of a session, then find after you enter subsequent commands that the problem has disappeared.

To correct the problem, rename these journal files, re-enter the command, and if the command succeeds, you can delete the journal files.

3.3 Verifying the Repository

The CDO VERIFY command scans your repository files to confirm that the repository is structurally correct. Perform a verify operation in the following situations:

- before and after you upgrade a repository
- after you restore a repository from a backup
- after you move a repository to a new location
- when an error message indicates the need for verification
- when you suspect a problem, such as after a system failure
- when you get errors from programs that access the repository
- when you notice unexpected behavior or missing information during repository activity

Use the CDO VERIFY/NOFIX command periodically to confirm the integrity of the repository. The frequency with which you verify repositories depends on your site's policies and the amount of repository activity at your site.

Use the CDO VERIFY/ALL/NOFIX command to verify repositories on a regular basis.

Note

In versions of Oracle CDD/Repository prior to Version 6.1, when you issued the VERIFY/ALL command, the /FIX qualifier was included by default. In Version 6.1 and higher, the default is /NOFIX. However, you can continue to use /FIX as the default by defining the CDD$VERIFY_ALL_FIX logical to be any value. Define the CDD$VERIFY_ALL_FIX logical at the process level or higher.

In addition, if you issue the VERIFY/ALL command without specifying a /FIX or a /NOFIX qualifier, and if you have not defined the CDD$VERIFY_ALL_FIX logical, a %CDO-I-VF_ALL_NOFIX informational message will be displayed immediately and the verify operation will continue.

If you issue the VERIFY command with the /FIX qualifier, make sure you have full backups of the repository and all other repositories that are linked to it *before* you attempt the fix.

For a full description of the VERIFY command and its qualifiers, see the *Oracle CDD/Repository CDO Reference Manual*.

To verify the structural condition of a repository, perform the following steps:

1. Before you issue the VERIFY command, prohibit all use of the repository. Refer to Section 3.1.2 for details.
2. Invoke the CDO utility, and enter the VERIFY/ALL/NOFIX command at the CDO prompt:

   ```
   $ REPOSITORY OPERATOR
   CDO> VERIFY/ALL/NOFIX disk:[anchor_dir]
   CDO> EXIT
   ```

   In some cases, it may be necessary to run VERIFY more than once. A VERIFY/ALL/FIX command should be followed by VERIFY/ALL/NOFIX to ensure that the repository is completely verified. If an error occurs after you use the /NOFIX qualifier, then run VERIFY/ALL/FIX again.
3. Review the results. If problems exist, correct them. For example, you may need to upgrade the repository if you have recently installed a new version of Oracle Rdb. Refer to Section 3.6 for upgrade instructions.

Most other problems can be corrected with VERIFY/ALL/FIX, as follows:

a. Confirm whether the repository you are verifying is linked to other repositories. Refer to Section 3.1.1 for information on how to locate linked repositories.

   Be sure that you upgrade all linked repositories.

b. Issue VERIFY/ALL/FIX for each repository. Wait for each VERIFY command to complete before issuing the next VERIFY command.

   Do *not* issue concurrent VERIFY commands against linked repositories. By default, VERIFY locks the repository it is verifying. A concurrent VERIFY will report errors when it attempts to access a repository that is locked.

c. After each VERIFY/ALL/FIX command completes, you can issue a VERIFY/ALL/NOFIX command to verify the structural condition of the repository. Again, wait for each VERIFY command to complete before issuing the next VERIFY command.

Restrictions

- The verify operation needs unrestricted access to a number of internal objects and structures to successfully complete its job. You must have SYSPRV or BYPASS privilege to run VERIFY/FIX.
- To use the VERIFY command, you need SHOW access to the repository. You also need SHOW access to any elements that you are verifying, such as cross-repository relationships.
- You cannot perform a remote verify operation.

3.3.1 Logging Repository Verification

During a verify operation, you can display information and error text on your default output device by including the /LOG qualifier when you issue the VERIFY command.

The following example shows the output from the VERIFY/LOG command. The output from this example indicates that there are no problems. The informational `CDD-I-SKIPTYPE` message indicates that the operation is not being performed for the given type. No user action is required.

```
$ REPOSITORY OPERATOR
CDO> VERIFY /ALL/NOFIX/LOG SYS$COMMON:[CDDPLUS.PERSONNEL]
%CDD-I-VFRECVR, VERIFY recovering journaled changes...
%CDD-I-NORECOVER, dictionary does not need recovery
%CDD-I-VFALL, VERIFY checking location, internal, and external references...
%CDD-I-OKLOC, dictionary SYS$COMMON:[CDDPLUS.PERSONNEL] location self-reference is okay
%CDD-I-SKIPTYPE, skipping protocol type MCS_ELEMENT_TYPE
%CDD-I-SKIPTYPE, skipping protocol type MCS_RELATION_TYPE
%CDD-I-SKIPTYPE, skipping protocol type MCS_METHOD
%CDD-I-SKIPTYPE, skipping protocol type MCS_MESSAGE
%CDD-I-SKIPTYPE, skipping protocol type MCS_MSGARG
%CDD-I-SKIPTYPE, skipping protocol type MCS_DATA_TYPE
%CDD-I-SKIPTYPE, skipping protocol type MCS_PROPERTY_TYPE
%CDD-I-SKIPTYPE, skipping protocol type CDD$LINK_TYPE
%CDD-I-SKIPTYPE, skipping protocol type MCS_VALIDATION
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_DEFAULT_METHOD
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_MESSAGE
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_MSGARG
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_DATATYPE
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_PREAMBLE
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_POSTAMBLE
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_RELATION
%CDD-I-SKIPTYPE, skipping protocol type MCS_RELATION_MEMBER
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_PROPERTY
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_RELATION_PROPERTY
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_LINK_PROPERTY
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_COMPUTED_PROPERTY
%CDD-I-SKIPTYPE, skipping protocol type CDD$HAS_LINK
%CDD-I-SKIPTYPE, skipping protocol type MCS_IMPLEMENTS_RELATION
%CDD-I-SKIPTYPE, skipping protocol type MCS_IMPLEMENTS_LINK
%CDD-I-SKIPTYPE, skipping protocol type MCS_IMPLEMENTS_METHOD
%CDD-I-SKIPTYPE, skipping protocol type MCS_HAS_SUPERTYPE
   .
   .
   .
```

3.3.2 Recovering the Directory System

The purpose of the VERIFY/REBUILD_DIRECTORY command is to rebuild *only* the directory system in a repository. The directory system does not include any database files with the extension .RDA, .RDB, or .SNP. It also does not include the subdirectories used for configuration management, namely CONTEXTS.DIR, DELTAFILES.DIR, PARTITIONS.DIR, and their contents.

Use the CDO VERIFY/REBUILD_DIRECTORY command when the directory entries for the repository anchor are so corrupt that all other VERIFY qualifiers fail.

VERIFY will ask if you are satisfied with your backup. Make sure you have an adequate backup of the repository before running the VERIFY/REBUILD_DIRECTORY command. The default answer to this question is NO. See Section 3.1 for the backup procedure.

If you issue VERIFY/REBUILD_DIRECTORY with the /LOG qualifier, Oracle CDD/Repository provides numerous informational messages about the rebuilding process.

The following is an example of using the VERIFY/REBUILD_DIRECTORY /LOG command:

```
CDO> VERIFY/REBUILD_DIRECTORY/LOG SYS$COMMON:[CDDPLUS.PERSONNEL]

Are you satisfied with the backup of your repository? [Y/N] (N): y

%CDD-I-VFREBLD, VERIFY rebuilding directory system...
%CDD-I-REBUILD, rebuilding directory structure for
  SYS$COMMON:[CDDPLUS.PERSONNEL]
%CDD-I-OKLOC, dictionary location SYS$COMMON:[CDDPLUS.PERSONNEL]
  self-reference is okay
CDO>
```

The VERIFY/REBUILD_DIRECTORY command first deletes all files in the anchor except the three subdirectories, DELTAFILES.DIR, PARTITIONS.DIR, CONTEXTS.DIR, and the database files. It then rebuilds the deleted files from the contents of the database. If any directory or database files are missing or inaccessible, the VERIFY/REBUILD_DIRECTORY operation will fail.

Restriction

You must have SYSPRV or BYPASS privilege to successfully run VERIFY /REBUILD_DIRECTORY.

3.3.3 Reducing the Snapshot File

A snapshot is a picture of the database for read-only transactions. To improve performance, reduce the size of the snapshot file by using the /COMPRESS qualifier with the VERIFY command, as follows:

```
CDO>  VERIFY/COMPRESS disk:[anchor_dir]
CDO>
```

Restrictions

The following restrictions apply to using the /COMPRESS qualifier:

- You must be the only user of the database at the time when you enter the command.
- VERIFY/COMPRESS must be the first command you enter after starting a CDO session; otherwise, CDO reports a conflict error with other users.

3.4 Moving a Repository

This section describes how to move a repository. Before you move a repository from one location to another, you need to consider several things:

- Is the old repository location on the same cluster as the new location?
- Are the versions of Oracle Rdb and Oracle CDD/Repository for the repository you are moving compatible with the versions on the new system?
- Is this a standalone repository or a distributed repository (a repository that is linked to other repositories through external references)?

  If this repository has external references, are you also moving the other repositories and will the other repositories be accessible through the network during and after the move?
- Are you moving DMU dictionary files?
- Does the account from which you are performing the move have SYSPRV or BYPASS privileges?

The following are some general guidelines for moving a repository. These guidelines will help you determine the correct method to use for your situation, and will make the operation easier.

Repository Locations

If you are moving a repository from one location on a cluster to another location on the same cluster, then use the CDO MOVE REPOSITORY command. See Section 3.4.1 for instructions on using the CDO MOVE REPOSITORY command.

If you are moving a repository from one system to another system that is not on the same cluster, you are performing a remote move operation. You cannot use the CDO MOVE REPOSITORY command for a remote move operation. See Section 3.4.2 for instructions on performing a remote move operation.

Oracle Rdb and Oracle CDD/Repository Versions

It is important to consider the versions of Oracle CDD/Repository and Oracle Rdb at both the source and target locations before you move a repository.

If you are moving a repository within the same cluster (using the CDO MOVE REPOSITORY command), this will not be a problem. But, if you are moving a repository from one system to another system that is not on the same cluster (performing a remote move operation), the system to which you are moving the repository cannot be running a version of Oracle CDD/Repository that is lower than the version of Oracle CDD/Repository where the repository currently

resides. Likewise, the system cannot be running a version of Oracle Rdb that is lower than the current version of the repository database.

However, you can move a single repository that has no external references from a system that is running a lower version of Oracle Rdb and Oracle CDD/Repository to a system that is running higher versions of Oracle Rdb and Oracle CDD/Repository. In that case, you must upgrade the repository after you move it. See the upgrade procedure in Section 3.6.

If you are moving one or more repositories that have external references to other repositories, there are some precautions and restrictions to consider. This information is provided in Section 3.4.2.

If you are not sure if the repository has external references, or if you want to display all repositories that a specific repository references, use one of the following commands:

- In Version 5.0 and higher, use the CDO SHOW REPOSITORIES command:

```
CDO> SET DEFAULT disk:[anchor_dir]
CDO> SHOW REPOSITORIES
```

- In Version 4.*n* use the CDO SHOW GENERIC command:

```
CDO> SHOW GENERIC CDD$ANCHOR/FULL disk:[anchor_dir]CDD$PROTOCOLS.CDD$SELF
```

DMU Dictionaries

The precautions and restrictions that apply to moving repositories described in Section 3.4.1 and Section 3.4.2 do not apply to moving DMU dictionary files. See Section 3.4.3 for instructions on how to move DMU dictionaries.

Privileges

With the exception of moving a DMU dictionary, most of the steps involved in either method for moving a repository require the account from which you perform the move operation to have SYSPRV or BYPASS privilege.

The CDO MOVE REPOSITORY command, the CDO VERIFY/LOCATION/FIX command, and both upgrade utilities (CDO CONVERT/REPOSITORY and CDDX) require SYSPRV or BYPASS privileges. Depending on the method you use, some of these commands will be executed during the move operation.

The following sections include detailed instructions for each method of moving a repository and the restrictions that apply to that method.

3.4.1 Moving a Repository to a Different Location on the Same Cluster

To move a repository to a different location on the same cluster, use the CDO MOVE REPOSITORY command. The CDO MOVE REPOSITORY command resolves all pointers to the old repository location to indicate the new location. The target location must be an OpenVMS directory that contains no files.

Note

When you use the CDO MOVE REPOSITORY command, you must specify the entire name, *including the device and directory name*, of both location and target files. If you specify only directory names, the move may not be successful.

The following command moves the repository at SYS$COMMON:[CDDPLUS] to DISK$1:[CORPORATE.MIS]:

```
CDO> MOVE REPOSITORY SYS$COMMON:[CDDPLUS]
cont> TO DISK$1:[CORPORATE.MIS].
CDO>
```

Restrictions

- When the repository you are moving contains elements that are related to elements in a remote repository, the CDO MOVE REPOSITORY command can resolve these references only if the network link is viable at the time of execution. If the network link is not viable, the MOVE REPOSITORY command fails and you must try the operation again.
- The move repository operation needs unrestricted access to a repository because it will automatically perform a CDO VERIFY/LOCATION /FIX operation. The account from which you issue the CDO MOVE REPOSITORY command must have SYSPRV or BYPASS privilege, which is required for the verify operation; otherwise, CDO will issue an error message and abort the move repository operation.
- The CDO MOVE REPOSITORY command is not supported for moving a repository to a node that is not in the same cluster (a remote move operation).
- Both the source and the target disk must be accessible to the system you are logged in to for the move operation.

3.4.2 Moving a Repository from One System to Another System Not on the Same Cluster

This section describes how to move a repository from one system to another system that is not on the same cluster. This is considered a remote move operation. It is recommended that you review all the restrictions in Section 3.4.2.6 before you perform this operation. You should also review the information in Chapter 6, which describes remote access setup, operations, and troubleshooting tips.

How Distributed Repositories Are Affected by a Remote Move

There are basically four different methods to perform a remote move. (These methods are described in detail later in this section.) Each method involves using the combined OpenVMS and Oracle RMU Backup procedure, copying or loading from media, and using OpenVMS and Oracle RMU restore operations. In some situations, these steps are followed by the CDO VERIFY command. In other situations, such as when you are moving distributed repositories, the CDO VERIFY command cannot be used and you must manually change the external references in the CDD$DATABASE.RDB file using SQL.

Oracle CDD/Repository stores the location of a repository's self-reference and any external repositories that the repository references, inside the CDD$DATABASE.RDB file. If all repositories linked to each other through external references are not accessible through the network during or after the move operation, or if distributed repositories have been moved simultaneously, there is no way to resolve the external references between them through the CDO VERIFY command.

If the verify operation cannot find the referenced repositories where it expects to find them (because they have been moved, are not on the network, or network proxy accounts are not set up correctly), verify removes the external references. This makes objects that depended on these references unusable. There is no way to relink the objects after the link has been broken. Use caution when you move distributed repositories.

Version Number Requirements for a Remote Move

The version of Oracle Rdb on the target system must be at the same version or at a later version than that of the source system.

If you are moving distributed repositories, then all the repositories that are linked must be at compatible versions of Oracle CDD/Repository. See Table 3–2 for a list of compatible versions.

Use Table 3–1 to determine the correct solution for your remote move operation.

Table 3–1 Remote Move Operation Choices

Distributed Repository Has External References?	Moving Multiple Repositories?	Move Solution	Proxy Account Required?
No	n/a	1	No
Yes	No	2	Yes
Yes	Yes (individually)	3	Yes
Yes	Yes (simultaneously)	4	No

The following sections provide step-by-step instructions for each move solution.

3.4.2.1 Solution 1—Moving a Single Repository with No External References

Use this solution to move a single repository that has no external references to any other repositories.

1. Make a backup of the repository you are moving, using the combined OpenVMS and the Oracle RMU Backup procedure, as described in Section 3.1.
2. Move the two backup files (.RBF and .BCK) to the new system. If both the source and target systems are accessible on the network, use the DCL COPY command or the FTSV SPOOL COPY command. Otherwise, load the files onto media and unload them at the new system.
3. Restore the database backup file (.RBF) to the new location, which should be an empty directory.

   ```
   $ RMU/RESTORE/NOCDD/DIRECTORY=new_dev:[new_anchor_dir]/NOAFTER_JOURNAL -
   _$ backup_file.RBF
   ```

   Press the Return key in response to any prompts for storage area information.
4. Restore the OpenVMS backup file (.BCK) to the new location, which should be the same directory into which you restored the corresponding .RBF file.

   ```
   $ BACKUP/LOG backup_file.BCK/SAVE new_dev:[new_anchor_dir]
   ```
5. Issue the CDO VERIFY/LOCATION/FIX command to verify the repository in its new location.

6. Run the SYS$LIBRARY:CDD$UPGRADE.COM command procedure if you need to upgrade the repository.

3.4.2.2 Solution 2—Moving a Single Distributed Repository

Use this solution to move a single, distributed repository.

Caution

Before you begin this procedure, carefully review Section 3.4.2.5 to ensure that all proxy accounts are properly set up and functioning as expected, login command procedures on default proxy accounts are correct, the versions of Oracle CDD/Repository and Oracle Rdb are compatible, and the test sample has run successfully. Otherwise, the verify operation could remove external references rather than update them.

To use this solution, all repositories must be available through network proxy accounts during the move operation so that the verify operation can fix the external references. All repositories must be available through network proxy accounts after the move for the repositories to continue to share data. See Section 3.4.2.5 before you begin.

Perform the following steps:

1. Make a backup of the repository you are moving, using the combined OpenVMS and Oracle RMU Backup procedure, as described in Section 3.1.

   It is recommended that you also make backups of all referenced repositories, as a precaution.

2. Move the two backup files (.RBF and .BCK) to the new system.

   If both the source and target systems are accessible on the network, use the DCL COPY command or the FTSV SPOOL COPY command. Otherwise, load the files onto media and unload them at the new system.

3. Restore the database backup file (.RBF) to the new location, which should be an empty directory.

   ```
   $ RMU/RESTORE/NOCDD/DIRECTORY=new_dev:[new_anchor_dir]/NOAFTER_JOURNAL -
   _$ backup_file.RBF
   ```

   Press the Return key in response to any prompts for storage area information.

4. Restore the OpenVMS backup file (.BCK) to the new location, which should be the same directory into which you restored the .RBF file.

   ```
   $ BACKUP/LOG backup_file.BCK/SAVE new_dev:[new_anchor_dir]
   ```

5. Issue the CDO VERIFY/LOCATION/EXTERNAL_REFERENCES/FIX command to verify the repository in its new location.
6. Run the SYS$LIBRARY:CDD$UPGRADE.COM command procedure if you need to upgrade the repository.
7. From the system of the old location, use the DCL DELETE command to delete the repository that you moved. Do not use the CDO DELETE REPOSITORY command. Deleting the repository from the old location saves disk space and prevents accidental use of the invalid repository.

3.4.2.3 Solution 3—Moving Multiple Distributed Repositories (Individually)

Use this solution to move multiple, distributed repositories. Move one repository at a time.

Caution

Before you begin this procedure, carefully review Section 3.4.2.5 to ensure that all proxy accounts are properly set up and functioning as expected, login command procedures on default proxy accounts are correct, the versions of Oracle CDD/Repository and Oracle Rdb are compatible, and the test sample has run successfully. Otherwise, the verify operation could remove external references rather than update them.

To use this solution, all repositories must be available through network proxy accounts during the move operation so that the verify operation can fix the external references. All repositories must be available through network proxy accounts after the move for the repositories to continue to share data. See Section 3.4.2.5 before you begin.

If the repositories are not all available on the network during the move operation, you must use Move Solution 4.

Perform the following steps for each repository being moved, *one repository at at time.*

1. Make a backup of the first repository you are moving, using the combined OpenVMS and Oracle RMU Backup procedure, as described in Section 3.1.
2. Move the two backup files (.RBF and .BCK) to the new system.

If both the source and target systems are accessible on the network, use the DCL COPY command or the FTSV SPOOL COPY command. Otherwise, load the files onto media and unload them at the new system.

3. Restore the database backup file (.RBF) to the new location, which should be an empty directory.

   ```
   $ RMU/RESTORE/NOCDD/DIRECTORY=new_dev:[new_anchor_dir]/NOAFTER_JOURNAL -
   _$ backup_file.RBF
   ```

   Press the Return key in response to any prompts for storage area information.

4. Restore the OpenVMS backup file (.BCK) to the new location, which should be the same directory into which you restored the .RBF file.

   ```
   $ BACKUP/LOG backup_file.BCK/SAVE new_dev:[new_anchor_dir]
   ```

5. Issue the CDO VERIFY/LOCATION/EXTERNAL_REFERENCES/FIX command to verify the repository in its new location.

6. Run the SYS$LIBRARY:CDD$UPGRADE.COM command procedure if you need to upgrade the repository.

7. From the system of the old location, use the DCL DELETE command to delete the repository that you moved. Do not use the CDO DELETE REPOSITORY command. Deleting the repository from the old location saves disk space and prevents accidental use of the invalid repository.

8. Perform steps 1 through 7 for each repository separately, repeating the steps for every repository that you move.

3.4.2.4 Solution 4—Moving Multiple Distributed Repositories (Simultaneously)

Use this solution to simultaneously move multiple, distributed repositories.

Caution

Use this solution with caution and only when necessary. Follow the steps in this procedure exactly; otherwise the repository will not be usable.

To use this solution, all repositories must be at compatible versions of Oracle CDD/Repository. See Table 3–2 for a list of compatible versions.

Perform the following steps:

1. Make a backup of each repository you are moving, using the combined OpenVMS and Oracle RMU Backup procedure, as described in Section 3.1.

Be sure to give meaningful names to each pair of backup files (.BCK and .RBF), so you can identify them when you copy them to the new system.

2. Move all pairs of backup files (.RBF and .BCK) to the new system.

   If both the source and target systems are accessible on the network, use the DCL COPY command or the FTSV SPOOL COPY command. Otherwise, load the files onto media and unload them at the new system.

3. Restore the database backup files (.RBF) to their new locations, which must be empty directories.

   ```
   $ RMU/RESTORE/NOCDD/DIRECTORY=new_dev:[new_anchor_dir]/NOAFTER_JOURNAL -
   _$ backup_file.RBF
   ```

   Press the Return key in response to any prompts for storage area information.

4. Restore the OpenVMS backup files (.BCK) to their new locations, which should be the same directories into which you restored the .RBF files.

   ```
   $ BACKUP/LOG backup_file.BCK/SAVE new_dev:[new_anchor_dir]
   ```

   Do not issue any CDO VERIFY commands at this time; otherwise, all external references will be lost.

5. Manually modify each repository's self-reference and external references by directly accessing the CDD$DATABASE.RDB file through SQL, and changing the old node and anchor directory locations to the new node and anchor directory locations.

   Note

   The syntax shown in the following steps is critical for a successful modification and should be followed exactly.

   Perform the following steps:

   a. Invoke SQL and enter the following commands:

      ```
      $ SQL
      SQL> ATTACH 'FILENAME NEW_DISK:[NEW_ANCHOR_DIR_1]CDD$DATABASE';
      SQL> SET TRANSACTION READ WRITE;
      ```

b. Display a list of all repositories referenced in this repository:

```
SQL> SELECT CDD$$A_NAD_NODE_NAME ,
cont>  CDD$$A_NAD_ANCHOR_NAME FROM CDD$$O_ANCHOR ;
 CDD$$A_NAD_NODE_NAME
   CDD$$A_NAD_ANCHOR_NAME
 OLD_NODE::
   OLD_DISK:[OLD_ANCHOR_DIR_1]
   >>
   >>
   >>
 OLD_NODE::
   OLD_DISK:[OLD_ANCHOR_DIR_2]
   >>
   >>
   >>
 OLD_NODE::
   OLD_DISK:[OLD_ANCHOR_DIR_3]
   >>
   >>
   >>
3 rows selected
SQL>
```

The first repository listed is always this repository's self-reference. In this example it is:

```
OLD_DISK:[OLD_ANCHOR_DIR_1]
```

c. Modify the repository's self-reference by carefully changing each location from the old location to the new location. Include the node and directory specification *exactly* as it appears in the listing. Use all uppercase letters, use the double colons (::) after the node name, and put all names (both the old and new) within single quotes, as shown.

```
SQL> UPDATE CDD$$O_ANCHOR
SQL> SET CDD$$A_NAD_NODE_NAME = 'NEW_NODE::',
cont> CDD$$A_NAD_ANCHOR_NAME = 'NEW_DISK:[NEW_ANCHOR_DIR_1]'
cont> WHERE CDD$$A_NAD_ANCHOR_NAME = 'OLD_DISK:[OLD_ANCHOR_DIR_1]';
```

d. Modify the external references. In this step, the two external references are being moved simultaneously from:

   OLD_NODE::OLD_DISK:[OLD_ANCHOR_DIR_2] and
   OLD_NODE::OLD_DISK:[OLD_ANCHOR_DIR_3] to
   NEW_NODE::NEW_DISK:[NEW_ANCHOR_DIR_2] and
   NEW_NODE::NEW_DISK:[NEW_ANCHOR_DIR_3], respectively.

```
SQL> UPDATE CDD$$O_ANCHOR
SQL> SET CDD$$A_NAD_NODE_NAME = 'NEW_NODE::',
cont>  CDD$$A_NAD_ANCHOR_NAME = 'NEW_DISK:[NEW_ANCHOR_DIR_2]'
cont> WHERE CDD$$A_NAD_ANCHOR_NAME = 'OLD_DISK:[OLD_ANCHOR_DIR_2]';

SQL> UPDATE CDD$$O_ANCHOR
SQL> SET CDD$$A_NAD_NODE_NAME = 'NEW_NODE::',
cont>  CDD$$A_NAD_ANCHOR_NAME = 'NEW_DISK:[NEW_ANCHOR_DIR_3]'
cont> WHERE CDD$$A_NAD_ANCHOR_NAME = 'OLD_DISK:[OLD_ANCHOR_DIR_3]';
```

e. Display a list again of all the repositories referenced in this repository; check that they were updated correctly before you commit the transaction.

```
SQL> SELECT CDD$$A_NAD_NODE_NAME ,
cont> CDD$$A_NAD_ANCHOR_NAME FROM CDD$$O_ANCHOR ;
 CDD$$A_NAD_NODE_NAME
   CDD$$A_NAD_ANCHOR_NAME
 NEW_NODE::
   NEW_DISK:[NEW_ANCHOR_DIR_1]
   >>
   >>
   >>
 NEW_NODE::
   NEW_DISK:[NEW_ANCHOR_DIR_2]
   >>
   >>
   >>
 NEW_NODE::
   NEW_DISK:[NEW_ANCHOR_DIR_3]
   >>
   >>
   >>
3 rows selected
SQL>
SQL> COMMIT;
SQL> DISCONNECT DEFAULT;
SQL> EXIT
```

6. Repeat steps a through e for all distributed repositories that you have moved.

Do not issue any CDO VERIFY commands in any of these repositories until after all repositories have been moved and modified.

7. After you have modified the CDD$DATABASE.RDB file for all the repositories you have moved, issue the CDO VERIFY/LOCATION /EXTERNAL_REFERENCES/NOFIX command to check that you have correctly updated all the location pointers in all the repositories.

   Do not use the /FIX qualifier if there are errors in the pointers. Rather, invoke SQL again and check your modifications. Be sure you have entered the names in uppercase letters, used the double colon (::), and single quotes where needed. Make any necessary corrections, exit SQL, and retry the CDO VERIFY/LOCATION/EXTERNAL_REFERENCES/NOFIX command.

3.4.2.5 Distributed Repository Move Operations Requiring Remote Network Proxy Access

This section applies if you plan to use Move Solution 2 and Move Solution 3. Both of these move solutions require that all repositories be available through network proxy accounts during, and possibly after, the move.

Before you use Move Solution 2 or Move Solution 3, perform the steps described in this section to prepare for the move operation. You should also review the instructions for remote access setup described in Chapter 6. After you perform these steps, run the remote move test sample in Example 3–1. Resolve all problems in the test move before you move your repositories.

1. Check version number compatibility

   Oracle CDD/Repository Version 5.3-08 (ECO) and higher versions include several corrections and enhancements that affect remote repository access. These changes make Version 5.3-08 (ECO) and higher incompatible for remote access with lower versions of Oracle CDD/Repository. Therefore, if one of the distributed repositories involved in the move is at either Version 5.3-08 or higher, then all the repositories must be at Version 5.3-08 or higher.

   This is an important consideration if you are moving a repository from a system that is running a lower version of Oracle CDD/Repository (the source) than the version of Oracle CDD/Repository on the system to which you are moving the repository (the target). See Table 3–2 for the compatibility requirements for versions of Oracle CDD/Repository.

Table 3–2 lists the version of Oracle CDD/Repository that is required on the target, based on the version of Oracle CDD/Repository at the source.

Table 3–2 Oracle CDD/Repository Version Compatibility

Version at Source	Version Required at Target
V4.*n*	V4.*n*
V5.0	V5.0
V5.1 - V5.3-06 (ECO)	V5.1 - V5.3-06 (ECO)
V5.3-08 (ECO) or later	V5.3-08 (ECO) or later

2. Set up proxy accounts

   Default proxy accounts must be set up on each system involved in the remote access. Issue the CDO SHOW REPOSITORIES command to display all the repositories that a specific repository references.

   You must set up a proxy account for each of the repositories that was displayed by the SHOW REPOSITORIES command. You will also need to set up proxy accounts for the system (or systems) to which you are moving the repository.

   For information on setting up proxy accounts, invoke the OpenVMS Authorize utility and type HELP ADD/PROXY. Be sure you use the /DEFAULT qualifier when you set up these proxy accounts.

3. Check LOGIN.COM files

   The LOGIN.COM command procedures for the default proxy accounts that you set up will be invoked during any remote operation. Therefore, you must examine all of the LOGIN.COM files to ensure that adequate privileges and other setup requirements are run automatically. For example, if your remote system has an Oracle Rdb multiversion environment, a specific version is required to activate the proper Oracle RMU and Oracle Rdb images on that system.

Remote Move Operation Test Sample

Example 3–1 creates two empty repositories on a Version 5.3-09 source node (the old node), links them by creating external references to each other, moves one of the repositories from the source node to a target node that is running Version 6.1 (the new node), verifies, and then upgrades the moved repository.

It is recommended that you use the sample in Example 3–1 to test a remote move operation before you move your repositories.

Be sure that you set up your remote default proxy accounts and verify that the LOGIN.COM files at these accounts are correct for remote access.

Substitute the actual location and anchor name of the old node for OLD_NODE /OLD_ANCHOR in the sample, and your new node location and anchor name for NEW_NODE/NEW_ANCHOR.

If errors occur when you perform the verify operation (during remote access), review the section on remote setup errors in Chapter 6 for troubleshooting tips.

This sample contains two repositories that are distributed and shows how to move one of the repositories to a remote node. The systems are set up as follows:

Source node = OLD_NODE
Oracle CDD/Repository = Version 5.3-09
Oracle Rdb = Version 5.1

Remote target node = NEW_NODE
Oracle CDD/Repository = Version 6.1
Oracle Rdb = Version 6.1

Example 3–1 Remote Move Operation Test Sample

```
$ CREATE/DIRECTORY OLD_DISK:[OLD_ANCHOR_1]
$ CREATE/DIRECTORY OLD_DISK:[OLD_ANCHOR_2]
$ REPOSITORY OPERATOR
Welcome to CDO V2.3
The CDD/Repository V5.3 User Interface
Type HELP for help
CDO> DEFINE REPOSITORY OLD_DISK:[OLD_ANCHOR_1].
CDO> DEFINE REPOSITORY OLD_DISK:[OLD_ANCHOR_2].

CDO> SET DEFAULT OLD_DISK:[OLD_ANCHOR_1]
CDO> DEFINE FIELD FLD_1 DATATYPE TEXT SIZE 25.

CDO> SET DEFAULT OLD_DISK:[OLD_ANCHOR_2]
CDO> DEFINE RECORD REC_1. OLD_DISK:[OLD_ANCHOR_1]FLD_1. END.

CDO> SET DEFAULT OLD_DISK:[OLD_ANCHOR_1]
CDO> SHOW REPOSITORY
```

(continued on next page)

Example 3–1 (Cont.) Remote Move Operation Test Sample

```
 The following REPOSITORIES are referenced:

    OLD_NODE::OLD_DISK:[OLD_ANCHOR_2]

CDO> SET DEFAULT OLD_DISK:[OLD_ANCHOR_2]
CDO> SHOW REPOSITORY

 The following REPOSITORIES are referenced:

    OLD_NODE::OLD_DISK:[OLD_ANCHOR_1]

CDO> EXIT
$
```

Move OLD_DISK:[OLD_ANCHOR_2] to the remote target node, NEW_NODE, using the following steps:

1. From the source node, OLD_NODE, make backups of OLD_DISK:[OLD_ANCHOR_2].

   ```
   $ RMU/BACKUP OLD_DISK:[OLD_ANCHOR_2]CDD$DATABASE V53_DICT2.RBF
   $ BACKUP/EXCLUDE=(.SNP,.RDB,.RDA)
   _From: OLD_DISK:[OLD_ANCHOR_2...]
   _To:   V53_DICT2.BCK/SAV
   ```

2. From the target node, copy the two backup files to NEW_NODE.

   ```
   $ COPY
   _From: OLD_NODE::OLD_DISK:[OLD_LOCATION]V53_DICT2.RBF,V53_DICT2.BCK
   _To:   NEW_NODE::NEW_DISK:[NEW_LOCATION]*
   ```

3. Restore the two backup files into an empty directory on NEW_NODE.

   ```
   $ CREATE/DIRECTORY NEW_DISK:[NEW_ANCHOR_2]

   $ RMU/RESTORE/NOCDD/DIR= NEW_DISK:[NEW_ANCHOR_2] /NOAFTER_JOURNAL
   _Backup: V53_DICT2.RBF
   _Storage_area:
   %RMU-I-LOGCONVRT, database root converted to current structure level
   %RMU-S-CVTDBSUC, database NEW_DISK:[NEW_ANCHOR_2]CDD$DATABASE.RDB;1
     successfully converted from version V5.1 to V6.1
   %RMU-I-CVTCOMSUC, CONVERT committed for
     NEW_DISK:[NEW_ANCHOR_2]CDD$DATABASE.RDB;1 to version V6.1
   ```

```
$ BACKUP/LOG V53_DICT2.BCK/SAV NEW_DISK:[NEW_ANCHOR_2]
%BACKUP-S-CREATED, created NEW_DISK:[NEW_ANCHOR_2]00000000.30000000;1
%BACKUP-S-CREATED, created
   NEW_DISK:[NEW_ANCHOR_2]30000000CDD$PROTOCOLS.40000000;1
%BACKUP-S-CREATED, created
   NEW_DISK:[NEW_ANCHOR_2]CDD$DIRECTORY.CDD;1
%BACKUP-S-CREDIR, created directory
   NEW_DISK:[NEW_ANCHOR_2.CONTEXTS]
%BACKUP-S-CREDIR, created directory
   NEW_DISK:[NEW_ANCHOR_2.DELTAFILES]
%BACKUP-S-CREDIR, created directory
   NEW_DISK:[NEW_ANCHOR_2.PARTITIONS]
```

Caution

The default proxy accounts *must* already be set up at this point for NEW_NODE on OLD_NODE and for OLD_NODE on NEW_NODE. The LOGIN.COM files for the default account must be correct for remote access. If the proxy accounts are not correctly in place, performing the verify operation can delete the external references rather than update them.

4. From NEW_NODE, invoke CDO and fix the location and external references using the CDO VERIFY command.

```
$ REPOSITORY OPERATOR
CDO> VERIFY/LOCATION/EXTERNAL_REFERENCES/FIX NEW_DISK:[NEW_ANCHOR_2]
CDO> EXIT
```

5. From NEW_NODE, perform an upgrade from Version 5.3-09 to Version 6.1.

   Use the SYS$LIBRARY:CDD$UPGRADE.COM command procedure, as shown, or use the CONVERT/REPOSITORY comand.

   If you use the CONVERT/REPOSITORY command rather than execute the SYS$LIBRARY:CDD$UPGRADE.COM command procedure, be sure that the CONVERT/REPOSITORY command is the first command you issue in the next CDO session.

```
$ @SYS$LIBRARY:CDD$UPGRADE

                    CDD$UPGRADE.COM

This procedure is used to upgrade repositories that were created
under a previous version of Oracle CDD/Repository.

Please enter the OpenVMS anchor of the repository to be upgraded.

DEVICE:[DIRECTORY] : NEW_DISK:[NEW_ANCHOR_2]

Are you satisfied with the backup of your repository? [Y/N] (N): Y

------------------------------------------------------
 What version are you upgrading this repository from?

  (If you are only exporting the repository, what
  version is the repository you are exporting?)

  Note: Upgrade is not required from V6.1 to V7.0.

     [ V50 | V51 | J51 | V53 | V61 | V70 ]
------------------------------------------------------
  V53

There are no more questions.

Converting repository NEW_DISK:[NEW_ANCHOR_2] to V6.1...

%CDO-I-UPGRADE_SUCCEED, dictionary successfully upgrade to new protocols

CDD$UPGRADE completed successfully.
```

3.4.2.6 Restrictions for Moving a Repository to Another System

The following restrictions apply if you are moving a repository from one system to another system that is not on the same cluster:

- Moving a repository to a different location that is not on the same cluster (performing a remote move operation) is not supported by the CDO MOVE REPOSITORY command.
- You need to perform additional steps if the repository you are moving contains the following:
  - binary objects (instances of MCS_BINARY type and its subtypes), where the MCS_storeType property has a value of MCS_STORETYPE_EXTERNAL

    Use the CDO DIRECTORY/TYPE=MCS_BINARY command to determine if you have binary types in your repository.
  - user-defined methods (instances of the MCS_METHOD type), where the MCS_funcType property has a value of MCS_METHOD_EXTERNAL_CODE

Use the CDO commands SHOW FILE_ELEMENT MCS_BINARY and SHOW FILE_ELEMENT MCS_METHOD to locate binary and method objects in your repository.

Perform these additional steps:

1. Move the referenced files to the system where you moved the repository.
2. If you are using a different directory hierarchy on your new system, it may be necessary to change the value of the MCS_storedIn property to point to the new file location. Use the CDO CHANGE FILE_ELEMENT command to change the values. In addition, if the referenced file is an executable image and you are moving the repository from an OpenVMS VAX system to an OpenVMS Alpha system, you must port the image.
3. If you are moving a repository from an OpenVMS VAX system to an OpenVMS Alpha system, you must port images that contain external method code. (The images that must be ported are specified as the value of the MCS_application property.)

- You must have the same values for the rights identifiers CDD$SYSTEM and CDD$EXTENDER on both systems. In addition, to ensure you have the same access on both systems, the RIGHTSLIST.DAT files should contain the same values for all identifiers and owner UICs used on repository objects.

  Check the value of the CDD$SYSTEM identifier from a privileged account on both the source and target machines. Use the following commands:

```
$ SET DEFAULT SYS$SYSTEM
$ RUN AUTHORIZE
UAF> SHOW /ID CDD$SYSTEM
```

Caution

Do not change the value for CDD$SYSTEM if there are viable repositories on the target system; otherwise, these repositories will have privilege access problems. If there are other repositories on the system that rely on the current value of CDD$SYSTEM, then you must change the protection on the repositories you have moved after you move them. For information on correcting repository protection, see Chapter 7.

If the hex value for CDD$SYSTEM is not the same on both machines, you must change it or modify the protection on the repository after you move it.

If the target node has no existing repositories that are being preserved, the CDD$SYSTEM identifier can be removed and replaced with the value from the source system. To do this, use the following commands:

```
UAF> REMOVE /ID CDD$SYSTEM
UAF> ADD /ID CDD$SYSTEM /VALUE=IDENTIFIER:%Xhex_value
```

Add only the last five characters of the hex value. The beginning 800 number is not used.

- The version of Oracle Rdb on the target system must be the same version or a later version than that of the source system.

  All distributed repositories must use compatible versions of Oracle CDD/Repository. See Section 3.4.2.5.

3.4.3 Moving a DMU Dictionary to Another System

Perform the following steps to move a DMU dictionary from one system to another system:

1. Locate any subdictionaries that are part of your logical dictionary structure. On the system from which you are moving the dictionary, use the following DMU LIST/TYPE=SUBDICTIONARY command:

   ```
   $ MCR DMU
   DMU> SET DEFAULT CDD$TOP
   DMU> LIST/TYPE=SUBDICTIONARY CDD$TOP>
   ```

   The listing includes the OpenVMS file specification of all subdictionaries. For example:

   ```
      .
      .
      .
   PERSONNEL <SUBDICTIONARY>: DB3:[CASADAY.CDD]PERS.DIC
      .
      .
      .
   DMU> EXIT
   ```

2. Copy the CDD$DICTIONARY:CDD.DIC dictionary from the old system to the new system using the DCL COPY command.
3. Change the definition of the CDD$DICTIONARY logical name in the SYS$STARTUP:CDDSTRTUP.COM procedure to point to the directory on the new system where you placed the dictionary file.
4. Copy all subdictionaries (that were listed when you performed step 1) from the old system to the new system using the DCL COPY command.

Be sure to place the subdictionary files in the same location within the OpenVMS directory structure on the new system as on the old system.

Otherwise, you must rename each subdictionary you copy to point to the new location, as follows:

```
DMU> RENAME/SUBDICTIONARY=file-specification path-name
```

For example:

```
DMU> RENAME/SUBDICTIONARY=DISK7:[DICT.CDD]PERS.DIC PERSONNEL
```

3.5 Deleting a Directory or an Entire Repository

Use the CDO DELETE DIRECTORY command to delete an empty repository directory.

For example:

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

Use the CDO DELETE REPOSITORY command to delete an entire repository. For example:

```
CDO> DELETE REPOSITORY DISK1:[SMITH.TEST].
```

The CDO DELETE REPOSITORY command allows you to delete all the elements in a repository, and to delete the repository itself. You must have DELETE privilege to the repository.

Caution

When Oracle CDD/Repository deletes a complete repository, all files in the anchor directory are deleted, including the database files. Therefore, you should store only files created by Oracle CDD/Repository in an OpenVMS directory that is dedicated to a repository. Do not store other files in the OpenVMS directory; otherwise, CDO deletes these files and all files in subdirectories under that directory when you delete the repository.

Before you delete a repository, make sure that it does not contain elements used by another element in a different repository (such as records that are based on fields in another repository). Otherwise, an error occurs when you issue the DELETE REPOSITORY command. To see if an element is used by another element, enter the CDO SHOW REPOSITORIES/FULL command.

If necessary, copy the elements to another repository and change the anchor and path to the elements, where appropriate. After you copy the elements that have relationships to other repositories, you must execute the VERIFY/EXTERNAL_REFERENCES/FIX command on the repository to remove all internal relationships with those repositories.

