Oracle® CDD/Repository

CDO Reference Manual

Release 7.0.1 for OpenVMS

August 1999

Part No. A70149-01

ORACLE®

Oracle CDD/Repository CDO Reference Manual, Release 7.0.1 for OpenVMS

Part No. A70149-01

Contents

Part II CDO Parameters

2 Field and Record Properties

Figures

Tables

Send Us Your Comments

Oracle CDD/Repository CDO Reference Manual, Release 7.0.1 for OpenVMS
Part No. A70149-01

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.

- 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). You can send comments to us in the following ways:

- E-mail — nedc_doc@us.oracle.com
- FAX — 603-897-3316 Attn: Oracle CDD/Repository Documentation
- Postal service:
  Oracle Corporation
  Oracle CDD/Repository Documentation
  One Oracle Drive
  Nashua, NH 03062-2698
  USA

If you would like a reply, please include your name and contact information.

If you have problems with the software, please contact your local Oracle Support Center.

Preface

This manual provides reference material for the Common Dictionary Operator (CDO) utility. It describes the syntax and semantics of all CDO commands.

Intended Audience

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

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

The audience does *not* include users who access CDD/Repository through the DMU utilities. These users can consult Figure 1 for appropriate manuals.

Document Structure

This manual is organized in two parts as follows:

- Part I CDO Commands
  - Chapter 1 provides syntax diagrams and rules for CDO commands.
- Part II CDO Parameters
  - Chapter 2 provides syntax diagrams and rules for CDO field and record properties.

- Chapter 3 provides syntax diagrams and rules for CDO RMS database properties.
- Chapter 4 provides syntax diagrams and rules for CDO expressions.
- Chapter 5 provides rules for CDO edit strings.
- Appendix A provides a mapping of keywords for CDO DEFINE_RMS_DATABASE command.
- Appendix B provides explanations of CDO OpenVMS logical names.

Associated Documents

See the CDD/Repository Documentation Chart for more information on associated documents and reading paths.

See online help for a glossary of defined terms.

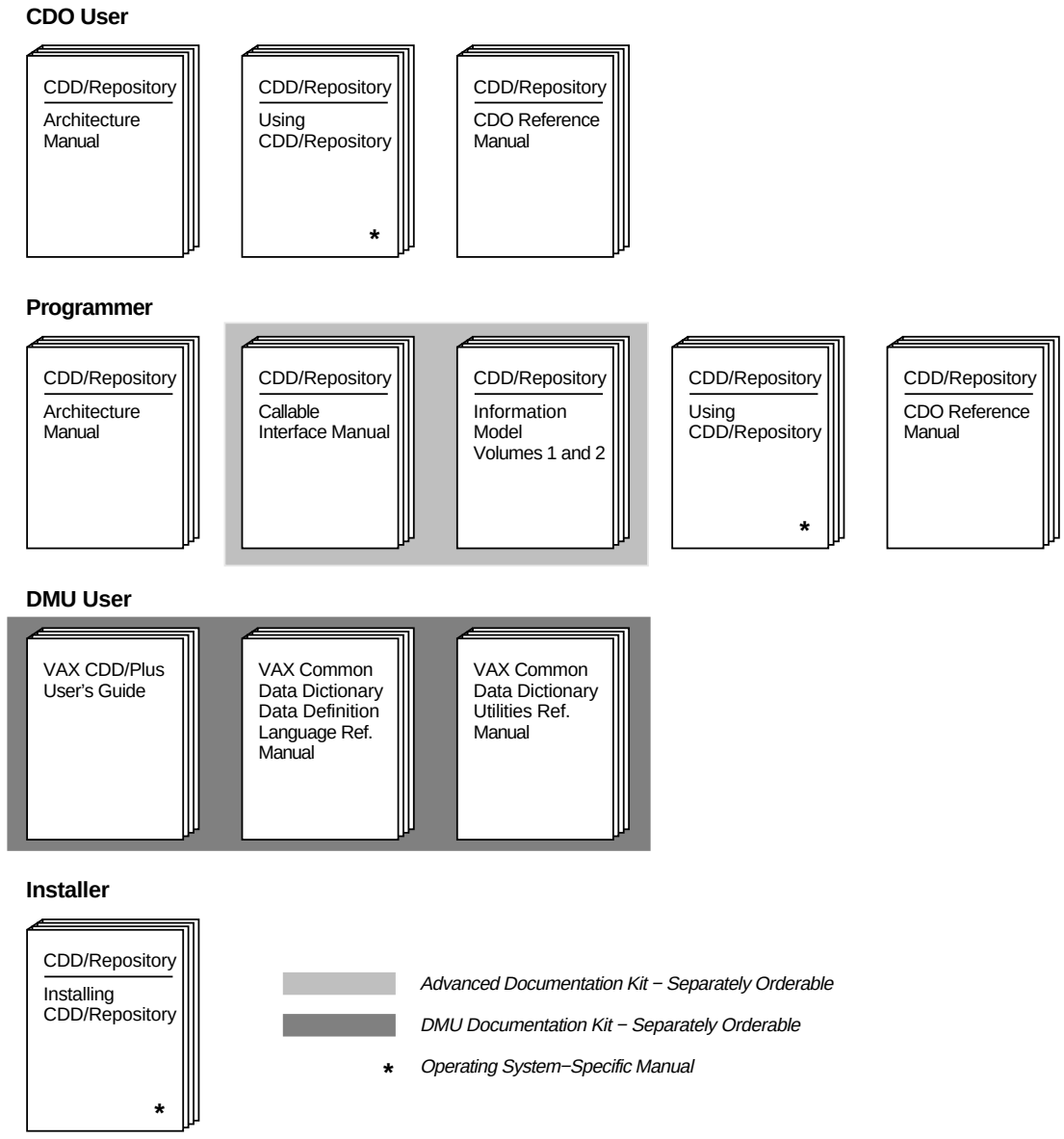
Within this manual, the title *CDD/Repository Information Model* refers to both *CDD/Repository Information Model Volume I* and *CDD/Repository Information Model Volume II*.