After you execute the VERIFY/EXTERNAL_REFERENCES/FIX command, issue the SHOW REPOSITORIES/FULL command to confirm that there are no longer any relationships. In some cases, you many need to run the CDO VERIFY/EXTERNAL_REFERENCES/FIX command twice to completely remove the relationships.

Once the relationships are removed, you can delete the repository.

The CDO DELETE command also lets you to delete protection schemas and individual repository elements. See Chapter 7 for information on deleting protection schemas.

For a full list of qualifiers and more information on the CDO DELETE commands, see the *Oracle CDD/Repository CDO Reference Manual*.

3.6 Upgrading a Dictionary or Repository

This section describes how to upgrade existing dictionaries or repositories.

The terms dictionary and repository are basically synonymous, and both are accepted by the product. The term dictionary refers to the product prior to Version 5.0; repository is the term used in Version 5.0 and later.

Note

DMU dictionaries do not require an upgrade.

The recommended method of upgrading existing repositories is to use the CDD$UPGRADE.COM command procedure. The CDD$UPGRADE.COM command procedure automatically invokes the appropriate upgrade utility based on the version number you specify. The procedure is provided with the Oracle CDD/Repository installation.

If you are performing an upgrade from a Version 5.*n* repository to a later version repository, CDD$UPGRADE.COM runs the CDO CONVERT utility using the CDO CONVERT/REPOSITORY command. This is called a minor upgrade.

If you are performing an upgrade from a Version 4.*n* dictionary, the CDD$UPGRADE.COM procedure runs CDDX, the repository translation utility, using the REPOSITORY EXPORT and REPOSITORY IMPORT commands. This type of upgrade is called a major upgrade.

Note

Refer to *Installing Oracle CDD/Repository on OpenVMS Systems* and the Read Before Installing letter for release-specific information and to determine the upgrade requirements between versions of Oracle CDD/Repository.

If your repository has already been upgraded to the latest version and you run the CDD$UPGRADE.COM procedure, it assumes that you want to perform a major upgrade and it will run CDDX.

Each of these utilities can be run manually. Section 3.6.3 describes how to use the CDO CONVERT utility; Section 3.6.4 describes how to use CDDX.

3.6.1 Preparing for an Upgrade of a Dictionary or Repository

Before you upgrade any dictionary or repository, complete the following steps for each dictionary or repository:

1. Ensure that you have SYSPRV or BYPASS privilege, which is necessary to perform an upgrade.
2. Check that your page file quota (PGFLQUOTA) is sufficient for the upgrade procedure.

   If you are performing a major upgrade (from a Version 4.*n* dictionary), the upgrade procedure checks the size of the CDD$DATABASE.RDB file, which contains repository information. The REPOSITORY EXPORT portion of the upgrade may require increased page file quota (PGFLQUOTA) while it generates the intermediate CDD$UPGRADE.CDDX export file. The .CDDX export file requires approximately half the number of blocks used by the CDD$DATABASE.RDB file. The number and complexity of the objects in your repository may also affect the size of the .CDDX export file.

   If the EXPORT operation is being performed on a Version 5.*n* or higher repository, the upgrade procedure checks the combined size of the CDD$DATA.RDA and CDD$DATABASE.RDA files, which contain repository information.

If you are upgrading a Version 5.*n* repository to a higher version (performing a minor upgrade) using the CDO CONVERT/REPOSITORY command, check the value of the VIRTUALPAGECNT system parameter. If the value is less than 100,000, the upgrade will fail. For best performance, increase the VIRTUALPAGECNT value to a *minimum* of 200,000 pages.

Table 3–3 lists the PGFLQUOTA values required for the upgrade.

Table 3–3 Oracle Rdb Database and PGFLQUOTA Values

CDD$DATABASE.RDB Size	PGFLQUOTA Value
0-20,000 blocks	50,000-200,000 pages
20,000-35,000 blocks	70,000-200,000 pages
35,000-55,000 blocks	100,000-200,000 pages
55,000+ blocks	200,000 pages

During the upgrade, if the values for PGFLQUOTA, VIRTUALPAGECNT, or disk space are below the required amounts or suggested minimums, the procedure displays a warning message giving you the option to continue or to cancel the upgrade.

3. Verify the repository. See Section 3.3 for instructions on verifying a repository.
4. Make sure you have an adequate backup of the repository before you perform the upgrade. See Section 3.1 for the backup procedure.
5. The CDD$UPGRADE.COM command procedure automatically defines the CDD$SEPARATOR logical name as a slash character (/) before running EXPORT.

   However, if you issue the REPOSITORY EXPORT command directly from DCL rather than use the CDD$UPGRADE.COM procedure, and the repository contains elements with a period (.) in their names, redefine CDD$SEPARATOR as a slash character (/) before you export the repository. Otherwise, the REPOSITORY IMPORT command cannot correctly interpret the names you exported.

   For example:

   ```
   $ DEFINE/USER CDD$SEPARATOR "/"
   ```

   Defining CDD$SEPARATOR as a slash character has no negative effect if there are no names containing periods.

6. If you are installing Oracle CDD/Repository, check for the SYS$SHARE:CDA$ACCESS.EXE image. This image is required by the upgrade images that are installed by Oracle CDD/Repository.

7. Oracle CDD/Repository adds new DCL commands during the installation. If you are installing Oracle CDD/Repository and the process you are in was created before the installation, then log out and log back in before executing the CDD$UPGRADE.COM command procedure, or run CDD$UPGRADE.COM from the installing account.

3.6.2 Executing CDD$UPGRADE.COM

Invoke the CDD$UPGRADE.COM command procedure, as follows:

```
$ @SYS$LIBRARY:CDD$UPGRADE
```

When CDD$UPGRADE.COM is executed, it displays the following sequence of questions:

1. The procedure asks you to enter the OpenVMS anchor of the repository to be upgraded:

   ```
   Please enter the OpenVMS anchor of the repository to be upgraded.
   ```

   If an export file named CDD$UPGRADE.CDDX is located in this anchor, the command procedure skips the export operation and performs the import operation. Otherwise, it proceeds to the next step.

   If you are installing Oracle CDD/Repository, it is recommended that the compatibility repository be the first repository you upgrade, and that you upgrade it soon after the installation.

2. Make sure you have an adequate backup of the repository before you continue the upgrade. The CDD$UPGRADE.COM procedure will delete the repository and save only the .CDDX export file. The procedure displays the following prompt:

   ```
   Are you satisfied with the backup of your repository? [ Y | N ] (N):
   ```

   If you do not have a backup of your repository, reply NO (the default) to this question.

Note

Previously, if you were running the CDD$UPGRADE.COM procedure, the default response to this question was Yes. In Version 6.1 the default was changed to No to be consistent with the default in the other upgrade utilities (CDDX and CDO CONVERT/REPOSITORY). You must explicitly enter Yes to continue the upgrade.

If you answer No, the procedure stops and the DCL level prompt ($) is displayed. Perform a regular backup now so you have a backup if the upgrade procedure fails, then rerun the CDD$UPGRADE.COM command procedure. See Section 3.1 for the backup procedure instructions.

If you reply Yes, Oracle CDD/Repository proceeds with the upgrade.

3. If you are performing an upgrade from Version 5.0 or higher, the procedure asks you to specify the version of the repository that you are upgrading. For example, if you are upgrading from Version 5.1 to Version 7.0, enter V51 at the prompt.

   If you are upgrading from a Version 4.*n* dictionary, this prompt does not appear.

```
------------------------------------------------------
  What version are you upgrading this repository from?

  (If you are only exporting the repository, what
  version is the repository you are exporting?)

  Note: Upgrade is not required from V6.1 to V7.0.

    [ V50 | V51 | J51 | V53 | V61 | V70 ]
------------------------------------------------------
```

4. If you are upgrading from a Version 4.*n* dictionary, or if you are only exporting the repository (the repository is at the currently installed version), the following prompt appears:

   ```
   Do you ONLY want to perform an EXPORT of this repository? [Y | N] :
   ```

   Choose whether to just export or to export *and* import your repository.

   If you answer No (the default) to this question, the upgrade procedure performs both the export and an import of your repository.

Caution

If you have extended your repository by modifying the product-supplied protocols through the Oracle CDD/Repository callable interface or through Oracle CDD/Administrator, and you want to perform an upgrade using REPOSITORY EXPORT and REPOSITORY IMPORT, you must answer Yes to the EXPORT ONLY question. Perform the additional steps in Section 3.6.2.2, then import the file, as described in Section 3.6.5. Otherwise, you will lose the extensions or any instances of extensions.

If you answer Yes at the EXPORT ONLY question, the upgrade procedure only exports your repository, creates an intermediate export file called CDD$UPGRADE.CDDX (the default), and places the export file in the anchor directory of the repository being upgraded. See Table 3–3 for PGFLQUOTA values required for the upgrade.

You must answer Yes to this question and perform an EXPORT ONLY upgrade in the following situations:

- You are using CDD$UPGRADE to convert your OpenVMS VAX repository to an OpenVMS Alpha system.

  For an OpenVMS Alpha system installation, you must perform the EXPORT ONLY portion of the upgrade on an OpenVMS VAX system, then perform a separate import operation on an OpenVMS Alpha system. Refer to Section 3.6.5 for instructions on how to perform an import operation.

- You are performing an upgrade using the REPOSITORY EXPORT and REPOSITORY IMPORT commands and your repository has been extended by modifying the product-supplied protocols through the Oracle CDD/Repository callable interface or through Oracle CDD/Administrator.

  Refer to Section 3.6.2.2 for instructions on upgrading an extended repository.

If you choose to perform an EXPORT ONLY upgrade, you must run the CDD$UPGRADE.COM procedure again to complete the upgrade. If you are installing Oracle CDD/Repository, run the IVP separately after you have completed the second run of CDD$UPGRADE.COM.

5. The upgrade procedure displays the following prompt, asking if you want to grant READ and WRITE access to the current repository to user [*,*].

```
By default, all users [*,*] will have READ-ONLY access to
this repository.  Do you want to give all users READ/WRITE
access to this repository?  [Y | N] :
```

   Version 4.*n* granted all users READ and WRITE access to the dictionaries by default. For Version 5.0 and higher, the default is READ-ONLY access.

   If you answer Yes (the default) and the repository you are upgrading is linked to other repositories, the DEFINE PROTECTION command will fail unless all repositories have been upgraded. In this case, you must enter the DEFINE PROTECTION command from CDO after you have upgraded all the linked repositories.

   The following is an example of how to manually set the protection and the default protections for a repository. Both commands are required to achieve the protection level specified in the CDD$UPGRADE.COM command procedure.

```
$ REPOSITORY OPERATOR
CDO> DEFINE PROTECTION FOR REPOSITORY <anchor> -
cont> POSITION 15 IDENT [*,*] ACCESS CHANGE+SHOW+DEFINE.
CDO> DEFINE PROTECTION FOR REPOSITORY <anchor>  -
cont> POSITION 15 IDENT [*,*] DEFAULT_ACCESS ALL.
CDO>
```

6. The upgrade procedure then displays the following information:

```
%CDDX-I-CVT_SUCCESS, repository SYS$COMMON:[CDDPLUS] converted.

Exporting repository SYS$COMMON:[CDDPLUS]...
Importing into repository SYS$COMMON:[CDDPLUS]...

%CDDX-I-DATA, ANSI X3.XXX-199Y EXPORT FILE
%CDDX-I-STMP, FILE EXPORTED ON 21-MAY-1992 11:48:51.70
%CDDX-I-DATA, CDD V4.3-2-0 EXPORT of repository SYS$COMMON:[CDDPLUS],
%CDDX-I-DATA,  repository major: 15,
%CDDX-I-DATA,  repository minor: 18,
%CDDX-I-DATA,  MAX index: 0000000000000CE2
%CDD-I-FIXSNP, dictionary SYS$COMMON:[CDDPLUS] snapshot file size has
               been reset
```

Run the CDD$UPGRADE.COM command procedure for each repository.

During a minor upgrade, the procedure creates a temporary context file in the context subdirectory of the anchor. This file, called CDD$CI_CONTEXT.DIR, is deleted after a successful upgrade, unless the upgrade procedure fails or is aborted.

If this file remains in the context subdirectory of the anchor, any subsequent attempts to run the upgrade procedure fail with the following error message:

```
MCS-E-DIREXISTS, directory already exists
```

If this error occurs during an upgrade procedure, exit CDO. Check for the CDD$CI_CONTEXT.DIR file in the subdirectory of the anchor. If the file exists, delete it, then retry the upgrade procedure.

3.6.2.1 Submitting CDD$UPGRADE.COM in Batch Mode

If you are upgrading a Version 4.*n* dictionary, the upgrade procedure can take several hours for a dictionary that is over 100,000 disk blocks in size. If you run CDD$UPGRADE.COM in batch mode, be sure to include all the required input parameters.

The input parameters are as follows:

- Anchor specification—Enter the OpenVMS anchor of the repository to be upgraded.
- Enter `Yes` or `No` for the question: `Are you satisfied with the backup of your repository?`
- Current repository version—Enter one of the following valid version numbers: V4, V5 (for V5.0), V50, V51, V52, V53, V61, or V70.
- Enter `Yes` or `No` for the question: `Do you ONLY want to perform an EXPORT of the repository?`
- Output device—The upgrade procedure will display at your terminal (SYS$OUTPUT) by default; otherwise, output is not displayed. In batch mode, enter " ".
- Enter `Yes` or `No` for the question: `Give all users [*,*] READ/WRITE access?`

The following is an example of how to submit CDD$UPGRADE.COM in batch mode using these input parameters:

```
$  SUBMIT /NOPRINT/NOTIFY/LOG=CDDUPG.LOG -
_$  /PARAM=("SYS$COMMON:[CDDPLUS]","Y","V4","N"," ","N") CDD$UPGRADE.COM
```

3.6.2.2 Upgrading an Extended Repository

If you are performing a minor upgrade and you have extended your repository, the CDO CONVERT/REPOSITORY command will automatically perform the steps you need. However, if you are using the REPOSITORY EXPORT and REPOSITORY IMPORT operation and you have extended your repository, perform the following steps before you import the metadata into a new repository. If you do not perform these additional steps, you will lose the extensions and any instances of extensions.

1. Check that you have the .CDDX export file of the anchor directory for the repository to be upgraded. If you do not have an export file, run the CDD$UPGRADE.COM command procedure and specify that you want to perform an export only, as described in Section 3.6.2, or use CDDX with the REPOSITORY EXPORT command to create an export file, as described in Section 3.6.4.
2. After you export the repository, delete the original repository (the one you exported) using DCL DELETE. Be sure you do *not* delete the .CDDX file.
3. Define a new repository in the same location.
4. Apply the extensions to the new repository.
5. Import the repository using CDDX with the REPOSITORY IMPORT command, as described in Section 3.6.5, or rerun the CDD$UPGRADE.COM procedure.

3.6.3 Using the CDO CONVERT/REPOSITORY Command

The CDO CONVERT utility was designed for performing a minor upgrade (from a Version 5.*n* repository to a later Version 5.*n*, a Version 6.1, or a Version 7.0 repository). When you perform a minor upgrade, the CDD$UPGRADE.COM procedure invokes the CDO CONVERT utility and uses the CDO CONVERT/REPOSITORY command.

Note

If the CDD$CI_CONTEXT.DIR file exists in the context subdirectory of the anchor, the upgrade procedure will fail with the following error message:

```
MCS-E-DIREXISTS, directory already exists
```

If this error occurs during an upgrade procedure, exit CDO. Check for the CDD$CI_CONTEXT.DIR file in the subdirectory of the anchor. If the file exists, delete it, then retry the upgrade procedure.

The steps in this section describe how to manually run the CDO CONVERT/REPOSITORY utility. You must have SYSPRV or BYPASS privilege to run this utility.

1. Make sure you have prepared for the upgrade by following the steps in Section 3.6.1.
2. Invoke CDO, then enter the CONVERT/REPOSITORY command, as follows:

   ```
   $ REPOSITORY
   CDO> CONVERT/REPOSITORY device:[directory]
   ```

3. Confirm that you have an adequate backup of the repository. The procedure displays the following prompt:

   ```
   Are you satisfied with the backup of your repository? [Y/N] (N):
   ```

   If you reply `No` (the default) the procedure advises you to back up your repository, and the CDO prompt is displayed. Exit CDO, and perform a regular backup of your repository, as described in Section 3.1, then rerun CDO CONVERT/REPOSITORY.

   If you have an adequate backup, enter `Yes`. CONVERT/REPOSITORY upgrades your repository to the current version by updating the protocols to the ones supplied with the new version of Oracle CDD/Repository. If your repository has been extended, the extensions and instances remain intact.

3.6.4 Creating an Export File Using the CDDX Utility

The CDDX translation utility was designed for performing a major upgrade (from a Version 4.*n* dictionary to a Version 5.*n*, a Version 6.1, or a Version 7.0 repository). If you are performing a major upgrade, the CDD$UPGRADE.COM procedure invokes CDDX.

Note

Do not use the CDDX utility for backing up, verifying, or moving a dictionary or repository. See Section 3.1, Section 3.3, and Section 3.4 for information on how to perform those operations.

To perform CDDX REPOSITORY EXPORT and REPOSITORY IMPORT you must have SYSPRV or BYPASS privilege.

For online help on export and import operations, type `HELP CDDX` at the DCL level prompt ($).

You can use the REPOSITORY EXPORT command of CDDX to manually create an export file of your repository metadata. After you export the repository, use the REPOSITORY IMPORT command to import the repository metadata from the export file.

1. Make sure you have prepared for the upgrade by following the steps in Section 3.6.1.
2. From DCL, issue the REPOSITORY EXPORT command, as follows:

   ```
   $ REPOSITORY EXPORT [qualifiers] repository export-filename
   ```

   If you are upgrading a pre-Version 4.3 dictionary, you must add the /CONVERT qualifier to the EXPORT command.

   The following list describes the qualifiers and parameters for the REPOSITORY EXPORT command.

   - /LOG

     The /LOG qualifier produces a line of text for every element in the repository. Use the /LOG qualifier only when you suspect a problem, because a large repository produces a large amount of output. The default is /NOLOG.

   - /VERSION=*n*

     The /VERSION qualifier specifies the version of the repository you are exporting. Valid version numbers are: V4, V5 or V50 (for Version 5.0), V51, V52, V53, V61, and V70. V52 (Version 5.2) is valid for OpenVMS Alpha systems only.

     If you execute the REPOSITORY EXPORT command directly from DCL, the list of valid version numbers is not displayed.

     The default version is /VERSION=V4.

   - /SCHEMA

     The /SCHEMA qualifier specifies the repository metadata or types to be exported. This qualifier allows user extensions or other product-supplied extensions to the metadata to be exported. The default is /SCHEMA; however, if you are performing a major upgrade you cannot override the default.

     If you specify the /NOSCHEMA qualifier, no repository types (protocols) are exported. This means that no user or other product extensions to the metadata will be exported. This speeds up the export process, especially for small repositories.

   - /CONVERT

Use the /CONVERT qualifier when you are exporting pre-Version 4.3 dictionaries. When you specify /CONVERT, CDDX first converts the Version 4.0, 4.1, or 4.2 dictionary to Version 4.3 before exporting it. The conversion will be committed, regardless of whether the export succeeds. A repository that cannot be converted will not be exported. The default is /NOCONVERT.

If you specify the /CONVERT qualifier, the following informational message will be displayed when the conversion is completed and before the actual EXPORT starts:

```
%CDDX-I-CVT_SUCCESS  repository device:[directory] converted
```

- *repository*

  The OpenVMS anchor of the repository to be exported.

- *export-filename*

  The name of the export file to be created. The default file extension is .CDDX.

The REPOSITORY EXPORT command creates an intermediate export file of your repository using the file name you supply. It places the file in your current directory by default, or you can specify another device and directory. Be sure you do *not* delete this export file.

Note

If your repository is corrupt and you issue the REPOSITORY EXPORT command, EXPORT issues warning messages. Oracle CDD/Repository attempts to export accessible portions of corrupt elements or skips over elements that are too incomplete to be exported.

3.6.5 Importing a Repository from an Export File

The REPOSITORY IMPORT command of CDDX reloads the exported repository data from a repository export file into a new, empty repository. IMPORT assumes that the location you specify to import the export file into contains either an empty directory, or a directory with a new empty repository at the current version. If you specify an empty directory, IMPORT creates a new repository and imports your metadata into it.

Perform the following steps:

1. Check for a .CDDX export file, which was created during the export. If you do not have the export file, run the CDD$UPGRADE.COM command procedure and specify that you want to perform an export

only of the repository, as described in Section 3.6.2, or use CDDX with the REPOSITORY EXPORT command, as described in Section 3.6.4.

2. Make sure you have an adequate backup of the repository. See Section 3.1 for the backup procedure.

3. Delete the original repository (the one you exported). Do *not* delete the .CDDX file.

4. If you have extended the repository by modifying the product-supplied types (protocols) using the Oracle CDD/Repository callable interface or through CDD/Adminstrator, define a new repository in the anchor from which the original repository was exported.

   If you do not have an extended repository, ignore this step. Skip to step 6 and issue the REPOSITORY IMPORT command.

5. If you have extended your repository, apply any schema modifications to the new repository.

   If you fail to apply the schema changes, you cannot import instances of extensions. When REPOSITORY IMPORT is executed, you will see error messages for all definitions that could not be imported. However, REPOSITORY IMPORT will continue to import all the definitions it can.

6. From DCL, issue the REPOSITORY IMPORT command. Specify the file name of the intermediate .CDDX export file, and the name of the anchor for the repository you just exported and deleted.

   ```
   $ REPOSITORY IMPORT /LOG export-filename disk:[anchor_dir]
   ```

   The only qualifier for REPOSITORY IMPORT is /LOG, which is optional. The /LOG qualifier produces a line of text for every element in the repository. Use the /LOG qualifier only when you suspect a problem, because a large repository produces a large amount of output. The default is /NOLOG.

   CDDX imports the exported repository to the specified anchor. (If the anchor is empty, CDDX creates a new repository for you.) You *must* import your repository into the same OpenVMS anchor directory from which it was exported.

7. After importing the repository, compress the repository to reduce the size of the snapshot file, CDD$DATABASE.SNP, which may become large during the IMPORT operation. Enter the following command:

   ```
   $ REPOSITORY OPERATOR VERIFY/COMPRESS disk:[anchor_dir]
   ```

3.6.6 Sample Upgrade of Version 4.3 Dictionary

The following is a sample upgrade of a Version 4.3 dictionary:

Note

The following output is a sample; the version numbers, dates, and times that display on your system will be different.

```
$ @SYS$LIBRARY:CDD$UPGRADE
             CDD$UPGRADE.COM
  This procedure is used to upgrade repositories that were created
  under a previous version of Oracle CDD/Repository.
  Please enter the OpenVMS anchor of the repository to be upgraded.
  DEVICE:[DIRECTORY] : CDD$1:[CDD.TEST_DICT]
   Are you satisfied with the backup of your repository? [Y/N] (N): y
  Do you ONLY want to perform an EXPORT of this repository? [ Y | N ]
  [ N ] :  n
  By default, all users [*,*] will have READ-ONLY access to this
  repository.  Do you want to give all users READ/WRITE access to
  this repository?  [ Y | N ]
  [ Y ] :  n
  There are no more questions.
  Exporting repository CDD$1:[CDD.TEST_DICT]...
  Importing into repository CDD$1:[CDD.TEST_DICT]...
%CDDX-I-DATA, ANSI X3.xxx-199y EXPORT FILE
%CDDX-I-STMP, File exported on 24-AUG-1994 11:48:51.70
%CDDX-I-DATA, CDD V4.3-2-0 EXPORT of repository CDD$1:[CDD.TEST_DICT]
%CDDX-I-DATA,  repository major: 15,
%CDDX-I-DATA,  repository minor: 18,
%CDDX-I-DATA,  MAX index: 0000000000000C8C
  CDD$UPGRADE completed successfully.
```

4
Enhancing Repository Performance

This chapter provides tuning concepts to enhance repository performance.

4.1 Performance Concepts

It is important to understand the concepts of system performance, because many variables can affect repository performance, such as:

- different applications running on the system (system load)
- SYSGEN parameter settings
- disk usage
- amount of memory available to an application
- process quotas
- logical settings
- concurrent repository access

Although performance expectations differ with applications, what really matters is elapsed time. Total elapsed time is affected by the size of the processor used, the number of I/O requests, and the number of page faults. Therefore, timing will vary, depending on the system used and other processes running simultaneously on the system.

Memory influences the number of page faults. Usually, as more memory is used, the number of page faults increases and the computer takes longer to process the application. You can see the effect of memory on elapsed time by enabling the Ctrl/T function, then typing a Ctrl/T key sequence. This displays statistics, as shown in the following example:

```
$ SET CONTROL=T
TLEO:: 11:36:57   (DCL)   CPU=00:00:55.95 PF=20417 IO=9550 MEM=191
$
TLEO:: 11:37:00   (DCL)   CPU=00:00:55.96 PF=20422 IO=9551 MEM=149
```

You need to be aware of these factors to understand application performance, because repository performance is discussed in terms of these factors.

4.2 General System Tuning Tasks

For general system tuning, use the OpenVMS Autogen utility (AUTOGEN). You execute the AUTOGEN command procedure when your system is installed, upgraded, or whenever you make significant changes to your workload. To run AUTOGEN, use the following two steps:

1. Enter the following command:

   ```
   $ @SYS$UPDATE:AUTOGEN SAVPARAMS TESTFILES FEEDBACK
   ```

   Review the AUTOGEN report for changes.

2. If you approve the report, reboot the system by entering the following command:

   ```
   $ @SYS$UPDATE:AUTOGEN GENPARAMS REBOOT
   ```

For more details on how to use AUTOGEN, see the OpenVMS documentation on how to set up an OpenVMS system.

You can improve performance through the following tasks:

- Relocate the page and swap files by placing them on a disk that is separate from the repositories.
- Spread the repositories across several disks.
- Increase the power of the processors.
- Distribute the workload of various applications across multiple processors.
- Decrease the startup time of a CDO command by keeping the database root file (CDD$DATABASE.RDB) of your repository open. To do this, enter the following command:

  ```
  $ RMU/OPEN CDD$DATABASE.RDB
  ```

For additional information on measuring and improving system performance, see the OpenVMS documentation set.

The following Oracle Rdb documentation also provides helpful information:

- *Oracle Rdb7 Guide to Database Maintenance*
- *Oracle Rdb7 Guide to Database Performance and Tuning*
- *Oracle RMU Reference Manual*

4.3 Performance Optimizations

This section describes performance optimizations that have been incorporated into Oracle CDD/Repository.

- Users attach only once to the repository.

  Because attaching to the database is expensive in terms of elapsed time, interactive repository sessions are optimized to attach the user to the database once. Therefore, remaining in a single CDO session is more efficient than exiting and entering CDO.

- Type definitions are cached only once in the repository.

  Objects are cached in memory throughout the transaction, or throughout the callable interface transaction. Again, remaining in a single CDO session, rather than exiting and reentering CDO, is recommended.

- Information is stored in memory to save time and database I/O requests.

  Less time is spent waiting for objects to be fetched from memory and more time is spent processing. By using longer transactions you can improve performance; however, longer transactions increase contention. Using a series of CDO commands in a single transaction is described in more detail in Section 4.3.3 through Section 4.7.

- Directory files are opened only for create and delete operations.

  The directory hierarchy search is based on OpenVMS directory file names. The depth of hierarchy in the directory path does not affect performance.

- The number of names per directory has been optimized.

  This optimization benefits any operation that reads an element by name or modifies the contents of a directory with a large number of entries. It is recommended that you keep directories uncluttered for optimal performance. The recommended maximum number of names in a repository directory is listed in Table 4–1.

- Logical name translations are saved.

  When you use logicals in element names, Oracle CDD/Repository saves the translations for use when the logical name is used again in the same CDO interactive session.

- Large database integrate operations have been optimized.

  The database integrate operation has been optimized in a number of ways, including one load into memory of all record and field definitions used when comparing the metadata in the database and repository.

For example, in one case, the integration of a large database into a nonexistent path that took 1 hour 15 minutes in Version 5.0 takes less than 30 minutes.

- CDO uses read-only transactions in VERIFY/NOFIX operations.

  CDO uses read-only transactions for verify operations, except when the /FIX, /REBUILD_DIRECTORY, or /COMPRESS qualifiers are specified. Using read-only transactions reduces the amount of locking in the underlying repository database.

- The CDD$MAX_OBJECTS_IN_MEMORY logical name can be used as a tuning method.

  The CDD$MAX_OBJECTS_IN_MEMORY logical name can assist in tuning I/O intensive repository applications. Using the RDM$BIND_BUFFERS logical name in conjunction with the CDD$MAX_OBJECTS_IN_MEMORY logical name further reduces I/O and improves overall performance of the applications.

  Using these logical names as a tuning method to improve performance is explained in more detail in Section 4.5.3.

4.3.1 Internal Operations During a Session

The following is an example of the operations that occur inside Oracle CDD/Repository during a session. Suppose the following CDO command is executed:

```
CDO> DEFINE FLD1 DATATYPE is TEXT SIZE is 30.
```

Oracle CDD/Repository performs the following internal operations:

1. attaches to the database
2. starts a read/write transaction on the database
3. loads the type definitions into memory
4. reads the directory into memory
5. creates the field in memory
6. validates and sets the field property values
7. creates a relationship and a history entry in memory
8. determines the request needed to store this field
9. stores the field as a row in a relation in the database
10. stores the history and relationship in the database

11. writes the field name to the directory buffer in memory
12. stores the modified directory in the database
13. commits the transaction

Creating a record would have similar steps. For example:

```
CDO> DEFINE RECORD REC1.
cont> FLD1. END RECORD.
CDO> SHOW RECORD REC1
  .
  .
  .
```

The repository is already attached to the database and has already loaded the type definitions. To read the objects, Oracle CDD/Repository performs the following steps:

1. starts a read-only transaction
2. finds the name REC1 in the directory (in memory)
3. uses a key stored with the name to find the record
4. determines the request needed to read this record
5. reads the record from the database
6. builds a key for each type of relationship the record can own
7. determines the request needed to read this relationship
8. reads the relationship from the database
9. uses the key stored in the relationship to find the member
10. determines the request needed to read this field
11. reads the field from the database
12. commits the transaction

4.3.2 CHANGE and DEFINE FIELD Command Enhancements

When you issue the CDO CHANGE FIELD command on an object with multiple relationships to an object in an external repository, Oracle CDD/Repository reads the information from memory and no longer performs unnecessary (and possibly remote) node lookups.

When you issue the CDO DEFINE FIELD command and the object is defined in a repository by using an object with relationships in another repository, CDO no longer copies these objects into the new repository.

In the following example, two repositories [.DICT1] and [.DICT2], are defined.

Dictionary 1 contains one global field B, field F1 based on B, and five hundred records (R1 to R500) also based on B. Dictionary 2 contains an additional record, R501, using the same BASED ON field F1 from Dictionary 1.

In versions of Oracle CDD/Repository prior to Version 5.3, the result in [.DICT2] was the following:

- Record R501 was created.
- A local copy of field F1 was created to provide fast access to the field.
- Five hundred local read copies of the records R1 to R500, defined in [.DICT1], were created in the new repository, [.DICT2].

Note

The copy of field F1, which is a clone, cannot be updated in [.DICT2]; it must be updated in [.DICT1], the repository in which it was defined.

In Oracle CDD/Repository Version 5.3 and higher, only record R501 is created, and the local copy of F1 is created for fast access to the field. The 500 local read copies of records R1 to R500 are not created.

```
$ REPOSITORY OPERATOR
CDO>  DEFINE REPOSITORY [.DICT1].
CDO>  DEFINE REPOSITORY [.DICT2].
CDO>  SET DEFAULT [.DICT1]
CDO>  DEFINE FIELD B DATATYPE TEXT SIZE 25.       ! global field
CDO>  DEFINE FIELD F1 BASED ON B.
CDO>  DEFINE RECORD  R1.
cont> F1. END.
   .
   .
   .    .
CDO>  DEFINE RECORD R500.
cont> F1. END.
CDO>  SET DEFAULT [.DICT2]
CDO>  DEFINE RECORD R501.
cont> [.DICT1]F1. END.
CDO>  EXIT
```

Example 4–1 shows the information that CDO displays for these objects when you issue various SHOW commands.

Example 4–1 CDO SHOW Command Examples

```
 $ REPOSITORY OPERATOR
   .
   .
   .
CDO> SHOW DEFAULT
[.DICT2]
    = DEVICE:[USER.DICT2]

CDO> DIRECTORY
 Directory DEVICE:[USER.DICT2]

CDD$PROTOCOLS                                    DIRECTORY
R501(1)                                          RECORD

CDO> SHOW RECORD R501
Definition of record R501
|   Contains field           F1

CDO> SHOW USED R501
Members of DEVICE:[USER.DICT2]R501(1)
|   DEVICE:[USER.DICT1]F1(1)       (Type : FIELD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS

CDO> SHOW USED/FULL R501
Members of DEVICE:[USER.DICT2]R501(1)
|   DEVICE:[USER.DICT1]F1(1)       (Type : FIELD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
|   |   DEVICE:[USER.DICT1]B(1)        (Type : FIELD)
|   |   |   via CDD$DATA_ELEMENT_BASED_ON

CDO> SHOW USES R501
   .
   .
   .
CDO> SHOW USED/FULL DEVICE:[USER.DICT1]F1(1)
Members of DEVICE:[USER.DICT1]F1(1)
|   DEVICE:[USER.DICT1]B(1)        (Type : FIELD)
|   |   via CDD$DATA_ELEMENT_BASED_ON
```

(continued on next page)

Example 4–1 (Cont.) CDO SHOW Command Examples

```
CDO> SHOW USES/FULL DEVICE:[USER.DICT1]F1(1)
Owners of DEVICE:[USER.DICT1]F1(1)
|   DEVICE:[USER.DICT1]R1(1)       (Type : RECORD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
   .
   .
   .
|   DEVICE:[USER.DICT2]R501(1)     (Type : RECORD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
CDO>
```

4.3.3 CDO Commands Improve Performance

A single repository transaction is defined as an individual CDO command, or a series of CDO commands between the START_TRANSACTION and COMMIT (or ROLLBACK) commands. This provides a substantial savings for large command procedures.

For example, defining 100 fields in a single CDO transaction is 80% faster than defining the same number of fields, each in a separate transaction.

Caution

While a transaction is open, locks are held against the repository database. Using the CDO START_TRANSACTION command can reduce the concurrency of the repository database.

The internal operations that occur after the CDO DEFINE RECORD command is executed are similar to those described in Section 4.3.1.

```
   .
   .
   .
CDO> START_TRANSACTION
CDO> DEFINE RECORD REC2.
cont> FLD1. END RECORD.
   .
   .
   .
CDO> SHOW RECORD REC2
   .
   .
   .
CDO> COMMIT
```

In this case, the repository is already attached to the database and has already loaded the type definitions. The CDO SHOW command executes only the following steps to read the record:

1. Finds the name REC2 in the directory (in memory).
2. Finds the name REC2, the relationship to FLD1, and FLD1 (loaded in memory when the record was defined).

When you use the CDO START_TRANSACTION and COMMIT commands, the overhead that is associated with these commands is incurred once in the repository and once in the database, rather than once for each CDO command between the START_TRANSACTION and COMMIT commands. Objects are retrieved from memory instead of from disk.

4.4 Designing the Repository Structure

The repository administrator can take an active role in designing the repository and maintaining the optimal performance of the repository. Different applications require different repository design trade-offs to take advantage of the specifics of the application.

Designing a repository involves:

- designing the repository structure
- adjusting the working set of the individual user
- managing the physical repositories

The repository contains files for the directory hierarchy, and binary and journal files. This repository structure is implemented on an Oracle Rdb relational database. Directory contents are stored in the relational database for reliability.

Good planning prior to setting up your repository can result in a more secure and consistent repository. A well-designed repository provides the following benefits:

- logical organization
- easy documentation and maintenance
- controlled metadata access
- optimum performance

Designing the repository can involve several tasks, such as creating a logical, understandable directory structure, choosing the proper location of physical repositories, implementing a single repository structure on all nodes within the logical framework, or organizing objects within the structure by function.

For optimal performance, when you make design decisions about where the data will reside, it is recommended that you limit the number of names in a repository directory to those listed in Table 4–1.

Table 4–1 Number of Names Defined

Oracle CDD/Repository Version	Recommended Maximum Number
5.0	200
5.1 or higher	500-1000

For example, a user that was performing an ACMS build noticed that a repository directory contained over 2000 definitions. The user moved the critical data to a different directory and cut the repository elapsed time on the build operation by 50%.

4.4.1 Deciding Where to Place Definitions

Many users prefer to organize repository metadata in a way that isolates separate user activities. You can divide your repository into directories and subdirectories, or your organization may have special needs, such as security or maintenance requirements, that call for multiple physical repositories.

There are a number of ways to structure your repository:

- by application

  In an organization where the departments must share most of the data, you can structure your repository according to application areas. For example, you can create separate directories for development, testing, and production modes of an application or system life cycle.

  Using logical names and search lists can facilitate movement from one phase of the cycle to another.

  You can set aside one repository directory for storing those elements that are shared throughout the organization. You can then create application-specific directories for elements that are less widely shared.

- by organizational entity

  Organizing your repository by departments is most useful in situations where different divisions of the organization have different data and security needs. For example, a company with two separate departments might choose to create two separate physical repositories. Selected employees might have access to the elements in both repositories, while other employees could access only one repository or the other.

  You can achieve a similar effect by setting up two repository subdirectories in the repository hierarchy and protecting the elements in each subdirectory from unqualified users.

- by individual user

  Individuals who work separately on independent projects may need a personal repository or a personal directory structure within a large repository. You can implement this structure in the repository hierarchy by assigning directories to individuals for their own use or allowing individuals to create their own repositories on the system, if the system is large enough to handle the load. This would let you distribute the load over more disks.

In many cases, an organization benefits most from a combination of these solutions. If your logical repository includes personal, department, and systemwide repositories, you can store elements that change frequently in a personal directory and elements that are permanent in a widely used system location.

4.4.2 Configuring a Repository over a Network

With Oracle CDD/Repository you can have any number of repository hierarchies stored on devices connected across a network. To improve performance, Oracle CDD/Repository maintains local copies of some remote definitions that are referenced in local definitions. Although the network does not affect repository performance for read operations, the network does affect performance for define and modify operations.

The more remote copies a definition has, the longer it takes to change it. For a change to take effect, all nodes in a distributed repository must be running and connected to the network.

How to implement a distributed repository is discussed in Chapter 6.

4.4.3 Using Logical Names to Preserve Structure

Repository definitions that use or are used by definitions in other repositories refer to those definitions with a fully qualified name that includes node, device, and directory names. Therefore, when you move a repository, you potentially invalidate any references to the repository from other repositories.

Oracle CDD/Repository translates to a concealed logical name, so by using concealed device logical names, you can avoid problems when moving disks. A **concealed device logical name** is a systemwide logical name that the system associates with a specific device. If you have a logical name associated with a device name, you can use the logical name instead of the formal device name.

When you change the location of a repository, you need only change the device assigned to the logical name, instead of changing every reference to the device.

You can use the file SYS$MANAGER:SYLOGICALS.COM to assign systemwide logical names. When you start up your system, it executes the command procedure SYLOGICALS.COM as part of the STARTUP.COM procedure and adds the logical names to the system logical name table. (If your system is not in a cluster, you can use the procedure included in the file SYLOGICALS.COM as a template for assigning logical names.)

For a directory specification to be part of a concealed logical name, you must define it so that it translates to a rooted directory. For example, if you have a database in DISK1:[CORPORATE.MIS.PERSONNELDB], and you want to use concealed logicals to access the database, you can use one of the methods in the following examples:

```
$ DEFINE/SYSTEM/TRANS=CONCEALED PB DISK1:[CORPORATE.MIS.]
$ REPOSITORY OPERATOR
CDO> DEFINE REPOSITORY PB:[PERSONNELDB]
```

or

```
$ DEFINE/SYSTEM/TRANS=CONCEALED DB_DISK $5$DUA1:
$ REPOSITORY OPERATOR
CDO> DEFINE REPOSITORY DB_DISK:[CORPORATE.MIS.PERSONNELDB]
```

4.4.4 Adjust the Working Set of the User

The working set of the individual user affects performance. If the working set is too low for the types of applications that are running, page faulting increases and performance is degraded.

Table 4–2 shows the minimum working set quotas (WSQUO) and paging file limit (PGFLQUOTA) that are recommended.

Table 4–2 Recommended Quotas

Quota	Recommended Minimum Setting
Working Set Quota Minimum	4,096
Working Set Quota, Large Operations	10,240
Memory Page File Limit	40,000

4.4.5 Managing the Physical Repositories

The repository administrator must consider when to distribute and when to replicate data. Users benefit from a design that combines both distributed data (linked repositories that share data definitions) and replicated data (separate repositories).

The advantages and disadvantages of linked repositories and separate repositories must be weighed when seeking improved performance in a repository application. As repository administrator, you should replicate data that does not change often, and distribute other data so that you can easily manage changes in a few places. Ideally, the amount of distributed data should be balanced with the amount of replicated data.

4.4.5.1 Linked Repositories

An advantage of linked repositories is that they allow central modification of shared definitions. CDO provides the BASED ON field property that lets you create local copies of the commonly used definitions. This means that READ access to the distributed data is local.

However, linked repositories require access to all the repositories for modifications. Because data is distributed and all linked repositories must be available, updates are expensive. Performance is significantly degraded if there are too many linked repositories.

In addition, performing backups on linked repositories becomes more complicated, since all repositories must be backed up at the same time to obtain a synchronous snapshot of the environment.

4.4.5.2 Separate Repositories

Separate repositories that contain replicated data offer faster access to definitions because of local availability and reduced contention. In addition, separate repositories let you phase in modifications of shared definitions, and separate repositories are easier to back up because they are not dependent on other repositories. However, too much replicated data can be a management nightmare.

4.5 Maintaining the Repository

You maintain repository application performance by analyzing the repository design with database analysis tools, and by monitoring and tuning available system resources.

Maintaining the repository is an ongoing process. It involves monitoring I/O performance, and identifying and resolving repository database performance problems.

4.5.1 Monitoring Repository I/O Performance and Locking

Use the OpenVMS Monitor utility (MONITOR) to monitor I/O performance by collecting data from a system that is running or from a previously created recording file.

During the first few weeks that Oracle CDD/Repository is installed, check the actual number of locks your system is using with the OpenVMS Monitor utility.

Issue the `MONITOR LOCK` command to display the maximum number of locks outstanding during the monitor period. You can use this value to fine-tune the LOCKIDTBL_MAX and RESHASHTBL system parameters.

For a complete description of the Monitor utility, the LOCKIDTBL_MAX and RESHASHTBL parameters, and tuning information, see the OpenVMS documentation.

Use the Oracle Rdb Performance Monitor to display database information that can help you identify repository performance problems. The Oracle RMU Show Statistics command opens the Oracle Rdb Performance Monitor on a character-cell terminal and displays current information about security audit characteristics, version numbers, active databases, active database users, active recovery units, or database statistics related to database activity, such as locking and contention on your node.

For example:

```
$  RMU/SHOW STATISTICS disk:[anchor_dir]CDD$DATABASE
```

For more information, see the *Oracle RMU Reference Manual*.

4.5.2 Moving Repositories Off the System Disk

If repositories reside on the system disk, often contention on the disk degrades performance. Therefore, you can greatly improve performance by moving repositories off the system disk. For instructions on how to move repositories, see Section 3.4.

Using multifile CDD$DATABASE does not improve performance significantly when the size of your repository database is constant. However, you can spread repository files over multiple disks for performance improvements.

If I/O is the bottleneck, try placing the database recovery unit journal file (CDD$DATABASE.RUJ) and the snapshot file (CDD$DATABASE.SNP) on separate disks. See Section 4.7 for more information.

4.5.3 Trading I/O for Memory

The most effective tuning method for improving performance in repository applications is trading I/O operations for memory. The two logical names, CDD$MAX_OBJECTS_IN_MEMORY and RDM$BIND_BUFFERS, control the size of memory cache in the repository and the database.

These logical names also affect access time for reading objects for integrate operations and verifications of large databases, and the deletion of an Oracle CODASYL DBMS schema.

The following sections describe how using these logical names can improve performance.

4.5.3.1 Controlling the Number of Objects in Memory

The CDD$MAX_OBJECTS_IN_MEMORY logical name controls the number of objects cached in memory per session, reduces the rereads from memory, and saves up to 50% of the CPU time. An object can be:

- a field
- a record
- a relationship between a record and a field
- a history entry and the relationship between a record and one of its history entries

Increasing the number of objects held in memory reduces the amount of I/O performed by a large application, such as on a large database integrate operation.

The number of objects in memory should be increased for SQL INTEGRATE operations of large databases.

In Version 4.3, the number of objects that could be cached was unlimited. In Oracle CDD/Repository Version 5.*n* and higher, you can limit the number of objects that can be cached. The default is 10,000, but you can change this number by defining the CDD$MAX_OBJECTS_IN_MEMORY logical name. For example, the number might be reduced to between 100 and 1000 objects for small operations like CDO commands, or increased to 100,000 for large integrate operations.

Each repository application needs to balance sufficient process memory with system memory. Because process memory is a page file quota (PGFLQUOTA) and system memory is a system parameter (VIRTUALPAGECNT), reducing the number of objects in memory lets you work with a smaller page file quota.

4.5.3.2 Controlling Pages in Cache Memory

Page faults increase with increased memory usage. The RDM$BIND_ BUFFERS logical name controls the number of pages the database caches in memory and reduces the overall direct I/O by an order of magnitude. Although reducing this number is not necessary or helpful for repository applications (the default is 50), increasing the number can *dramatically* enhance performance for large operations such as SQL integrate operations of large databases.

Using the RDM$BIND_BUFFERS logical name with the CDD$MAX_ OBJECTS_IN_MEMORY logical name also:

- increases the number of buffers that the underlying database holds in memory
- further reduces I/O
- improves overall performance of the applications

For example:

```
$ DEFINE CDD$MAX_OBJECTS_IN_MEMORY 20000
$ DEFINE RDM$BIND_BUFFERS 200
```

The commands in Example 4–2 change a domain that is used in 70 tables in a database that contains 100 tables.

Table 4–3 shows how I/O is traded for memory during the integrate operation. (The INTEGRATE DOMAIN feature was implemented in Version 6.1 of Oracle CDD/Repository.)

Example 4–2 Trading I/O with Memory During an Integrate Operation

```
$ SQL
SQL> ATTACH 'FILENAME TRDBRESTORE';
SQL> ALTER DOMAIN CDD$$A_NAME IS CHAR(500);
SQL> COMMIT;
SQL> EXIT
```

After completing the change in the domain, integrate the change into the repository, as follows:

```
$ SQL
SQL> ATTACH 'PATHNAME TRDBRESTORE';
SQL> INTEGRATE DOMAIN CDD$$A_NAME ALTER DICTIONARY;
SQL> COMMIT;
SQL> EXIT
```

Next, tune the database by increasing the values of CDD$MAX_OBJECTS_IN_MEMORY and RDM$BIND_BUFFERS logical names.

Performing an SQL integrate operation of the changed domain, and increasing the values of CDD$MAX_OBJECTS_IN_MEMORY and RDM$BIND_BUFFERS logical names reduces CPU time, direct I/O, and overall elapsed time, as shown in Table 4–3.

Table 4–3 SQL INTEGRATE Changed Domain

RDM$BIND_BUFFERS Value	CDD$MAX_OBJECTS_IN_MEMORY Value	Direct I/O	Page Faults	CPU (Minutes)	Elapsed Time (Minutes)
50	50	11,651	57,868	6:08	9:46
200	50	1,142	73,043	5:47	6:37
200	20000	1,111	93,098	3:27	4:17

Notice that page faults increase when both the CDD$MAX_OBJECTS_IN_MEMORY and RDM$BIND_BUFFERS logical names are increased. You must adjust the working set of the user to reduce page faults. Refer to Section 4.4.4 for information on adjusting the working set.

4.5.4 Using the CDD$WAIT Logical Name

The CDD$WAIT logical name controls how Oracle CDD/Repository handles lock conflicts. If CDD$WAIT is not defined, there are no restrictions on access. Read and write requests can access a repository at any time, and they will abort if there is an outstanding lock on the resource that is needed.

PROTECTED locking forces single threading of repository write requests; it also allows long batch jobs to complete.

EXCLUSIVE locking single threads all requests and is the same as turning off the snapshots. For this reason, exclusive locking is for single-user repositories. Exclusive locking also restricts the repository for maintenance operations. Only one access to the repository is allowed; read and write requests are queued and all operations are serialized.

Caution

Because CDD$WAIT is a process logical name, all repositories you access are affected by how CDD$WAIT is defined.

4.5.5 Improving Repository Concurrency and Performance

In environments where concurrent repository access is causing deadlocks, lowering the SYSGEN parameter DEADLOCK_WAIT to a value that is less than 10 seconds (the default) can improve performance in some situations. You should lower the DEADLOCK_WAIT parameter value only if most of the work being performed on your system involves operations that access Oracle Rdb databases; otherwise, making this change could have adverse affects. Use caution when making this change, because it will affect all applications that are running on your system.

Using the SQL ALTER DATABASE statement to enable global buffers is another way to improve performance in environments where concurrent repository access occurs.

These adjustments should be made with caution. Before implementing any changes on your system, you should test the adjustments in a simulated environment to see if performance improves in your particular situation.

4.6 Preventing Disk Quota Errors

Using the OpenVMS System Management utility (SYSMAN), system managers can create disk quota files to limit the disk consumption of users. The quota file records the current usage and the maximum disk consumption for all users.

However, a problem can occur if disk quotas are set on the disk where a repository resides, and users do not have disk quotas enabled for them on that disk. If you do not have disk quotas and you try to define something in the repository, the following error message can occur:

```
$ REPOSITORY OPERATOR
CDO> DEFINE FIELD LAST_NAME
cont> DATATYPE IS TEXT
cont> SIZE IS 20.
%CDO-E-ERRDEFINE, An error occurred while trying to define something.
%CDD-F-NOJNLCRE, cannot create journal file in anchor
-RMS-E-OCRE, ACP FILE CREATE FAILED
-NONAME-W-NOMSG MESSAGE NUMBER 00000000
```

To prevent this problem, use the following steps to create a rights identifier called CDD_USER, and control access to the compatibility repository directory.

The CDD_USER identifier will own all the space allocated to the repository files in the compatibility repository. That way, no individual user needs quotas on the system disk.

1. Use the OpenVMS Authorize utility (AUTHORIZE) to create a rights identifier called CDD_USER.

   Create the CDD_USER identifier with the RESOURCE attribute so that holders of the identifier can charge resources to it. To create the CDD_ USER identifier, enter the following commands:

   ```
   $ SET DEFAULT SYS$SYSTEM
   $ RUN AUTHORIZE
   UAF> ADD/IDENTIFIER CDD_USER/ATTRIBUTES=RESOURCE
   UAF> EXIT
   ```

2. Grant the CDD_USER identifier the same quotas as any other user of Oracle CDD/Repository. To avoid running out of journal file space, allocate the same amount of disk space to CDD_USER as the amount used by the repository. Any user holding the CDD_USER identifier can use this disk space. Since an empty repository takes up approximately 15,000 blocks, it is recommended that you allow between 50,000 and 250,000 blocks, depending on how much the repository is used.

To add the entry for the CDD_USER rights identifier, enter the following commands:

```
$ SET DEFAULT SYS$SYSTEM
$ RUN SYSMAN
SYSMAN> DISKQUOTA ADD CDD_USER/DEVICE=YOURDISK/PERMQUOTA=50000
```

3. Create the anchor for the compatibility repository with the CDD_USER identifier as the owner. For example, if your compatibility repository anchor is SYS$SYSROOT:[000000]CDDPLUS.DIR, enter the following command:

   ```
   $ CREATE/DIRECTORY/OWNER=CDD_USER SYS$SYSROOT:[000000]CDDPLUS.DIR
   ```

4. Grant the CDD_USER identifier with the resource attribute to all users of the repository:

   ```
   $ SET DEFAULT SYS$SYSTEM
   $ RUN AUTHORIZE
   UAF> GRANT/IDENTIFIER CDD_USER/ATTRIBUTE=RESOURCE user-name
   UAF> EXIT
   ```

The only users who can use the repository if these steps are followed are users with the CDD_USER identifier and users with quotas explicitly set for them on the compatibility repository's disk.

4.7 Improving the I/O Performance of the Repository Database

The following operations can be performed to effectively tune the repository database and make the application run under optimal conditions:

- move the database to multiple disks to reduce contention
- reduce database extensions to improve I/O performance
- reduce snapshots of the database to improve I/O performance
- reduce index node depth

4.7.1 Moving the Repository Database to Multiple Disks

To improve I/O performance, spread the repository database over multiple disks to reduce disk contention and reduce I/O.

Note

Moving the database will dramatically increase the complexity of performing backup operations.

There are three options you can use:

- Move the database file.

  Move the database file when I/O contention is a major problem. Enter the following command to move the database file:

  ```
  $  RMU/MOVE_AREA disk:[anchor_dir]CDD$DATABASE CDD$DATA
  _$  /DIRECTORY=[directory_spec]
  ```

- Move the database journal file.

  You can move the recovery unit journal file (.RUJ) for an individual process. If you lose the .RUJ file, the repository is irrevocably corrupt, and you must revert to a backup. Enter the following command to move the recovery unit journal file:

  ```
  $  DEFINE RDMS$RUJ DISK:[RDM$RUJ]
  ```

- Move the repository binary files.

  You can move the repository binary files only when a repository is created. Enter the following command to move the binary files:

  ```
  $ REPOSITORY OPERATOR DEFINE REPOSITORY disk:[anchor_dir] -
  _$ ALTERNATE_ROOT another_disk:[dir].
  ```

4.7.2 Reducing Repository Database Extensions

As a repository database grows, it will be extended. Extensions can cause extra I/O operations, which slow performance considerably.

Use the Oracle RMU Dump command to assemble an overall picture of the Oracle Rdb relational database. For example, the output from an Oracle RMU Dump operation reveals clues about the number of times the database has been extended.

In Example 4–3 the current physical page count of the storage area CDD$DATA is 1246, and the database has been extended five times.

Example 4–3 Oracle RMU Dump Output Showing Extensions

```
$ RMU/DUMP SYS$COMMON:[CDDPLUS]CDD$DATABASE
   .
   .
   .
Storage area CDD$DATA
    Area ID number is 2
    Storage area page format is mixed
    Filename is SYS$COMMON:[CDDPLUS]CDD$DATA.RDA;1
    Page size is 8 blocks
    Initial data page count was 501
    Current physical page count is 1246
    Extends are enabled
      - Area has been extended 5 times
      - Extend area by 20%, minimum of 99 pages, maximum
        of 9999 pages
      - Volume set spreading is enabled
    Area has space management pages
      - Current SPAM page count is 1
      - Interval is 4008 data pages
      - Thresholds are 70%, 85%, and 95%
    Snapshots are allowed and enabled
    Snapshot area ID number is 4
    Area last backed up at 7-JULY-1994 12:18:08.27
    Area has never been incrementally restored
```

Example 4–4 shows how to remove the database extensions to improve I/O performance. In this example, a regular repository backup is performed. The database is then exported and deleted. Finally, the database is imported into an empty directory and the physical page count allocation is reset.

Note

If you use the SQL CREATE STORAGE AREA clause with the SQL IMPORT statement, only the storage area parameters you specify are created. Other unspecified parameters assume the default values; they do not inherit the existing values. If you do not use the CREATE STORAGE AREA clause with the IMPORT statement, the default values are inherited from the export file. For more information on storage area parameters, see the *Oracle Rdb7 SQL Reference Manual*.

The page allocation is determined by the current physical page count. In Example 4–4, 1250 pages are allocated because the current physical page count of the storage area is 1246.

Example 4–4 Removing the Database Extensions

```
$ BACKUP/VERIFY/EXCLUDE=(.RDA,.RDB,.SNP) -
_$ disk:[anchor_dir] disk:[different-dir]filename.BCK/SAVE

$ RMU/BACKUP disk:[anchor_dir]CDD$DATABASE -
_$ disk:[different-dir]filename.RBF

$ SQL
SQL> EXPORT DATABASE FILE disk:[anchor_dir]CDD$DATABASE.RDB
cont> INTO EXPORTEDFILE.RBR;

SQL> DROP DATABASE FILENAME disk:[anchor_dir]CDD$DATABASE.RDB;

SQL> IMPORT DATABASE FROM EXPORTEDFILE.RBR
cont> FILENAME disk:[anchor_dir]CDD$DATABASE
cont> CREATE STORAGE AREA CDD$DATA
cont> ALLOCATION IS 1250
cont> PAGE SIZE IS 8 BLOCKS
cont> PAGE FORMAT IS MIXED
cont> THRESHOLDS ARE (70,85,95)
cont> INTERVAL IS 4008;
SQL> EXIT

$ RMU/DUMP disk:[anchor_dir...]CDD$DATABASE
```

The Oracle RMU Dump command output now shows that the area has never been extended. The reduction in I/O will improve the performance of the repository application.

```
Storage area CDD$DATA
   .
   .
   .
 - Initial data page count was 1245
 - Current physical page count is 1250
Extension...
 - Extends are enabled
 - Extend area by 20%, minimum of 99 pages, maximum of 9999
   pages
 - Volume set spreading is enabled
 - Area has never been extended
   .
   .
   .
```

See the *Oracle RMU Reference Manual* for more information on the Oracle RMU Dump command.

4.7.3 Reducing Snapshots of the Database to Improve I/O

A snapshot is a picture of the database for read-only transactions. Read-only repository transactions are:

- CDO SHOW commands
- language compilations that use the repository
- DMU access of CDO format definitions

By turning off snapshots for single-user repositories, performance is increased by as much as 20% in the extra I/O saved in the update operations. Disabling the snapshot utility on the database benefits a single, read-only user. This operation reduces the lock conflicts of concurrent accessors.

Periodically, reduce the size of the snapshot file by using the /COMPRESS qualifier with the VERIFY command. For example:

```
CDO> VERIFY/COMPRESS disk:[anchor_dir]
```

The following restrictions apply to using VERIFY/COMPRESS:

- You must be the only user of the repository when you issue the VERIFY /COMPRESS command.
- VERIFY/COMPRESS must be the first command you enter after starting a CDO session.

Three snapshot options are available: ENABLED IMMEDIATE, ENABLED DEFERRED, and DISABLED.

4.7.3.1 Enabled Immediate Snapshots

By enabling snapshot files, read-only users can access snapshot files and avoid record-locking conflicts with users who are updating the database. This option is the default.

```
SQL> ALTER DATABASE FILENAME dict1:CDD$DATABASE
cont>  SNAPSHOT IS ENABLED IMMEDIATE;
```

4.7.3.2 Enabled Deferred Snapshots

The deferred option saves overhead by writing to the snapshot file only when a read-only transaction is in progress. Use the deferred option when you often have large updates with no readers in the repository. To defer snapshot files, enter the following SQL command:

```
SQL>  ALTER DATABASE FILENAME dict1:CDD$DATABASE
cont>  SNAPSHOT IS ENABLED DEFERRED;
```

4.7.3.3 Disabled Snapshots

If snapshot files are disabled, attempts to ready the database in read-only mode start a read/write transaction. In general, use the disabled option only with single-user repositories. To disable snapshot files, enter the following SQL command:

```
SQL>  ALTER DATABASE FILENAME dict1:CDD$DATABASE
cont>  SNAPSHOT IS DISABLED;
```

You can also improve performance by scheduling times when you allow read-only access. For example:

1. Enable snapshot files when update activity is low.
2. Allow read-only access for a certain amount of time.
3. Disable snapshot files again until the next scheduled read-only access time.

For more information on controlling snapshot files, refer to the *Oracle Rdb7 Guide to Database Maintenance*.

4.7.4 Reducing Index Node Depth

Use the Oracle RMU Analyze command to see how deep the index structure is for CDD$DATABASE.RDB.

If the index structure is too deep (three or four levels are generally too deep), consider moving some entities to different repositories. If that is not appropriate, use the Oracle RMU Analyze Placement command to see the index path lengths. For each index affected, use the SQL DROP INDEX statement, then the SQL CREATE INDEX statement.

5
Managing Repository Elements

A data administrator maintains elements centrally, converting existing elements and distributing them to other users. Data administrators can build a conceptual model of an organization's data using consistent terminology. Programmers can then include the elements in their programs, ensuring consistency and establishing control of a critical organizational resource.

This chapter shows you how to perform the following tasks:

- define Oracle CDD/Repository elements
- display Oracle CDD/Repository elements
- change Oracle CDD/Repository elements
- copy Oracle CDD/Repository elements
- delete and purge Oracle CDD/Repository elements

5.1 Creating a Framework for Defining Elements

This section describes how you set up your environment by creating a framework in which you define your elements. To create this framework you use partitions, contexts, and collections.

Before you set up any real production repositories, please read all the Oracle CDD/Repository documentation.

5.1.1 Defining the Partition Hierarchy

Partitions are named divisions of the repository that model and implement work-flow levels in support of your software development methodology.

You can organize your repository into a hierarchical structure by defining partitions that mirror the approval process for your environment. Elements exist in only one partition, and they are promoted as they qualify for the next level.

The first partition is the root partition, and it is linked to subsequent partitions by a parent-child relationship. For example:

```
CDO> DEFINE PARTITION RELEASED_SYSTEMS AUTOPURGE.
CDO> DEFINE PARTITION FIELDTEST_RELEASE
cont> PARENT_PARTITION IS RELEASED_SYSTEMS AUTOPURGE.
CDO> DEFINE PARTITION SECOND_BASELEVEL
cont> PARENT_PARTITION IS FIELDTEST_RELEASE AUTOPURGE.
CDO> DEFINE PARTITION FIRST_BASELEVEL
cont> PARENT_PARTITION IS SECOND_BASELEVEL.
CDO> DEFINE PARTITION FRONT_END
cont> PARENT_PARTITION IS FIRST_BASELEVEL AUTOPURGE.
CDO> DEFINE PARTITION BACK_END
cont> PARENT_PARTITION IS FIRST_BASELEVEL
cont> LOOKASIDE_PARTITION IS FRONT_END AUTOPURGE.
```

In this example, the LOOKASIDE_PARTITION makes the contents of FRONT_END visible to BACK_END. The AUTOPURGE keyword ensures that intermediate versions of elements in the partition are purged when you promote the latest version.

For more information about partitions, see the *Oracle CDD/Repository Architecture Manual*.

5.1.2 Creating a Context

Once you have designed your partition hierarchy, you can further control your repository elements by creating a **context**. A context contains pointers to the top of a collection hierarchy and the bottom of a partition hierarchy. Contexts usually correspond to a developer or development tasks. Within a context, you can have reserved or controlled elements.

The file system hierarchy under your context reflects the directory hierarchy in your repository. As you define your context, you must also specify a base partition. This **base partition** specifies which elements of a repository you can see. You can view only those elements that are in your base partition or higher parent partitions. The following example defines the contexts of your repository:

```
CDO> DEFINE CONTEXT DEVELOPMENT_CONTEXT
cont> BASE_PARTITION IS FIRST_BASELEVEL
cont> DEFAULT_ATTACHMENT IS LATEST_CHECKIN.
CDO> DEFINE CONTEXT BILL_CONTEXT
cont> BASE_PARTITION IS FRONT_END.
CDO> DEFINE CONTEXT SUSAN_CONTEXT
cont> BASE_PARTITION IS BACK_END.
CDO> DEFINE CONTEXT QA_CONTEXT
cont> BASE_PARTITION IS FIELDTEST_RELEASE.
```

The base partition, or the lowest visible partition, is set for four different contexts. Each context has a unique view of the partition hierarchy. A context also points to the collection hierarchy. The top of the partition hierarchy is a version of a particular collection whose members can also be versions of other collections. At the bottom of the partition hierarchy are versions that make up the system under development. Within a configuration context, you can manipulate the complete system or subsystems. By changing or creating configuration contexts, you can get different views of the system.

If you do not use a context, the fields and records you define are considered uncontrolled elements because they are not contained in a partition hierarchy. See Section 5.2.1 for a discussion on the difference between controlled and uncontrolled elements in the repository.

5.1.3 Defining CDD$CONTEXT

Once your context exists, you can define the logical name CDD$CONTEXT to be the name of the context. For example:

```
$ DEFINE CDD$CONTEXT "DISK1:[CORPORATE.MIS]FIRST_BASELEVEL.DEVELOPMENT_CONTEXT"
```

When you enter CDO, your context is automatically set. If the context is in your default directory, you do not have to specify your entire pathname.

You can also enter the CDO SET CONTEXT command after you enter a CDO session and set your default context to another context. This context lasts for the duration of the session or until you change it.

If you have not defined the context and issued the SET CONTEXT command, CDO issues a warning and ignores the command for that session.

5.1.4 Creating a Collection

A **collection** specifies what versions of elements make up your system at a particular time. Specifying a particular version of a collection gives your context a view of a particular version or snapshot of your system. A collection can also be a versioned element; successive versions of the collection represent successive configurations of the system.

You must set a context before you can define or reserve a collection.

The collection hierarchy has a parallel directory hierarchy. A single directory for each part of a project is reasonable. For example:

```
CDO> SET CONTEXT DEVELOPMENT_CONTEXT
CDO> DEFINE COLLECTION COMPILER_C.
CDO> RESERVE COLLECTION COMPILER_C
CDO> DEFINE COLLECTION FRONT_END_C.
CDO> DEFINE COLLECTION BACK_END_C.
CDO> DEFINE COLLECTION PARSER.
CDO> REPLACE COLLECTION COMPILER_C
```

5.2 Creating Field and Record Definitions

You can define and store many types of elements in a repository. This section describes field and record definitions. See Chapter 8 for information about creating database definitions in Oracle CDD/Repository.

You can create field and record definitions in CDO using one of the following methods:

- the CDO screen editor
- the CDO DEFINE command
- a command procedure created at DCL level ($) but executed within CDO

Chapter 2 describes the CDO screen editor. It is menu driven and generates the appropriate commands for you as you create the definitions.

Using a text editor, you can create any of the definitions in this chapter as a command procedure. You can then execute the command procedure at the CDO prompt with the AT command (@). For more information about creating definitions using command procedures, see Section 1.7.

5.2.1 Using Uncontrolled or Controlled Elements

A **controlled element** is one that you create within a partition and context, then constrain using the CDO CONSTRAIN command. Using the CONSTRAIN command places the element on the base partition of the context.

For example:

```
CDO> DEFINE PARTITION FIRST_QUARTER.
CDO> DEFINE CONTEXT SUBSCRIPTIONS BASE_PARTITION IS FIRST_QUARTER.
CDO> SET CONTEXT SUBSCRIPTIONS
CDO> CONSTRAIN FIELD/CLOSURE=TO_BOTTOM TEST_A
```

The /CLOSURE=TO_BOTTOM qualifier of the CDO CONSTRAIN command constrains all descendants of field TEST_A that are uncontrolled. Once you have constrained an element, you can use the CDO PROMOTE command to move a version of the element to higher partitions.

The CONSTRAIN command is irreversible. Once an element is controlled, you can never make it uncontrolled.

When you modify controlled versions of an element, you must use the CDO RESERVE and REPLACE commands. Changing controlled elements and using the version control model is described in more detail in Section 5.4.

If an element exists outside the normal restrictions of the version control model and it does not reside in any partition, it is said to be **uncontrolled**. An uncontrolled element can be changed in place—it is not necessary to reserve and replace it when making changes.

You cannot have a controlled version of an element dependent on an uncontrolled version. For example, you cannot use an uncontrolled field definition in a controlled record definition. CDO responds with an error message if you try to control an element that depends on an uncontrolled element.

To determine if an element is controlled or uncontrolled, enter the CDO SHOW command, as follows:

```
CDO> SHOW FIELD /ALL TEST_A

Definition of field TEST_A
|   acl
(IDENTIFIER=[CDD,JONES],ACCESS=READ+WRITE+MODIFY+ERASE+
  SHOW+DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[*,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+
  OPERATOR+ADMINISTRATOR)
|   Created time             6-MAY-1991 13:30:37.27
|   Modified time            6-MAY-1991 13:30:42.53
|   Owner                    [CDD,JONES]
|   Status                   Available
|   Freeze time              6-MAY-1991 13:30:37.27
|   Controlled               No
   .
   .
   .
```

5.2.2 Creating Controlled Field Definitions

A field definition is the smallest element that you can store and access in the repository. Field definitions can include properties ranging from data type information to validation criteria.

Note

Before creating field and record definitions, set the default repository directory to the location where you want the definitions created. Otherwise, you must specify the full pathname each time you create a definition. For more information about setting the default repository directory, see Section 1.5.1 and Section 1.5.2.

The following example shows how to create a controlled field with the CDO DEFINE FIELD command. To define a controlled field you must first set a context and reserve a collection.

```
CDO> SET CONTEXT DEVELOPMENT_CONTEXT
CDO> RESERVE COLLECTION COMPILER_C
CDO> DEFINE FIELD FIELD_EXAMPLE
cont>   DESCRIPTION IS /*A TEST FIELD CREATED TO */
cont>         /*DEMONSTRATE A RANGE OF PROPERTIES*/
cont>  AUDIT IS /*NOTICE THAT THE TEXT EXPLANATION OF */
cont>         /*THE REASON OR HISTORY OF A FIELD MAY BE INPUT*/
cont> DATATYPE IS UNSIGNED LONGWORD 3
cont> INITIAL_VALUE IS 3
cont> MISSING_VALUE IS -1
cont> NAME FOR BASIC IS BASIC_NAME_EXAMPLE
cont> QUERY_HEADER IS "QUERY_EXAMPLE"
cont> QUERY_NAME IS "PROMPT EXAMPLE"
cont> VALID IF FIELD_EXAMPLE < 1000.
CDO> REPLACE COLLECTION COMPILER_C
```

5.2.2.1 Defining a One-Dimensional Array Field

The following example defines a controlled field as a one-dimensional array. The field definition includes three properties:

- The INITIAL_VALUE property allows an initial value to be assigned by an application at run time. (Some applications cannot support initial values.)
- The OCCURS property specifies the number of elements in the array.
- The AUDIT property adds an entry to the definition's history list.

```
CDO> RESERVE COLLECTION PERSONNEL
CDO> DEFINE FIELD 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> REPLACE COLLECTION PERSONNEL
```

5.2.2.2 Defining a Two-Dimensional Array Field

The following example defines a controlled field to be a two-dimensional array. The first dimension has a lower bound of 5 and an upper bound of 20. 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.)

```
CDO> RESERVE COLLECTION PERSONNEL
CDO> DEFINE FIELD 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> REPLACE COLLECTION PERSONNEL
CDO>
```

By default, Oracle CDD/Repository arrays are row major; however, products that support column major arrays translate the order accordingly. You can explicitly specify an array as column major.

5.2.2.3 Using the BASED ON Property

By using the BASED ON property, you can create definitions that vary from existing definitions. You can add more properties to a definition that is based on another, or you can override properties that were assigned to the original definition.

An important feature of the BASED ON property is that if you change the original definition, definitions based on that definition automatically reflect the change.

In the following example, the field definition REGION_CODE is based on the definition of POSTAL_CODE. Because the field REGION_CODE is controlled, the based on field POSTAL_CODE must also be controlled. You cannot define a controlled field using a based on field that is uncontrolled.

The properties assigned to the field POSTAL_CODE are automatically assigned to the field REGION_CODE using the BASED ON property.

For REGION_CODE, the length of the field is increased and an EDIT_STRING property is added. Because the field definition POSTAL_CODE is in another directory, you must specify the full pathname.

```
CDO> RESERVE COLLECTION CONTRACT
CDO> DEFINE FIELD REGION_CODE
cont>   DESCRIPTION IS /* Based on field POSTAL_CODE*/
cont>   BASED ON DISK$01:[CORPORATE.MIS]PERSONNEL.SALARIED.POSTAL_CODE
cont>   DATATYPE IS TEXT
cont>   SIZE IS 9
cont>   EDIT_STRING IS 99999"-"9999.
CDO> REPLACE COLLECTION CONTRACT
CDO>
```

See the *Oracle CDD/Repository CDO Reference Manual* for complete descriptions of the field properties.

5.2.3 Creating Record Definitions

You can create simple record definitions by including previously defined fields. You can then combine field and record definitions into complex record definitions.

Alternatively, you can use the field definitions directly in a layered application to build up complex data structures.

The following example creates a record definition FULL_NAME. The record definition includes the names of previously defined fields that are located in the same directory as the record FULL_NAME.

```
CDO> RESERVE COLLECTION PERSONNEL
CDO> DEFINE RECORD FULL_NAME.
cont>    FIRST_NAME.
cont>    MIDDLE_INIT.
cont>    LAST_NAME.
cont> END FULL_NAME RECORD.
CDO> REPLACE COLLECTION PERSONNEL
CDO>
```

The fields and structures included in a record definition must already be defined. You cannot define elements in Oracle CDD/Repository by including them in other definitions.

You can also use a command procedure to create a record definition and execute the command procedure from CDO. See Section 1.7 for more information about executing command procedures in CDO.

5.2.3.1 Nesting Definitions

You can nest record definitions (include a record definition within another record definition). In the following example, EMPLOYEE_REC includes the record definition FULL_NAME:

```
CDO> RESERVE COLLECTION PERSONNEL
CDO> DEFINE RECORD EMPLOYEE_REC.
cont>    FULL_NAME.
cont>    DEPENDENTS.
cont>    WAGE_CLASS.
cont>    START_DATE.
cont> END EMPLOYEE_REC RECORD.
CDO> REPLACE COLLECTION PERSONNEL
CDO>
```

5.2.3.2 Using the VARIANTS Clause

The VARIANTS clause of the CDO DEFINE RECORD command defines a set of two or more definitions that provide alternative descriptions for the same portion of a record definition.

This lets you map different data types to the same storage location. You can use VARIANT definitions to specify a tag expression whose value is used at run time to determine the current VARIANT.

The following example shows the EMPLOYEE_REC record definition with the RETIRED and DISMISSED variants for the field JOB_CODE:

```
CDO> RESERVE COLLECTION PERSONNEL
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> REPLACE COLLECTION PERSONNEL
CDO>
```

In this example, two variant definitions exist within one VARIANTS clause. At run time, the product using CDO compares the value of each expression in the variant definition to the value in the tag variable. If D is specified for JOB_CODE, then DISMISSED is the current variant definition.

You can include as many variants as you want in a record definition, and you can repeat the VARIANTS clause as required.

5.2.3.3 Creating Top-Down Definitions

CDO lets you 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 containing structures. A **structure** is a named group within a record definition. The structure lets you refer to a field within the record definition and to use this entity as a dependency, even though it is not a separate element.

For example, the OCCURS clause in the following record definition of FAMILY depends on another field definition, NUM_OF_KIDS:

```
CDO> RESERVE COLLECTION PERSONNEL
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> REPLACE COLLECTION PERSONNEL
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 can include elements that have been defined.

The record definitions shown in this chapter represent only a few of the optional clauses and properties; for complete details, see the *Oracle CDD/Repository CDO Reference Manual*.

5.2.4 Understanding DMU and CDO Record Format Differences

A record that is copied from a DMU format dictionary differs from a similar record that is copied from a CDO format dictionary. There is a difference in the CDDL and CDO DEFINE RECORD syntax and the entities to which the syntax translates. The DEFINE RECORD command in CDDL syntax specifies the pathname where the record is to be defined. It does not create a structure with the name specified in the DEFINE RECORD command. In CDO syntax, a CDDL record without a structure maps to a *field*; but, a CDDL record with a structure maps to a *record*. The structure itself is the record definition.

The language compilers must use the record name rather than the structure name when including the record. The name of the structure is the name that a compiler would call the record name. The name specified in the DEFINE RECORD command is only a path to get to the record definition.

The DEFINE RECORD command in CDO syntax specifies that a record is to be created using the name in that command as both the pathname of the record (directory name) and its record name (processing name).

5.2.5 Creating Relationships

Oracle CDD/Repository implicitly creates relationships whenever you explicitly connect elements in some way. For example, a record definition relates 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.

An element is an **owner** of a relationship when it uses or depends on another element. An element is a **member** of a relationship when it is used by another element. A relationship has one owner and one member. An element can be an owner and a member of many different relationships.

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

The following example shows how to assign the field definition GIVEN_NAME the same properties as FIRST_NAME using the BASED ON property. This definition implicitly relates the two fields. Any subsequent change to the definition of FIRST_NAME (the member) can affect GIVEN_NAME (the owner); however, a change to GIVEN_NAME will not affect FIRST_NAME.

```
CDO> SET DEFAULT CDD$TOP.PERSONNEL.CONTRACT
CDO> DEFINE FIELD GIVEN_NAME
cont>   BASED ON CDD$TOP.PERSONNEL.SALARIED.FIRST_NAME.
CDO>
```

If you subsequently define a new version of an element, Oracle CDD/Repository does not include the new version 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 CDO CHANGE command you 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 5.4.10.

5.2.5.1 Viewing Relationships

Use the CDO SHOW commands to review the relationships that your definitions create. The CDO SHOW USED_BY command in the following example 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 example displays the actual properties for GIVEN_NAME:

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

The record definition FULL_NAME is defined as containing the fields FIRST_NAME, MIDDLE_INIT, and LAST_NAME. The subsequent CDO SHOW RECORD command displays the relationships 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>
```

In the following example, the CDO SHOW USES command lists the elements that use the field FIRST_NAME and the relationships that the elements 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>
```

5.2.6 Displaying Relationships Between Elements

To display all of the elements that use a specified definition, use the CDO SHOW USES command. With this information, you can then consider the impact of changing the definition by creating a new version.

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>
```

For more information on how to track changes to definitions, see Section 5.4.5.

When you define fields and records, the most common relationships that are created are:

- CDD$DATA_ELEMENT_BASED_ON—created when one field is based on another.

- CDD$DATA_AGGREGATE_CONTAINS—created when a record definition includes field or record definitions.
- CDD$DATA_ELEMENT_COMPUTED_VALUE—created when an expression in a field definition depends on the resolution of the expression at run time.

Oracle CDD/Repository stores relationships in addition to those listed.

You can relate a definition in a repository to a definition in another repository on the same node or on a different node in a network.

For example, the field ID_NUM is based on a field SOC_SEC in another repository 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, Oracle CDD/Repository creates a local copy of remote CDO definitions. When the CDO DEFINE command is executed, Oracle CDD/Repository searches for a local copy of the remote field definition SOC_SEC. If a local copy does not exist, Oracle CDD/Repository creates a local copy. Therefore, when a remote definition is first defined, the network link between the local and remote node must be viable.

For remote access, you must be running compatible versions of Oracle CDD/Repository and Oracle Rdb on each node. See Section 6.3 for more information on remote access.

5.2.7 Defining Logical Names to Access Multiple Repositories

You can define a logical name to create a link between two elements located on different nodes. For example:

```
$ DEFINE ALL_FIELD-
_$ NODE1::USER1:[TEAM.TESTDATA],-
_$ NODE2::USER2:[PUBLIC]
```

After you have defined the logical name, you can locate all FIELD element types with the same name, word, or phrase in common within a field name. For example:

```
CDO> DIRECTORY ALL_FIELD.CDDP.SHARE.FIELD.*PERSON*

  Directory NODE1::USER1:[TEAM.TESTDATA]CDDP.SHARE.FIELD

PERSON_LCL_BADGE_NO;1
PERSON_NAME;2
PERSON_NAME;1
SVC_CALL_ASEND_PERSON_INDCR;2
SVC_PERSON_SUPRT_LVL_CD;1
SVC_PERSON_SUPRT_LVL_NAME;2

  Directory NODE2::USER2:[PUBLIC]CDDP.SHARE.FIELD

 CNTCT_PERSON_NAME;1
```

5.3 Displaying Repository Definitions

The CDO DIRECTORY command provides an easy way to check on the contents and structure of a portion of your repository. When you issue the DIRECTORY command, Oracle CDD/Repository displays the name of your current directory, then lists each definition, displaying the name and type of the definition.

In the following example, the DIRECTORY command lists the contents of the top directory called [CORPORATE.MIS]:

```
CDO> SET DEFAULT DISK$01:[CORPORATE.MIS]
CDO> DIRECTORY
Directory DISK$01:[CORPORATE.MIS]
ASSET_REC                                                   RECORD
CDD$PROTOCOLS                                               DIRECTORY
EMPLOYEE_NAME                                               RECORD
INVENTORY_CODE                                              FIELD
MODEL_NUMBER                                                FIELD
PAY_REC                                                     RECORD
PERSONNEL                                                   DIRECTORY
SERIAL_NUMBER                                               FIELD
CDO>
```

You can also use wildcards to list definitions in a directory and in specified subdirectories.

5.3.1 Listing Specific Types of Definitions

Use the optional /TYPE qualifier with the CDO DIRECTORY command to limit the type of definitions that the DIRECTORY command displays. In the following example, only the record definitions in the directory are displayed:

```
CDO> DIRECTORY /TYPE=RECORD

Directory DISK$01:[CORPORATE.MIS]PERSONNEL.CONTRACT
ASSET_REC                                                   RECORD
EMPLOYEE_NAME                                               RECORD
PAY_REC                                                     RECORD
CDO>
```

Use the /FULL qualifier with the CDO DIRECTORY command to obtain a list of definitions and all the related information in the repository for the definition, such as the creation date, modification date, protection provisions, and relative size. For example:

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

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

EMPLOYEE_NAME(1)                                  RECORD
Created:   4-AUG-1995 16:09:52.29     Revised:  4-AUG-1995 17:13:01.12
Owner: [13,10]
Access Cntrl List:
(IDENTIFIER=[13,10],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+
DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[VDD,CDDCDD],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+CHANGE+D
ELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[*,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+OPERATOR+ADMINISTRATOR)
Size: 1377       dictionary_format: CDO format

   .
   .
   .

INVENTORY_CODE(1)                               FIELD
Created: 6-MAR-1991 12:19:29.57  Revised: 6-MAR-1991 12:19:29.57
Size: 30 blocks
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)
```

```
   .
   .
   .
SERIAL_NUMBER(1)                                  FIELD
Created: 7-JAN-1991 19:10:23.42   Revised: 28-JAN-1991 15:18:20:56
Size: 28 blocks
Owner [13,62]
Access Control List:
(IDENTIFIER=[13,62],ACCESS=ALL)
(IDENTIFIER=PERSONNEL,ACCESS=READ+WRITE+CREATE)
(IDENTIFIER=[20,*],ACCESS=READ)
   .
   .
   .
```

5.3.2 Displaying Elements with SHOW Commands

You can display the properties of a specified definition with the CDO SHOW FIELD, SHOW RECORD, and SHOW DATABASE commands.

In the following example, the SHOW FIELD command displays the basic information stored in the repository for the field definition SUPERVISOR_NAME.

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

To display more information than the default output of the SHOW command, use one of the optional qualifiers.

The following example uses the /FULL qualifier:

```
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 the /BRIEF qualifier or the /FULL qualifier shows only the **user-specified** properties, which are stored in the directory by the user who created or changed the field. For more information about user-specified properties, see Section 5.2.2.

To display more information about a definition, use the /ALL qualifier. The CDO SHOW/ALL command displays the definition's **system-specified** properties, in addition to its user-specified properties. System-specified properties include the owner, creation date, modification date, protection provisions, and history list.