References to Products

The Oracle CDD/Repository documentation set to which this manual belongs often refers to the following products by their abbreviated names:

- In this manual, Oracle Rdb refers to Oracle Rdb for OpenVMS.
- OpenVMS means both the OpenVMS Alpha and OpenVMS VAX operating systems.
- The SQL interface to Oracle Rdb is referred to as SQL. This interface is the Oracle Rdb implementation of the SQL standard ANSI X3.135-1992, ISO 9075:1992, commonly referred to as the ANSI/ISO SQL standard or SQL92.
- COBOL means both the DIGITAL VAX COBOL and DIGITAL DEC COBOL language products.

Figure 1 CDD/Repository Documentation Chart



Conventions Used in This Document

Table 1 shows the conventions used in this manual:

Table 1 Documentation Conventions

Convention	Description
Ctrl/*x*	Ctrl/*x* indicates that you hold down the Ctrl key while you press another key or mouse button (indicated here by *x*).
{ }	In format descriptions, braces indicate required elements. You must choose one of the elements.
[ ]	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 indicates an item that can be repeated.
. . .	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 indicates complete titles of manuals.
UPPERCASE	Words in uppercase indicate a command, the name of a file, the name of a file protection code, or an abbreviation for a system privilege.
lowercase	In format descriptions, words in lowercase indicate parameters or arguments to be specified by the user.
Underlined text	In format descriptions, underlined keywords of a command are required. Keywords that are not underlined are optional.
$	A dollar sign ($) represents the OpenVMS DCL system prompt.
Δ	A delta symbol indicates a single space.

Part I

CDO Commands

This part describes the syntax and semantics of the CDO commands.

1
Command Descriptions

CDO commands allow you to define, modify, delete, display, and manipulate some elements in the repository. These elements are contexts, collections, partitions, and data definitions.

CDO command descriptions do not define terms and concepts described in the *Oracle CDD/Repository Architecture Manual* and the *Using Oracle CDD/Repository on OpenVMS Systems*.

@ (At Sign) Command

Format

@file-spec

Parameters

file-spec
Specifies the CDO command file to execute. File-spec can be a fully qualified path name, a relative path name, or a logical name. The default file type is .CDO.

Description

The @ (at sign) command reads and executes the CDO commands contained in the specified file as if you had entered these commands at the terminal. The file can contain any valid CDO commands, including other @ commands.

By default, CDO does not echo commands and comments in your file to the standard output location. You can override this default by including the SET VERIFY command as the first command in your file.

By default, CDO exits the file when it encounters an error. You can override this default by including the ON command in your file.

When CDO executes an EXIT command in the file, or reaches the end of the file, control returns to the command stream that invoked the file. That command stream can be the terminal or a previous file containing CDO commands. You can issue the @ command at the CDO prompt (CDO>). After the CDO commands execute, the CDO prompt returns.