The following SHOW FIELD/ALL command displays both the user-specified and system-specified properties of the field definition TEST_A:

```
CDO> SHOW FIELD/ALL TEST_A

Definition of field TEST_A
|   acl
(IDENTIFIER=[CDD,JONES],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+
 DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[*,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+
 OPERATOR+ADMINISTRATOR)
|   Created time             15-AUG-1991 14:57:31.43
|   Modified time            15-AUG-1991 14:57:31.43
|   Owner                    [CDD,JONES]
|   Status                   Available
|   Freeze time              15-AUG-1991 14:57:31.43
|   Controlled               No
|   Allow concurrent         No
|   Datatype                 text size is 2 characters
|   |   History entered by JONES ([CDD,JONES])
|   |       using CDO V2.0
|   |       to CREATE definition on 15-AUG-1991 14:57:31.26
```

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

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

Using the /FULL qualifier with the CDO SHOW RECORD command displays the included field definitions and properties, as follows:

```
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 issue the CDO SHOW DATABASE command, the database name, file name, fully qualified pathname, and user-specified file name are displayed, as shown in the following example:

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

To view the complete definition of an Oracle Rdb database, use the following command:

```
CDO> SHOW USED_BY /FULL element-name
```

Specify the name of the database as the element-name parameter. For example:

```
CDO> SHOW USED_BY /FULL FOO
Members of  DISK1:[USER1.REP3]FOO(1)
|   FOO                              (Type : CDD$RDB_DATABASE)
|   |   via CDD$DATABASE_SCHEMA
|   |   DISK1:[USER1.REP3]CDD$RDB_SYSTEM_METADATA.RDB$OBJECT_BLR(1) (Type: FIELD)
|   |   |   via CDD$RDB_DATA_ELEMENT
|   |   DISK1:[USER1.REP3]CDD$RDB_SYSTEM_METADATA.RDB$DESCRIPTION(1) (Type: FIELD)
|   |   |   via CDD$RDB_DATA_ELEMENT
|   |   DISK1:[USER1.REP3]CDD$RDB_SYSTEM_METADATA.RDB$CONSTRAINT_TYPE(1) (Type: FIELD)
|   |   |   via CDD$RDB_DATA_ELEMENT
   .
   .
   .
```

To see the history list for a definition, use the /AUDIT qualifier with the CDO SHOW FIELD or SHOW RECORD command.

Note

When an element is deleted, its history is not retained unless the logical name CDD$KEEP_HISTORY_ON_DELETE is defined. If this logical name exists, regardless of its value, the history of a deleted element is retained and moved to the CDD$PROTOCOLS.CDD$SELF element for the repository. To see the history, use the following CDO commands:

```
CDO> SET DEFAULT anchor_dir:CDD$PROTOCOLS
CDO> SHOW GENERIC MCS_DATABASE/ALL CDD$SELF
```

Syntax information for all the CDO SHOW commands is contained in the *Oracle CDD/Repository CDO Reference Manual*.

5.4 Changing Elements in CDO

To change an element you use the CDO CHANGE command. If the element is controlled, you must reserve the element, as well as the collection, then change the element. The following example changes one line in the FIELD_EXAMPLE field definition:

```
CDO> RESERVE COLLECTION COMPILER_C
CDO> RESERVE FIELD FIELD_EXAMPLE
CDO> CHANGE FIELD FIELD_EXAMPLE
cont> VALID IF FIELD_EXAMPLE < 900.
CDO> REPLACE FIELD FIELD_EXAMPLE
CDO> REPLACE COLLECTION COMPILER_C
```

You change an uncontrolled element in place using the CDO CHANGE command. When you make a **change in place**, it is not necessary to reserve or replace the element you are changing. The element retains the version number it had before the changes were made.

After you change an element with the CDO CHANGE command, you can no longer access the original definition at compilation time.

Note

The CDO DEFINE and CHANGE commands, like the CDO DELETE command, require a terminating period at the end of the command line.

5.4.1 Documenting Changes

It is good practice to document all changes. You can include remarks with the DESCRIPTION and AUDIT properties for both the CDO CHANGE and CDO DEFINE commands.

Use the DESCRIPTION property to describe the element and the AUDIT property to comment on the changes made to the element.

5.4.2 Using the Version Control Model

Elements residing in a repository can evolve or change over time. Each evolution results in a new version of that element. Both old and new versions need to be managed. You must know the following information when an element changes:

- the most current version of the element
- the element versions that make up a particular program or system
- the record of element versions in use at a particular time
- the history of change

To modify a repository element, use the CDO RESERVE command, which creates a copy of the element to be changed. This copy is called a **ghost** version. When you reserve the element, a new version number that is one higher than the latest version in the partition is assigned to the ghost before you make any changes or replace the element. The ghost is available only to you and is invisible to other repository users. The subsequent CDO REPLACE command makes the new version of the modified element generally available.

If you have not modified the reserved element, you can issue a CDO UNRESERVE command, which deletes the ghost version.

5.4.3 Tracking Parallel Versions of a Project

If both a production version and a development version of a controlled element exist in parallel, these parallel versions are called **branches**. At some time these versions must be reconciled and merged.

The CDO session in Example 5–1 shows an example of creating a branch version of a field, then merging the branch into the main line of descent.

Note

In this example, the repository DISK1:[USER.DIC]TEST_MERGE has already been defined.

First, set up the partition hierarchy and define the context and collection.

Example 5–1 Version Control Example

```
   .
   .
   .
CDO> SET DEFAULT DISK1:[USER.DIC]TEST_MERGE
CDO> DEFINE PARTITION FINAL_REVIEW AUTOPURGE.
CDO> DEFINE PARTITION LIMITED_REVIEW PARENT_PARTITION FINAL_REVIEW.
CDO> DEFINE PARTITION DRAFT_REVIEW PARENT_PARTITION LIMITED_REVIEW.

CDO> DEFINE CONTEXT PERSONNEL BASE_PARTITION DRAFT_REVIEW.
CDO> SET CONTEXT PERSONNEL
CDO> DEFINE COLLECTION EMPLOYEE_RECORDS.
CDO> RESERVE COLLECTION EMPLOYEE_RECORDS
```

Next, define a record containing some fields.

```
CDO> DEFINE FIELD LAST_NAME DATATYPE TEXT SIZE 30.
CDO> DEFINE FIELD MID_INITIAL DATATYPE TEXT SIZE 1.
CDO> DEFINE FIELD FIRST_NAME DATATYPE TEXT SIZE 20.
CDO> DEFINE RECORD EMPLOYEE_NAME.
   LAST_NAME.
   FIRST_NAME.
   MID_INITIAL.
   END RECORD.
CDO> REPLACE COLLECTION EMPLOYEE_RECORDS
```

(continued on next page)

Example 5–1 (Cont.) Version Control Example

```
CDO> DIRECTORY
 Directory DISK1:[USER.DIC]TEST_MERGE
DRAFT_REVIEW                                MCS_PARTITION
EMPLOYEE_NAME(1)                            RECORD
EMPLOYEE_RECORDS(2)                         MCS_COLLECTION
EMPLOYEE_RECORDS(1)                         MCS_COLLECTION
FINAL_REVIEW                                MCS_PARTITION
FIRST_NAME(1)                               FIELD
LAST_NAME(1)                                FIELD
LIMITED_REVIEW                              MCS_PARTITION
MID_INITIAL(1)                              FIELD
PERSONNEL                                   MCS_CONTEXT
```

Then, make changes to field FIRST_NAME. Use the /CLOSURE=TO_TOP qualifier when reserving the field FIRST_NAME and all its ancestors.

```
CDO> RESERVE FIELD /CLOSURE=TO_TOP FIRST_NAME
CDO> CHANGE FIELD FIRST_NAME(2) DESCRIPTION "FIRST_NAME(2)".
CDO> REPLACE FIELD /CLOSURE=TO_TOP FIRST_NAME(2)

CDO> DIRECTORY
 Directory DISK1:[USER.DIC]TEST_MERGE
DRAFT_REVIEW                                MCS_PARTITION
EMPLOYEE_NAME(2)                            RECORD
EMPLOYEE_NAME(1)                            RECORD
EMPLOYEE_RECORDS(3)                         MCS_COLLECTION
EMPLOYEE_RECORDS(2)                         MCS_COLLECTION
EMPLOYEE_RECORDS(1)                         MCS_COLLECTION
FINAL_REVIEW                                MCS_PARTITION
FIRST_NAME(2)                               FIELD
FIRST_NAME(1)                               FIELD
LAST_NAME(1)                                FIELD
LIMITED_REVIEW                              MCS_PARTITION
MID_INITIAL(1)                              FIELD
PERSONNEL                                   MCS_CONTEXT
```

Create a branch until new version of the field is tested.

```
CDO> RESERVE RECORD /CLOSURE=TO_TOP EMPLOYEE_NAME
CDO> RESERVE FIELD /BRANCH=TEST FIRST_NAME
CDO> CHANGE FIELD FIRST_NAME(2:TEST:1)
cont> DESCRIPTION "FIRST_NAME(2:TEST:1)".
CDO> REPLACE FIELD FIRST_NAME(2:TEST:1)
CDO> REPLACE RECORD /CLOSURE=TO_TOP EMPLOYEE_NAME
```

(continued on next page)

Example 5–1 (Cont.) Version Control Example

```
CDO> DIRECTORY
 Directory DISK1:[USER.DIC]TEST_MERGE
DRAFT_REVIEW                                MCS_PARTITION
EMPLOYEE_NAME(3)                            RECORD
EMPLOYEE_NAME(2)                            RECORD
EMPLOYEE_NAME(1)                            RECORD
EMPLOYEE_RECORDS(4)                         MCS_COLLECTION
EMPLOYEE_RECORDS(3)                         MCS_COLLECTION
EMPLOYEE_RECORDS(2)                         MCS_COLLECTION
EMPLOYEE_RECORDS(1)                         MCS_COLLECTION
FINAL_REVIEW                                MCS_PARTITION
FIRST_NAME(2)                               FIELD
FIRST_NAME(2:TEST:1)                        FIELD
FIRST_NAME(1)                               FIELD
LAST_NAME(1)                                FIELD
LIMITED_REVIEW                              MCS_PARTITION
MID_INITIAL(1)                              FIELD
PERSONNEL                                   MCS_CONTEXT
```

Make some more changes to the field and record elements.

```
CDO> RESERVE RECORD /CLOSURE=TO_TOP EMPLOYEE_NAME
CDO> RESERVE FIELD FIRST_NAME(2:TEST:1)
CDO> CHANGE FIELD FIRST_NAME(2:TEST:2)
cont> DESCRIPTION "FIRST_NAME(2:TEST:2)".
CDO> REPLACE FIELD FIRST_NAME(2:TEST:2)
CDO> REPLACE RECORD /CLOSURE=TO_TOP EMPLOYEE_NAME
CDO> DIRECTORY
 Directory DISK1:[USER.DIC]TEST_MERGE
DRAFT_REVIEW                                MCS_PARTITION
EMPLOYEE_NAME(4)                            RECORD
EMPLOYEE_NAME(3)                            RECORD
EMPLOYEE_NAME(2)                            RECORD
EMPLOYEE_NAME(1)                            RECORD
EMPLOYEE_RECORDS(5)                         MCS_COLLECTION
EMPLOYEE_RECORDS(4)                         MCS_COLLECTION
EMPLOYEE_RECORDS(3)                         MCS_COLLECTION
EMPLOYEE_RECORDS(2)                         MCS_COLLECTION
EMPLOYEE_RECORDS(1)                         MCS_COLLECTION
FINAL_REVIEW                                MCS_PARTITION
FIRST_NAME(2)                               FIELD
FIRST_NAME(2:TEST:2)                        FIELD
FIRST_NAME(2:TEST:1)                        FIELD
FIRST_NAME(1)                               FIELD
LAST_NAME(1)                                FIELD
LIMITED_REVIEW                              MCS_PARTITION
MID_INITIAL(1)                              FIELD
```

(continued on next page)

Example 5–1 (Cont.) Version Control Example

```
PERSONNEL                                MCS_CONTEXT
```

Attach to composite makes FIRST_NAME(2) a controlled element under composite EMPLOYEE_RECORDS.

```
CDO> RESERVE COLLECTION EMPLOYEE_RECORDS
CDO> DETACH FIELD FIRST_NAME(2:TEST:2) FROM EMPLOYEE_RECORDS
CDO> ATTACH FIELD FIRST_NAME(2) TO EMPLOYEE_RECORDS
CDO> REPLACE COLLECTION EMPLOYEE_RECORDS
CDO> DIRECTORY
 Directory DISK1:[USER.DIC]TEST_MERGE
DRAFT_REVIEW                             MCS_PARTITION
EMPLOYEE_NAME(4)                         RECORD
EMPLOYEE_NAME(3)                         RECORD
EMPLOYEE_NAME(2)                         RECORD
EMPLOYEE_NAME(1)                         RECORD
EMPLOYEE_RECORDS(6)                      MCS_COLLECTION
EMPLOYEE_RECORDS(5)                      MCS_COLLECTION
EMPLOYEE_RECORDS(4)                      MCS_COLLECTION
EMPLOYEE_RECORDS(3)                      MCS_COLLECTION
EMPLOYEE_RECORDS(2)                      MCS_COLLECTION
EMPLOYEE_RECORDS(1)                      MCS_COLLECTION
FINAL_REVIEW                             MCS_PARTITION
FIRST_NAME(2)                            FIELD
FIRST_NAME(2:TEST:2)                     FIELD
FIRST_NAME(2:TEST:1)                     FIELD
FIRST_NAME(1)                            FIELD
LAST_NAME(1)                             FIELD
LIMITED_REVIEW                           MCS_PARTITION
MID_INITIAL(1)                           FIELD
PERSONNEL                                MCS_CONTEXT
```

(continued on next page)

Example 5–1 (Cont.) Version Control Example

Make more changes.

```
CDO> RESERVE FIELD /CLOSURE=TO_TOP FIRST_NAME(2)
CDO> CHANGE FIELD FIRST_NAME(3) DESCRIPTION "FIRST_NAME(3)".
CDO> REPLACE FIELD /CLOSURE=TO_TOP FIRST_NAME(3)
CDO> DIRECTORY
 Directory DISK1:[USER.DIC]TEST_MERGE
DRAFT_REVIEW                                MCS_PARTITION
EMPLOYEE_NAME(4)                            RECORD
EMPLOYEE_NAME(3)                            RECORD
EMPLOYEE_NAME(2)                            RECORD
EMPLOYEE_NAME(1)                            RECORD
EMPLOYEE_RECORDS(7)                         MCS_COLLECTION
EMPLOYEE_RECORDS(6)                         MCS_COLLECTION
EMPLOYEE_RECORDS(5)                         MCS_COLLECTION
EMPLOYEE_RECORDS(4)                         MCS_COLLECTION
EMPLOYEE_RECORDS(3)                         MCS_COLLECTION
EMPLOYEE_RECORDS(2)                         MCS_COLLECTION
EMPLOYEE_RECORDS(1)                         MCS_COLLECTION
FINAL_REVIEW                                MCS_PARTITION
FIRST_NAME(3)                               FIELD
FIRST_NAME(2)                               FIELD
FIRST_NAME(2:TEST:2)                        FIELD
FIRST_NAME(2:TEST:1)                        FIELD
FIRST_NAME(1)                               FIELD
LAST_NAME(1)                                FIELD
LIMITED_REVIEW                              MCS_PARTITION
MID_INITIAL(1)                              FIELD
PERSONNEL                                   MCS_CONTEXT
```

Merge the branch into the main line of descent.

```
CDO> RESERVE FIELD /CLOSURE=TO_TOP FIRST_NAME(3)
CDO> MERGE FIELD FIRST_NAME(4) WITH FIRST_NAME(2:TEST:2)
CDO> REPLACE FIELD /CLOSURE=TO_TOP FIRST_NAME(4)
CDO> SHOW RESERVATIONS
There are no elements reserved
```

5.4.4 Showing the Effects of Changes

In the CDO environment, you can browse through the repository to monitor and analyze usage. The knowledge of how your repository is used can help you:

- evaluate when a directory structure is overloaded
- evaluate how far-reaching a change to an element will be

- decide when elements become obsolete
- decide when elements should be purged
- restructure your repository hierarchy

The following sections explain how to obtain information about the relationships between the definitions stored in the repository.

5.4.5 Tracking Changes to Definitions

When you consider changing a definition, you need to know the following:

- which other elements use the definition
- which elements automatically include changes
- which elements are flagged with a notice if an inconsistency occurs

5.4.6 Using the SHOW UNUSED Command

The SHOW UNUSED command displays whether a parent-child relationship exists between elements. For example:

```
CDO> SHOW UNUSED FIELD_B
DISK1:[SMITH.DICT]FIELD_B(1) is referenced by or references other element(s)
```

This example shows that FIELD_B has either a parent or child relationship with another element.

The following example shows that the new version of FIELD_A does not have a relationship with any other elements.

```
CDO> SHOW UNUSED FIELD_A(2)
DISK1:[SMITH.DICT]FIELD_A(2)     (TYPE : FIELD)
```

You can use the SHOW UNUSED command to determine whether it is safe to purge or delete repository elements.

5.4.7 Accessing Notices About Changes

When you change an element, or when you create a new version of an element, Oracle CDD/Repository sends a notice to elements that are owners of relationships with the element being modified. An Oracle CDD/Repository **notice** is an encoded report of the status of an element. A notice signals that an element that is a member of a relationship has been changed. A member of a relationship can send notices to the following:

- the owner of the relationship
- the owner's ancestors

- both the owner and the owner's ancestors

Any product that supports Oracle CDD/Repository can read notices that are attached to CDO definitions.

Whenever you change a definition, you potentially affect other definitions or sources that use that definition. You must decide if sources should use the original or the latest version of a definition. You may need to change your source program and recompile it. Or, you may need to integrate the definition in the database.

If you want to know the impact of performing a change in place with the CDO CHANGE command before you make the change, use the CDO SHOW WHAT_IF command. The SHOW WHAT_IF command lists the elements that will be affected if the element you specify is changed.

Each affected element would receive an "`element is possibly invalid...`" message, if you issue the CHANGE command for the specified element.

Note

The notices produced by CDO apply only to the protocols that are supplied with Oracle CDD/Repository. If you create protocols for your own repository definitions, you can establish a different scheme for signaling notices.

For example:

```
CDO> SHOW WHAT_IF/FULL PART_NO
Signaled owners of DISK1:[USER1.REP3]PART_NO(1)
|   DISK1:[USER1.REP3]MULTI(1)  (Type : CDD$DATABASE)
|   |   via CDD$DATABASE_SCHEMA            ( 2 unshown intermediate nodes)
```

When you maintain copies of definitions in a database, the database copies are not automatically updated after a change in the repository. Oracle CDD/Repository will flag database definitions in the repository with a message about the inconsistency.

In the following example, the field EMPLOYEE_ID is changed. CDO notifies you that the field EMPLOYEE_ID is used by the Oracle Rdb database DEPT1.

```
CDO> CHANGE FIELD EMPLOYEE_ID
cont> DATATYPE IS TEXT SIZE IS 15.
%CDO-I-DBMBR, database SYS$COMMON:[CDDPLUS]DEPT1(1)
may need to be INTEGRATED
```

To update this change in the database, use the SQL INTEGRATE statement. See Chapter 8 for more information on integrating changes.

Oracle CDD/Repository considers all changes and new versions of elements to be incompatible with prior versions of that element, with the exception of a change to the history list of an element. Oracle CDD/Repository flags all dependent uses with notices when a new version is created.

5.4.7.1 Reading Notices

To read notices attached to an element, use the CDO SHOW NOTICES command. In the following example, the SHOW NOTICES command displays notices attached to the record ADDRESS_REC.

```
CDO> SHOW NOTICES CDD$TOP.PERSONNEL.ADDRESS_REC

SYS$COMMON:[CDDPLUS]PERSONNEL.ADDRESS_REC(1) uses an entity
which has new versions, triggered by CDD$DATA_ELEMENT
SYS$COMMON:[CDDPLUS]PERSONNEL.ZIP_CODE(1)
```

Oracle CDD/Repository generates a notice to users of ADDRESS_REC that a new version of ZIP_CODE(1) exists. Because the element ZIP_CODE is used by ADDRESS_REC you must redefine ADDRESS_REC to use the new version of ZIP_CODE.

5.4.7.2 Clearing Notices

After all users have been warned of a change in the definition, the warning notice may become superfluous. You can delete notices with the CDO CLEAR NOTICES command. For example, the following command clears the notice attached to the record definition ADDRESS_REC:

```
CDO> CLEAR NOTICES ADDRESS_REC
CDO>
```

Do not clear notices from a definition until you are certain that all sources have been altered to accommodate the changes.

5.4.8 Modifying the Access Rights to Allow Changes

By default, only the owner of a definition has the privileges to change it. If you have CONTROL access rights, you can change the default protection on an element.

To find out if you have access rights to change or define elements, use the CDO SHOW PROTECTION or CDO SHOW PRIVILEGES command. The SHOW PROTECTION command displays the access control list (ACL) for the definition you specify.

In the following example, a user in group 20 has both CHANGE and DEFINE access rights to the field RATE:

```
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=[20,*],ACCESS=READ+WRITE+MODIFY+ERASE+
    SHOW+DEFINE+CHANGE+DELETE)
    (IDENTIFIER=[SECRETARIES,*],ACCESS=SHOW)
    (IDENTIFIER=[SALES,JONES],ACCESS=NONE)
```

The CDO SHOW PRIVILEGES command displays your current privileges for the definition. You see the following SHOW PRIVILEGES output only if you belong to group 20:

```
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 7 discusses access rights in more detail.

5.4.9 Creating New Versions of Definitions

When you store several versions of the same definition in the repository, 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 CDO RESERVE, CDO DEFINE, and CDO REPLACE commands rather than use the CDO CHANGE command.

Creating new versions, rather than changing the original definition, leaves an audit trail. You need DEFINE access rights to an existing definition to create new versions of it.

When you define a record and include a field without specifying the field version number, Oracle CDD/Repository includes the highest version available at the time. 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. See Section 5.4.10 for more information.

5.4.9.1 Changing Field Definitions in Place

When you change a field or record definition with the CDO CHANGE command, definitions that automatically include that field or record definition, such as related field and record definitions, include the change. Repository definitions that do not automatically include the changed definition are flagged with a notice. For example:

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

In this example, the field definition BADGE_NO is changed using the CDO CHANGE FIELD command. The record definition EMPLOYEE_REC automatically includes the change to BADGE_NO. Therefore, EMPLOYEE_ REC does not receive a notice. However, now the copy of BADGE_NO contained in the database DEPT1 does not match the BADGE_NO definition in the repository, so Oracle CDD/Repository flags the database with a notice.

To make the database and repository match, use the SQL INTEGRATE statement. See Chapter 8 for more information on INTEGRATE.

In Section 5.2.2.1, the field SIMPLE_LIST was given an initial value. The following command changes the definition of SIMPLE_LIST so that the definition no longer has an INITIAL VALUE property. The AUDIT clause then becomes a history list entry that describes the change.

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

You can change properties, 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. In this example, the CHANGE FIELD command changes the data type of the field definition in the repository, not the data type of the stored data.

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

Note

If you change the data type of a field and the field is used in an Oracle Rdb hashed index, you must delete the index to change the field, then re-create the index.

The following command changes the array created in the field SIMPLE_LIST from a 5-element array to a 10-element array:

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

Caution

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

The CDO CHANGE command lets you change properties such as missing values, special names for languages, query headers, and so on. For information on the field properties that you can change, see the CHANGE FIELD command description in the *Oracle CDD/Repository CDO Reference Manual*.

5.4.9.2 Changing Record Definitions in Place

You can also modify record definitions with the CHANGE command. For more information about the changes you can make, see the CHANGE RECORD command description in the *Oracle CDD/Repository CDO Reference Manual*.

The following command 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 property. The included field becomes the last field in the record definition. The following example adds the fields WAGE_STATUS and CLASS_CODE to the record definition EMPLOYEE_REC:

```
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>
```

The DEFINE keyword that is used in the CDO CHANGE RECORD command is not the same as the CDO DEFINE command. You use the DEFINE keyword to define a new field as part of the record, but the field definition must already exist. You cannot implicitly create a field definition using the DEFINE keyword in the CDO CHANGE RECORD command.

To make changes to a variant definition in a record definition, use the VARIANTS clause with the CHANGE RECORD command. You must indicate the position of the variant you are changing so that Oracle CDD/Repository can identify it. To indicate a variant's position, use the VARIANT . . . END VARIANT keywords for the variants that precede the variant you are changing.

The following example uses the keyword DEFINE to specify the new field RATE in the second variant in the VARIANTS clause of the record definition EMPLOYEE_REC. The first variant is not changing; therefore, only the VARIANT . . . END VARIANT keywords are referenced.

```
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>
```

5.4.9.3 Using New Versions of Field Definitions in Records

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

- use the CDO screen editor
- use the CDO EXTRACT command to display the record definition, then modify the definition using the RESERVE/CHANGE/REPLACE or DEFINE commands

When you create a new version of a definition, Oracle CDD/Repository flags all repository elements that use the previous version of the element.

For example, suppose the record EMPLOYEE_REC contains the field definition BADGE_NO:

```
CDO> DEFINE RECORD EMPLOYEE_REC.
cont> BADGE_NO.
cont> END EMPLOYEE_REC RECORD.
```

You can use the CDO DEFINE command to create a new version of the field BADGE_NO.

```
CDO> DIRECTORY BADGE_NO
 Directory SYS$COMMON:[CDDPLUS]PERSONNEL
BADGE_NO(1)                                    FIELD
CDO> DEFINE FIELD BADGE_NO
cont> DATATYPE IS TEXT
cont> SIZE IS 9.
CDO> DIRECTORY BADGE_NO
 Directory SYS$COMMON:[CDDPLUS]PERSONNEL
BADGE_NO(2)                                   FIELD
BADGE_NO(1)                                   FIELD
```

The record EMPLOYEE_REC and the Oracle Rdb database DEPT1 do not automatically use the new version of BADGE_NO. Oracle CDD/Repository flags EMPLOYEE_REC and DEPT1 with notices warning you that a new version of BADGE_NO exists. For example:

```
CDO> SHOW NOTICES EMPLOYEE_REC
SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_REC(1) uses an entity which has
new versions, triggered by
CDD$DATA_ELEMENT SYS$COMMON:[CDDPLUS]PERSONNEL.BADGE_NO(1)
CDO> SHOW NOTICES DEPT1
SYS$COMMON:[CDDPLUS]PERSONNEL.DEPT1(1) uses an entity which has
new versions, triggered by
CDD$DATA_ELEMENT SYS$COMMON:[CDDPLUS]PERSONNEL.BADGE_NO(1)
```

You have the option of using the new version of BADGE_NO.

For example, suppose that a supporting product, such as a database or program, also uses EMPLOYEE_REC. If the supporting product has a relationship to EMPLOYEE_REC, Oracle CDD/Repository attaches a notice to that product's repository definition to warn the user that a new version exists.

Section 5.2.6 contains more information about analyzing the impact of changes and tracking repository usage.

5.4.9.4 Extracting 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 properties in the editor, the CDO 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 write the output of the EXTRACT command to a file rather than to your terminal. To do this, use the SET OUTPUT command. With the output from the EXTRACT command in a file, you can use a text editor to change the command syntax, add new properties, 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.
!
$ LOGOUT
!
!Execute the modified command file.
!
CDO> @[WORKSPACE]FIRST_NAME
```

5.4.10 Changing Relationships

When two definitions are related, the relationship exists between the owner and the specified version of the member. When you do not specify a version of the member, Oracle CDD/Repository relates the owner 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 and DEFINE commands. When you use the CDO screen editor to change a definition, the effect is the same as when you use the DEFINE command. CDO creates a new version of the definition.

5.4.10.1 Changing a Member with the CHANGE Command

The original definition of FULL_NAME did not specify versions. For example:

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

FULL_NAME owns three relationships with one relationship between each of the field definitions FULL_NAME. Each of the included fields are members of a relationship with FULL_NAME.

Suppose that you want to change MIDDLE_INIT to increase its size to three characters to allow for the convention NMN for no middle name. If you change the original definition with the CHANGE command, 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.

You have now increased the size of MIDDLE_INIT by two additional characters. Because you changed MIDDLE_INIT in place instead of creating a new version, all definitions that use MIDDLE_INIT incorporate this change automatically. Oracle CDD/Repository does not attach a notice to FULL_NAME or any other field or record definitions that automatically include the change.

5.4.10.2 Changing a Member with the DEFINE Command

Consider again the previous example of the record definition FULL_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.

Oracle CDD/Repository does not automatically include new versions of definitions in the definitions that use them. The newly created MIDDLE_INIT(2) is not related to FULL_NAME(1). You must change or redefine the owner of a relationship 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 then 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, Oracle CDD/Repository attaches a notice that a new version of MIDDLE_INIT exists to each user of the previous version.

Applications often require nested relationships. For example, the record definition FULL_NAME is itself a member of a relationship with EMPLOYEE_NAME.

```
CDO> SHOW RECORD EMPLOYEE_NAME
Definition of record EMPLOYEE_NAME
|   Contains record          FULL_NAME
```

You can list all dependent uses of a definition with the SHOW USES /FULL command, as follows:

```
CDO> SHOW USES /FULL FULL_NAME
Owners of SYS$COMMON:[CDDPLUS]PERSONNEL.FULL_NAME(1)
|   SYS$COMMON:[CDDPLUS]PERSONNEL.EMPLOYEE_NAME(1)  (Type : RECORD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
```

5.4.11 Recompiling Application Programs

Notices are passive. It is up to you to change the source code, redefine the record, integrate the repository and database, or take other appropriate action.

The languages include definitions from Oracle CDD/Repository at compile time. Any changes to definitions are not reflected in the programs unless they are recompiled. A change to a definition can require a related change in the program source code. If you do not recompile a source program, the executable code will continue to reflect the original definitions.

If you access CDO definitions from a product that does not support Oracle CDD/Repository, CDO does not issue warning messages. It is up to the user who makes the change to the definition to isolate all programs that access the changed definition, and to notify the appropriate users to make any required changes to their sources. When all users have accommodated new versions, you can remove definitions with the PURGE or DELETE commands.

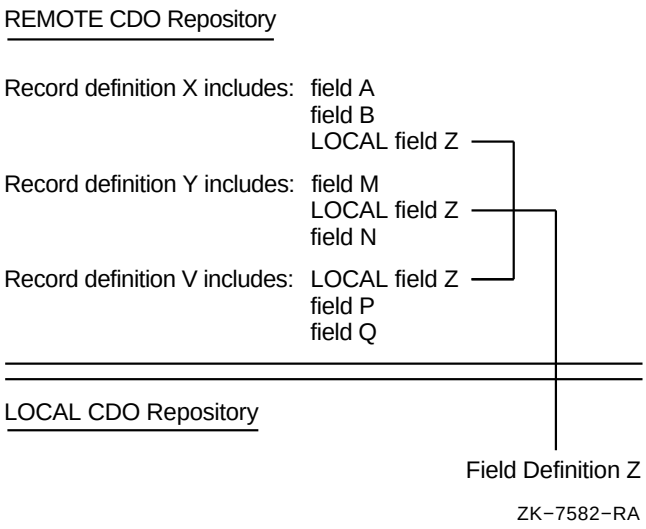
Note

Version 1 of an element is never purged. For example, if you have FIELD_A(1) and FIELD_A(2) in your directory, and you issue the PURGE command from CDO, you get a "normal successful completion" message. If you issue the PURGE command from DATATRIEVE (DTR), DTR issues a "there is no prior version of this entity" message. In both cases, FIELD_A(1) is not purged.

5.4.12 Changing Elements in a Network

Figure 5–1 shows how Oracle CDD/Repository uses local copies for faster performance when definitions are related through a network.

Figure 5–1 Creating Local Copies of Remote Dictionary Definitions



When you change a local definition with the CDO CHANGE command, Oracle CDD/Repository automatically updates all remote copies of the definition. The node for the remote repository that contains copies of the definition must be accessible at the time of the change; otherwise, Oracle CDD/Repository does not allow the change and issues an error message.

Using the example in Figure 5–1, suppose you change version 1 of the field definition Z on the local node. In this example, the network link is viable, and Oracle CDD/Repository updates the copies of field Z in the record definitions X, Y, and V on the remote node.

Because Oracle CDD/Repository automatically includes the changed field Z in the record definitions, these elements are not flagged with a warning message about the change.

However, when you create a new version of a local definition with the CDO DEFINE command, the copy on the remote node remains unchanged. Instead, Oracle CDD/Repository flags users of the definition with a notice that a new version exists.

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.

5.5 Copying Definitions

You can copy definitions when you need duplicates for production or testing. The CDO COPY command lets you copy a specified definition and the relationships it owns. You must supply the name of the definition to be copied and specify the name of the directory into which you are copying the definition (the target directory).

In the following example, the record definition EMPLOYEE_REC is copied from the SALARIED directory to the CONTRACT directory:

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

You can use wildcard characters for both the source directory and the target directory. In the following example, the COPY command copies all the definitions from the PERSONNEL directory to the SALARIED directory using the asterisk (*) wildcard character:

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

If the target directory to be copied does not exist, Oracle CDD/Repository creates it when you copy the definitions. In the following example, the COPY command copies all definitions in the SALARIED directory and all the directories under it to the SALARIED.MAIN directory using the ellipsis (. . .) as the wildcard character:

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

See the *Oracle CDD/Repository CDO Reference Manual* for rules on using wildcard characters in the CDO COPY command.

When you copy a definition, Oracle CDD/Repository does not add an entry to the history list auditing the copy operation. However, Oracle CDD/Repository does copy the existing history list to the new definition.

You can use the CDO COPY command to copy a definition from one repository to another. The COPY command is valid across a network. When more than one version of the specified definition exists, Oracle CDD/Repository 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, Oracle CDD/Repository creates a new version of the definition.

For example, if you make a subsequent change to the original record definition EMPLOYEE_REC in the SALARIED directory, the copied definition in the directory SALARIED.MAIN will not reflect this change.

Restrictions

- Do not copy controlled elements. You should use the CDO EXTRACT command and redefine the element, or use the CDO DEFINE command, to copy controlled elements.

  In general, it is better not to mix controlled and uncontrolled elements in the same directory. The following example shows how versions of an element are affected by copying an uncontrolled element to another directory.

  In a directory named ACCOUNTS, a context has been defined and set. A collection named CREDIT has been defined and reserved. Field F(1) is defined within the context.

  In another directory named HISTORY, field F is defined. It is an uncontrolled element. If you copy field F from the HISTORY directory to the ACCOUNTS directory, field F(2) is created, and it is uncontrolled. If you subsequently define field F in the ACCOUNTS directory, field F(3) will be created and it will also be uncontrolled.

  Use the CDO CONSTRAIN command to make all the versions of the field F controlled and associated with the collection.

  ```
  CDO> CONSTRAIN FIELD F
  ```

  Use the CDO UPDATE COMPOSITE command to make the collection refer to the latest version of field F.

  ```
  CDO> UPDATE COMPOSITE CREDIT
  ```

- Oracle CDD/Repository does not support quoted passwords in Oracle CDD/Repository pathnames.

  To access a remote repository (a repository running on a system other than your host), you must obtain a proxy account. For example, if you want to copy an element to or from a remote repository and you specify your password in the name string used in the copy command, you will get an error message similar to the following:

```
CDO> COPY NODE1"account password"::[CDDTEST.SERVER.ADIRCDD]F1 F1

COPY NODE1"ACCOUNT PASSWORD"::[CDDTEST.SERVER.ADIRCDD]F1 F1
                                                        ^
%CDO-E-KWSYNTAX, syntax error in command line at or near F1

CDO> COPY NODE1"account password"::[CDDTEST.SERVER.ADIRCDD]
%CDO-E-ERRCOPY, error copying an object
-CDD-F-IDPC, invalid directory path component in name 'NODE1"CDDTEST'
-CDD-F-ILLCH, illegal character in directory name

CDO> COPY NODE1::[CDDTEST.SERVER.ADIRCDD]F1
CDO> COPY NODE1::[CDDTEST.SERVER.ADIRCDD]F1
%CDO-E-ERRCOPY, error copying an object
-CDD-F-IDPC, invalid directory path component in name 'NODE1"CDDTEST'
-CDD-F-ILLCH, illegal character in directory name
```

5.6 Copying Relationships

The CDO COPY command preserves all relationships. When you copy both the owner and the member of a relationship, 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 you copy both FIRST_NAME and GIVEN_NAME in the same CDO command, Oracle CDD/Repository creates a new relationship between the new copies of both.

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

When you copy a member but not the owner, Oracle CDD/Repository creates a new relationship because no definitions own the copied member. For example, if you copy FIRST_NAME but not GIVEN_NAME, no definition uses the copy of FIRST_NAME. The old relationship remains intact.

5.7 Deleting Elements

If an element is not used by any application, you can delete the element with the CDO DELETE command. By default, only the owner of an element has the privileges to delete it; you need DELETE access to repository elements to purge or delete them. However, if you have CONTROL access to the element you can change the default protection. See Section 7.7 for more information on access rights to repository elements.

If you specify a wildcard character (*) for the version number, all versions of the element are deleted from the repository. In the following example, all versions of the field GIVEN_NAME are deleted:

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

You can supply several element names with a single DELETE command if the elements 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>
```

To delete or purge elements in a remote CDO repository, the network link between the local and remote node must be viable. See Section 5.4.12 for more information on remote access.

Using the /DESCENDANTS Qualifier

You can delete definitions that are members of a relationship with the specified element. In the following example, the /DESCENDANTS qualifier is used with the DELETE RECORD command to delete the record definition FAMILY and all the definitions that the record uses.

If a member of a relationship to FAMILY is also a member of a relationship with another definition, Oracle CDD/Repository does not delete the descendant.

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

Deleting Elements in a START_TRANSACTION and COMMIT Session

If you delete a record inside a START_TRANSACTION and COMMIT session, local fields within that record are marked for deletion at the end of the transaction, provided that they remain unused at the end of the transaction. In CDO, there is no way to reuse those local fields. However, it is possible to reuse them in the Oracle CDD/Repository application programming interfaces and layered products or in applications that use Oracle CDD/Repository. Therefore, the local fields cannot be automatically deleted at the point in the transaction where the record is deleted.

If you define records that include structures and then attempt to delete them, CDO does not delete the relationships between records. A CDD-E-INUSE error occurs, as shown in the following example:

```
   .
   .
   .
CDO> DEFINE FIELD CURRENCY_EXCHANGE_RATE
cont> DESCRIPTION IS /*Not available.*/
cont> DATATYPE IS SIGNED LONGWORD SCALE -6
cont> VALID IF (RATE_OF_EXCHANGE_STD > 0).
CDO> DEFINE RECORD TEST_REC.
cont> CURRENCY_EXCH_RATE BASED ON CURRENCY_EXCHANGE_RATE.
cont> END.
CDO> DIR
   .
   .
   .
CDD$PROTOCOLS                    DIRECTORY
CURRENCY_EXCHANGE_RATE(1)        FIELD
   .
   .
   .
TEST_REC(1)                      RECORD
CDO> START_TRANSACTION
CDO> DELETE RECORD TEST_REC.
CDO> DELETE FIELD CURRENCY_EXCHANGE_RATE(1).
%CDD-E-INUSE, element is the member of a relationship; it cannot be deleted
CDO> COMMIT
%CDD-F-MUSTABORT, previous errors require that session abort
CDO> ROLLBACK
CDO> SHOW USES/FULL CURRENCY_EXCHANGE_RATE(1)
Owners of DISK:[NAME.CDD]currency_exchange_rate(1)
|   CURRENCY_EXCH_RATE           (Type : FIELD)
|   |   via CDD$DATA_ELEMENT_BASED_ON
|   |   DISK:[NAME.CDD]TEST_REC(1)   (Type : RECORD)
|   |   |   via CDD$DATA_AGGREGATE_CONTAINS
CDO> EXIT
```

You must either delete the record and field in separate transactions, or to accomplish this all in one transaction, use the CDO ENTER command to enter the local field, making it accessible to CDO. Delete the local field after deleting the record, then delete the global field. For example:

```
   .
   .
   .
CDO> DEFINE FIELD CURRENCY_EXCHANGE_RATE
cont> DESCRIPTION IS /*Not available.*/
cont> DATATYPE IS SIGNED LONGWORD SCALE -6
cont> VALID IF (RATE_OF_EXCHANGE_STD > 0).
CDO> DEFINE RECORD TEST_REC.
cont> CURRENCY_EXCH_RATE BASED ON CURRENCY_EXCHANGE_RATE.
cont> END.
CDO> DIR
   .
   .
   .
CDD$PROTOCOLS                    DIRECTORY
CURRENCY_EXCHANGE_RATE(1)        FIELD
   .
   .
   .
TEST_REC(1)                      RECORD
CDO> START_TRANSACTION
CDO> ENTER FIELD CURRENCY_EXCH_RATE FROM RECORD TEST_REC
CDO> DELETE FIELD CURRENCY_EXCH_RATE.
CDO> DELETE RECORD TEST_REC.
CDO> DELETE FIELD CURRENCY_EXCHANGE_RATE(1).
CDO> COMMIT
CDO> ROLLBACK
CDO> EXIT
```

5.8 Purging Definitions

When you have created more than one version of an element, you can purge earlier versions with the CDO PURGE command. The PURGE command deletes all but the first and last version of an element. You cannot delete the first version, and an intermediate version is not purged if a branch line descends from it. You cannot purge an element that is a member of a relationship, such as a field that is used in a database or record.

The following command deletes all but the first version and the latest version of the field SALARY_MAX:

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

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.

You can use a wildcard character (*) with the PURGE command. In the following example, a wildcard is used and all fields that are not related to others in the current default directory are purged:

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

6
Implementing a Distributed Repository

This chapter contains examples of distributing repositories on multiple devices. It also discusses the concepts of remote access, messaging, and journal files.

6.1 Introduction to a Distributed Repository

With Oracle CDD/Repository you can access metadata in repositories that are distributed on multiple devices, such as:

- on a single node
- on multiple nodes in a VMScluster
- on local area networks
- on wide area networks

A distributed repository is not by definition a remote repository. You can have separate repositories on a single node. However, a distributed repository consists of multiple physical repositories on the same node or cluster that are used as one logical repository.

A logical repository 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 repository directories and group definitions within these directories. You can use both explicit naming and search lists to create the logical repository structure.

A remote repository is a repository that is on a node or cluster other than that from which you invoke Oracle CDD/Repository. To access a remote repository, you must obtain a proxy account on that node. You cannot define a repository remotely; you must log in to the remote system and define the repository locally. Remote access is described in more detail in Section 6.3.

The design of a distributed repository must ensure that the integrity of the metadata is maintained across all repositories and that the metadata is available at all times.

Before you create your repository directory structure, consider what type of structure can lend itself to the anticipated repository usage. Section 4.4 describes repository design alternatives and considerations.

Since one logical repository can include many physical repositories, you can list several repositories in one anchor specification, including remote repositories. When you use a search list to access a set of logical repositories, the repositories are accessed in the order in which you name them in your definition. Oracle CDD/Repository stores all new definitions in the first repository you name.

When retrieving metadata, Oracle CDD/Repository searches the repositories in the order in which you named them and returns the first instance of the metadata that it finds. Since new data is always entered in the first repository that is named in an anchor specification, Oracle CDD/Repository always returns the most recent version of a data definition.

Within the CDO interface, you can switch from one physical repository to another and access definitions in any of the repositories to which you have access.

6.2 Distributing Repository Elements

You can distribute repository elements in two ways:

- using the CDO ENTER command
- creating relationships across repositories

The following examples show how to distribute both controlled and uncontrolled repository elements. For an explanation of the difference between uncontrolled and controlled elements, see Chapter 5.

For complete information on the CDO ENTER command and any of the CDO commands used in these examples, see the *Oracle CDD/Repository CDO Reference Manual*.

The distributed repository in these examples consists of two repositories that are located on different disks (DISK$1 and DISK$2) that are on the same system. The examples show that a field can be defined in one repository and accessed from another repository because the repositories are distributed.

In Example 6–1, the anchor repositories and a directory in each repository are created using CDO.

Example 6–1 Distributing Uncontrolled Elements Using the CDO ENTER Command

```
$ REPOSITORY OPERATOR
CDO> DEFINE REPOSITORY DISK$1:[CORPORATE].
CDO> DEFINE REPOSITORY DISK$2:[BRANCH].
CDO> DEFINE DIRECTORY DISK$1:[CORPORATE]FIELDS.
CDO> DEFINE DIRECTORY DISK$2:[BRANCH]ACCOUNTS.
```

Set default to the corporate repository DISK$1:[CORPORATE]FIELDS and define a field called CORP_A. The CDO SHOW FIELD command displays the field.

```
CDO> SET DEFAULT DISK$1:[CORPORATE]FIELDS
CDO> DEFINE FIELD CORP_A DATATYPE TEXT SIZE 5 CHARACTERS.
CDO> SHOW FIELD/FULL CORP_A
Definition of field CORP_A
|   Datatype                text size is 5 characters
```

Set default to the branch repository DISK$2:[BRANCH]ACCOUNTS and enter a directory name for the field located in the corporate repository DISK$1:[CORPORATE]FIELDS. The CDO SHOW FIELD command displays the field name as it is defined in the corporate repository.

```
CDO> SET DEFAULT DISK$2:[BRANCH]ACCOUNTS
CDO> ENTER FIELD BRANCH_A FOR DISK$1:[CORPORATE]FIELDS.CORP_A
CDO> SHOW FIELD/FULL BRANCH_A
Definition of field CORP_A
|   Datatype                text size is 5 characters
```

To demonstrate that you can change an element from the other repository, change the size attribute of the field in the branch repository. The CDO SHOW FIELD command displays the actual field name as defined in the corporate repository and shows that its size attribute has changed.

```
CDO> CHANGE FIELD BRANCH_A DATATYPE IS TEXT SIZE IS 10 CHARACTERS.
CDO> SHOW FIELD/FULL BRANCH_A
Definition of field CORP_A
|   Datatype                text size is 10 characters
```

Set default to the corporate repository and display the field. Note that the change was actually made in the corporate repository.

```
CDO> SET DEFAULT DISK$1:[CORPORATE]FIELDS
CDO> SHOW FIELD/FULL CORP_A
Definition of field CORP_A
|   Datatype                text size is 10 characters
CDO> EXIT
```

Example 6–2 uses the repositories and elements created in Example 6–1. From the branch repository DISK$2:[BRANCH]ACCOUNTS, a record is defined using the CORP_A field, which is located in the corporate repository. This creates a distributed relationship between the record and the field. The CDO SHOW RECORD command displays the field name as defined in the corporate repository.

Example 6–2 Distributing Uncontrolled Elements by Creating a Relationship

```
$ REPOSITORY OPERATOR
CDO> SET DEFAULT DISK$2:[BRANCH]ACCOUNTS
CDO> DEFINE RECORD BRANCH_REC.
CDO>    DISK$1:[CORPORATE]FIELDS.CORP_A.
CDO>    END.
CDO> SHOW RECORD BRANCH_REC
Definition of record BRANCH_REC
|   Contains field           CORP_A
```

The CDO SHOW USED_BY command displays detailed information about the field in the corporate repository. The field is now used by the new record in the branch repository.

```
CDO> SHOW USED_BY BRANCH_REC
Members of DISK$2:[REPO2]ACCOUNTS.BRANCH_REC(1)
|   DISK$1:[REPO1]FIELDS.CORP_A(1)  (Type : FIELD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
```

Set default to the corporate repository. The CDO SHOW USES command displays detailed information about the record in the branch repository that uses the field in the corporate repository.

```
CDO> SET DEFAULT DISK$1:[CORPORATE]FIELDS
CDO> SHOW USES CORP_A
Owners of DISK$1:[REPO1]FIELDS.CORP_A(1)
|   DISK$2:[REPO2]ACCOUNTS.BRANCH_REC(1)  (Type : RECORD)
|   |   via CDD$DATA_AGGREGATE_CONTAINS
```

6.3 Accessing Remote Repository Elements

This section describes how to configure remote access operations.

6.3.1 Setup for Remote Access

In this section, the term **client** refers to the local node from which Oracle CDD/Repository is invoked. The term **server** refers to the remote node on which a repository is located.

Perform the following steps:

1. Make sure that all images for Oracle CDD/Repository Version 5.3-08 (ECO) or higher have been installed on all systems that will participate in remote operations.

   To verify the installed version of Oracle CDD/Repository, check the image identification information by using the DCL ANALYZE/IMAGE command on the images SYS$SYSTEM:CDO.EXE, SYS$MESSAGE:CDDEXC.EXE, SYS$SHARE:CDDSHR.EXE, and SYS$SYSTEM:CDD$REMOTE5.EXE. For example:

   ```
   $ ANALYZE/IMAGE SYS$SHARE:CDDSHR.EXE
   ```

   The image identifier CDD V5.3-08, CDD V5.3-09, CDD V6.1, CDD V6.1-01, or CDD V7.0 should be displayed.

2. Make sure that NCP (the Network Control Program utility) points to the correct CDD$REMOTE5.COM file by looking at the command file that NCP returns. To do this, enter the following commands if you are running DECnet Phase IV:

   ```
   $ RUN SYS$SYSTEM:NCP
   NCP> SHOW OBJECT CDD$REMOTE5
   ```

   Look at the command file that NCP returns. Make sure that CDD$REMOTE5.COM is calling CDD$REMOTE5.EXE with the image identifier CDD V5.3-08 or higher.

   If NCP does not recognize the CDD$REMOTE5 object, then the Oracle CDD/Repository installation did not complete correctly. Reinstall Oracle CDD/Repository.

3. If you are on a node that is running DECnet Phase V, enter the following commands:

   ```
   $ RUN SYS$SYSTEM:NCL
   NCL> SHOW SESSION CONTROL APPLICATION CDD$REMOTE5
   ```

4. If DECnet Phase V does not recognize the CDD$REMOTE5 server object, enter the following commands:

```
$ RUN SYS$SYSTEM:NCL
NCL> CREATE NODE 0 SESSION CONTROL APPLICATION CDD$REMOTE5
NCL> SET NODE 0 SESSION CONTROL APPLICATION CDD$REMOTE5 -
_NCL> IMAGE NAME SYS$SYSTEM:CDD$REMOTE5.EXE
NCL> SET NODE 0 SESSION CONTROL APPLICATION CDD$REMOTE5 -
_NCL> ADDRESS {NAME=CDD$REMOTE5}
NCL> EXIT
```

5. If you have a multiversion Oracle Rdb environment, check that you have set the proper version number on the server for the repository you want to access on the server.

   You can do this at the system level by modifying the CDD$REMOTE5.COM file, or by modifying the LOGIN.COM file in the proxy account. Enter the following command where *nn* represents the version number:

   ```
   $ @SYS$SHARE:DECRDB$SETVER nn
   ```

   For example:

   ```
   $ @SYS$SHARE:DECRDB$SETVER 6.0
   ```

   Or, if you have Version 7.0 of Oracle Rdb installed, enter:

   ```
   $ @SYS$SHARE:RDB$SETVER 7.0
   ```

   To check the current version of Oracle Rdb, enter the following command:

   ```
   $ @SYS$SHARE:DECRDB$SHOVER
   ```

   Or, if you have Version 7.0 of Oracle Rdb installed, enter:

   ```
   $ @SYS$SHARE:RDB$SHOVER
   ```

6. Make sure that you have set up the proxy account on the server node, as follows:

   ```
   $ RUN SYS$SYSTEM:AUTHORIZE
   UAF> ADD/PROXY client::username server-username/DEFAULT
   UAF> SHOW/PROXY client::username
   ```

   The /DEFAULT qualifier is required.

7. A repository cannot be created remotely. It must be created locally.

   You must set host to the server node, and create the repository locally on the server node. You can then populate the repository from the client node.

6.3.2 Remote Access Operations

The following steps outline the communications between the client and server repositories:

1. The client node determines that remote access is necessary. This is typically accomplished by specifying a node name (other than the client) on a CDO command. For example, the following command would cause a remote operation:

   ```
   CDO> DIRECTORY SERVER::pathname
   ```

   where *pathname* is *node::device:[anchor-dir]*. If you are using a logical name for the *pathname*, a terminating colon (:) is required after the logical name for remote access, as follows:

   ```
   CDO> SET DEFAULT SERVER::logical_name:
   ```

2. The remote session is attempted through DECnet.

3. On the server node, a DECnet object is defined as CDD$REMOTE5. After successfully connecting to the CDD$REMOTE5 object, the SYS$SYSTEM:CDD$REMOTE5.COM command procedure is executed.

   The CDD$REMOTE5.COM command procedure is invoked only after the SYLOGIN.COM and the user's LOGIN.COM command procedures are executed. The LOGIN.COM file that is executed depends on the network proxy that may be established.

   The CDD$REMOTE5.COM procedure contains only one line, as follows:

   ```
   $ RUN SYS$SYSTEM:CDD$REMOTE5.EXE
   ```

   In an Oracle Rdb multiversion environment, the default version of Oracle Rdb (if there is one) will be used. However, the repository that is being accessed may not have been created with the same version of Oracle Rdb. Therefore, you must set the Oracle Rdb version that will be active in this network session.

   You can customize the CDD$REMOTE5.COM procedure; however, it is not recommended because the CDD$REMOTE5 procedure is replaced after each subsequent installation of Oracle CDD/Repository. Rather, you should customize the SYLOGIN.COM command procedure or the user's LOGIN.COM command procedure to specify which Oracle Rdb version is necessary for the remote repository.

For instance, if the default Oracle Rdb version for the system is Version 5.1 and the repository was created using Oracle Rdb Version 6.1, then add the following line to either one of the login command procedures:

```
$ @SYS$LIBRARY:DECRDB$SETVER 6.1
```

4. Finally, SYS$SYSTEM:CDD$REMOTE5.EXE is executed, and it will invoke Oracle CDD/Repository.
5. The client can explicitly end the session by exiting CDO.
6. If the session ends as a result of a broken communication link, the transaction is rolled back.

Restriction

During a remote operation the compatibility repository consists of only CDO objects; DMU objects cannot be accessed.

6.3.3 Setup Errors

This section lists errors that may occur during setup:

-CDD-E-NOMODIFY Error

```
%CDO-E-ERRDEFINE, error defining an object
-CDD-E-NOMODIFY, no privilege to modify *name is unknown*
```

When accessing remote repositories, check that the remote OpenVMS directory is not Group and World protected against WRITE access.

-CDD-F-NOATTACH

```
%CDO-E-ERRDIRE, error displaying a directory
-CDD-F-NOATTACH, unable to attach to dictionary database
```

One common reason for this error is that the database on the server node does not match the Oracle Rdb version. You should change either the CDD$REMOTE5.COM file or the LOGIN.COM file to set the version of Oracle Rdb. Or, set the version at the system level.

This error can also indicate a privilege problem. Check that the remote OpenVMS directory is not Group and World protected against WRITE access.

-RDB-E-REQ_SYNC

```
%CDO-E-ERRCHANGE, error changing object
-CDD-E-ERRMODIFY, error modifying entities
-RDB-E-REQ_SYNC, host program out of synchronization with
  the specified request
```

This error occurs if there is an Oracle Rdb image mismatch problem. Make sure that the version of Oracle Rdb that was used to create the repository on the server node is the current version of Oracle Rdb for the session. To set the version of Oracle Rdb for the session, see Section 6.3.1.

Check the version identifiers of your installed Oracle Rdb version against the version identifiers specified in the Oracle Rdb documentation.

-SYSTEM-F-LINKEXIT

```
%CDO-E-ERRDIRE, error displaying a directory
-CDD-E-PRPCCONERR, error connecting to server on node ABCDEF
-SYSTEM-F-LINKEXIT, network partner exited
```

This error occurs for many reasons. Check the NETSERVER.LOG file on the server node for additional information. Also check the following:

- Make sure all nodes are known to each other.

  ```
  $ RUN SYS$SYSTEM:NCP
  NCP> SHOW NODE server::
  ```

- Make sure proxies are defined correctly on the server for the client user, as follows:

  ```
  $ RUN SYS$SYSTEM:AUTHORIZE
  UAF> SHOW/PROXY client::username
  ```

- Make sure that NCP is pointing to the CDD$REMOTE5.COM that is calling the correct CDD$REMOTE5.EXE image. See Section 6.3.1 for instructions.

-CDD-PRPCXERR

```
%CDO-E-ERRDIRE, error during directory
-CDD-E-HADTOABORT, could not commit transaction, had to roll back
-CDD-PRPCXERR, error writing to node !AF
```

This error occurs when the link to the remote node is lost during the operation.

%CDD-F-BADLOC

```
%CDD-F-BADLOC, dictionary NODE1::$DEVICE1:[CDDTEST.AN] has self-reference
of NODE1::$DEVICE1:[CDDTEST.AN]
```

This error occurs when a remote verify operation is attempted. You should set host to the server node and perform a verify operation on the server repository locally. Performing a remote verify operation is currently unsupported.

-SYSTEM-F-INVLOGIN

```
 %CDO-E-ERRSTARTSESS, error starting a session
 -CDD-E-PRPCCONERR, error connecting to server on node
 -SYSTEM-F-INVLOGIN, login information invalid at remote node
```

Your proxy account is not set up correctly. See Section 6.3.1 for instructions.

6.3.4 Errors You May See During Remote Operations

The following are errors you may see during remote operations. A recommended action follows each error.

-CDD-F-NOATTACH

```
  CDO> SET DEFAULT SERVER::CDD$COMPATIBILITY
  CDO> SHOW DEFAULT
SERVER::CDD$COMPATIBILITY
  CDO> DIRECTORY
 %CDD-F-NOATTACH, unable to attach to dictionary database
 %CDO-E-ERRDIRE, error displaying a directory
 -CDD-F-NOATTACH, unable to attach to dictionary database
```

Perform the following steps to correct the problem:

1. Check the network object, as follows:

   ```
   $ RUN SYS$SYSTEM:NCP
   NCP> SHOW OBJECT CDD$REMOTE5
   ```

2. If the object does not exist, add it:

   ```
   $ @SYS$COMMON:[SYSMGR]CDD$REMOTE5_NCP.COM
   ```

3. Check file protections on the SYS$SYSTEM:CDD$REMOTE5.EXE and SYS$SYSTEM:CDD$REMOTE5.COM files. If necessary, change the file protections so they are less restrictive.

4. Start Oracle CDD/Repository:

   ```
   $ @SYS$STARTUP:CDDSTRTUP.COM
   ```

-RDB-E-NO_RECORD

```
SQL> CREATE DATABASE FILENAME DEPT2 PATHNAME
SERVER::SYS$COMMON:[CDDPLUS]CDD_PLUS$EXAMPLES.DEPT1;
%CDD-E-NO_CRE_DB, error creating database definition in dictionary
-RDB-E-NO_RECORD, access by dbkey failed because dbkey is no longer
 associated with a record
```

Make sure that all participating nodes have Oracle CDD/Repository Version 5.3-08 (ECO) or higher installed.

-RDMS-F-ROOTMAJVER

```
SQL> CREATE DATABASE FILENAME DEPT2 PATHNAME
DISK:[CDDPLUS]WORKSHOPS.DEPT1;
%CDD-F-NOATTACH, unable to attach to dictionary database
-RDB-F-WRONG_ODS, the on-disk structure of database filename is not
 supported by version of facility being used
-RDMS-F-ROOTMAJVER, database format 40.0 is not compatible with
 software version 41.0
```

Make sure that the version of Oracle Rdb that was used to create the server repository is set as the current version of Oracle Rdb for the session. To do this, see Section 6.3.1.

-RDO-F-CDDERR

```
RDO> DEFINE DATABASE DEPT1 IN 'DISK:[A.DIRECTORY]CDD$DATABASE.RDB'.
%RDO-F-PATHREQCDD, a pathname is specified but no CDD is available
-RDO-F-CDDERR, error occurred during CDD signin, CDD will not be
 used
-LIB-E-ACTIMAGE, error activating image CDDSHR
RDO> SHOW DICTIONARY
%RDO-F-CDDDEFERR, unable to update or read the CDD default path
-RDO-F-CDDERR, error occurred during CDD signin, CDD will not be
 used
-LIB-E-READERR, error reading !AS
RDO> SHOW DICTIONARY
The current CDD dictionary is  *None*
```

Reinstall Oracle CDD/Repository.

6.3.5 Limiting Remote Access

By setting up privileges and quotas, you can limit remote access to repository elements in the following ways:

- If you specify ACCESS=NONE for the identifier SYS$REMOTE, all users on other nodes are denied access to the repository definition over DECnet whether or not they have a proxy account.
- If you specify ACCESS=NONE for the identifier DECnet, Oracle CDD/Repository denies access to users on other nodes who do not have proxy accounts. For users with proxy accounts, Oracle CDD/Repository grants access based on the UIC and identifiers of the proxy account, as described in the next section.
- If you specify an ACE for the identifier DECNET, Oracle CDD/Repository grants the access permissions of that ACE to users on other nodes who do not have proxy accounts.

See the OpenVMS documentation for information about setting up proxy accounts.

6.3.6 Using Proxy Access to Remote Repository Elements

When you access elements in a remote repository, Oracle CDD/Repository uses DECnet to connect to a server process on the target node. The server accesses the repository locally and performs the requested actions on your behalf. Thus, your access rights in the remote repository are the access rights granted to the user account on the target node under which the server runs. By using proxy accounts, you can grant remote repository access without granting SHOW access to all users.

Note

The default DECnet account is not equipped with sufficient process quotas to run Oracle CDD/Repository.

If your user account on your local system does not have a proxy account on the target node, the server runs under the target node's default nonprivileged DECnet account, usually user DECNET with the UIC DECNET. If your user account on the local system has a proxy account on the target node, the server runs under that proxy account and has the UIC and any other rights identifiers and privileges of that account.

For example, user SMITH on node A accesses a repository on node B. In the absence of a proxy account, the server runs on node B under the nonprivileged DECnet account. The access permissions granted to identifier DECNET govern SMITH's access to the remote repository. If user account JONES on node B has been set up as a proxy account for user SMITH on node A, when SMITH on node A accesses repositories on node B, Oracle CDD/Repository grants access based on the UIC of account JONES and on any other rights identifiers that JONES may hold.

6.4 Using Oracle CDD/Repository Notices for Repository Integrity

Whenever you create a new version of an element, CDO sends a notice to the elements that depend on the changed element (such as a database or a compiled module). All repositories to which the changed element has a relationship must be accessible for CDO to successfully send the notice.

Notices tell you when your definitions are no longer synchronized because an element has changed.

6.5 Using the Journal File

When you define an element in your repository, Oracle CDD/Repository writes the metadata to an Oracle Rdb database and a journal file that are on the local node.

The journal file keeps track of operations in the primary database. Its file extension is CDD$JNL_*date_and_time*.

Oracle CDD/Repository writes the metadata to all linked repositories within a single transaction. Oracle CDD/Repository then checks that all repositories with relationships to the changed element are prepared to commit the transaction. If one repository signals a failure, the whole transaction fails and Oracle CDD/Repository sends a failure flag to each involved repository.

Oracle CDD/Repository does not delete the journal file. It is used if you decide to roll back the operation.

If all involved repositories signal success, Oracle CDD/Repository sends a commit command to each involved repository. If the operation successfully completes, Oracle CDD/Repository deletes the journal file.

7

Managing Repository Protection

Oracle CDD/Repository provides the following five levels of protection for your repository:

- OpenVMS protection on devices, directories, and files
- Oracle CDD/Repository protection on the repository itself
- Oracle CDD/Repository protection on directories in the repository
- Oracle CDD/Repository protection on elements in the repository
- Oracle CDD/Repository protection on types in the repository

Each user on your system is identified by an access control entry (ACE) in an access control list (ACL). An ACE consists of the following two parts:

- one or more identifiers that specify a user or set of users and the kind of process used: user identification code (UIC), general, and system-defined
- a set of access rights that determines what operations each user can perform: READ, WRITE, EXECUTE, and DELETE

To access a repository, its directories, and elements, you must first be granted access through the ACL. The ACL specifies the access that a particular user has to a particular repository element. Oracle CDD/Repository saves the ACLs that are associated with an element for a particular user when that element is read from memory from the repository. While you are in a CDO session, you are authorized those exact privileges for the element. To acquire changes to authorized access privileges, you must exit and reenter the CDO session.

7.1 Understanding Oracle CDD/Repository Protection Schema

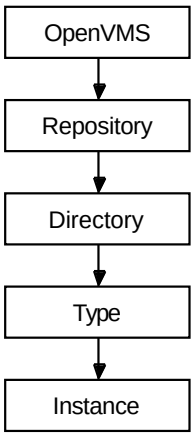
Oracle CDD/Repository checks for protection in the following sequence:

1. the OpenVMS UIC-based protection to repository files
2. the Oracle CDD/Repository access rights to the repository itself to perform the requested operation in that repository

3. the directory that contains the elements for access rights to the requested operation
4. the ACL on the type of which the element is an instance for the right to perform the operation
5. the ACL on the element for the right to perform the operation

Figure 7–1 shows the sequence that Oracle CDD/Repository follows in checking users' access rights to their files and directories.

Figure 7–1 How Oracle CDD/Repository Checks Access Rights



ZK–3486A–RA

Oracle CDD/Repository determines access rights to an element or directory by an exact match to a user's identifier in an ACE. Oracle CDD/Repository searches ACLs sequentially beginning with the first ACE and continues to search until it finds the first match or reaches the last ACE.

When Oracle CDD/Repository finds the first match, it grants only those access rights specified by the ACE. If Oracle CDD/Repository cannot find a match in an existing ACL, it does not grant any access rights for that element or directory.

If an ACL does not exist, Oracle CDD/Repository grants all users all access rights for that element or directory.

7.2 Setting OpenVMS Protections

The OpenVMS protection provisions apply to the node, device, and directory or directories where your repository resides.

The following sections describe three types of identifiers that are set at the OpenVMS DCL level: UIC (user identification code), general, and system-defined.

See the OpenVMS documentation for more information about OpenVMS identifiers.

7.2.1 Using UIC Identifiers

UICs identify each user on a system. A UIC can be in either numeric or alphanumeric format. The following examples of valid UIC identifiers identify the same user on a system:

```
[SALES, M_JONES]
M_JONES
[360,12]
```

You can use the wildcard character (*) as part of a UIC identifier. For example, if you want to specify any user in group 360, you can enter [360,*] as the UIC identifier.

7.2.2 Adding UIC-Based Protection

Use the DCL SET PROTECTION command to create UIC-based protection to specify access rights to a file or a directory for general users. For example:

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

The previous example specifies that all groups of users have READ, WRITE, and EXECUTE rights to the directory where your repository resides. Only SYSTEM and OWNER users are granted DELETE privileges.

You also can add UIC-based protection with the DCL CREATE/DIRECTORY /PROTECTION command.

When you create a repository, Oracle CDD/Repository ensures that Owner (the creator of the repository) has all access rights to any OpenVMS directories created for the repository, regardless of the UIC-based protection already defined for Owner on the OpenVMS directory in which the repository is being created. All access rights except DELETE access for System, Group, and World for any OpenVMS directories created for the repository are copied from the OpenVMS directory in which the repository is being created.

7.2.3 Using General Identifiers

The system manager for your OpenVMS system defines general identifiers in the system rights database, SYS$SYSTEM:RIGHTSLIST.DAT, to identify groups of users on the system. Examples of general identifiers might include DATAENTRY, SECRETARIES, MANAGERS, and so forth.

7.2.4 Using System-Defined Identifiers

OpenVMS automatically creates system-defined identifiers at installation time when it creates the rights database. The system assigns one or more of these identifiers, depending on the type of process a user executes: BATCH, DIALUP, INTERACTIVE, LOCAL, NETWORK, or REMOTE.

7.2.5 Specifying Multiple Identifiers

You can specify multiple identifiers by combining two or more identifiers with plus signs (+). A multiple identifier applies to only those users who match *all* of the individual identifiers in a multiple identifier. You cannot use more than one UIC in an ACE with a multiple identifier.

For example, the multiple identifier SECRETARIES+INTERACTIVE specifies only those users who have the SECRETARIES identifier enabled and who are also logged in interactively.

7.3 Protecting the Anchor Directory

Oracle CDD/Repository gives itself the identifier CDD$SYSTEM when it creates a journal or directory file. You do not need to modify the ACLs of any other files in the anchor directory. The Oracle CDD/Repository installation procedure sets the protection on the compatibility repository. The installation procedure also creates the template repository and places ACLs on it. All repositories that are later defined using the template are given the same ACLs.

Assigning READ, WRITE, EXECUTE, DELETE, and CONTROL ACEs to CDD$SYSTEM protects a directory so that OpenVMS utilities (including the BACKUP utility) cannot directly access repository files unless you invoke the utilities from an account with SYSPRV or BYPASS, such as the system management account.

Caution

Do not delete the CDD$SYSTEM identifier or you will not be able to access any repository files.

You should not assign the CDD$SYSTEM identifier to any users. Oracle CDD/Repository grants and revokes the identifier as needed.

7.3.1 Protecting the Anchor Directory with ACLs

Oracle CDD/Repository places an ACL on repository anchors when you perform the following tasks:

- create a new repository
- define a repository template
- move a repository
- verify a repository with the CDO VERIFY/LOCATION/FIX command, the CONVERT/REPOSITORY command, or the CDO VERIFY/ALL command if the CDD$VERIFY_ALL_FIX logical name is defined

Unless you change the default protection for your repository, Oracle CDD/Repository protects your anchor directory with the following ACL:

```
(IDENTIFIER=CDD$SYSTEM,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
(IDENTIFIER=[*,*],ACCESS=READ+EXECUTE)
(IDENTIFIER=CDD$SYSTEM,OPTIONS=DEFAULT+NOPROPAGATE,ACCESS=READ+WRITE+
  EXECUTE+DELETE+CONTROL+BIT_5+BIT_6+BIT_7+BIT_8+BIT_9+BIT_10+BIT_11+
  BIT_12+BIT_13+BIT_14+BIT_15+BIT_16+BIT_17+BIT_18)
(IDENTIFIER=[username],OPTIONS=DEFAULT+NOPROPAGATE,ACCESS=READ+WRITE+
  EXECUTE+DELETE+CONTROL+BIT_5+BIT_6+BIT_7+BIT_8+BIT_9+BIT_10+BIT_11+
  BIT_12+BIT_13+BIT_14+BIT_15+BIT_16+BIT_17+BIT_18)
(IDENTIFIER=[*,*],OPTIONS=DEFAULT+NOPROPAGATE,ACCESS=BIT_8)
```

With this ACL, only repository files can be created in an anchor directory by anyone other than the user who created the repository.

You can change the default protection on the anchor directory by using either one of the following methods:

- the DCL SET ACL/ACL command:

  ```
  $ SET ACL/ACL=(IDENTIFIER=CDD$SYSTEM,ACCESS=READ+WRITE+EXECUTE+DELETE+
  CONTROL) [SMITH]DIC.DIR

  $ SET ACL/ACL=(IDENTIFIER=[*,*],ACCESS=READ) [SMITH]DIC.DIR
  ```

- the ACL editor:

  ```
  $ SET ACL/EDIT [SMITH]DIC.DIR
  $ EDIT/ACL [SMITH]DIC.DIR
  ```

7.3.2 Displaying the ACL

To display the ACL, use the DCL SHOW ACL command or the DIRECTORY /SECURITY command. For example:

```
$ SHOW ACL [SMITH]REPO1.DIR
Object type: FILE,  Object name: DISK$400:[SMITH]REPO1.DIR;1,
on 10-DEC-1995 16:52:02.15
    (IDENTIFIER=CDD$SYSTEM,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
    (IDENTIFIER=[*,*],ACCESS=READ+EXECUTE)
    (IDENTIFIER=CDD$SYSTEM,OPTIONS=DEFAULT+NOPROPAGATE,ACCESS=READ+WRITE+
    EXECUTE+DELETE+CONTROL+BIT_5+BIT_6+BIT_7+BIT_8+BIT_9+BIT_10+BIT_11+
    BIT_12+BIT_13+BIT_14+BIT_15+BIT_16+BIT_17+BIT_18)
    (IDENTIFIER=[MCKEEN],OPTIONS=DEFAULT+NOPROPAGATE,ACCESS=READ+WRITE+
    EXECUTE+DELETE+CONTROL+BIT_5+BIT_6+BIT_7+BIT_8+BIT_9+BIT_10+BIT_11+
    BIT_12+BIT_13+BIT_14+BIT_15+BIT_16+BIT_17+BIT_18)
    (IDENTIFIER=[*,*],OPTIONS=DEFAULT+NOPROPAGATE,ACCESS=BIT_8)
```

7.4 Preventing Repository Corruption

The Oracle CDD/Repository security facilities are a safeguard against unauthorized access to the repository. However, these facilities are not guaranteed to prevent deliberate attempts to corrupt a repository.

For security reasons, you should be cautious in granting DELETE or CONTROL access rights to repository users. CONTROL should be granted only to the data administrator, the system manager, and users with personal directories. Remember also that users with OpenVMS BYPASS or SYSPRV privileges can override Oracle CDD/Repository and OpenVMS file security provisions. In general, you should grant users the minimum privileges they need to work in their portions of the repository.

In addition to ACLs, repositories are protected by Oracle Rdb locking mechanisms. Oracle CDD/Repository uses these mechanisms to protect repository elements from simultaneous updates. When a user is modifying an element, another user cannot modify the same element until the first user finishes, regardless of the user's access rights.

7.5 Protecting a Repository

Oracle CDD/Repository checks access rights to a repository when you try to attach to a specific repository. Any user can write in any repository with little or no restriction. However, you need CHANGE access to a repository if you want to change the default ACL on it. You also need CHANGE access for repositories when you relate a definition in a repository to a definition in another repository.

For example, suppose you define record REC_1 in repository REP_A and include a field called F1 in REC_1. Field F1 is in another repository called REP_B. You need CHANGE access at the repository level for both repository REP_A and REP_B, because during the CDO DEFINE RECORD command Oracle CDD/Repository must open each repository, as follows:

- If field F1 has already been used by some other definition in repository REP_A, a clone of F1 exists in REP_A. In that case, Oracle CDD/Repository opens only repository REP_B, which contains the master copy of the field, and updates the master copy with information that F1 is now also used by REC_1 in repository REP_A.
- If this is the first time that field F1 is being referenced in repository REP_A, Oracle CDD/Repository must open REP_A (for WRITE access) and make a clone of field F1 in repository REP_A.

  Oracle CDD/Repository then opens repository REP_B (for WRITE access) because it contains the existing field. Oracle CDD/Repository also opens any other repositories that use the field and updates copies of the field with information that repository REP_A also contains a copy of the field.

The following example grants SYSTEM access rights for the PERSONNEL repository:

```
CDO> DEFINE PROTECTION FOR REPOSITORY PERSONNEL
cont> POSITION 2
cont> IDENTIFIER [SYSTEM]
cont> ACCESS SHOW+DEFINE+CHANGE+DELETE+CONTROL
```

When you perform backup and verify operations, you need sole access to a repository for the operation to be successful. You must be the only user at the time you run the VERIFY command.

If you do not have DELETE access to the repository database, CDO prevents you from moving or deleting your repository. However, you can still use the CDO VERIFY/REBUILD_DIRECTORY command, provided you have SYSPRV or BYPASS privilege.

You can change the default ACL on a repository if you have CHANGE access, by using the CDO DEFINE PROTECTION command. The following example changes the default access for PERSONNEL:

```
CDO> DEFINE PROTECTION FOR REPOSITORY PERSONNEL
cont> POSITION 2
cont> IDENTIFIER [SYSTEM]
cont> DEFAULT_ACCESS READ+WRITE+DELETE+CONTROL
```

7.6 Protecting Repository Directories

Repository directories are part of the hierarchy you can create within a repository. Any user who has access to the repository can list directory contents. Oracle CDD/Repository grants four directory access rights to users. Table 7–1 describes each of these access rights.

Table 7–1 Access Rights for Directories

Access Right	Access Permitted
[NO]SHOW	Read and copy directory contents
[NO]CHANGE	Create new elements and new versions of elements in the directory; delete elements from the directory
[NO]DELETE	Delete an empty directory
[NO]CONTROL	Define, change, and delete directory ACLs

When you display the access rights for a directory using the CDO SHOW PROTECTION or the CDO SHOW PRIVILEGES command, Oracle CDD/Repository lists the access rights defined on that directory. Of those access rights, only the ones that appear in Table 7–1 apply to directories. All other access rights are irrelevant to directories, so they are ignored.

The CHANGE access right for directories differs from the CHANGE access right for elements. If you create a new element or new version of an element in a directory, the contents of the directory change, but Oracle CDD/Repository does not create a new directory.

When you use the CDO CHANGE command to change an element, you do not need the CHANGE access right for the directory because you are not changing the contents of the directory. Only the CDO DEFINE and DELETE commands change the contents of the directory.

The default ACL for each new repository directory and its contents is the same as the default protection for repository elements. See Section 7.7.1 for more information on this default protection.

The default protection defined at the repository level is given to all directories and elements that are created in that repository. The default protection of a repository is restrictive in terms of shared access. Therefore, if the elements are to be shared, you should change the default protection on the repository before creating elements in that repository. For example, if a group of developers is going to work in a repository, you can change the default protection so that the entire development group, as well as the individual who created the element, has SHOW and CHANGE access.

7.7 Protecting Repository Elements

Each repository element has an ACL associated with it. Oracle CDD/Repository grants users access rights to repository elements. Table 7–2 describes each of these access rights.

Table 7–2 Access Rights for Elements

Access Right	Access Permitted
[NO]SHOW	Read and copy elements
[NO]DEFINE	Create new elements and new versions of elements
[NO]CHANGE	Change elements
[NO]DELETE	Delete and purge elements
[NO]CONTROL	Define, change, and delete ACLs

When you display the access rights for an element using the CDO SHOW PROTECTION or the CDO SHOW PRIVILEGES command, Oracle CDD/Repository lists the access rights defined for that element. Of those access rights, only the ones that appear in Table 7–2 apply to elements. All other access rights are ignored.

7.7.1 Using Default Protections for Repository Elements

If you are the creator of a repository element, you have full access to it, including CONTROL access, regardless of the ACL settings. All other users have only SHOW access. To change the default protection, you must have CONTROL access.

Unless you have changed the default protection, Oracle CDD/Repository provides the following two ACEs for each new repository element:

1. The first ACE gives the creator (owner) of the element all element (SHOW+DEFINE+CHANGE+DELETE+CONTROL) access rights and all type (READ+WRITE+MODIFY+ERASE) access rights:

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

2. The second ACE gives all other users the SHOW access right to the element:

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

The presence of SHOW, DEFINE, CHANGE, and DELETE access rights in an element's ACL does not guarantee that Oracle CDD/Repository grants you these rights. Oracle CDD/Repository only grants you these rights if they are confirmed by the READ, WRITE, MODIFY, and ERASE rights in the ACL for the type of which the element is an instance.

7.7.2 Protecting Interrelated Elements

When you read a repository record, Oracle CDD/Repository checks the ACLs for each included field and record, and displays only the fields and records to which you have access.

For example, the CUSTOMER_REC record contains the field POSTAL_CODE. If you have SHOW access to the CUSTOMER_REC record, but you do not have access to the POSTAL_CODE field, Oracle CDD/Repository does not show you the entire record definition for CUSTOMER_REC.

If a record contains four fields and you have access to the record, but you only have access to three of the four fields, Oracle CDD/Repository issues an error message in response to the SHOW RECORD command. You can, however, list the record contents with the CDO DIRECTORY command.

Caution

When you have no access rights or access to only part of an element, and, if you attempt to include a record in a language program, the included record will be incomplete.

7.8 Protecting Repository Types

Oracle CDD/Repository supplies types to define its own metadata. The following are examples of types that Oracle CDD/Repository uses to create other types:

- MCS_PROPERTY_TYPE
- MCS_ELEMENT_TYPE
- MCS_HAS_PROPERTY
- MCS_HAS_RELATION
- MCS_RELATION_MEMBER

The SYSTEM account (system UIC) is the owner of all the types supplied by Oracle CDD/Repository, including the record (CDD$DATA_AGGREGATE) and field (CDD$DATA_ELEMENT) types.

7.8.1 Using Default Protection for Supplied Types

When Oracle CDD/Repository grants access rights to a user, it considers two ACLs: the ACL of the element itself and the ACL of the element's type. Oracle CDD/Repository grants an access right to a user only if the ACL of the element's type confirms the access right.

Oracle CDD/Repository grants these rights by default for all types supplied by Oracle CDD/Repository. Table 7–3 describes the access rights that must be present in a type's ACL to confirm a user's access to an instance of the type.

Table 7–3 Access Rights for Types

Privilege	Access Right Confirmed
READ	SHOW access to an instance of the type
WRITE	DEFINE access to an instance of the type
MODIFY	CHANGE access to an instance of the type
ERASE	DELETE access to an instance of the type

The default ACL for types supplied by Oracle CDD/Repository consists of the following three ACEs:

- The first ACE gives the creator (owner) of a user-defined subtype of the type all access rights. The element access rights are necessary so that owners can create instances of their type.

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

- The second ACE gives the SYSTEM account all access rights for types and CHANGE, CONTROL, and SHOW element access rights:

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

- The third ACE gives all users all access rights for types and the SHOW element access right:

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

Most repository users can ignore the rights assigned to an element by the element's type unless users on their system define their own types or change the default protection for types. Changes in the default type protection affect all instances of that type and can create a conflict of access rights. Layered products that use repository elements check the type access rights at run time.

For example, the field type, CDD$DATA_ELEMENT, which is supplied by Oracle CDD/Repository, lists the READ access right for all users in its default ACL. By confirming the SHOW access right, Oracle CDD/Repository allows all users to read and copy elements that are of this type. If the SHOW access right is not confirmed, then you cannot use the CDO SHOW or COPY commands with elements based on this type.

The default ACL for all users of the field type CDD$DATA_ELEMENT and record type CDD$DATA_AGGREGATE contains the following access rights: READ, WRITE, MODIFY, and ERASE.

This means that if you have CONTROL access to an element (an instance of these types), you can grant other users SHOW, DEFINE, CHANGE, and DELETE access rights to the elements based on these types, and Oracle CDD/Repository will confirm these rights.

7.8.2 Using Default Protection for User-Defined Types

The default ACL for user-defined types consists of the following two ACEs:

- The first ACE gives the creator (owner) of a user-defined type all type and element access rights. The element access rights are necessary so that owners can create instances of their type.

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

- The second ACE gives all users all type access rights and the SHOW access right to a user-defined type:

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

7.8.3 Using CDD$EXTENDER to Change the Protection for User-Defined Types

CDD$EXTENDER is an identifier that allows repository administrators and tool developers to define metadata. Previously only the SYSTEM identifier granted the proper access rights to define your own metadata.

You can grant the CDD$EXTENDER access rights if you have system privileges.

You can change the default protection for a user-defined type by using the identifier CDD$EXTENDER in the following ACL:

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

This example does not allow DELETE access to the user-defined type.

7.9 Displaying the Protection on an Element

To display your access rights for a particular element, use the CDO SHOW PRIVILEGES command. In the following example, the SHOW PRIVILEGES command displays the current privileges for the HOME_PHONE field.

```
CDO> SHOW PRIVILEGES FOR FIELD HOME_PHONE

 Directory SYS$COMMON:[CDDPLUS]PERSONNEL

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

The CDO SHOW PROTECTION command displays the complete ACL for the specified element. For example:

```
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,HAN],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
  CHANGE)
 (IDENTIFIER=[20,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE)
```

7.10 Organizing Access Control Entries (ACEs)

To create new entries in the ACL for an element, use the CDO DEFINE PROTECTION command. You must have CONTROL access rights to use the DEFINE PROTECTION command.

In the following example, a new ACE is defined. The user with the identifier [SECRETARIES,SUSAN] is granted SHOW and DEFINE access to the record definition ADDRESS. Then, the new access entry is inserted as the second ACE, using the POSITION 2 qualifier. The original second ACE is now the new third ACE, since there were originally only two ACEs.

```
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)
```

```
CDO> DEFINE PROTECTION FOR RECORD ADDRESS
cont> POSITION 2 IDENTIFIER [SECRETARIES,SUSAN]
cont> ACCESS SHOW+DEFINE.

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 lets you 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.

For example, the following command creates a new ACE for the field RATE and specifies that the new entry be placed after the ACE with the identifier [21,12]:

```
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)

CDO> DEFINE PROTECTION FOR FIELD RATE AFTER [21,12]
cont> IDENTIFIER [30,13]
cont> ACCESS NONE.

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>
```

When you do not specify the position or the identifier for the new ACE, Oracle CDD/Repository places it in the first position by default. You can use wildcard characters (*, %, . . .) to define the protection for all of the elements in a particular directory.

The following command defines the fourth ACE for all versions of all record elements in the BENEFITS directory:

```
CDO> DEFINE PROTECTION FOR RECORD BENEFITS.*;* POSITION 4
cont> IDENTIFIER [PERSONNEL,*] ACCESS SHOW.
```

Note

If you define a new ACE with an identifier that matches the identifier in an existing ACE, Oracle CDD/Repository deletes the original ACE for that identifier and creates a new ACE with the rights and position you specify in the DEFINE PROTECTION command.

Three additional terms make listing access rights easier:

- ALL – Enables all rights
- NOALL – Disables all rights
- NONE – Denies all rights

For example, to grant a user all access rights except CONTROL, 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.

7.11 Changing Access Control Entries (ACEs)

To modify the protection for particular elements, use the CDO CHANGE PROTECTION command. You must have CONTROL access rights to use the CDO CHANGE PROTECTION command. The new access rights you specify do not replace the old rights; instead, Oracle CDD/Repository combines them with the old set of access rights to produce a new set.

The CDO CHANGE PROTECTION command lets you alter an existing ACE. You can identify the ACE either by position or by identifier.

Note

If you do not specify the position or an identifier in the CHANGE PROTECTION command, Oracle CDD/Repository makes the change to the first ACE, by default.

In the following example, the CHANGE PROTECTION command changes the fourth ACE in the ACL so that the user has no access to the FIRST_NAME field. Oracle CDD/Repository changes the current fourth ACE instead of creating a new ACE.

```
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)

CDO> CHANGE PROTECTION FOR FIELD FIRST_NAME POSITION 4 ACCESS NONE.
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>
```

7.12 Deleting an Access Control Entry (ACE) or an Access Control List (ACL)

To delete an ACE or an ACL, use the CDO DELETE PROTECTION command. You must have CONTROL access rights to use the CDO DELETE PROTECTION command. For example:

```
CDO> DELETE PROTECTION /LOG FOR RECORD EMPLOYEE_REC POSITION 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 CDO DELETE PROTECTION 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

CDO> SHOW PROTECTION FOR RECORD EMPLOYEE_REC

  Directory SYS$COMMON:[CDDPLUS]PERSONNEL

EMPLOYEE_REC(1)
CDO>
```

After you delete the ACL for a repository element, it has no associated protection provisions, and all users have full access to the element.

7.13 Understanding Remote Access Rights

When you create a copy (a clone) of an element, the following occurs:

- If the clone is a copy of an element from a repository on the same node or node in the same cluster, the copy has the same ACLs as the original.
- If the clone is a copy of an element from a remote repository—one that is not on the same node (or node in the same cluster) from which you invoked Oracle CDD/Repository—the copy does not have the ACLs of the original. ACLs are not set because the remote ACLs have no meaning on the local node.