You can issue the @ command as a foreign command at the system level. You can append the @ command to the REPOSITORY OPERATOR command. You can also include the REPOSITORY OPERATOR @ command in an OpenVMS command procedure.

To execute a CDO command procedure with a default file type of .CDO, you do not need to specify the file type.

If the file type for a CDO command procedure is not .CDO, you must specify the file type to execute the command procedure.

After the CDO commands execute, the system prompt returns. If you intend to use your file as an initialization file, you need not issue the @ command. Instead, name your file CDO$INIT.CDO and place it in the directory from which you invoke CDO. CDO then automatically executes this file at the start of each CDO session.

You can also define CDO$INIT as a logical name specifying a device, directory, and file name. If you use such a logical name, the file does not need to be in your default directory when you invoke CDO.

Examples

1.
```
$ DEFINE CDO$INIT SYS$LOGIN:CDO$INIT.CDO
$ SET VERIFY
$ SET DEFAULT USER$DISK:[BOB.DICT]
$ SHOW DEFAULT
CDO> DIRECTORY
```

   In this example, the CDO$INIT.CDO initialization file sets your default repository directory. Oracle CDD/Repository automatically executes the initialization file when you invoke CDO from the OpenVMS directory that contains it.

2.
```
CDO>  @START
CDO> SET DEFAULT USER$DISK:[BOB.DICT]
CDO> DIRECTORY
Directory USER$DISK:[BOB.DICT]
CDDPLUS                          DIRECTORY
CDO>
```

   The START.CDO command procedure in this example places you in the [BOB.DICT] directory, then lists the definitions in that directory. The SET VERIFY command in the previous example instructs CDO to display each subsequent command on the terminal screen before execution.

3.
```
CDO> @EMPLOYEES.PROCEDURE
```

   In this example, the @ (at sign) command executes the CDO commands in the EMPLOYEES.PROCEDURE command procedure.

4.
```
CDO>  @CDDNODE::SYS$DISK:[SMITH.REP]CHANGE.PROCEDURE
```

   In this example, the file specification incorporates a fully qualified path name and a user-supplied file type. The @ (at sign) command executes the CHANGE.PROCEDURE file.

5.
```
CDO> @START
```

   In this example, the file specification incorporates a file name and the default file type (.CDO). The @ (at sign) command executes the START.CDO file.

6.
```
$ DEFINE CDO$INIT SYS$LOGIN:CDO$INIT.CDO
$ TYPE SYS$LOGIN:CDO$INIT.CDO
$ SET VERIFY
$ SET DEFAULT device:[CDDPLUS]MYDIR
$ SHOW DEFAULT
$ REPOSITORY OPERATOR
```

   In this example, when CDO is invoked, SYS$LOGIN:CDO$INIT.CDO is executed immediately before the CDO prompt is displayed.

ATTACH Command

Format

ATTACH process-name

Parameters

process-name
Specifies the process to which control passes. The process must be an existing process. If the process name contains blanks, lowercase characters, or other special characters, enclose the name in double quotation marks (" ").

Description

The ATTACH command passes control from the current process to a parent process or a subprocess.

Examples

1.
```
CDO>  ATTACH JIM SMITH
$
```

In this example, the ATTACH command passes control from the current parent process at the CDO prompt to the JIM SMITH subprocess at the system prompt.

2.
```
CDO>  ATTACH "Jim Smith"
$
```

In this example, the ATTACH command passes control from the current parent process at the CDO prompt to the Jim Smith process at the system prompt. The process name is entered in lowercase characters, which requires double quotation marks.

3.
```
CDO> SPAWN
$ SHOW DEFAULT
  USER1:[SMITH]
$ ATTACH Smith
CDO> SHOW DEFAULT
  USER1:[SMITH.REP]
CDO> ATTACH Smith_1
$SHOW DEFAULT
%DCL-S-RETURNED, control returned to process SMITH_1
```

In this example, the SPAWN command creates a subprocess, and the ATTACH commands pass control back and forth between the spawned

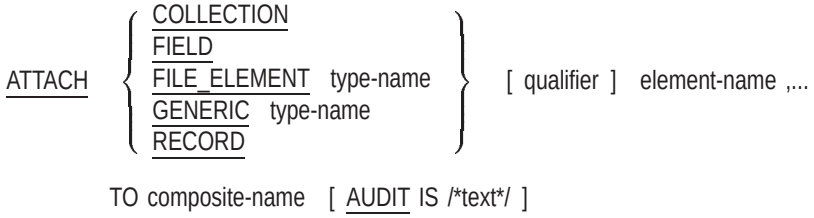
subprocess and the parent process.