You cannot display the entire ACL for a repository definition on a remote node; you can show only your ACE from the ACL. (The CDO SHOW PRIVILEGE command lists only your ACE; the CDO SHOW PROTECTION command lists the entire ACL.) See Chapter 6 for more information on remote access.

8

Using Oracle CDD/Repository with Oracle Rdb

This chapter describes how to use Oracle CDD/Repository concurrently with Oracle Rdb. Included are examples of how to create, change, and track repository definitions, which can be shared by multiple Oracle Rdb databases.

To understand this chapter you should be familiar with the repository concepts discussed in Chapter 1 of *Using Oracle CDD/Repository on OpenVMS Systems* and the basic concepts of Oracle Rdb.

8.1 Introduction

Oracle Rdb is a relational database product that supports the features of Oracle CDD/Repository. When you create a database, other databases (and potentially other products) can share the same database metadata to reduce redundancies. Sharing metadata provides consistency across databases and other software products as you develop an application.

There are a variety of CDO commands that are useful for Oracle Rdb users, because they let you perform the following tasks:

- define metadata
- track metadata
- set protection on metadata
- delete or modify metadata

Oracle CDD/Repository keeps track of all users of a particular repository element, so you can analyze the effects of a change to metadata used throughout an application. Whenever metadata is changed, a message about the change is associated with the repository description of any Oracle Rdb databases using the metadata.

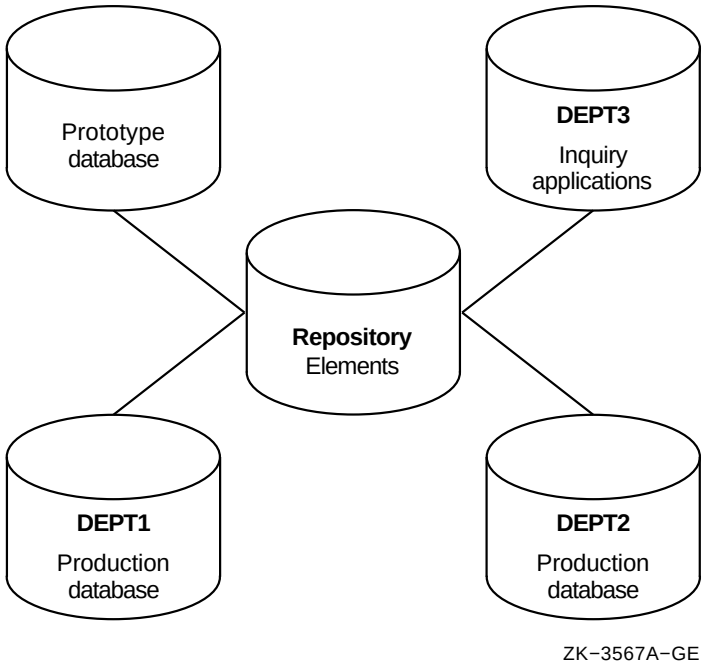
As an Oracle Rdb user, you gain access to repository definitions through CDO to create, modify, delete, or inquire about elements to which you have the appropriate access privileges.

When you use SQL to attach to a database by repository pathname, SQL informs you if a message about a repository change is associated with the database. You can read the message from CDO.

Messages warn you that repository definitions may be inconsistent with database metadata and that the definitions may need to be synchronized. This synchronization is discussed in Section 8.3.3. You can create and maintain standard definitions in Oracle CDD/Repository, which are replicated in any number of Oracle Rdb databases, then share them with other applications.

Figure 8–1 shows a prototype database, two department-wide production databases, and end-user applications that share the same elements through Oracle CDD/Repository.

Figure 8–1 Sharing Repository Definitions Among Database Products



Because the elements reside in the repository and not in any particular database, you can maintain them independently and share them with other databases.

For example, using CDO, you can define a field and base other fields on it, as follows:

```
$ REPOSITORY OPERATOR
CDO> DEFINE FIELD ID_NUMBER
cont> DESCRIPTION IS 'Corporate Standard ABRII'
cont> DATATYPE IS TEXT SIZE IS 5.
CDO> DEFINE FIELD EMPLOYEE_ID
cont> AUDIT IS 'copied from Corporate Standard 5-01-91'
cont>  BASED ON ID_NUMBER.
CDO> DEFINE FIELD STANDARD_DATE
cont> DESCRIPTION IS 'The corporate standard for date'
cont> DATATYPE IS DATE.
```

After defining fields and records in the repository using CDO, you can share them across multiple Oracle Rdb databases by issuing SQL statements to define the fields (which are called domains in SQL) and records (called tables or relations in SQL) from the repository through pathnames.

Section 8.2 contains more information about this process.

8.2 Creating Shareable Definitions

To create definitions in a repository and make them shareable between the repository and the database or multiple databases, use the following process:

1. Use CDO to define the fields (domains) and records (tables) for a database.
2. Use SQL to define the database.
3. Use SQL to copy the fields (domains) and records (tables) from the repository to the database.
4. Use SQL to create local views, constraints, and indexes. You can mix local and shared database definitions of domains and tables, as needed.
5. Use SQL to include any changed definitions in your repository.
6. Share the definitions, as needed, among other Oracle Rdb databases.

8.2.1 Examples of Creating Shareable Definitions

The following series of examples shows how to use a repository to create definitions for multiple Oracle Rdb databases. These examples assume the following:

- A repository is located at SYS$COMMON:[CDDREP] (the anchor).
- A directory called PERS, which is in the repository, contains the database definitions.

Example 8–1 shows how to use CDO to define a repository field called ID_NUMBER and base another field, EMPLOYEE_ID, on it.

Example 8–1 Defining Shareable Fields in CDO

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]PERS
CDO> DEFINE FIELD ID_NUMBER
cont> DESCRIPTION IS 'Corporate Standard ABRII'
cont> DATATYPE IS TEXT SIZE IS 5.
CDO> DEFINE FIELD EMPLOYEE_ID
cont> AUDIT IS 'copied from Corporate Standard 5-01-91'
cont>  BASED ON ID_NUMBER.
```

Define a repository field, STANDARD_DATE, and base other fields on it:

```
CDO> DEFINE FIELD STANDARD_DATE
cont> DESCRIPTION IS 'The corporate standard for date'
cont> DATATYPE IS DATE.
CDO> DEFINE FIELD START_DATE
cont> DESCRIPTION IS 'employee start date'
cont> BASED ON STANDARD_DATE.

CDO> DEFINE FIELD END_DATE
cont> DESCRIPTION IS 'employee termination date'
cont> BASED ON STANDARD_DATE.

CDO> DEFINE FIELD SALARY_AMOUNT
cont> DATATYPE IS SIGNED QUADWORD SCALE -2.
CDO> DEFINE FIELD BIRTHDAY
cont> DESCRIPTION IS 'employee date of birth'
cont> BASED ON STANDARD_DATE.
```

Define other repository fields for the database table called EMPLOYEES.

```
CDO> DEFINE FIELD LAST_NAME
cont> DESCRIPTION IS 'Employee last name'
cont> DATATYPE IS TEXT SIZE IS 14.

CDO> DEFINE FIELD FIRST_NAME
cont> DESCRIPTION IS 'Employee first name'
cont> DATATYPE IS TEXT SIZE IS 10.
```

(continued on next page)

Example 8–1 (Cont.) Defining Shareable Fields in CDO

```
CDO> DEFINE FIELD MIDDLE_INITIAL
cont> DESCRIPTION IS 'Employee middle initial'
cont> DATATYPE IS TEXT SIZE IS 1.

CDO> DEFINE FIELD ADDRESS_DATA_1
cont> DESCRIPTION IS 'Employee home street address'
cont> DATATYPE IS TEXT SIZE IS 25.

CDO> DEFINE FIELD ADDRESS_DATA_2
cont> DESCRIPTION IS 'Employee street or PO Box address'
cont> DATATYPE IS TEXT SIZE IS 25.

CDO> DEFINE FIELD CITY
cont> DESCRIPTION IS 'Employee home city address'
cont> DATATYPE IS TEXT SIZE IS 20.

CDO> DEFINE FIELD STATE
cont> DESCRIPTION IS 'Employee home state address'
cont> DATATYPE IS TEXT SIZE IS 2.

CDO> DEFINE FIELD POSTAL_CODE
cont> DESCRIPTION IS 'Employee home 9-digit postal code'
cont> DATATYPE IS TEXT SIZE IS 9.

CDO> DEFINE FIELD SEX
cont> DESCRIPTION IS 'M=male or F=female'
cont> DATATYPE IS TEXT SIZE IS 1.

CDO> DEFINE FIELD STATUS_CODE
cont> DESCRIPTION IS 'F=full-time or P=part-time employee'
cont> DATATYPE IS TEXT SIZE IS 1.
```

Use CDO to verify that the fields have been defined, as shown in Example 8–2.

Example 8–2 Listing Field Definitions

```
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]PERS
CDO> DIRECTORY *

Directory SYS$COMMON:[CDDREP]PERS

ADDRESS_DATA_1(1)                               FIELD
ADDRESS_DATA_2(1)                               FIELD
BIRTHDAY(1)                                     FIELD
CITY(1)                                         FIELD
EMPLOYEE_ID(1)                                  FIELD
END_DATE(1)                                     FIELD
FIRST_NAME(1)                                   FIELD
ID_NUMBER(1)                                    FIELD
LAST_NAME(1)                                    FIELD
MIDDLE_INITIAL(1)                               FIELD
POSTAL_CODE(1)                                  FIELD
SALARY_AMOUNT(1)                                FIELD
SEX(1)                                          FIELD
STANDARD_DATE(1)                                FIELD
START_DATE(1)                                   FIELD
STATE(1)                                        FIELD
STATUS_CODE(1)                                  FIELD
```

For more information on an element, issue a CDO SHOW FIELD /ALL command.

```
CDO> SHOW FIELD/ALL CITY

Definition of field CITY
 |   acl              (IDENTIFIER=[SYS1,SMITH],ACCESS=READ+WRITE+
MODIFY+ERASE+SHOW+DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[*,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+OPERATOR+
ADMINISTRATOR)
 |    Created time            4-JAN-1994 11:30:08.07
 |    Description             'Employee home city address'
 |    Modified time           4-JAN-1994 11:30:08.07
 |    Owner                   [SYS1,SMITH]
 |    Datatype                text size is 20 characters
 |     |    History entered by SMITH ([SYS1,SMITH])
 |     |      using CDO V2.3
 |     |      to CREATE definition on 4-MAY-1994 11:30:02.02
   .
   .
   .
```

(continued on next page)

Example 8–2 (Cont.) Listing Field Definitions

As you define additional elements, you can identify what other elements use them.

```
   .
   .
   .
CDO> SHOW USES STANDARD_DATE
Owners of SYS$COMMON:[CDDREP]PERS.STANDARD_DATE(1)
 |    SYS$COMMON:[CDDREP]PERS.END_DATE(1)    (Type : FIELD)
 |     |    via CDD$DATA_ELEMENT_BASED_ON
 |    SYS$COMMON:[CDDREP]PERS.START_DATE(1)  (Type : FIELD)
 |     |    via CDD$DATA_ELEMENT_BASED_ON
 |    SYS$COMMON:[CDDREP]PERS.BIRTHDAY(1)    (Type : FIELD)
 |     |    via CDD$DATA_ELEMENT_BASED_ON
   .
   .
   .
```

Example 8–3 shows how to use CDO to create a repository record.

First, define a record in the repository for the database table EMPLOYEES. Then define a record in the repository for the SALARY_HISTORY table in the database.

Example 8–3 Defining a Record

```
   .
   .
   .
CDO> DEFINE RECORD EMPLOYEES
cont>  DESCRIPTION IS /* CORPORATE EMPLOYEE INFORMATION */.
cont> EMPLOYEE_ID.
cont> LAST_NAME.
cont> FIRST_NAME.
cont> MIDDLE_INITIAL.
cont> ADDRESS_DATA_1.
cont> ADDRESS_DATA_2.
cont> CITY.
cont> STATE.
cont> POSTAL_CODE.
cont> BIRTHDAY.
cont> SEX.
cont> STATUS_CODE.
cont> END EMPLOYEES RECORD.
```

(continued on next page)

Example 8–3 (Cont.) Defining a Record

```
   .
   .
   .
CDO> DEFINE RECORD SALARY_HISTORY
cont> DESCRIPTION IS /* INFO ABOUT EACH JOB HELD */.
cont> EMPLOYEE_ID.
cont> SALARY_AMOUNT.
cont> START_DATE.
cont> END_DATE.
cont> END SALARY_HISTORY RECORD.
   .
   .
   .

CDO> SHOW RECORD EMPLOYEES
Definition of record EMPLOYEES
 |    Description              'CORPORATE EMPLOYEE INFORMATION'
 |    Contains field           EMPLOYEE_ID
 |    Contains field           LAST_NAME
 |    Contains field           FIRST_NAME
 |    Contains field           MIDDLE_INITIAL
 |    Contains field           ADDRESS_DATA_1
 |    Contains field           ADDRESS_DATA_2
 |    Contains field           CITY
 |    Contains field           STATE
 |    Contains field           POSTAL_CODE
 |    Contains field           BIRTHDAY
 |    Contains field           SEX
 |    Contains field           STATUS_CODE

CDO> SHOW RECORD SALARY_HISTORY
Definition of record SALARY_HISTORY
 |    Description              'INFO ABOUT EACH JOB HELD'
 |    Contains field           EMPLOYEE_ID
 |    Contains field           SALARY_AMOUNT
 |    Contains field           START_DATE
 |    Contains field           END_DATE
CDO> EXIT
```

The EMPLOYEES and SALARY_HISTORY definitions that you created using CDO can be shared by many Oracle Rdb databases.

Example 8–4 defines two department databases, DEPT1 and DEPT2, that share the EMPLOYEES and SALARY_HISTORY definitions.

Create the DEPT1 database in SQL using a relative pathname. The full pathname for DEPT1 is SYS$COMMON:[CDDREP]PERS.DEPT1. Create the DEPT2 database using the full pathname with the anchor representing the repository origin.

Example 8–4 Using Repository Definitions to Create a Database

```
$ DEFINE CDD$DEFAULT SYS$COMMON:[CDDREP]
$ SQL
SQL> CREATE DATABASE FILENAME DEPT1
cont>              PATHNAME PERS.DEPT1
cont>              DICTIONARY IS REQUIRED;
SQL> DISCONNECT ALL;

SQL> CREATE DATABASE FILENAME DEPT2
cont>              PATHNAME SYS$COMMON:[CDDREP]PERS.DEPT2;
SQL> DISCONNECT ALL;

SQL> ATTACH 'PATHNAME SYS$COMMON:[CDDREP]PERS.DEPT1';
SQL> SHOW TABLES
User tables in database with path name SYS$COMMON:[CDDREP]PERS.DEPT1(1)
No tables found
   .
   .
   .
```

Create the EMPLOYEES table and the SALARY_HISTORY table in SQL using the repository definitions:

```
   .
   .
   .
SQL> CREATE TABLE
cont> FROM SYS$COMMON:[CDDREP]PERS.EMPLOYEES;
SQL> CREATE TABLE
cont> FROM SYS$COMMON:[CDDREP]PERS.SALARY_HISTORY;
```

(continued on next page)

Example 8–4 (Cont.) Using Repository Definitions to Create a Database

```
SQL> SHOW TABLES
User tables in database with pathname SYS$COMMON:[CDDREP]PERS.DEPT1(1)
     EMPLOYEES
     SALARY_HISTORY
SQL> COMMIT;

SQL> SHOW TABLE EMPLOYEES

CDD Pathname:   SYS$COMMON:[CDDREP]PERS.EMPLOYEES(1)

Comment on table EMPLOYEES:
CORPORATE EMPLOYEE INFORMATION

Columns for table EMPLOYEES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)          EMPLOYEE_ID
LAST_NAME                      CHAR(14)         LAST_NAME
FIRST_NAME                     CHAR(10)         FIRST_NAME
MIDDLE_INITIAL                 CHAR(1)          MIDDLE_INITIAL
ADDRESS_DATA_1                 CHAR(25)         ADDRESS_DATA_1
ADDRESS_DATA_2                 CHAR(25)         ADDRESS_DATA_2
CITY                           CHAR(20)         CITY
STATE                          CHAR(2)          STATE
POSTAL_CODE                    CHAR(9)          POSTAL_CODE
BIRTHDAY                       DATE             BIRTHDAY
SEX                            CHAR(1)          SEX
STATUS_CODE                    CHAR(1)          STATUS_CODE

   .
   .
   .
SQL> SHOW TABLE SALARY_HISTORY

CDD Pathname:   SYS$COMMON:[CDDREP]PERS.SALARY_HISTORY(1)

Comment on table SALARY_HISTORY:
INFO ABOUT EACH JOB HELD

Columns for table SALARY_HISTORY:
Column Name                    Data Type        Domain
-----------                    ---------        ------
EMPLOYEE_ID                    CHAR(5)          EMPLOYEE_ID
SALARY_AMOUNT                  QUADWORD(2)      SALARY_AMOUNT
START_DATE                     DATE             START_DATE
END_DATE                       DATE             END_DATE
```

(continued on next page)

Example 8–4 (Cont.) Using Repository Definitions to Create a Database

```
SQL> COMMIT;
SQL> DISCONNECT ALL;
```

Use SQL to set the appropriate protection rights on metadata in the database, then commit the transaction.

Use CDO to set protection on the repository definitions. By default, CDO grants all privileges including CONTROL to Owner; World is granted the SHOW privilege only.

See Chapter 7 for a discussion of repository protection. After committing the transaction, you can track any modifications to the Oracle Rdb domains (fields) and tables (records) through Oracle CDD/Repository by using the CDO SHOW USES command.

8.2.2 Creating Record Constraints

In Oracle CDD/Repository, you can specify conditions, known as constraints, that affect adding or modifying data in a database table (CDO record). CDO provides syntax for record constraints, including specification of NOT NULL, PRIMARY KEY, FOREIGN KEY, UNIQUE, and CHECK (arbitrary search condition constraint) for fields and records.

See the *Oracle CDD/Repository CDO Reference Manual* and the Oracle CDD/Repository release notes for information on the CDO DEFINE RECORD command and command syntax.

Each constraint can be named and supplied with evaluation attributes DEFERRABLE and NOT DEFERRABLE in SQL.

The following examples show the use of the record constraints feature.

Example 8–5 uses the CDO DEFINE RECORD command syntax to establish constraints on the PARTS record in the ACME Company database.

ACME wants the PART_ID to be the primary key and also the PART_NO to be a unique value across all possible parts. By not specifying whether the constraints are deferrable, ACME accepts the default evaluation time. In CDO, the default evaluation time for constraints is NOT DEFERRABLE. Constraints are evaluated at statement time.

Using CDO, ACME defines the record PARTS with the following attributes:

Primary key PARTS_PMK
Unique constraint PARTS_UNQ
Check constraint PART_CST
Foreign key constraint PART_FRK

Note

For the purposes of this example, it is assumed that the field definitions referred to in the record definitions have already been defined in the repository.

Example 8–5 Using the CDO DEFINE RECORD Command to Establish Primary, Unique, Check, and Foreign Key Constraints

```
   .
   .
   .
CDO> DEFINE RECORD PARTS
cont> CONSTRAINT PARTS_PMK PRIMARY KEY PART_ID
cont> CONSTRAINT PARTS_UNQ UNIQUE PART_NO
cont> CONSTRAINT PART_CST CHECK
cont> (ANY P IN PARTS WITH (PART_ID IN PARTS = PART_ID_USED_IN IN PARTS))
cont> CONSTRAINT PART_FRK
cont>   FOREIGN KEY PART_NO REFERENCES PARTS PART_ID.
cont> PART_NO.
cont> PART_ID.
cont> PART_ID_USED_IN.
cont> PART_QUANT.
cont> END.
```

(continued on next page)

Example 8–5 (Cont.) Using the CDO DEFINE RECORD Command to Establish Primary, Unique, Check, and Foreign Key Constraints

```
CDO> SHOW RECORD PARTS/FULL
Definition of record PARTS
|   Contains field           PART_NO
|   |   Datatype                signed word
|   Contains field           PART_ID
|   |   Datatype                signed longword
|   Contains field           PART_ID_USED_IN
|   |   Based on                ID_DOM
|   |   |   Datatype                signed longword
|   Contains field           PART_QUANT
|   |   Datatype                signed word
|   Constraint PARTS_PMK  primary key PART_ID NOT DEFERRABLE
|   Constraint PARTS_UNQ  unique PART_NO NOT DEFERRABLE
|   Constraint PART_CST (ANY (P IN PARTS WITH (PART_ID IN PARTS EQ
PART_ID_USED_IN IN PARTS))) NOT DEFERRABLE
|   Constraint PART_FRK  foreign key PART_NO references PARTS PART_ID
NOT DEFERRABLE
```

A database at Smitti's Grocery Store provides the PRODUCE record used in Example 8–6 to display combinations of NOT NULL and DEFERRABLE field attributes in a record definition.

Each item sold at the store contains several attributes that can be defined in a record: UPC number, weight, price, and quantity.

Because NOT NULL is specified on each field, when the store later inserts values into the PRODUCE record, a value must be supplied for *each* field in the PRODUCE record. The field PRICE has the DEFERRABLE attribute, allowing its value to be evaluated at commit time.

Note

If you do not specify a name for the NOT NULL constraint, as displayed for the field UPC_CODE, Oracle CDD/Repository generates its own name; in this instance, the name given is CDD$NOT_NULL_ 0096C4CF2C3448EA.

If an error pertaining to this constraint occurs, the output of the error will contain this generated name and can cause confusion. Oracle Corporation recommends that you always specify a constraint name when defining constraints.

Example 8–6 Using the CDO DEFINE RECORD Command with NOT NULL and DEFERRABLE Field Attributes

```
   .
   .
   .
CDO> DEFINE RECORD PRODUCE.
cont> UPC_CODE NOT NULL DEFERRABLE.
cont> WEIGHT CONSTRAINT WNOTNULL NOT NULL.
cont> PRICE CONSTRAINT PNOTNULL NOT NULL DEFERRABLE.
cont> QUANTITY CONSTRAINT QNOTNULL NOT NULL NOT DEFERRABLE.
cont> END.

CDO> SHOW RECORD PRODUCE/FULL
Definition of record PRODUCE
|   Contains field          UPC_CODE
|   |   Datatype                signed longword
|   Contains field          WEIGHT
|   |   Datatype                signed word
|   Contains field          PRICE
|   |   Datatype                signed longword
|   Contains field          QUANTITY
|   |   Datatype                signed word
|   Constraint CDD$NOT_NULL_0096C4CF2C3448EA  not null UPC_CODE DEFERRABLE
|   Constraint WNOTNULL  not null WEIGHT NOT DEFERRABLE
|   Constraint PNOTNULL  not null PRICE DEFERRABLE
|   Constraint QNOTNULL  not null QUANTITY NOT DEFERRABLE
```

Oracle CDD/Repository now supports defining a local field within a record. The local field must be based on an existing field.

When a repository record that contains a local field definition is used in the database, Oracle Rdb defines a local copy of the field and a global definition of the field it is based on in the database.

Example 8–7 shows how you can set an attribute on the new repository field:

Example 8–7 Setting an Attribute on the New Repository Field

```
CDO> DEFINE FIELD A
cont> DESCRIPTION IS /* global field */
cont> DATATYPE IS LONGWORD.
CDO> DEFINE RECORD R.
cont> LOC_FLD
cont> DESCRIPTION IS /* a local def to record */
cont> BASED ON A.
cont> END.

CDO> SHOW RECORD/FULL R
Definition of record R
|   Contains field          LOC_FLD
|   |   Description             /* a local def to record  */
|   |   Based on                A
|   |   |   Datatype                signed longword

CDO> DIRECTORY
Directory disk:[dir.V53_DICT]

A(1)                                            FIELD
CDD$PROTOCOLS                                   DIRECTORY
R(1)                                            RECORD
CDO> EXIT
```

Although the field LOC_FLD exists internally, it cannot be displayed using a CDO DIRECTORY command because it does not contain a directory name. Use the CDO ENTER command to assign a directory name to the field, then display the definition using the CDO DIRECTORY command.

Note

The CDO ENTER command does not apply a default ACL to the element you are entering. Therefore, if the element did not have an ACL prior to being entered, it will not have one after it is entered in the directory system. You will need to set the ACL.

In the following example, the CDO ENTER command enters a directory name in the local directory for fields A and C, which are located in the CORPORATE directory.

```
CDO> DEFINE REPOSITORY USER$ANCHOR_DIR.
CDO> DEFINE REPOSITORY USER$ANCHOR_DIR2.
CDO> DEFINE DIRECTORY USER$ANCHOR_DIR:CORPORATE.
CDO> DEFINE DIRECTORY USER$ANCHOR_DIR2:FARWAY.
CDO> SET DEFAULT USER$ANCHOR_DIR:CORPORATE
CDO> DEFINE FIELD A.
CDO> DEFINE FIELD B.
CDO> DEFINE FIELD C.
CDO> SET DEFAULT USER$ANCHOR_DIR2:FARWAY
CDO> ENTER FIELD A FOR USER$ANCHOR_DIR:CORPORATE.A
CDO> ENTER FIELD C FOR USER$ANCHOR_DIR:CORPORATE.C
CDO> EXIT
```

Or, you can use SQL to display the field and its attributes. For example:

```
$ SQL
SQL> CREATE DATABASE PATHNAME LOCAL_DB FILENAME LOCAL_DB;
SQL> CREATE TABLE FROM R;
SQL> SHOW DOMAIN A
A                               INTEGER
 Comment:       global field
CDD Pathname:  disk:[dir.V53_DICT]A;1

SQL> SHOW DOMAINS
User domains in database with pathname LOCAL_DB
A                               INTEGER

SQL> SHOW DOMAINS A
A                               INTEGER
 Comment:       global field
CDD Pathname:  disk:[dir.V53_DICT]A;1

SQL> SHOW TABLE R
Information for table R

CDD Pathname:   disk:[dir.V53_DICT]R;1
```

```
Columns for table R:
Column Name                    Data Type        Domain
-----------                    ---------        ------
LOC_FLD                        INTEGER          A
Comment:       a local def to record

Table constraints for R:
No constraints found

Constraints referencing table R:
No constraints found

Indexes on table R:
No indexes found

Storage Map for table R:
No Storage Map found

Triggers on table R:
No triggers found

SQL> SHOW ALL TABLES (COLUMNS) R
Information for table R

CDD Pathname:   disk:[dir.V53_DICT]R;1

Columns for table R:
Column Name                    Data Type        Domain
-----------                    ---------        ------
LOC_FLD                        INTEGER          A
Comment:       a local def to record
SQL>
```

The ADDRESS record for an EMPLOYEES database in Example 8–8 shows the use of local field definitions using BASED ON attributes from the CDO DEFINE RECORD syntax diagram.

In this example, the fields FIRST_NAME and LAST_NAME are both based on the 30-character text string, NAME_STRING, and contain descriptions. The field ZIP_CODE must be specified when you insert values into the record. This requirement is established by the NOT NULL attribute.

Example 8–8 Using CDO DEFINE RECORD Field Attributes to Specify BASED ON Attributes

```
CDO> DEFINE RECORD ADDRESS.
cont>  FIRST_NAME
cont>           DESCRIPTION IS /* employee first name */
cont>           BASED ON NAME_STRING.
cont>  LAST_NAME
cont>           DESCRIPTION /* employee last name */
cont>           BASED ON NAME_STRING.
cont>  STREET.
cont>  CITY.
cont>  STATE.
cont>  ZIP_CODE NOT NULL.
cont> END.

CDO> SHOW RECORD ADDRESS/FULL
Definition of record ADDRESS
|   Contains field           FIRST_NAME
|   |   Description             /* employee first name  */
|   |   Based on                NAME_STRING
|   |   |   Datatype                text size is 30 characters
|   Contains field           LAST_NAME
|   |   Description             /* employee last name  */
|   |   Based on                NAME_STRING
|   |   |   Datatype                text size is 30 characters
|   Contains field           STREET
|   |   Datatype                text size is 40 characters
|   Contains field           CITY
|   |   Datatype                text size is 20 characters
|   Contains field           STATE
|   |   Datatype                text size is 2 characters
|   Contains field           ZIP_CODE
|   |   Datatype                text size is 5 characters
|   Constraint CDD$NOT_NULL_0096C4D0443E0F10 not null ZIP_CODE NOT DEFERRABLE
```

The CAR record in Example 8–9 is from a database at Herb's Auto Dealer. The record has several fields to define a car's make, year, color, cost, and the date sold.

The local field MAKE is based on the CAR_MAKES field, and the local field CAR_YEAR, which defined the year of the car, is based on the YEAR field. The COST field contains the attributes ALIGNED ON QUAD and NOT NULL. The local field DATE_SOLD is based on the DATE field.

Example 8–9 Using CDO DEFINE RECORD Field Attributes to Specify BASED ON and ALIGNED ON Attributes

```
CDO> DEFINE RECORD CAR.
cont> MAKE
cont>       DESCRIPTION IS /* make of car */
cont>       BASED ON CAR_MAKES.

cont> CAR_YEAR
cont>       DESCRIPTION IS /* year of car */
cont>       BASED ON YEAR.

cont> COLOR DESCRIPTION IS /* color of car */.
cont> COST DESCRIPTION IS /* price of car */
cont>       BASED ON PRICE ALIGNED ON QUAD NOT NULL.
cont> DATE_SOLD
cont>       DESCRIPTION IS /* date of sale */
cont>       BASED ON DATE.
cont> END.

CDO> SHOW RECORD CAR/FULL
Definition of record CAR
|   Contains field           MAKE
|   |   Description             /* make of car  */
|   |   Based on                CAR_MAKES
|   |   |   Datatype                text size is 10 characters
|   Contains field           CAR_YEAR
|   |   Description             /* year of car  */
|   |   Based on                YEAR
|   |   |   Datatype                signed word
|   Contains field           COLOR
|   |   Description             /* color of car  */
|   |   Based on                COLOR
|   |   |   Datatype                text size is 12 characters
|   Contains field           COST
|   |   Aligned on quadword boundary
|   |   Description             /* price of car  */
|   |   Based on                PRICE
|   |   |   Datatype                signed longword
|   Contains field           DATE_SOLD
|   |   Description             /* date of sale  */
|   |   Based on                DATE
|   |   |   Datatype                date
|   Constraint CDD$NOT_NULL_0096C4D3A9B805E6 not null COST NOT DEFERRABLE
```

Example 8–10 displays how to use fields from different directories in record definitions when using the BASED ON attribute.

The repository in this example has two directories: CUST containing customer information, and EMPLOY containing employee information.

In the top level default directory, two fields are created: FIRST_NAME and LAST_NAME. These fields are used in the record definition for CUSTOMER and EMPLOYEE in their respective directories. Defining fields using BASED ON within record definitions can eliminate some CDD-E-DUPGLOBAL errors.

Example 8–10 Defining Fields Using BASED ON Attributes Within Record Definitions in Different Directories

```
   .
   .
   .
CDO> DEFINE DIRECTORY CDD$DEFAULT.CUST.
CDO> DEFINE DIRECTORY CDD$DEFAULT.EMPLOY.
CDO> DEFINE FIELD CDD$DEFAULT.FIRST_NAME
cont> DATATYPE IS TEXT SIZE IS 10.
CDO> DEFINE FIELD CDD$DEFAULT.LAST_NAME
cont> DATATYPE IS TEXT SIZE IS 15.
CDO> SET DEFAULT CDD$DEFAULT.CUST
CDO> DEFINE RECORD CUSTOMER.
cont> CUST_FNAME BASED ON CDD$DEFAULT.FIRST_NAME.
cont> CUST_LNAME BASED ON CDD$DEFAULT.LAST_NAME.
cont> END.

CDO> SHOW RECORD/FULL CUSTOMER
Definition of record CUSTOMER
|   Contains field          CUST_FNAME
|   |   Based on                FIRST_NAME
|   |   |   Datatype              text size is 10 characters
|   Contains field          CUST_LNAME
|   |   Based on                LAST_NAME
|   |   |   Datatype              text size is 15 characters
CDO> SET DEFAULT CDD$DEFAULT.EMPLOY
CDO> DEFINE RECORD EMPLOYEE.
cont> EMP_FNAME BASED ON CDD$DEFAULT.FIRST_NAME.
cont> EMP_LNAME BASED ON CDD$DEFAULT.LAST_NAME.
cont> END.

CDO> SHOW RECORD/FULL EMPLOYEE
Definition of record EMPLOYEE
|   Contains field          EMP_FNAME
|   |   Based on                FIRST_NAME
|   |   |   Datatype              text size is 10 characters
|   Contains field          EMP_LNAME
|   |   Based on                LAST_NAME
|   |   |   Datatype              text size is 15 characters
```

8.2.3 Using Conditional Expressions

This section describes the support for conditional expressions between Oracle Rdb and Oracle CDD/Repository, which was implemented in Version 6.1 of Oracle CDD/Repository.

A conditional expression (or Boolean expression) represents the relationship between two value expressions. A value expression returns a value that can be a string, a number, or a null value. A conditional expression returns a value of true, false, or null (missing). Conditional expressions consist of value expressions and relational or logical operators.

In Oracle Rdb a conditional expression is referred to as a CASE expression, and in Oracle CDD/Repository it is referred to as a COMPUTED BY field property.

Note

To use this functionality Oracle CDD/Repository requires SQL Version 6.0 or higher. In addition, you must specify the RDB060 option on the ATTACH statement to generate the CASE information to be stored in the repository. To do this, enter the following:

```
SQL> ATTACH 'PATHNAME FOO RDB060';
```

However, be aware that by attaching in this manner, Oracle Rdb now assumes that the version of the repository it is attached to supports all Oracle Rdb Version 6.0 features. Refer to the Oracle Rdb release notes for a list of features that Oracle CDD/Repository currently supports.

The following examples demonstrate using records that contain conditional expressions between Oracle Rdb and Oracle CDD/Repository, as well as some of the limitations of this functionality. Included are examples that use DATATRIEVE as a mechanism to store records with CHOICE statements. This information is stored in the repository as computed by fields within records that are then used to create tables in Oracle Rdb.

8.2.3.1 Restrictions

- Tables containing SQL CASE expressions that return a NULL value are not correctly displayed in Oracle CDD/Repository once they are entered as records into the repository. As shown in the following example, the NULL value is missing in the CDO SHOW command following the THEN expression.

Although it is not displayed, the NULL value is stored correctly in the expression. Also, if the ELSE clause is not specified in the SQL CASE expression, SQL defaults to returning NULL. Oracle CDD/Repository does not display the NULL value correctly once the record is entered into the repository.

```
SQL> CREATE TABLE T
cont> (TERMINATION_DATE INTEGER,
cont>  WORK_STATUS COMPUTED BY
cont>   CASE
cont>   WHEN (TERMINATION_DATE <> 0)  THEN NULL
cont>   ELSE 1
cont> END);
SQL> COMMIT;

   .
   .
   .
CDO> ENTER RECORD T FROM DATABASE FOO
CDO> SHOW RECORD T/FULL
Definition of record T
|   Contains field     TERMINATION_DATE
|   | Based on  SQL$INT
|   | |   Datatype      signed longword
|   Contains field     WORK_STATUS
|   | Datatype  text size is 0 characters
|   |   Computed by  ( IF (TERMINATION_DATE NE 0) THEN
```

- When you display records in SQL that were originally defined as DATATRIEVE CHOICE statements, the computed by source is not displayed. You must use the Oracle RMU Extract command on the database to determine the full description of the table containing the computed by expression.

  The DATATRIEVE CHOICE statement is seen as a nested IF/THEN/ELSE expression in CDO. This is then converted to a CASE expression by Oracle Rdb.

- Data types in complex computed expressions may be stored differently in CDO than in SQL. When complex CASE expressions are defined in SQL attached by pathname and stored in the repository, Oracle CDD/Repository may compute different data types than SQL. This will not affect other databases that use the record definition from the repository because Oracle Rdb will compute its own data type. However, it may affect the size of structures allocated from the record in the repository when used with languages such as Pascal, COBOL, C, and FORTRAN.

- Oracle CDD/Repository supports only the searched CASE expression and not the simple CASE expression.

The following syntax shows the simple CASE expression, which is currently not supported by Oracle CDD/Repository:

```
CASE value-expr-0
   WHEN value-expr-1 THEN value-expr-2
   ...
   [ ELSE value-expr-n ]
END
```

If you define a table in SQL containing the simple CASE expression when attached by pathname, Oracle CDD/Repository displays the following error:

```
%CDD-E-INVALID_CASE_BL, invalid CASE BLR
```

When this error occurs, redefine the CASE expression by using the searched case expression form, which is supported by Oracle CDD/Repository, as follows:

```
CASE
   WHEN value-expr-1 THEN value-expr-3
   ...
   [ ELSE value-expr-n ]
END
```

- The SQL statements in the following example are allowed in Oracle CDD/Repository, but they are invalid in Oracle Rdb.

```
CDO> DEFINE FIELD FLD1 DATATYPE SIGNED LONGWORD
cont> COMPUTED BY FLD1 * 0.25.
CDO> DEFINE RECORD REC1.
cont>  FLD_IN_REC BASED ON FLD1.
cont>  END.
   .
   .
   .
SQL> CREATE TABLE FROM REC1;

%RDB-E-NO_METADATA_UPDATE, metadata update failed
-RDMS-F-GFLDNOEX, There is not a global field named fld1 in the database.
```

8.2.3.2 CDO Syntax for COMPUTED BY Field Property

The following syntax shows how to define computed by expressions in CDO.

```
COMPUTED BY { value-expr }
            { IF cond-expr THEN value-expr [ELSE value-expr]}
```

For example:

```
CDO> DEFINE FIELD c_other
cont> COMPUTED BY
cont> IF (c ge 2 and c le 4)
cont> OR (c ge 11 and c le 20)
cont> THEN 1 ELSE 0.
```

Example 8–11 defines a computed by field, includes the field in a record, then integrates the record into the database.

The data type must be included in the computed field for it to be included in the database. The data type is used by the preprocessors in source programs.

Example 8–11 Defining a COMPUTED BY Field and Integrating a Record into a Database

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> DEFINE FIELD EMP_NAME DATATYPE TEXT 30.
CDO> DEFINE FIELD EMP_ID DATATYPE SIGNED WORD.
CDO> DEFINE FIELD SALARY DATATYPE SIGNED LONGWORD.
CDO> DEFINE FIELD TAX DATATYPE SIGNED LONGWORD
cont> COMPUTED BY SALARY * 0.33.

CDO> DEFINE RECORD EMPLOYEE.
cont>  EMP_NAME.
cont>  EMP_ID.
cont>  SALARY.
cont>  TAX.
cont> END.

CDO> DEFINE FIELD GROUP_NAME DATATYPE TEXT 30.
CDO> DEFINE FIELD EMPS_IN_GROUP DATATYPE SIGNED LONGWORD
cont> COMPUTED BY COUNT OF E IN EMPLOYEE.

CDO> DEFINE RECORD GROUP.
cont>  GROUP_NAME.
cont>  EMPS_IN_GROUP.
CDO> END.
CDO> EXIT

$ SQL
SQL> ATTACH 'PATHNAME FOO';
SQL> CREATE TABLE FROM EMPLOYEE;
SQL> CREATE TABLE FROM GROUP;
SQL> COMMIT;
```

Although only basic computed by expressions can be defined through CDO, Oracle CDD/Repository can support more complex IF-THEN expressions when they are defined through DATATRIEVE.

Example 8–12 demonstrates defining an Oracle CDD/Repository record for COBOL 88 conditionals. This example shows how records containing computed by fields, which were originally defined in the repository, can be used in a database.

First, the field VALUE_FIELD and computed by fields VALUE_A, VALUE_B, VALUE_C, and INVALID_VALUE are defined using CDO. Then, the COBOL88_REC record is created using the fields.

SQL is invoked and a table is created from the record definition in the repository. The Oracle RMU Extract command displays the table definition, and performs a simple query on the table.

Example 8–12 Defining a Record for COBOL 88 Conditionals

```
CDO> DEFINE FIELD VALUE_FIELD DATATYPE IS TEXT SIZE IS 1.
CDO> DEFINE FIELD VALUE_A
cont>  COMPUTED BY IF VALUE_FIELD EQ "A" THEN 1 ELSE 0.
CDO> DEFINE FIELD VALUE_B
cont>  COMPUTED BY IF VALUE_FIELD EQ "B" THEN 1 ELSE 0.
CDO> DEFINE FIELD VALUE_C
cont>  COMPUTED BY IF VALUE_FIELD EQ "C" THEN 1 ELSE 0.
CDO> DEFINE FIELD INVALID_VALUE
cont> COMPUTED BY
cont> IF (VALUE_FIELD GE "D" AND VALUE_FIELD LE "Z")
cont> THEN 1 ELSE 0.
CDO> DEFINE RECORD COBOL88_REC.
cont>    VALUE_FIELD.
cont>    VALUE_A.
cont>    VALUE_B.
cont>    VALUE_C.
cont>    INVALID_VALUE.
cont> END RECORD.
```

(continued on next page)

Example 8–12 (Cont.) Defining a Record for COBOL 88 Conditionals

```
CDO> SHOW RECORD COBOL88_REC/FULL
Definition of record COBOL88_REC
|   Contains field       VALUE_FIELD
|   |   Datatype            text size is 1 characters
|   Contains field       VALUE_A
|   |   Computed by         ( IF (VALUE_FIELD EQ "A") THEN 1 ELSE 0)
|   Contains field       VALUE_B
|   |   Computed by         ( IF (VALUE_FIELD EQ "B") THEN 1 ELSE 0)
|   Contains field       VALUE_C
|   |   Computed by         ( IF (VALUE_FIELD EQ "C") THEN 1 ELSE 0)
|   Contains field       INVALID_VALUE
|   |   Computed by         ( IF ((VALUE_FIELD GE "D") AND (VALUE_FIELD
LE "Z")) THEN 1 ELSE 0)

SQL> ATTACH 'PATH TESTDB';
SQL> CREATE TABLE FROM COBOL88_REC;
SQL> SHOW TABLE(COLUMN) COBOL88_REC;
Information for table COBOL88_REC

CDD Pathname:   DISK1:[CORPORATE.RDB60.CDD53]COBOL88_REC;1

Columns for table COBOL88_REC:
Column Name                    Data Type        Domain
-----------                    ---------        ------
VALUE_FIELD                    CHAR(1)          VALUE_FIELD
VALUE_A                        SMALLINT
 Computed:      ( IF (VALUE_FIELD EQ "A") THEN 1 ELSE 0)
VALUE_B                        SMALLINT
 Computed:      ( IF (VALUE_FIELD EQ "B") THEN 1 ELSE 0)
VALUE_C                        SMALLINT
 Computed:      ( IF (VALUE_FIELD EQ "C") THEN 1 ELSE 0)
INVALID_VALUE                  SMALLINT
 Computed:      ( IF ((VALUE_FIELD GE "D") AND (VALUE_FIELD LE "Z"))
THEN 1 ELSE 0)
```

(continued on next page)

Example 8–12 (Cont.) Defining a Record for COBOL 88 Conditionals

```
!
! Show output from Oracle RMU Extract command for record in database.
!
$ RMU/EXTRACT/ITEM=TABLE/OPTION=NODICT TESTDB.RDB
-- RMU/EXTRACT for Oracle Rdb
--
--                              Table Definitions
--
--------------------------------------------------------------------------------
create table COBOL88_REC (
    VALUE_FIELD VALUE_FIELD,
    VALUE_A
        computed by case
            when (VALUE_FIELD = 'A') then 1
            else 0
        end,
    VALUE_B
        computed by case
            when (VALUE_FIELD = 'B') then 1
            else 0
        end,
    VALUE_C
        computed by case
            when (VALUE_FIELD = 'C') then 1
            else 0
        end,
    INVALID_VALUE
        computed by case
            when ((VALUE_FIELD >= 'D')
                and (VALUE_FIELD <= 'Z')) then 1
            else 0
        end);

!
! Show a query on the table in the database.
!
SQL> SELECT * FROM COBOL88_REC;
 VALUE_FIELD   VALUE_A   VALUE_B   VALUE_C   INVALID_VALUE
 B                   0         1         0               0
 G                   0         0         0               1
 C                   0         0         1               0
 Y                   0         0         0               1
4 rows selected
```

Example 8–13 demonstrates defining a table with a CASE expression in SQL Version 6.0 when attached to the database by pathname.

The CASE expression is stored in the repository as a COMPUTED BY expression. To see the record in the repository, the record must first be entered and then displayed, as shown. The record can then be used from the repository by other databases to create table definitions.

Table TRAIN contains three columns: MODEL_NUMBER, PART_COUNT, and NUMBER. NUMBER is a computed column; it contains the value 100 when the MODEL_NUMBER is 0 and the PART_COUNT is greater than 0, else the number contains the value of the MODEL_NUMBER.

Example 8–13 Defining a Table with a CASE Expression Attached by PATHNAME

```
SQL> ATTACH 'PATH TESTDB';
SQL> CREATE TABLE TRAIN
cont> (MODEL_NUMBER INTEGER,
cont> PART_COUNT INTEGER,
cont> NUMBER COMPUTED BY
cont>   CASE
cont>      WHEN (MODEL_NUMBER = 0)
cont>         AND (PART_COUNT >0) THEN 100
cont>      ELSE
cont>         MODEL_NUMBER
cont>      END);

SQL> SHOW TABLE(COLUMN) TRAIN
Information for table TRAIN

Columns for table TRAIN:
Column Name                    Data Type        Domain
-----------                    ---------        ------
MODEL_NUMBER                   INTEGER
PART_COUNT                     INTEGER
NUMBER                         INTEGER
 Computed:     by
                       case
                           when (model_number = 0)
                              and (part_count >0) then 100
                           else
                              model_number
                            end

SQL> COMMIT;
```

(continued on next page)

Example 8–13 (Cont.) Defining a Table with a CASE Expression Attached by PATHNAME

```
   .
   .
   .
CDO> ENTER RECORD TRAIN FROM DATABASE TESTDB
CDO> SHOW RECORD TRAIN/FULL
Definition of record TRAIN
|   Contains field       MODEL_NUMBER
|   |   Based on            SQL$INT
|   |   |   Datatype            signed longword
|   Contains field       PART_COUNT
|   |   Based on            SQL$INT
|   |   |   Datatype            signed longword
|   Contains field       NUMBER
|   |   Datatype            signed word
|   |   Computed by         ( IF ((MODEL_NUMBER EQ 0) AND
(PART_COUNT GT 0) ) THEN 100 ELSE MODEL_NUMBER)
   .
   .
   .
```

Example 8–14 shows a table named VALUE_ATTRIBUTES created in SQL when the repository is attached to the database by pathname. This table contains several columns: VAL, BOUND, MIXES, NONULL, and ABS. This table contains several computed by columns. The record is then entered into the repository and displayed. This record can now be used by other databases.

In this example, SQL computes a data type for the computed columns BOUND and NONULL that is different from the data type Oracle CDD/Repository computes. SQL generates the data types BIGINT for BOUND and integer for NONULL, while Oracle CDD/Repository generates the data types signed longword for BOUND and signed word for NONULL.

This can happen in complex computed expressions when they are included in the repository from SQL. The data type that is stored in Oracle CDD/Repository may not be correct in this scenario; however, this will not affect other databases that include this record, because SQL still computes the proper data type.

It may, on the other hand, affect the size of structures generated by languages when you include these records in a program. See the restrictions described in Section 8.2.3.1.

At the end of this example, notice that Pascal allocates an integer for BOUND and a word for NONULL when the record VALUE_ATTRIBUTES is included.

Example 8–14 Complex Computed Expressions Example

```
$ SQL
SQL> ATTACH 'PATH TESTDB';
SQL> CREATE TABLE VALUE_ATTRIBUTES
cont> (VAL INTEGER,
cont>  BOUND COMPUTED BY  (CASE
cont>                       WHEN VAL > 100 THEN 10
cont>                          ELSE VAL
cont>                     END) * 10,
cont>  MIXES COMPUTED BY  (CASE
cont>                       WHEN VAL > 100 THEN 10
cont>                            ELSE VAL
cont>                     END) * 10
cont>                  -  (CASE
cont>                       WHEN VAL > 100 THEN 10
cont>                            ELSE VAL
cont>                     END),
cont>  NONULL COMPUTED BY (CASE
cont>                       WHEN VAL IS NULL THEN 0
cont>                            ELSE VAL
cont>                     END),
cont>  ABS COMPUTED BY    (CASE
cont>                       WHEN VAL < 0 THEN -VAl
cont>                            ELSE VAL
cont>                     END)
cont>  );
SQL> SHOW TABLE(COLUMN) VALUE_ATTRIBUTES
Information for table VALUE_ATTRIBUTES
```

(continued on next page)

Example 8–14 (Cont.) Complex Computed Expressions Example

```
Columns for table VALUE_ATTRIBUTES:
Column Name                     Data Type        Domain
-----------                     ---------        ------
VAL                             INTEGER
BOUND                           BIGINT
 Computed:       by (case
                                     when val > 100 then 10
                                       else val
                                end) * 10
MIXES                           BIGINT
 Computed:       by (case
                                     when val > 100 then 10
                                        else val
                                end) * 10
                 - (case
                                     when val > 100 then 10
                                        else val
                                end)
NONULL                          INTEGER
 Computed:       by (case
                                      when val is null then 0
                                         else val
                                end)
ABS                             INTEGER
 Computed:       by (case when val < 0 then -val else val end)
```

(continued on next page)

Example 8–14 (Cont.) Complex Computed Expressions Example

```
SQL> COMMIT;
!
! Enter the record in the repository so it can be displayed.
!
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> ENTER RECORD VALUE_ATTRIBUTES FROM DATABASE TESTDB
CDO> SHOW RECORD VALUE_ATTRIBUTES/FULL
Definition of record VALUE_ATTRIBUTES
|   Contains field          VAL
|   |   Based on                SQL$INT
|   |   |   Datatype                signed longword
|   Contains field          BOUND
|   |   Datatype                signed longword
|   |   Computed by             (( IF (VAL GT 100) THEN 10 ELSE VAL)
* 10)
|   Contains field          MIXES
|   |   Datatype                signed quadword
|   |   Computed by             ((( IF (VAL GT 100) THEN 10 ELSE VAL)
* 10) - ( IF (VAL GT 100) THEN 10 ELSE VAL))
|   Contains field          NONULL
|   |   Datatype                signed word
|   |   Computed by             ( IF (VAL MISSING ) THEN 0 ELSE VAL)
|   Contains field          ABS
|   |   Datatype                signed longword
|   |   Computed by             ( IF (VAL LT 0) THEN (- VAL) ELSE VAL)
   .
   .
   .

------------------------------------------------------------------
!
! Now include this record from the repository into a Pascal program
! using the %DICTIONARY directive
!
00003      0  0  %DICTIONARY 'CDD$DEFAULT.VALUE_ATTRIBUTES/LIST'
00004  DC  0  0 { CDD Path Name => CDD$DEFAULT.VALUE_ATTRIBUTES }
00005  D   0  0 VALUE_ATTRIBUTES =  PACKED RECORD
00006  D   0  0      VAL :  INTEGER;
00007  D   0  0      BOUND :  INTEGER;
00008  D   0  0      MIXES : [BYTE(8)] RECORD END;
00009  D   0  0      NONULL : [WORD] -32768..32767;
00010  D   0  0      ABS :  INTEGER;
00011  D   0  0      END; { record VALUE_ATTRIBUTES }
```

(continued on next page)

Example 8–14 (Cont.) Complex Computed Expressions Example

```
!
! Now include this record from the repository into a C program
! using the  #dictionary directive and see size of variables allocated.
!
    1           #dictionary "cdd$default.value_attributes"
        D       /* CDD Path Name is "cdd$default.value_attributes" */
        D       struct value_attributes
        D       {
        D           int val;
        D           int bound;
        D           struct { char cc_cdd$_unsupported_#1 [8]; } mixes;
        D           short nonull;
        D           int abs;
        D       };
%CC-I-UNSUPPTYPE, The CDD description for "mixes"
                specifies a data type not supported in C.
```

Example 8–15 displays two records originally defined in DATATRIEVE. These records are then used to create tables in Oracle Rdb. Also, tables originally defined in Oracle Rdb containing CASE expressions when attached by pathname can also be used by DATATRIEVE. In other words, the record definitions can be used in both directions.

RDB ⟹ CDD ⟹ DATATRIEVE and DATATRIEVE ⟹ CDD ⟹ RDB

However, some data type conversion restrictions exist. See the appropriate product documentation for more specific details regarding what data types are supported by each product.

The record SALARY_HISTORY_REC contains two fields: SALARY_AMOUNT and SALARY. Salary is computed by the SALARY_AMOUNT field value multiplied by some value (Social Security tax, for example) to determine the resulting salary.

The record YACHT_REC has two fields: PRICE and DISCOUNT_PRICE. The DISCOUNT_PRICE field is computed by the price multiplied by a value based on the price.

Example 8–15 displays the restriction when including DATATRIEVE records into a database. The computed by source is not stored in Oracle CDD/Repository from DATATRIEVE and, therefore, is not returned to SQL when a table is created from the repository.

When the BOAT table is displayed from SQL, it is not clear that the field DISCOUNT_PRICE is a computed by field. You must extract the table using the Oracle RMU Extract command to determine the exact table definition, including the computed by definition.

Note

Syntax does not exist in CDO for defining multiple IF-THEN statements within a computed by expression.

Example 8–15 Creating Records Originally Defined in DATATRIEVE

```
$!
$! Use DATATRIEVE to define SALARY_HISTORY_REC and YACHT_REC.
$!
$!
$ DATATRIEVE
DTR> DEFINE RECORD SALARY_HISTORY_REC USING
cont> 01 SALARY_HISTORY.
cont>  05 SALARY_AMOUNT LONG SCALE IS -2.
cont>  05 SALARY COMPUTED BY
cont>  CHOICE
cont>    SALARY_AMOUNT  LT 20000.50  THEN (SALARY_AMOUNT * .02)
cont>    SALARY_AMOUNT  LT 30000.23  THEN (SALARY_AMOUNT * .123)
cont>    SALARY_AMOUNT  LT 40000.67  THEN (SALARY_AMOUNT * .349)
cont>    ELSE (SALARY_AMOUNT * .4034)
cont>  END_CHOICE.
cont> ;
[Record is 4 bytes long.]
DTR> DEFINE RECORD YACHT_REC USING
cont> 01 BOAT.
cont>  06 PRICE REAL.
cont>  06 DISCOUNT_PRICE COMPUTED BY
cont>  CHOICE
cont>    PRICE LT 20000 THEN (PRICE * .9)
cont>    PRICE LT 30000 THEN (PRICE * .8)
cont>    PRICE LT 40000 THEN (PRICE * .7)
cont>    ELSE (PRICE * .6)
cont>  END_CHOICE
cont>  EDIT_STRING IS $$$,$$$.
cont> ;
[Record is 4 bytes long.]
DTR> EXIT
```

(continued on next page)

Example 8–15 (Cont.) Creating Records Originally Defined in DATATRIEVE

```
$!
$! Show the two records stored in the repository.
$!

$ REPOSITORY OPERATOR
CDO> SHOW RECORD YACHT_REC/FULL
Definition of record BOAT
|   Generic DTR$SOURCE_TEXT is
|                          'RECORD YACHT_REC USING'
|                          '01 BOAT.'
|                          '  06 PRICE REAL.'
|                          '  06 DISCOUNT_PRICE COMPUTED BY'
|                          '.CHOICE'
|                          '.    PRICE LT 20000 THEN (PRICE * .9)'
|                          '            PRICE LT 30000 THEN (PRICE * .8)'
|                          '            PRICE LT 40000 THEN (PRICE * .7)'
|                          '            ELSE (PRICE * .6)'
|                          '       END_CHOICE'
|                          '     EDIT_STRING IS $$$,$$$.'
|                          ';'
|   Contains field           PRICE
|   |   Datatype                 f_floating
|   Contains field           DISCOUNT_PRICE
|   |   DTR Edit_string          $$$,$$$
|   |   Computed by            ( IF (PRICE LT 20000) THEN (PRICE * 0.9)
ELSE ( IF (PRICE LT 30000) THEN (PRICE * 0.8) ELSE ( IF
(PRICE LT 40000) THEN (PRICE * 0.7) ELSE (PRICE * 0.6))))
```

(continued on next page)

Example 8–15 (Cont.) Creating Records Originally Defined in DATATRIEVE

```
CDO> SHOW RECORD SALARY_HISTORY_REC/FULL
Definition of record SALARY_HISTORY
|   Generic DTR$SOURCE_TEXT is
|                           'RECORD SALARY_HISTORY_REC USING'
|                           '01 SALARY_HISTORY.'
|                           '  05 SALARY_AMOUNT LONG SCALE IS -2.'
|                           '  05 SALARY COMPUTED BY'
|                           '  CHOICE'
|                           '       SALARY_AMOUNT  LT 20000.50  THEN
(SALARY_AMOUNT * .02)'
|                           '       SALARY_AMOUNT  LT 30000.23  THEN
(SALARY_AMOUNT * .123)'
|                           '       SALARY_AMOUNT  LT 40000.67  THEN
(SALARY_AMOUNT * .349)'
|                           '        ELSE (SALARY_AMOUNT * .4034)'
|                           '  END_CHOICE.'
|                           ';'
|   Contains field          SALARY_AMOUNT
|   |   Datatype                signed longword scale -2
|   Contains field          SALARY
|   |   Computed by             ( IF (SALARY_AMOUNT LT 20000.50) THEN
(SALARY_AMOUNT * 0.02) ELSE ( IF (SALARY_AMOUNT LT 30000.23)
THEN (SALARY_AMOUNT * 0.123) ELSE ( IF (SALARY_AMOUNT LT 40000.67) THEN
(SALARY_AMOUNT * 0.349) ELSE (SALARY_AMOUNT * 0.4034))))
CDO> EXIT

$!
$! Now create tables in Oracle Rdb from the records in DATATRIEVE.
$!

$ SQL
SQL> ATTACH 'PATH TESTDB';
SQL> CREATE TABLE FROM YACHT_REC;
SQL> CREATE TABLE FROM SALARY_HISTORY_REC;
SQL> SHOW TABLE(COLUMN) BOAT
Information for table BOAT

CDD Pathname:   DISK1:[CORPORATE.RDB60.CDD61]TRDB$SQL_CASE.YACHT_REC;1

Columns for table BOAT:
Column Name                     Data Type       Domain
-----------                     ---------       ------
PRICE                           REAL            PRICE
DISCOUNT_PRICE                  REAL
 Edit String:   $$$,$$$

SQL> SHOW TABLE(COLUMN) SALARY_HISTORY;
Information for table SALARY_HISTORY
```

(continued on next page)

Example 8–15 (Cont.) Creating Records Originally Defined in DATATRIEVE

```
CDD Pathname:

    DISK1:[CORPORATE.RDB60.CDD61]TRDB$SQL_CASE.SALARY_HISTORY_REC;1

Columns for table SALARY_HISTORY:
Column Name                    Data Type        Domain
-----------                    ---------        ------
SALARY_AMOUNT                  INTEGER(2)       SALARY_AMOUNT
SALARY                         DOUBLE PRECISION

$!
$! Use the Oracle RMU Extract command to display the full table definition
$! including the CASE expression.
$!

$ RMU/EXTRACT/ITEM=TABLE/OPTION=NODICT TESTDB.RDB
-- RMU/EXTRACT for Oracle Rdb
--
--                             Table Definitions
--
--------------------------------------------------------------------------------

SQL> CREATE TABLE BOAT (
cont>    PRICE PRICE,
cont>    DISCOUNT_PRICE
cont>        COMPUTED BY CASE
cont>            WHEN (PRICE < 20000) THEN (PRICE * 0.9)
cont>            WHEN (PRICE < 30000) THEN (PRICE * 0.8)
cont>            WHEN (PRICE < 40000) THEN (PRICE * 0.7)
cont>            ELSE (PRICE * 0.6)
cont>        END
cont>        EDIT STRING IS '$$$,$$$');

SQL> CREATE TABLE SALARY_HISTORY (
cont> SALARY_AMOUNT SALARY_AMOUNT,
cont> SALARY
cont>  COMPUTED BY CASE
cont>    WHEN (SALARY_AMOUNT < 20000.50) THEN (SALARY_AMOUNT * 0.02)
cont>    WHEN (SALARY_AMOUNT < 30000.23) THEN (SALARY_AMOUNT * 0.123)
cont>    WHEN (SALARY_AMOUNT < 40000.67) THEN (SALARY_AMOUNT * 0.349)
cont>    ELSE (SALARY_AMOUNT * 0.4034)
cont>  END);
```

8.2.3.3 Implicit Support for COALESCE and NULLIF

In addition to CASE expressions, SQL Version 6.0 and higher supports COALESCE and NULLIF, two special shorthand forms of a CASE expression. These special shorthand formats are also supported by Oracle CDD/Repository. This allows you to attach to a database by pathname and define expressions using COALESCE or NULLIF.

Note

Oracle CDD/Repository only generates the CASE expression equivalents of NULLIF and COALESCE when creating definitions from the repository.

COALESCE is used to allow inheritance from a series of value expressions. Only the first non-NULL value is inherited:

```
COALESCE(v1, v2, ..., vn)
```

This is equivalent to:

```
CASE
    WHEN v1 is not NULL THEN v1
    WHEN v2 is not NULL THEN v2
    .
    .
    .
    ELSE vn
END
```

NULLIF is used to substitute NULL when the two value expressions are equal.

```
NULLIF(v1,v2)
```

This is equivalent to:

```
CASE
    WHEN v1 = v2 THEN NULL
    ELSE v1
END
```

Use the record-change-clause of the CDO CHANGE RECORD command to add a new field or constraint. Use the DELETE clause to delete an existing field or constraint. See the *Oracle CDD/Repository CDO Reference Manual* and the Oracle CDD/Repository release notes for details on the CDO CHANGE RECORD command syntax.

If you specify the field name (no special keyword) and one of the attributes (field-constr, ALIGNED ON datatype, or NOALIGNED) in the CDO CHANGE RECORD command, then the field's attributes will be changed. If you specify the constraint name and one of the attributes, then the constraint attributes will be changed.

Example 8–16 uses the CDO CHANGE RECORD command syntax to add the telephone number to the PERSON record and to ensure that a value is supplied for all people.

Example 8–16 Using the CDO CHANGE RECORD Command to Add a New Field

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> CHANGE RECORD PERSON.
cont> DEFINE TELEPHONE_NUMBER
cont>    DESCRIPTION IS 'Home Telephone Number'
cont>    CONSTRAINT NO_PHONE_NUMBER
cont>    NOT NULL NOT DEFERRABLE.
cont> END.
cont> END.

CDO> SHOW RECORD PERSON/FULL
Definition of record PERSON
|   Contains field          NAME
|   |   Datatype               text size is 20 characters
|   Contains field          AGE
|   |   Datatype               signed word
|   Contains field          TELEPHONE_NUMBER
|   |   Description            /* Home Telephone Number */
|   |   Based on               TELEPHONE_NUMBER
|   |   |   Datatype               signed longword
|   Constraint NO_PHONE_NUMBER  not null TELEPHONE_NUMBER NOT DEFERRABLE
```

Example 8–17 returns to the PARTS record at the ACME Company, which was described in Example 8–5. Now the ACME Company decides the foreign key on the PARTS record is no longer useful. The PARTS record can be altered by using the CHANGE RECORD command with the DELETE clause specified with the constraint name.

The ACME Company also wants a new not null field, SUPPLIER, to identify the supplier of each part. To accomplish this, ACME uses the DEFINE clause as shown in Example 8–17.

Example 8–17 Using the CHANGE RECORD Command to Delete a Constraint and Add a NOT NULL Field

```
   .
   .
   .
CDO> CHANGE RECORD PARTS
cont>  DELETE CONSTRAINT PART_FRK.
cont>  DEFINE SUPPLIER
cont>       CONSTRAINT NO_SUPPLIER NOT NULL NOT DEFERRABLE.
cont>   END.
cont>  END.

CDO> SHOW RECORD PARTS/FULL
Definition of record PARTS
|   Contains field           PART_NO
|   |   Datatype                 signed word
|   Contains field           PART_ID
|   |   Datatype                 signed longword
|   Contains field           PART_ID_USED_IN
|   |   Based on                 ID_DOM
|   |   |   Datatype                 signed longword
|   Contains field           PART_QUANT
|   |   Datatype                 signed word
|   Contains field           SUPPLIER
|   |   Datatype                 text size is 20 characters
|   Constraint PARTS_PMK  primary key PART_ID NOT DEFERRABLE
|   Constraint PARTS_UNQ  unique PART_NO NOT DEFERRABLE
|   Constraint PART_CST (ANY (P IN PARTS WITH (PART_ID IN PARTS EQ
PART_ID_USED_IN IN P))) NOT DEFERRABLE
|   Constraint NO_SUPPLIER  not null SUPPLIER NOT DEFERRABLE
```

In Example 8–18, Smitti's Grocery Store decides the UPC_CODE, which was defined in Example 8–6, should be a primary key constraint for the PRODUCE record. To accomplish this, Smitti uses the CHANGE RECORD command and specifies the PRIMARY KEY on the local field UPC_CODE.

Smitti also wants to add the STATUS_CODE field to help determine if the item is currently in stock.

Example 8–18 Using the CHANGE RECORD Command to Specify a Primary Key Constraint

```
   .
   .
   .
CDO> CHANGE RECORD PRODUCE
cont> CONSTRAINT U_PKEY PRIMARY KEY UPC_CODE.
cont> DEFINE STATUS_CODE
cont>     DESCRIPTION IS /* in stock status */
cont>     BASED ON STATUS.
cont> END.
cont> END.

CDO> SHOW RECORD PRODUCE/FULL
Definition of record PRODUCE
|   Contains field          UPC_CODE
|   |   Datatype                signed longword
|   Contains field          WEIGHT
|   |   Datatype                signed word
|   Contains field          PRICE
|   |   Datatype                signed longword
|   Contains field          QUANTITY
|   |   Datatype                signed word
|   Contains field          STATUS_CODE
|   |   Description             /* if in stock  */
|   |   Based on                STATUS
|   |   |   Datatype                text size is 1 characters
|   Constraint CDD$NOT_NULL_0096C56941B9497D  not null UPC_CODE DEFERRABLE
|   Constraint WNOTNULL  not null WEIGHT NOT DEFERRABLE
|   Constraint PNOTNULL  not null PRICE DEFERRABLE
|   Constraint QNOTNULL  not null QUANTITY NOT DEFERRABLE
|   Constraint U_PKEY  primary key UPC_CODE NOT DEFERRABLE
```

In Example 8–19, the managers in charge of the EMPLOYEES database decide the ADDRESS record must be changed by deleting the FIRST_NAME field and redefining the CITY field to be based on the NAME_STRING.

Example 8–19 shows how they accomplish this by first deleting the FIRST_ NAME field and deleting the CITY field. Then the CITY field is redefined by specifying the field with new attributes and supplying an end.

Example 8–19 Using the CHANGE RECORD Command to Delete a Field, Then Redefine It

```
   .
   .
   .
CDO> CHANGE RECORD ADDRESS.
cont>   DELETE FIRST_NAME.
cont>   DELETE CITY.
cont>   DEFINE CITY BASED ON NAME_STRING.
cont>   END.
cont> END.

CDO> SHOW RECORD ADDRESS/FULL
Definition of record ADDRESS
|   Contains field           LAST_NAME
|   |   Based on                NAME_STRING
|   |   |   Description             /* standard name  */
|   |   |   Datatype                text size is 30 characters
|   Contains field           STREET
|   |   Datatype                text size is 40 characters
|   Contains field           STATE
|   |   Datatype                text size is 2 characters
|   Contains field           ZIP_CODE
|   |   Datatype                text size is 5 characters
|   Constraint CDD$NOT_NULL_0096C4D0443E0F10  not null ZIP_CODE NOT
DEFERRABLE
|   Contains field           CITY
|   |   Based on                NAME_STRING
|   |   |   Description             /* standard name  */
|   |   |   Datatype                text size is 30 characters
```

8.2.4 Differences Between CDO IN Clause and SQL IN Operator

This section explains the differences between using the CDO IN keyword, which is part of a relation-clause in a record selection expression (RSE), and the SQL IN operator.

The CDO IN keyword in a relation-clause declares context variables for a record stream or loop. For example, the context variable PART_ID specifies a temporary name that identifies the record stream to the product evaluating the clause:

```
PART_ID in PARTS
```

You then use the context variable to refer to fields from that relation. The relation name PARTS specifies the relation from which CDO takes the records in the record stream.

For more information, refer to the description of record selection expressions in the *Oracle CDD/Repository CDO Reference Manual*.

The SQL IN operator compares a value with another value or a collection of values. For example:

```
STATE in ('RI','MA','NY')
```

In SQL, value-expr IN (value-expr1,value-expr2,value-expr3) is the same as the complex predicate, as follows:

```
value-expr = value-expr1
OR
value-expr = value-expr2
OR
value-expr = value-expr3
```

Unlike SQL, CDO does not have an IN operator; therefore, to perform the same operation within a CDO expression, you must specify the complex predicate form. For example:

```
CDO> DEFINE RECORD REC1
cont> CONSTRAINT CONS CHECK ((STATE IN REC1 EQ "RI") OR
                             (STATE IN REC1 EQ "MA") OR
                             (STATE IN REC1 EQ "NY")).
cont> END.
```

8.3 Modifying Repository Definitions and Database Metadata

When you create an Oracle Rdb database, you can store metadata in just the database, or in the database and the repository. There may be times when the database metadata and the definitions in the repository no longer match. When they no longer match, you can update one source using the other.

The following sections describe situations that might cause discrepancies between the repository definitions and the database metadata, and give examples of how to modify them so that they match.

8.3.1 Modifying Repository Definitions Using CDO

You can modify repository definitions by using the CDO DEFINE command to create a new version of an element, or use the CDO CHANGE command to modify an element.

Oracle CDD/Repository lets you store several versions of the same element in much the same way as the OpenVMS operating system stores multiple versions of files. When you store several versions of the same definition, you can specify and use any of these versions. If you specify a definition without a

version number, CDO assigns the highest version number to the definition by default.

You can issue a CDO SHOW USES command to see if an older version is part of the database. For example:

```
CDO> SHOW USES DATE

Owners of SYS$COMMON:[CDDREP]PERS.DATE(1)
|    SYS$COMMON:[CDDREP]PERS.BIRTHDAY(2)    (Type : CDD$RDB_DATABASE)
|     |    via CDD$RDB_DATA_ELEMENT
|    SYS$COMMON:[CDDREP]PERS.BIRTHDAY(1)    (Type : CDD$RDB_DATABASE)
|     |    via CDD$RDB_DATA_ELEMENT
```

Use the CDO CHANGE commands to modify an element created by CDO. If the element is controlled, you can modify only the highest visible version, and you must reserve the element before you can change it.

A controlled element is one that you create within a partition and context, then constrain using the CDO CONSTRAIN command. Using the CONSTRAIN command places the element on the base partition of that context. You cannot modify the controlled versions without using the CDO RESERVE/REPLACE model.

Use the CDO CHANGE command to modify an uncontrolled element created by CDO. The CHANGE command changes the original definition without creating a new version of the definition. See the *Oracle CDD/Repository CDO Reference Manual* for more information on the CDO CHANGE, RESERVE, and REPLACE commands.

```
CDO> SHOW FIELD ID_NUMBER
Definition of field ID_NUMBER
 | Description              'Corporate Standard ABRII'
 | Datatype                 text size is 5 characters
CDO> CHANGE FIELD ID_NUMBER DATATYPE TEXT SIZE IS 6.
CDO> SHOW FIELD ID_NUMBER

 Definition of field ID_NUMBER
 | Description              'Corporate Standard ABRII'
 | Datatype                 text size is 6 characters
```

Because the ID_NUMBER field is changed in the repository, the database and the repository no longer match. Use the SQL INTEGRATE DATABASE statement to alter the database definitions inclusively to match those of the repository.

```
$ SQL
SQL> INTEGRATE DATABASE PATHNAME SYS$COMMON:[CDDREP]PERS.DEPT1
cont> ALTER FILES;
SQL> COMMIT;
```

Section 8.3.2 and Section 8.3.3 describe the SQL INTEGRATE statement in more detail.

8.3.2 Integrating a Single Object

In versions of Oracle CDD/Repository prior to Version 6.1, when you performed an integrate operation, *all* objects in the database supported by Oracle CDD/Repository were compared and updated, as needed, to make the database target match the repository source, or to make the repository target match the database source, depending on the specified direction for the integration.

The SQL INTEGRATE TABLE and INTEGRATE DOMAIN statements, implemented in Version 6.1, allow the following objects to be integrated:

- a single named global field (domain)
- a single named relation (table)
- any related objects, such as collating sequences, other domains, other tables and views, constraints, and indexes that:
  - are referenced by the named global field or relation
  - reference the named global field or relation if it has changed

Restrictions

- You cannot use the INTEGRATE DOMAIN or INTEGRATE TABLE statements to create an object in the database or repository; use these statements to modify an existing definition.
- In Oracle CDD/Repository Version 6.1 and higher, you can update an individual repository field using the SQL INTEGRATE DOMAIN statement. To update an individual record, use the SQL INTEGRATE TABLE statement. You must attach the repository to the database by issuing the ATTACH . . . PATHNAME statement before using the SQL INTEGRATE DOMAIN or SQL INTEGRATE TABLE statements.
- If you delete a definition by issuing an SQL DROP DOMAIN or SQL DROP TABLE statement while attached to the repository by PATHNAME, use the CREATE DOMAIN FROM PATHNAME statement to re-create the metadata. However, if you delete the definition using a CDO DELETE FIELD command, to rebuild the metadata from the Oracle Rdb database you must use the SQL INTEGRATE DATABASE statement or use the Oracle RMU Extract command and manipulate the SQL syntax to redefine the object in CDO.

For information on the SQL INTEGRATE statement syntax, including the syntax for updating an individual repository field using the INTEGRATE DOMAIN statement, or updating a repository record using the INTEGRATE TABLE statement, see the *Oracle Rdb7 SQL Reference Manual*.

8.3.3 Using SQL INTEGRATE to Synchronize the Repository and the Database

This section shows how you can use the SQL INTEGRATE statement to do the following:

- create repository definitions for the first time by using the database metadata as the source
- update repository definitions using the database metadata as the source
- update the database metadata using repository definitions as the source

The following list describes possible combinations of the SQL CREATE DATABASE statement and the SQL ATTACH statement and explains the actions you must take in each situation to update the repository:

- Create a database using the DICTIONARY IS REQUIRED option and the PATHNAME option of the SQL CREATE DATABASE statement. You can subsequently attach to the database by using the PATHNAME option of the SQL ATTACH statement.

  SQL updates the repository and the database with any changes you make during that attachment to the database. You do not need to use the SQL INTEGRATE statement.

  If you attach to the database using the FILENAME option of the SQL ATTACH statement, SQL produces an error message in response to subsequent data definition statements. This is because the database definition specified the DICTIONARY IS REQUIRED option, and the FILENAME option indicates that only the database is updated.

  When you specify the PATHNAME option on the SQL ATTACH statement, SQL updates both the repository and the database.

- Create a database using the DICTIONARY IS NOT REQUIRED (the default) option and the PATHNAME option of the SQL CREATE DATABASE statement. You subsequently attach to the database using the FILENAME option of the SQL ATTACH statement.

SQL stores the initial database definitions in the repository when you specify the PATHNAME option. Because the repository is not required, and because you attach to the database using the FILENAME option, any changes you make to the database metadata during that attachment to the database are entered only into the database, not into the repository.

To update repository definitions inclusively, use the SQL INTEGRATE DATABASE . . . PATHNAME statement with the ALTER DICTIONARY clause. This statement alters the repository definitions so that they are the same as those in the database. However, by altering definitions in the repository, you may affect other repository entities that refer to these definitions.

Example 8–20 is a typical situation where you would use the INTEGRATE DATABASE statement with the ALTER DICTIONARY clause.

- Create a database using the DICTIONARY IS NOT REQUIRED option (the default) of the SQL CREATE DATABASE statement, but *do not* specify the PATHNAME option. You can subsequently attach to the database using the FILENAME option of the SQL ATTACH statement.

  Because you do not specify the PATHNAME option, SQL does not store the original database metadata in the repository. When you specify the FILENAME option, SQL enters any database metadata changes you make during that attachment to only the database, not to the repository.

  To store existing database metadata in the repository for the first time, use the SQL INTEGRATE DATABASE . . . FILENAME statement with the CREATE PATHNAME clause. This statement creates repository definitions using the database as the source.

  Example 8–21 creates a database and stores the metadata in only the database. It then uses the SQL INTEGRATE statement with the CREATE PATHNAME clause to integrate the metadata from the database to the repository.

8.3.4 Logging Information During an Integrate Operation

Use the CDD$INTEGRATE_LOG logical name to display metadata definition information activity during an integrate operation. Define the CDD$INTEGRATE_LOG logical name to be either USER or ALL.

If you define the CDD$INTEGRATE_LOG logical name with a value of USER, Oracle CDD/Repository displays user metadata definitions during the integrate operation. For example:

```
Integrating entity: OF_DOM1
    type:  FIELD   action:  DEFINE   target:  DICTIONARY
```

If the CDD$INTEGRATE_LOG logical name is defined as ALL, Oracle CDD/Repository displays Oracle Rdb system and user metadata definitions during the integrate operation. Oracle Rdb system metadata definitions are any metadata names that begin with RDB$ or RDBVMS$. For example:

```
Integrating entity: RDB$USER_ID
    type:  FIELD   action:  DEFINE   target:  DICTIONARY
```

8.3.5 Modifying Repository Definitions Using SQL

If you create a database using the DICTIONARY IS NOT REQUIRED option and the PATHNAME option, SQL stores the initial definitions in the repository. If you subsequently make changes to the database definitions without using the PATHNAME option, SQL stores those changes only in the database, not in the repository.

If you specify the DICTIONARY IS REQUIRED clause when you create the database, then try to modify metadata using the FILENAME option, you will see an error message indicating that updates are not allowed.

Example 8–20 shows how to integrate the database with the repository. At the beginning of this example, the repository and the database match. The database contains a table called ORDER, and the repository contains a record also called ORDER.

The ORDER table has four fields (FIRST, SECOND, THIRD, and FOURTH) that are based on four domains (FIRST_DOM, SECOND_DOM, THIRD_DOM, and FOURTH_DOM).

Example 8–20 Modifying Repository Definitions Using the INTEGRATE Statement with the ALTER DICTIONARY Clause

```
   .
   .
   .
SQL> CREATE DATABASE FILENAME TEST1
cont>              PATHNAME SYS$COMMON:[CDDREP]TEST1;
SQL> CREATE DOMAIN FIRST_DOM CHAR(4);
SQL> CREATE DOMAIN SECOND_DOM CHAR(4);
SQL> CREATE DOMAIN THIRD_DOM CHAR(4);
SQL> CREATE DOMAIN FOURTH_DOM CHAR(4);
```

(continued on next page)

Example 8–20 (Cont.) Modifying Repository Definitions Using the INTEGRATE Statement with the ALTER DICTIONARY Clause

```
SQL> CREATE TABLE ORDER
cont>    (FIRST FIRST_DOM,
cont>    SECOND SECOND_DOM,
cont>    THIRD THIRD_DOM,
cont>    FOURTH FOURTH_DOM);
SQL> COMMIT;
SQL> DISCONNECT ALL;
```

Attach to the database with the FILENAME clause so the repository is not updated, then use the SQL SHOW TABLE statement to see what fields and domains are part of the table ORDER.

```
SQL> ATTACH 'FILENAME TEST1';
SQL> SHOW TABLE ORDER
Columns for table ORDER:
Column Name                    Data Type        Domain
-----------                    ---------        ------
FIRST                          CHAR(4)          FIRST_DOM
SECOND                         CHAR(4)          SECOND_DOM
THIRD                          CHAR(4)          THIRD_DOM
FOURTH                         CHAR(4)          FOURTH_DOM
   .
   .
   .
```

Create a new domain called FIFTH_DOM. Add a new column to the ORDER table. Name the column FIFTH and base it on the domain FIFTH_DOM:

```
SQL> CREATE DOMAIN FIFTH_DOM CHAR(4);
SQL> ALTER TABLE ORDER ADD FIFTH FIFTH_DOM;

   .
   .
   .
SQL> SHOW TABLE ORDER

Columns for table ORDER:
Column Name                    Data Type        Domain
-----------                    ---------        ------
FIRST                          CHAR(4)          FIRST_DOM
SECOND                         CHAR(4)          SECOND_DOM
THIRD                          CHAR(4)          THIRD_DOM
FOURTH                         CHAR(4)          FOURTH_DOM
FIFTH                          CHAR(4)          FIFTH_DOM
```

(continued on next page)

Example 8–20 (Cont.) Modifying Repository Definitions Using the INTEGRATE Statement with the ALTER DICTIONARY Clause

```
   .
   .
   .
SQL> COMMIT;
SQL> EXIT
```

Invoke CDO, and check the record ORDER. The field FIFTH is not part of the record ORDER. Now the database and the repository no longer match.

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> SHOW RECORD ORDER FROM DATABASE TEST1
Definition of the record ORDER
|   Contains field              FIRST
|   Contains field              SECOND
|   Contains field              THIRD
|   Contains field              FOURTH

CDO> EXIT
```

Enter SQL again. Use the SQL INTEGRATE DATABASE statement with the ALTER DICTIONARY clause to resolve this situation. Alter the repository to add the field FIFTH and the domain FIFTH_DOM to the repository using the database as the source.

Note

You must integrate the entire database; you cannot use the SQL INTEGRATE DOMAIN statement to create FIFTH_DOM in the repository. Creating individual objects using INTEGRATE is not supported.

The SQL INTEGRATE DATABASE statement implicitly attaches to the database, using the default authorization identifier.

```
$ SQL
SQL> INTEGRATE DATABASE PATHNAME TEST1 ALTER DICTIONARY;
SQL> COMMIT;
SQL> EXIT
```

Enter CDO again. Use the CDO SHOW RECORD command to see that the field FIFTH is now part of the record ORDER. The repository and the database match again.

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> SHOW RECORD ORDER FROM DATABASE TEST1
Definition of the record ORDER
|   Contains field              FIRST
|   Contains field              SECOND
|   Contains field              THIRD
|   Contains field              FOURTH
|   Contains field              FIFTH

CDO> EXIT
```

8.3.6 Creating Repository Definitions Using SQL

If you create a database using the DICTIONARY IS NOT REQUIRED option and you do not specify the PATHNAME option, SQL does not store the original definitions in the repository.

Example 8–21 shows how to store existing database file definitions in the repository for the first time.

This example creates a database called DOGS. It then creates a table for the breed of dog, POODLES. The columns in the table are types of poodles. The example shows how to use the SQL INTEGRATE statement with the CREATE PATHNAME clause to integrate the metadata from the database into the repository.

Example 8–21 Storing Existing Definitions in the Repository

```
$ SQL
SQL> CREATE DATABASE DOGS;
SQL> CREATE TABLE POODLES (STANDARD CHAR(10),
cont> MINIATURE CHAR(10), TOY CHAR(10));

   .
   .
   .
SQL> SHOW TABLE POODLES

Columns for table POODLES:
Column Name                    Data Type        Domain
-----------                    ---------        ------
STANDARD                       CHAR(10)
MINIATURE                      CHAR(10)
TOY                            CHAR(10)

   .
   .
   .

SQL> COMMIT;
SQL> EXIT
```

The CDO DIRECTORY command shows that the database DOGS is not part of the repository:

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> DIR
Directory SYS$COMMON:[CDDREP]DEPT32.FIELDMAN
FIRST(1)                                          FIELD
FOURTH(1)                                         FIELD
ORDER(1)                                          RECORD
PERSONNEL(1)                                      CDD$DATABASE
SECOND(1)                                         FIELD
TEST1(1)                                          CDD$DATABASE
THIRD(1)                                          FIELD
CDO> EXIT
```

(continued on next page)

Example 8–21 (Cont.) Storing Existing Definitions in the Repository

Enter SQL again. Use the SQL INTEGRATE DATABASE FILENAME statement to integrate the database DOGS into the repository:

```
$ SQL
SQL> INTEGRATE DATABASE FILENAME DISK01:[FIELDMAN.KENNELS]DOGS
cont> CREATE PATHNAME SYS$COMMON:[CDDREP]DEPT32.FIELDMAN.DOGS;
SQL> COMMIT;
SQL> EXIT
```

Enter CDO again, and check to see that the database DOGS has been integrated into the repository:

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> DIR
DOGS(1)                                          CDD$DATABASE
FIRST(1)                                         FIELD
FOURTH(1)                                        FIELD
ORDER(1)                                         RECORD
SECOND(1)                                        FIELD
TEST1(1)                                         CDD$DATABASE
THIRD(1)                                         FIELD
PERSONNEL(1)                                     CDD$DATABASE
   .
   .
   .
```

You can also use the CDO SHOW USED_BY command to see that the record (table) POODLES and the fields (columns) STANDARD, MINIATURE, and TOY are part of the database DOGS.