4.
```
CDO> SPAWN RUN SYS$SYSTEM:SQL$
SQL>
SQL> COMMIT
SQL> $ATTACH SMITH
CDO>
CDO> ATTACH SMITH_2
%DCL-S-RETURNED, control returned to process SMITH_2
SQL>
```

In this example, the SPAWN command creates a subprocess to invoke SQL and a secondary subprocess that runs SQL. When you are in CDO and want to reattach to your SQL subprocess, you can avoid subprocess quotas by attaching to the secondary subprocess.

ATTACH TO COMPOSITE Command

Format



Parameters

type-name
Specifies the type of file or generic element to which you are attaching.

element-name
Specifies the element to which you are attaching. You can substitute an asterisk (*) wildcard character for this parameter.

composite-name
Specifies the collection, field, record, file, or generic element to which you are attaching.

text
Adds information to the history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_ set of the session to DEC_KANJI.

Qualifiers

/LOG
/NOLOG (default)
Specifies whether CDO displays text identifying each element as the element is attached.

ATTACH TO COMPOSITE Command

Description

The ATTACH TO COMPOSITE command attaches a controlled element to the composite you specify. The element then becomes a child of the composite to which you are attaching.

Before you issue the ATTACH TO COMPOSITE command, you must have set a context and reserved a composite. The SHOW CONTEXT and SHOW RESERVATIONS commands indicate whether these conditions exist.

You can use the ATTACH TO COMPOSITE command in conjunction with the DEFINE, RESERVE, REPLACE, and DETACH commands to link collections in collection hierarchies. See the DEFINE COLLECTION command for an example of a collection hierarchy.

You can also use the DETACH FROM COMPOSITE and ATTACH TO COMPOSITE commands to move between lines of descent. See the *Oracle CDD/Repository Architecture Manual* for more information on lines of descent.

Examples

1.
```
CDO>  DEFINE PARTITION FIRST_QUARTER AUTOPURGE.
CDO>  DEFINE CONTEXT SALES
cont>   BASE_PARTITION FIRST_QUARTER.
CDO>  SET CONTEXT SALES
CDO>  DEFINE COLLECTION SALES_EACH_PRODUCT.
CDO>  CONSTRAIN FIELD PART_NUMBER
CDO>  RESERVE COLLECTION SALES_EACH_PRODUCT
CDO>  ATTACH FIELD PART_NUMBER TO SALES_EACH_PRODUCT
CDO>  REPLACE COLLECTION SALES_EACH_PRODUCT
```

In this example, the ATTACH TO COMPOSITE command attaches the PART_NUMBER field to the SALES_EACH_PRODUCT collection.

2.
```
CDO>  RESERVE COLLECTION EMPLOYEE_RECORDS
CDO>  DETACH FIELD FIRST_NAME(2:BRANCH:2) FROM EMPLOYEE_RECORDS
CDO>  ATTACH FIELD FIRST_NAME(2) TO EMPLOYEE_RECORDS
CDO>  REPLACE COLLECTION EMPLOYEE_RECORDS
```

In this example, the ATTACH TO COMPOSITE command attaches a version in the main line of descent, FIRST_NAME(2), to the EMPLOYEE_RECORDS collection. This allows you to create further versions in the main line, instead of in the branch line where you had been working.

CHANGE COLLECTION Command

Format

```
CHANGE  COLLECTION  collection-name

        [ DESCRIPTION IS /*text*/ ]
        [ NODESCRIPTION           ]   [ AUDIT IS /*text*/ ] .
```

Parameters

collection-name
Specifies the collection you are modifying.

text
Modifies information. Within the DESCRIPTION clause, this is information documenting the collection; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

Description

The CHANGE COLLECTION command modifies a collection by performing a change in place. CDO changes the values you specify, and other values remain the same.

Before you can issue the CHANGE COLLECTION command, you must issue the RESERVE COLLECTION command to reserve the collection. The SHOW COLLECTION or SHOW RESERVATIONS command indicates whether a condition is reserved. The RESERVE command creates a new version of the element.

Since a collection is a controlled element, CDO freezes previous versions and allows you to modify only the highest visible version.