```
   .
   .
   .
CDO> SHOW USED_BY/FULL DOGS

Members of SYS$COMMON:[CDDREP]DEPT32.FIELDMAN.DOGS(1)
|   SYS$COMMON:[CDDREP]DEPT32.FIELDMAN]DOGS    (Type : CDD$FILE)
|   |   via CDD$DATABASE_FILE
|   DOGS                              (Type : CDD$RDB_DATABASE)
|   |   via CDD$DATABASE_SCHEMA

   .
   .
   .
```

(continued on next page)

Example 8–21 (Cont.) Storing Existing Definitions in the Repository

```
|   SYS$COMMON:[CDDREP]CDD$RDB_SYSTEM_METADATA.RDB$CDD_NAME(1)(Type : FIELD)
|   |   |   |   via CDD$DATA_AGGREGATE_CONTAINS
|   |   POODLES                          (Type : RECORD)
|   |   |   via CDD$RDB_DATA_AGGREGATE
|   |   |   STANDARD                         (Type : FIELD)
|   |   |   |   via CDD$DATA_AGGREGATE_CONTAINS
|   |   |   |   SQL$10CHR                        (Type : FIELD)
|   |   |   |   |   via CDD$DATA_ELEMENT_BASED_ON
|   |   |   MINIATURE                        (Type : FIELD)
|   |   |   |   via CDD$DATA_AGGREGATE_CONTAINS
|   |   |   |   SQL$10CHR                        (Type : FIELD)
|   |   |   |   |   via CDD$DATA_ELEMENT_BASED_ON
|   |   |   TOY                              (Type : FIELD)
|   |   |   |   via CDD$DATA_AGGREGATE_CONTAINS
|   |   |   |   SQL$10CHR                        (Type : FIELD)
|   |   |   |   |   via CDD$DATA_ELEMENT_BASED_ON
CDO> EXIT
```

8.3.7 Making a Database Table Shareable in the Repository

To make a database table shareable in a repository, use the CDO ENTER command. For example:

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> ENTER RECORD POODLES FROM DATABASE DOGS
CDO>
```

The CDO ENTER command lets you assign a directory name to an element that exists in the repository, so that you can display and share the definition in the database. You must be sure that your current default directory does not contain a directory name that is the same as the processing name of the element to be shared. If it does, the CDO ENTER command will fail.

8.3.8 Updating the Database File Using the Repository Definitions

The previous sections described how the SQL INTEGRATE statement uses the database metadata to create and update repository definitions. Sometimes, you need to perform the operation in the opposite direction; that is, use repository definitions to update the database.

If you have created a table or domain with the SQL CREATE TABLE . . . FROM statement or the SQL CREATE DOMAIN . . . FROM statement, and then changed the definitions in the repository, you can use the SQL INTEGRATE DATABASE . . . ALTER FILES statement to update the database metadata to match the repository definitions.

The SQL INTEGRATE . . . ALTER FILES statement has no effect on metadata that was not created using the SQL CREATE TABLE . . . FROM or the SQL CREATE DOMAIN . . . FROM statement.

To update the database metadata so that it matches the repository definitions, use the SQL INTEGRATE DATABASE . . . PATHNAME . . . ALTER FILES statement.

Example 8–22 shows a typical situation where you would use the SQL INTEGRATE statement with the ALTER FILES clause. In the example, fields and records are defined in the repository. Then a table is created in SQL based on the repository definitions. The repository definitions are subsequently changed, and the database metadata and the repository definitions no longer match.

The SQL INTEGRATE statement resolves this situation by altering the database using the repository definitions as the source.

Example 8–22 Updating the Database to Match Repository Definitions

```
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]DEPT32.FIELDMAN
CDO> DEFINE FIELD FIRST DATATYPE IS TEXT 4.
CDO> DEFINE FIELD SECOND DATATYPE IS TEXT 4.

CDO> DEFINE RECORD ORDER.
CDO> FIRST.
CDO> SECOND.
CDO> END ORDER RECORD.
CDO> EXIT
```

Enter SQL, and create the database TEST1. Then create a table, ORDER, in the TEST1 database. Use the table just created in the repository, as follows:

```
$ SQL
SQL> CREATE DATABASE FILENAME TEST1 PATHNAME TEST1;
SQL> CREATE TABLE FROM SYS$COMMON:[CDDREP]DEPT32.FIELDMAN.ORDER;
SQL>
```

Use the SQL SHOW TABLE statement to see information about the table:

```
   .
   .
   .
SQL> SHOW TABLE ORDER

Columns for table TEST1.ORDER:
Column Name                    Data Type        Domain
-----------                    ---------        ------
FIRST                          CHAR(4)          FIRST
SECOND                         CHAR(4)          SECOND

   .
   .
   .
SQL> COMMIT;
SQL> EXIT
```

Enter CDO again. Verify which fields are in the record ORDER:

(continued on next page)

Example 8–22 (Cont.) Updating the Database to Match Repository Definitions

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> SHOW RECORD ORDER
Definition of record ORDER
|   Contains field          FIRST
|   Contains field          SECOND
```

Define the fields THIRD and FOURTH and add them to the record ORDER. Now the repository definitions and the database metadata no longer match:

```
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]DEPT32.FIELDMAN
CDO> DEFINE FIELD THIRD DATATYPE IS TEXT 4.
CDO> DEFINE FIELD FOURTH DATATYPE IS TEXT 4.
CDO> CHANGE RECORD ORDER.
CDO> DEFINE THIRD.
CDO> END.
CDO> DEFINE FOURTH.
CDO> END.
CDO> END ORDER RECORD.

CDO> SHOW RECORD ORDER
Definition of record ORDER
|   Contains field          FIRST
|   Contains field          SECOND
|   Contains field          THIRD
|   Contains field          FOURTH

CDO> EXIT
```

Enter SQL again. Use the SQL INTEGRATE statement to alter the database, making it the same as the repository. The SQL INTEGRATE statement implicitly attaches to the database:

```
$ SQL
SQL> INTEGRATE DATABASE PATHNAME SYS$COMMON:[CDDREP]DEPT32.FIELDMAN.TEST1
cont> ALTER FILES;
   .
   .
   .
```

(continued on next page)

Example 8–22 (Cont.) Updating the Database to Match Repository Definitions

The SQL SHOW TABLE statement indicates that the table ORDER has changed. SQL has integrated the THIRD and FOURTH domains into the database:

```
   .
   .
   .
SQL> SHOW TABLE ORDER

Columns for table ORDER:
Column Name                    Data Type        Domain
-----------                    ---------        ------
FIRST                          CHAR(4)          FIRST
SECOND                         CHAR(4)          SECOND
THIRD                          CHAR(4)          THIRD
FOURTH                         CHAR(4)          FOURTH

SQL> COMMIT;
SQL> EXIT
```

Table 8–1 summarizes the SQL statements and steps you use to modify repository definitions and database metadata using CREATE DATABASE and INTEGRATE DATABASE statements.

Table 8–1 Modify Repository Definitions and Database Metadata

To:	Use SQL Statements:
Update repository from database source	1. CREATE DATABASE . . . PATHNAME . . . DICTIONARY [IS/IS NOT] REQUIRED 2. ATTACH . . . PATHNAME
Update database only[1]	1. CREATE DATABASE . . . PATHNAME . . . DICTIONARY IS NOT REQUIRED 2. ATTACH . . . FILENAME
Update existing database from repository source	INTEGRATE DATABASE . . . PATHNAME . . . ALTER FILES
Create a new repository definition from existing database source	INTEGRATE DATABASE . . . FILENAME . . . CREATE PATHNAME
Update repository from existing database source	INTEGRATE DATABASE . . . PATHNAME . . . ALTER DICTIONARY
Update a modified field in the repository from the database	1. SQL CREATE DATABASE . . . DICTIONARY IS NOT REQUIRED 2. SQL CREATE DOMAIN 3. SQL ALTER DOMAIN 4. SQL INTEGRATE DOMAIN . . . ALTER DICTIONARY

[1]If you issue the SQL CREATE DATABASE . . . PATHNAME statement with the DICTIONARY IS REQUIRED clause and later use the ATTACH . . . FILENAME statement, an error will occur when you attempt to update the database because DICTIONARY IS REQUIRED was specified.

(continued on next page)

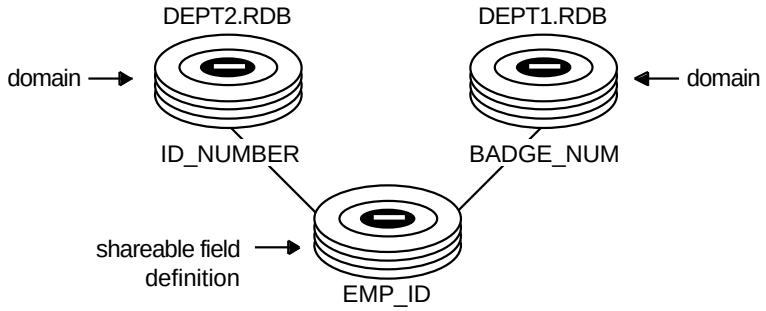
Table 8–1 (Cont.) Modify Repository Definitions and Database Metadata

To:	Use SQL Statements:
Update a modified record in the repository from the database	1. SQL CREATE DATABASE . . . DICTIONARY IS NOT REQUIRED 2. SQL CREATE TABLE 3. SQL ALTER TABLE 4. SQL INTEGRATE TABLE . . . ALTER DICTIONARY
Update a modified field in the database from the repository	1. SQL CREATE DATABASE . . . DICTIONARY IS REQUIRED 2. CDO DEFINE FIELD 3. CDO CHANGE FIELD 4. SQL INTEGRATE DOMAIN . . . ALTER FILES
Update a modified record in the database from the repository	1. SQL CREATE DATABASE . . . DICTIONARY IS REQUIRED 2. CDO DEFINE RECORD 3. CDO CHANGE RECORD 4. SQL INTEGRATE TABLE . . . ALTER FILES

8.4 Deleting Definitions Using SQL and CDO

You cannot use SQL to delete shared fields (domains) that reside in repositories. For example, Figure 8–2 shows two domains in two separate Oracle Rdb databases that depend on one Oracle CDD/Repository shareable field definition.

Figure 8–2 Shareable Fields in Oracle CDD/Repository



Because the EMP_ID field is used by both ID_NUMBER and BADGE_NUM, do not use SQL to delete EMP_ID. However, when you attach to a database by pathname and issue a DROP DOMAIN statement, you delete that database's connection between that field and the record in the repository. The field still exists in the repository for other databases that share the field. For example:

```
SQL> ATTACH 'PATHNAME DEPT2';
SQL> DROP DOMAIN ID_NUMBER;
SQL> COMMIT;
SQL> DISCONNECT ALL;
SQL> ATTACH 'PATHNAME DEPT1';
SQL> DROP DOMAIN BADGE_NUM;
SQL> COMMIT;
SQL> DISCONNECT ALL;
```

From SQL, update the DEPT1 and the DEPT2 databases from the repository, as follows:

```
$ SQL
SQL> INTEGRATE DATABASE PATHNAME SYS$COMMON:[CDDREP]PERS.DEPT1 ALTER FILES;
SQL> COMMIT;
SQL> DISCONNECT ALL;
SQL> INTEGRATE DATABASE PATHNAME SYS$COMMON:[CDDREP]PERS.DEPT2 ALTER FILES;
SQL> COMMIT;
SQL> DISCONNECT ALL;
```

Note

The INTEGRATE statement starts a transaction that can be completed by an SQL COMMIT statement, or canceled by an SQL ROLLBACK statement.

You can delete a field if it is not used by another element. Use the CDO SHOW USES command to see the uses of the field STANDARD_DATE, as in Example 8–23.

Example 8–23 Using the CDO SHOW USES Command to Track the Uses of a Field

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> SHOW USES STANDARD_DATE
Owners of SYS$COMMON:[CDDREP]PERS.STANDARD_DATE(1)
|    SYS$COMMON:[CDDREP]PERS.START_DATE(1)  (Type : FIELD)
|     |    via CDD$DATA_ELEMENT_BASED_ON
|    SYS$COMMON:[CDDREP]PERS.END_DATE(1)    (Type : FIELD)
|     |    via CDD$DATA_ELEMENT_BASED_ON
|    SYS$COMMON:[CDDREP]PERS.BIRTHDAY(1)    (Type : FIELD)
|     |    via CDD$DATA_ELEMENT_BASED_ON
```

Because the STANDARD_DATE field is used by other entities, you cannot delete it until you delete all the definitions that use it.

```
CDO> DELETE FIELD STANDARD_DATE.
%CDO-E-NOTDELETED, entity STANDARD_DATE not deleted
-CDD-E-INUSE, entity is the member of a relationship; it cannot be deleted
```

Example 8–24 shows that a shared definition also cannot be deleted from the repository using SQL. In this example, the repository record EMPLOYEES contains the field ADDRESS_DATA_2, and the database table EMPLOYEES contains the domain ADDRESS_DATA_2.

Example 8–24 Repository Definition Not Removed After SQL DROP COLUMN

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]PERS

CDO> SHOW RECORD EMPLOYEES
Definition of record EMPLOYEES
 |    Description             'CORPORATE EMPLOYEE INFORMATION'
 |    Contains field          EMPLOYEE_ID
 |    Contains field          LAST_NAME
 |    Contains field          FIRST_NAME
 |    Contains field          MIDDLE_INITIAL
 |    Contains field          ADDRESS_DATA_1
 |    Contains field          ADDRESS_DATA_2
 |    Contains field          CITY
 |    Contains field          STATE
 |    Contains field          POSTAL_CODE
 |    Contains field          BIRTHDAY
 |    Contains field          SEX
 |    Contains field          STATUS_CODE

$ SQL
SQL> ATTACH 'FILENAME DEPT1';
SQL> SHOW TABLE EMPLOYEES

CDD Pathname:   SYS$COMMON:[CDDREP]PERS.EMPLOYEES(1)

Comment on table EMPLOYEES:
CORPORATE EMPLOYEE INFORMATION
```

(continued on next page)

Example 8–24 (Cont.) Repository Definition Not Removed After SQL DROP COLUMN

```
Columns for table EMPLOYEES:
Column Name                    Data Type       Domain
-----------                    ---------       ------
EMPLOYEE_ID                    CHAR(5)         EMPLOYEE_ID
LAST_NAME                      CHAR(14)        LAST_NAME
FIRST_NAME                     CHAR(10)        FIRST_NAME
MIDDLE_INITIAL                 CHAR(1)         MIDDLE_INITIAL
ADDRESS_DATA_1                 CHAR(25)        ADDRESS_DATA_1
ADDRESS_DATA_2                 CHAR(25)        ADDRESS_DATA_2
CITY                           CHAR(20)        CITY
STATE                          CHAR(2)         STATE
POSTAL_CODE                    CHAR(9)         POSTAL_CODE
BIRTHDAY                       DATE            BIRTHDAY
SEX                            CHAR(1)         SEX
STATUS_CODE                    CHAR(1)         STATUS_CODE
```

If you enter the SQL ALTER TABLE statement and use the DROP COLUMN clause, the definition of the field ADDRESS_DATA_2 is not removed from the repository.

```
   .
   .
   .
SQL> ALTER TABLE EMPLOYEES
cont>      DROP COLUMN ADDRESS_DATA_2;
SQL> COMMIT;
SQL> EXIT;
```

(continued on next page)

Example 8–24 (Cont.) Repository Definition Not Removed After SQL DROP COLUMN

```
$ REPOSITORY OPERATOR
   .
   .
   .
CDO> SET DEFAULT SYS$COMMON:[CDDREP]PERS
CDO> SHOW RECORD EMPLOYEES
Definition of record EMPLOYEES
 |    Description              'CORPORATE EMPLOYEE INFORMATION'
 |    Contains field           EMPLOYEE_ID
 |    Contains field           LAST_NAME
 |    Contains field           FIRST_NAME
 |    Contains field           MIDDLE_INITIAL
 |    Contains field           ADDRESS_DATA_1
 |    Contains field           ADDRESS_DATA_2
 |    Contains field           CITY
 |    Contains field           STATE
 |    Contains field           POSTAL_CODE
 |    Contains field           BIRTHDAY
 |    Contains field           SEX
 |    Contains field           STATUS_CODE
```

However, the link between the database and the field is deleted.

```
   .
   .
   .
CDO> EXIT
$ SQL
SQL> SHOW TABLE EMPLOYEES

CDD Pathname:   SYS$COMMON:[CDDREP]PERS.EMPLOYEES(1)

Comment on table EMPLOYEES:
CORPORATE EMPLOYEE INFORMATION
```

(continued on next page)

Example 8–24 (Cont.) Repository Definition Not Removed After SQL DROP COLUMN

```
Columns for table EMPLOYEES:
Column Name                    Data Type       Domain
-----------                    ---------       ------
EMPLOYEE_ID                    CHAR(5)         EMPLOYEE_ID
LAST_NAME                      CHAR(14)        LAST_NAME
FIRST_NAME                     CHAR(10)        FIRST_NAME
MIDDLE_INITIAL                 CHAR(1)         MIDDLE_INITIAL
ADDRESS_DATA_1                 CHAR(25)        ADDRESS_DATA_1
CITY                           CHAR(20)        CITY
STATE                          CHAR(2)         STATE
POSTAL_CODE                    CHAR(9)         POSTAL_CODE
BIRTHDAY                       DATE            BIRTHDAY
SEX                            CHAR(1)         SEX
STATUS_CODE                    CHAR(1)         STATUS_CODE
```

To delete repository definitions, use the SQL DROP PATHNAME statement. The SQL DROP PATHNAME statement does not delete the database metadata; it deletes only the repository definitions. Shared definitions are not deleted.

8.5 Deciding Whether to Use Oracle CDD/Repository

If your application consists of many Oracle Rdb databases, you should decide whether or not to use Oracle CDD/Repository. Consider the following trade-offs when making the decision:

- Field and record definitions in the repository are stored in a generic format that can be interpreted appropriately by different products.
- You cannot rename a repository definition as you include it in a database.
- Shareable definitions created in CDO can be changed from either CDO or SQL.
- Whenever database metadata is changed without the corresponding repository change, inconsistencies can exist between copies of the definitions in the repository and database. If you use SQL and invoke a database using the PATHNAME clause, you will receive a message that the repository definitions have changed. For example:

```
      .
      .
      .
   SQL> ATTACH 'PATHNAME DEPT1';

   %SQL-I-DIC_DB_CHG1, A dictionary definition used by schema
     SYS$COMMON:[CDDREP]PERS.DEPT1(1) has changed
   -SQL-I-DIC_DB_CHG2, Use the INTEGRATE statement to resolve any
     differences between the dictionary and the database
   %CDD-I-MESS, entity has messages
      .
      .
      .
```

If your database design requires that all metadata updates be maintained in the repository, use the SQL DICTIONARY IS REQUIRED clause when you issue the SQL CREATE DATABASE statement.

Then, if you use SQL and attach to a database using the FILENAME clause and you attempt to manipulate database definitions, you will receive the following messages:

```
   .
   .
   .
SQL> CREATE DOMAIN EMPLOYEE_NAME IS CHAR (10);
%RDB-E-NO_META_UPDATE, metadata update failed
-RDMS-F-CDDISREQD, CDD required for metadata updates is not
   being maintained
SQL> ROLLBACK;
SQL> DISCONNECT ALL;
```

Storing database definitions in the repository provides a central source of shareable field and record definitions. To avoid data definition inconsistencies, attach to the database using the Oracle CDD/Repository pathname, with the SQL ATTACH . . . PATHNAME statement. By doing this, the database definitions are available to other products that use the repository.

In Oracle CDD/Repository and Oracle Rdb environments, you can use either CDO or SQL to define fields (domains) and records (tables). In some cases, however, using CDO has certain advantages because records and fields can be defined, shared, tracked, and controlled independently of a particular database. You must define fields and records in CDO or use the CDO ENTER command for them to be shareable, as described in Section 8.3.7.

You should consider implementing database definitions through Oracle CDD/Repository if you need to track and share definitions.

Not all database designs benefit equally from a centrally placed common repository. You need not create definitions in the repository if the definitions are local to one application and are likely to remain local, or if requirements for control of definitions are met by existing SQL statements.

8.6 Restoring a Database That Uses Shareable Repository Definitions

You can copy an Oracle Rdb database using the following methods:

- Oracle RMU Backup and Restore commands—for regular maintenance backups or disaster recovery
- SQL IMPORT and EXPORT statements—for unloading and reloading databases, restructuring physical database files, or migrating a database such as Oracle Rdb to another database
- Oracle RMU Extract command—re-creates an empty database; provides syntax for the database structure

8.6.1 Backing Up and Restoring Databases

Do not use the OpenVMS BACKUP procedure to back up your database. Use the Oracle RMU Backup command.

The Oracle RMU Backup command creates a backup copy of an Oracle Rdb database and places it in an .RBF 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 Oracle RMU Restore command to restore the database to the condition it was in when you backed it up.

The Oracle RMU Restore command re-creates all the relationships between the database structure and shared definitions individually defined in CDO.

You can rename or move the files that comprise a database by using the Oracle RMU Backup and Restore command combination. To move a multifile Oracle Rdb database, you *must* use the Oracle RMU Backup and Restore commands or the SQL EXPORT and IMPORT statements.

Do not use the DCL COPY command with a multifile database; otherwise, the resulting database will be corrupt and unusable.

For more information about how to perform an Oracle RMU Backup operation, see the Oracle Rdb documentation.

8.6.2 Restructuring and Reloading Databases

You can use the SQL EXPORT and IMPORT statements to perform the following tasks:

- migrate a database from one system to another
- change parameters, such as a 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 using SQL IMPORT and EXPORT statements, see the *Oracle Rdb7 SQL Reference Manual*.

8.7 Deleting Databases

You can delete Oracle Rdb databases with the SQL DROP DATABASE statement, which has the following two clauses:

- PATHNAME—Deletes the database and the repository definition for the database
- FILENAME—Deletes only the database

Caution

Use the SQL DROP DATABASE statement with care. You cannot use the SQL ROLLBACK statement to cancel an SQL DROP DATABASE statement.

The SQL DROP DATABASE statement deletes the database root file (.RDB), its snapshot files (.SNP), and any storage area files (.RDA), which include all data and all definitions. The PATHNAME clause deletes the repository database definitions from the repository in addition to these files.

Issue the SQL DROP DATABASE statement before attaching to the database or after issuing the DISCONNECT ALL statement, because you cannot delete a database when there are active users for that database.

When you use the PATHNAME clause, you can specify either one of the following:

- a full repository pathname, such as CDD$USER:[JONES.PERS]DEPT1
- a relative repository pathname, such as DEPT1

If you use a relative pathname, be sure that the current repository default directory is defined to be all of the path segments preceding the relative pathname.

Example 8–25 shows how to delete database DEPT1 using the SQL DROP DATABASE . . . PATHNAME statement. In this example, the database, DEPT1, is in the current Oracle CDD/Repository directory, CDD$USER:[JONES.PERS]:

Example 8–25 Deleting a Database Using the SQL DROP DATABASE PATHNAME Statement

```
$ REPOSITORY OPERATOR
CDO> SHOW DATABASE

Definition of database  DEPT1
|   database uses RDB database DEPT1
|   database in file DEPT1
|   |   fully qualified file CDD$USER:[JONES.PERS]DEPT1.RDB;
Definition of database  FOO1
|   database uses RDB database FOO1
|   database in file FOO1
|   |   fully qualified file CDD$USER:[JONES.PERS]FOO1.RDB;
Definition of database  TEST1
|   database uses RDB database TEST1
|   database in file TEST1
|   |   fully qualified file CDD$USER:[JONES.PERS]TEST1.RDB;
CDO> EXIT

$ SQL
SQL> ATTACH 'PATHNAME DEPT1';
SQL> SHOW DATABASES
Default alias:
    repository pathname is CDD$USER:[JONES.PERS]DEPT1
    Oracle Rdb database in file dept1
SQL> DISCONNECT ALL;
SQL> DROP DATABASE PATHNAME DEPT1;
SQL> ATTACH 'PATHNAME DEPT1';
%CDD-E-NOT_A_DB, dept1, is not the name of a database in the repository
```

Because you have successfully deleted DEPT1, SQL shows no databases in the current Oracle CDD/Repository directory when you issue the SQL ATTACH statement. Using CDO, you can also see that the database has been deleted from the repository.

```
   .
   .
   .
SQL> $ RUN SYS$SYSTEM:CDO
CDO> DIRECTORY

 Directory CDD$USER:[JONES.PERS]
   .
   .
   .
FOO1(1)                                          CDD$DATABASE
   .
   .
   .
TEST1(1)                                         CDD$DATABASE
   .
   .
   .
CDO> EXIT
```

When you delete a database using the SQL DROP DATABASE . . . FILENAME statement, the database is deleted, but the database definition remains in the repository.

You can use either a full or partial file specification when deleting the database using the DROP DATABASE . . . FILENAME statement.

Example 8–26 shows how to delete the database FOO1, which was created with the DICTIONARY IS NOT REQUIRED clause.

Example 8–26 Deleting a Database Using the SQL DROP DATABASE FILENAME Statement

```
   .
   .
   .
SQL> DROP DATABASE FILENAME 'FOO1';
SQL> SHOW DATABASES
%SQL-F-ERRATTDEF, Could not use database file specified by SQL$DATABASE
-RDB-E-BAD_DB_FORMAT, SQL$DATABASE does not reference a database known to Rdb
-RMS-E-FNF, file not found
SQL> ATTACH 'FILENAME FOO1';
%SQL-F-ERRATTDEC, Error attaching to database foo1
-RDB-E-BAD_DB_FORMAT, foo1 does not reference a database known to Rdb
-RMS-E-FNF, file not found
```

Although the database has been deleted, the definition remains in the repository:

(continued on next page)

Example 8–26 (Cont.) Deleting a Database Using the SQL DROP DATABASE FILENAME Statement

```
SQL> $ RUN SYS$SYSTEM:CDO
CDO> DIRECTORY

 Directory CDD$USER:[JONES.PERS]
   .
   .
   .
FOO1(1)                                          CDD$DATABASE
   .
   .
   .
TEST1(1)                                         CDD$DATABASE
   .
   .
   .
```

You can delete the CDD$DATABASE definition for FOO1 by using the CDO DELETE GENERIC command, as follows:

```
   .
   .
   .
CDO> DELETE GENERIC CDD$DATABASE FOO1.
CDO> EXIT
SQL> EXIT
$ DIRECTORY *.RDB

Directory CDD$USER:[JONES.PERS]

TEST1.RDB;1           1097
```

9

Managing RMS Files with CDO

This chapter explains the relationship between logical database file definitions and physical database files for OpenVMS Record Management Services (RMS) databases. It also describes how to perform the following tasks:

- create, modify, and delete RMS database file definitions
- create and use physical RMS database files
- use file definition, area definition, and key definition (for indexed files) properties
- create repository elements that can be used by programs and applications
- display database file definitions

9.1 Overview

If you use RMS to create and access files and to process records, you can create Oracle CDD/Repository definitions for these elements. Oracle CDD/Repository can be used for the following:

- storing RMS database definitions for reference in programs
- standardizing the definitions within an RMS database
- tracking the definitions used by an RMS database
- generating messages to notify you when the definitions used by an RMS database are changed
- protecting database definitions
- creating RMS database definitions that can be used by Oracle Rally applications

Each RMS file is an array of records of one type. Each record is a set of fields. An RMS database is an RMS file that is built according to a template contained in CDD$RMS_DATABASE. The CDD$RMS_DATABASE element type describes the logical definition of the file, which includes both a record definition and a file definition. Elements of type CDD$RMS_DATABASE are the RMS database definitions.

Table 9–1 lists the CDO commands that you can use to manipulate RMS files. These commands are described in more detail in the following sections.

Table 9–1 CDO Commands for Manipulating RMS Files

Command	Function
CHANGE DATABASE	Modifies the physical RMS database file; the ON clause moves the database; the AUDIT clause modifies history entry; the DESCRIPTION clause modifies information documenting the database.
DEFINE DATABASE	Creates the physical database file using an RMS database file definition, and puts the file definition into the repository.
DEFINE RMS_DATABASE	Creates a logical database file definition that includes descriptions of the RMS file and data structures within the file. Does not create the physical file.
DELETE DATABASE	Deletes the physical RMS database file (CDD$FILE) from disk and its CDD$DATABASE element from the repository.
DELETE RMS_DATABASE	Deletes the logical RMS database file definition from the repository.
SHOW DATABASE	Displays the file definition for the database and the field and record definitions used by the database.
SHOW NOTICES	Displays any NOTICES attached to the specified definition to indicate changes in that definition or a related one. See Chapter 5 for more information.
SHOW RMS_DATABASE	Displays the logical database file definition, its properties, and the record used by the database.
SHOW USED_BY	Displays the children of the database element.
SHOW USES	Displays the parents of the database element.

9.2 Creating RMS Database Definitions

Before you can define a physical RMS database file, you must create a logical RMS database file definition that describes the database in your repository. The physical database file is based on the RMS database definition. Use the CDO DEFINE RMS_DATABASE command to create a logical RMS database element of type CDD$RMS_DATABASE in the repository.

Arguments to the CDO DEFINE RMS_DATABASE command let you specify the properties and structure you want for the database file definition. All databases based on this definition share the specified properties and structure. The logical RMS database consists of only one record and file definition. However, one logical RMS database file definition can be owned by many physical RMS databases, where each physical RMS database owns a different CDD$DATABASE element.

The following naming conventions apply to the CDO DEFINE RMS_ DATABASE command:

- The record that you specify must exist in the repository.
- The name of the record that you specify cannot be the same as the name of the database.
- If you do not specify a full pathname for the database, Oracle CDD/Repository creates the RMS database element in your default directory.

The following example creates a logical RMS database file definition that specifies two keys:

```
CDO> DEFINE RMS_DATABASE EQUIPMENT_RMS.
cont>    RECORD PART_REC.
cont>    FILE_DEFINITION
cont>         MAX_RECORD_SIZE 41
cont>         ORGANIZATION INDEXED.
cont>    KEYS.
cont>      KEY 0
cont>         DUPLICATES
cont>         SEGMENT MANUFACTURER IN TYPE IN PART_REC
cont>         SEGMENT MODEL IN TYPE IN PART_REC.
cont>      KEY 1
cont>         CHANGES
cont>         DUPLICATES
cont>         SEGMENT MODEL IN TYPE IN PART_REC.
cont>    END KEYS.
cont> END.
```

Specifying File Properties

Properties allow you to fine-tune your RMS applications. For example, you can use a file property to preallocate the file where performance is important and the size of the file is predetermined.

Table 9–2 shows the three types of properties used in specifying a file. See the *Oracle CDD/Repository CDO Reference Manual* for more information on the syntax for each property.

Table 9–2 DEFINE RMS_DATABASE Properties

Property	Specifies
FILE_DEFINITION	Defines file characteristics and certain run-time options.
AREAS	Controls file or area space allocation on disk devices to optimize performance.
KEYS	Defines the characteristics of one or more keys in an indexed file.

9.3 Creating Physical Database Files

After creating your RMS database definition, you can use one or more CDO DEFINE DATABASE commands to create physical RMS database files based on that RMS database definition. A physical database file appears in the repository as an element of type CDD$DATABASE.

Example 9–1 shows how to build an RMS database in the repository to maintain a list of current employees in three departments.

The first step is to define the database elements that make up an employee record.

Example 9–1 Defining Field and Record Elements

```
CDO> DEFINE FIELD FIRST_NAME
cont> DATATYPE IS TEXT
cont> SIZE IS 20.
```

(continued on next page)

Example 9–1 (Cont.) Defining Field and Record Elements

```
CDO> DEFINE FIELD LAST_NAME
cont> DATATYPE IS TEXT
cont> SIZE IS 30.
CDO> DEFINE FIELD EMP_ID
cont> DATATYPE IS UNSIGNED LONGWORD.
CDO> DEFINE RECORD EMPLOYEE_REC.
cont> LAST_NAME.
cont> FIRST_NAME.
CONT> EMP_ID.
cont> END.
CDO>
```

After the elements are defined in the repository, create the logical RMS database file definition EMPLOYEE_STORAGE.

```
CDO> DEFINE RMS_DATABASE EMPLOYEE_STORAGE.
cont>    RECORD EMPLOYEE_REC.
cont>    FILE_DEFINITION
cont>      ALLOCATION 200
cont>      FILE_PROCESSING_OPTIONS CONTIGUOUS
cont>      ORGANIZATION INDEXED.
cont>    AREAS.
cont>      AREA 0
cont>         ALLOCATE 10
cont>         BUCKET_SIZE 5
cont>         EXTENSION 7.
cont>      AREA 1
cont>         ALLOCATE 15
cont>         BUCKET_SIZE 3
cont>         EXTENSION 11.
cont>      AREA 2
cont>         ALLOCATE 20
cont>         BUCKET_SIZE 7.
cont>    END.
```

(continued on next page)

Example 9–1 (Cont.) Defining Field and Record Elements

```
cont>    KEYS.
cont>      KEY 0
cont>         DUPLICATES
cont>         SEGMENT LAST_NAME IN EMPLOYEE_REC.
cont>      KEY 1
cont>         CHANGES
cont>         SEGMENT EMP_ID IN EMPLOYEE_REC.
cont>    END.
cont> END.
CDO>
```

Now, use the CDO DEFINE DATABASE command to create a physical RMS database file based on the logical RMS database file definition. You must reference an RMS database definition in the USING clause. The CDO DEFINE DATABASE command can be used to define only RMS databases, not Oracle Rdb databases. In this example, the EMPLOYEE_STORAGE definition is used.

Note

Do not create the physical RMS files in your anchor directory. If you create the RMS files in your anchor directory, Oracle CDD/Repository will delete them when you delete your repository.

```
CDO> DEFINE DATABASE SALES_FILE USING
EMPLOYEE_STORAGE ON DISK$07:[SUPPORT]EMP.DAT.
%CDO-I-FILECRE, file DISK$07:[SUPPORT]EMP.DAT;1 created
```

Use the CDO DIRECTORY command to check that the RMS database definition appears in your repository as a CDD$RMS_DATABASE, and the physical RMS file appears as a CDD$DATABASE element.

```
CDO> DIRECTORY
 Directory SYS$COMMON:[CDDREP]
EMPLOYEE_REC(1)                                 RECORD
EMPLOYEE_STORAGE(1)                             CDD$RMS_DATABASE
EMP_ID(1)                                       FIELD
FIRST_NAME(1)                                   FIELD
LAST_NAME(1)                                    FIELD
SALES_FILE(1)                                   CDD$DATABASE
CDO> EXIT
```

From the DCL prompt, use the following commands to check that you created the physical RMS database file in the OpenVMS directory:

```
$ SET DEFAULT DISK$07:[SUPPORT]
$ DIR/FULL EMP.DAT

Directory DISK$07:[SUPPORT]

EMP.DAT(1)                    File ID:  (10657,17,0)
Size:         3006/3006       Owner:    [VDD,USER01]
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: 48, Extend: 7, Maximum bucket size: 7
                    Global buffer count: 0, No version limit
Record format:      Variable length, maximum 54 bytes
Record attributes:  None
RMS attributes:     None
Journaling enabled: None
File protection:    System:RW, Owner:RWED, Group:RWED, World:RWED
Access Cntrl List:  None

Total of 1 file, 48/48 blocks.
```

9.4 Displaying Databases and Database Definitions

The CDO SHOW commands listed in Table 9–1 let you check the contents of the database and display the database definition.

The CDO SHOW DATABASE/BRIEF command, which is the default, displays the database name and description, record name, file organization properties, and fully qualified pathname of an RMS database. For example:

```
CDO> SHOW DATABASE/BRIEF SALES_FILE
Definition of database  SALES_FILE
|   database uses RMS database EMPLOYEE_STORAGE
|   |   database uses record EMPLOYEE_REC
|   |   file definition
|   |   |   file allocation 200
|   |   |   file contiguous
|   |   |   file organization index sequential
|   |   |   key 0
|   |   |   |   key duplicates
|   |   |   |   segment LAST_NAME IN EMPLOYEE_REC
|   |   |   key 1
|   |   |   |   key changes
|   |   |   |   segment EMP_ID IN EMPLOYEE_REC
|   |   |   storage area 0
|   |   |   |   area allocation 10
|   |   |   |   area bucket size 5
|   |   |   |   area extension 7
|   |   |   storage area 1
|   |   |   |   area allocation 15
|   |   |   |   area bucket size 3
|   |   |   |   area extension 11
|   |   |   storage area 2
|   |   |   |   area allocation 20
|   |   |   |   area bucket size 7
|   database in file DISK$07:[SUPPORT]EMP.DAT
|   |   fully qualified file DISK$07:[SUPPORT]EMP.DAT;
CDO>
```

The CDO SHOW RMS_DATABASE command displays the RMS database file definition, as in the following example:

```
CDO> SHOW RMS_DATABASE EMPLOYEE_STORAGE
Definition of RMS database  EMPLOYEE_STORAGE
|   database uses record EMPLOYEE_REC
|   file definition
|   |   file allocation 200
|   |   file contiguous
|   |   file organization index sequential
|   |   key 0
|   |   |   key duplicates
|   |   |   segment LAST_NAME IN EMPLOYEE_REC
|   |   key 1
|   |   |   key changes
|   |   |   segment EMP_ID IN EMPLOYEE_REC
|   |   storage area 0
|   |   |   area allocation 10
|   |   |   area bucket size 5
|   |   |   area extension 7
|   |   storage area 1
|   |   |   area allocation 15
|   |   |   area bucket size 3
|   |   |   area extension 11
|   |   storage area 2
|   |   |   area allocation 20
|   |   |   area bucket size 7
CDO>
```

If you have more than one RMS database definition in your repository and you do not specify an RMS database name in the CDO SHOW RMS_DATABASE command, CDO displays all of the RMS database definitions in your repository.

For detailed information on the CDO SHOW commands, see the *Oracle CDD/Repository CDO Reference Manual*.

9.5 Modifying Databases with CDO CHANGE DATABASE

You can use the CDO CHANGE DATABASE command to modify your RMS database in the following ways:

- move the database using the ON clause
- add comments using the DESCRIPTION clause
- modify or add an entry to the history list with the AUDIT clause

When you want to move a physical RMS database file to a new location, use the CDO CHANGE DATABASE command with the ON clause.

The following example moves the physical RMS database SALES_FILE to a new location on DISK$03:[MIS.MARKETING]EMP.DAT and updates the pointer to the file in the repository.

```
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>
```

The following example shows the CDO CHANGE DATABASE command used with the DESCRIPTION clause to modify comments:

```
CDO> CHANGE DATABASE SALES_FILE(1)
cont> DESCRIPTION IS /* Moved to new disk after reorganization. */
cont>.
CDO>
```

9.6 Deleting Databases and Database Definitions

You can delete either the physical RMS database file or the element that represents it in the repository by using the CDO DELETE DATABASE command. Oracle CDD/Repository prompts you to confirm the request before it deletes the database.

In the following example, the DELETE DATABASE command deletes the CDD$DATABASE element SALES_FILE(1) from the repository, and the RMS file from the OpenVMS directory.

```
CDO> DELETE DATABASE SALES_FILE(1).
deleting file DISK$07:[SUPPORT]EMP.DAT;, proceed? [Y/N] (N)Y
CDO>
```

To delete the logical RMS database element, first delete all the databases that use the RMS database definition. Then, use the CDO DELETE RMS_DATABASE command to delete the RMS database definition in the repository. Unlike the CDO DELETE DATABASE command, the CDO DELETE RMS_DATABASE command deletes the definition without prompting you for confirmation. In the following example, the CDO DELETE RMS_DATABASE command is used to delete the CDD$RMS_DATABASE element from the repository.

```
CDO> DELETE RMS_DATABASE EMPLOYEE_STORAGE.
CDO>
```

If you want notification that the RMS database definition was deleted, use the /LOG qualifier with the DELETE RMS_DATABASE command, as follows:

```
CDO> DELETE RMS_DATABASE /LOG EMPLOYEE_STORAGE.
%CDO-I-ENTDEL, entity SYS$COMMON:[CDDREP]EMPLOYEE_STORAGE(2) deleted
```

9.7 Using RMS File Definitions in Programs and Applications

This section describes how you can use information from databases in an Oracle Rally and DATATRIEVE application.

Oracle Rally is a forms-based application generator. By default, the RALLY CREATE command places an element in the repository for each application file (AFILE) and each Data Source Definition (DSD). These elements, RALLY$APPLICATION and RALLY$DATA_SOURCE_DEFINITION, are proxy objects.

Proxy objects contain information about and point to objects, but the actual Oracle Rally object is created and maintained in the AFILE. Because the repository keeps track of the location and change history for the corresponding Oracle Rally object, you can use CDO to display the location and creation history of Oracle Rally applications.

Oracle Rally builds DSDs based on the Oracle CDD/Repository pathname to an RMS file or a record definition. You can use DSDs to connect the information in the RMS database to form or report fields, or to variables in an AFILE.

Example 9–2 shows how an Oracle Rally and DATATRIEVE user defines an RMS database definition, then creates the RMS database for YACHTS.

After the database is defined, you can give the pathname of YACHTS_DB as the data source pathname in the RMS DSD creation form.

The example uses the sample definitions from the DATATRIEVE YACHTS demo. The YACHTS definitions are currently in DMU format and cannot be used by CDO. Therefore, the definitions must be converted to CDO format and entered individually so that they can be referenced in the CDO file and database definitions.

Example 9–2 Using Definitions in an Oracle Rally Application

```
   .
   .
   .
CDO> CONVERT CDD$TOP.DTR$LIB.DEMO.YACHT YACHT_REC(1)
CDO> ENTER RECORD TYPE FROM RECORD YACHT_REC
CDO> ENTER RECORD SPECIFICATIONS FROM RECORD YACHT_REC
CDO> ENTER FIELD MANUFACTURER FROM RECORD TYPE
CDO> ENTER FIELD MODEL FROM RECORD TYPE
```

Next, define the RMS database element using the YACHTS_REC record definition.

(continued on next page)

Example 9–2 (Cont.) Using Definitions in an Oracle Rally Application

```
CDO> DEFINE RMS_DATABASE YACHTS_RMS.
cont>    RECORD YACHT_REC.
cont>    FILE_DEFINITION
cont>      MAX_RECORD_SIZE 41
cont>      ORGANIZATION INDEXED.
cont>    KEYS.
cont>      KEY 0
cont>        DUPLICATES
cont>        SEGMENT MANUFACTURER IN TYPE IN YACHT_REC
cont>        SEGMENT MODEL IN TYPE IN YACHT_REC.
cont>      KEY 1
cont>        CHANGES
cont>        DUPLICATES
cont>        SEGMENT MODEL IN TYPE IN YACHT_REC.
cont>    END KEYS.
cont> END.
```

Use the logical RMS database element YACHTS_RMS to create the physical RMS database element YACHTS_DB in the YACHT.DAT file. Your current OpenVMS directory will be used unless you enter the full file specification for a different directory.

```
CDO> DEFINE DATABASE YACHTS_DB USING YACHTS_RMS ON YACHT.DAT.
```

Index

A

B

C

D

L

M

N

O

P

Q

R

S

T

U

V

W