Examples

1.
```
CDO> SET CONTEXT
CDO> DEFINE COLLECTION REGIONAL_SALES
cont> DESCRIPTION IS "COLLECTION IS REGION_5".
CDO>  RESERVE COLLECTION REGIONAL_SALES
CDO>  CHANGE COLLECTION REGIONAL_SALES
cont>   DESCRIPTION IS "COLLECTION DIRECTORY IS WEST_COAST".
CDO>  REPLACE COLLECTION REGIONAL_SALES
```

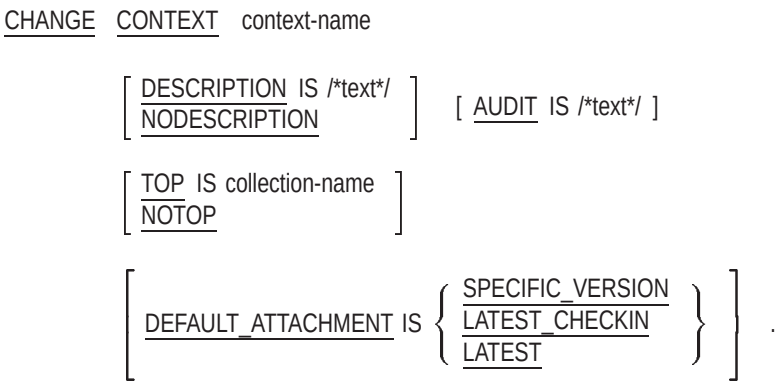
   In this example, the CHANGE COLLECTION command modifies the description clause of the REGIONAL_SALES collection.

2.
```
CDO>  RESERVE COLLECTION COMPILER_C
CDO>  CHANGE COLLECTION COMPILER_C
cont>   NODESCRIPTION
cont>   AUDIT IS "PHASE REVIEW".
CDO>  REPLACE COLLECTION COMPILER_C
```

   In this example, the CHANGE COLLECTION command removes the description clause and adds audit text.

CHANGE CONTEXT Command

Format



Parameters

context-name
Specifies the context you are modifying.

text
Modifies information. Within the DESCRIPTION clause, this is information documenting the context; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command and set the character_set of the session to DEC_KANJI.

collection-name
Specifies a new collection as the top collection for the context.

Description

The CHANGE CONTEXT command modifies a context by performing a change in place. CDO changes the values you specify, and other values remain the same.

Because a context is a nonversioned element, CDO does not accept a branch designation or a version number in the context name.

The TOP clause redefines the top collection property for the context. An error occurs if you attempt to redefine the top collection while you have any element reserved to your context. The SHOW CONTEXT or SHOW RESERVATIONS command indicates whether this condition exists.

The NOTOP keyword sets the top collection property to a null value.

The DEFAULT_ATTACHMENT clause redefines the default attachment property for the context. This property refers to the default behavior that occurs when you issue the UPDATE command. Choose one of the following keywords:

DEFAULT_ATTACHMENT Keyword	Behavior
SPECIFIC_VERSION	Does not detach the currently attached version.
LATEST_CHECKIN	Detaches the currently attached version and attaches the most recently checked in version.
LATEST	Detaches the currently attached version and attaches the lastest version, whether it is checked in or is a ghost. The LATEST keyword is the default.

Examples

```
CDO>  CHANGE CONTEXT DEVELOPMENT_CONTEXT
cont>   DESCRIPTION IS "ARCHIVING THIS CONTEXT"
cont>   "VERSION 5.0 DEVELOPMENT COMPLETED"
cont>   NOTOP
cont>   DEFAULT_ATTACHMENT IS SPECIFIC_VERSION.
```

In this example, the CHANGE CONTEXT command modifies the DEFAULT_ ATTACHMENT clause, the description text, and the TOP clause in the DEVELOPMENT_CONTEXT context.

CHANGE DATABASE Command

Format

```
CHANGE  DATABASE  rms-database-name

        [ DESCRIPTION IS /*text*/ ]      [ AUDIT IS /*text*/ ]
        [ NODESCRIPTION           ]

        [ ON file-name ] .
```

Parameters

rms-database-name
Specifies the physical RMS database you are modifying.

text
Modifies information. Within the DESCRIPTION clause, this is information documenting the database; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

file-name
Specifies a new disk location for the physical RMS database. This OpenVMS file name is a character string having from 1 to 1024 characters.

Description

The CHANGE DATABASE command modifies a physical RMS database element by performing a change in place. CDO changes the values you specify, and other values remain the same.

If an RMS database element is controlled, CDO freezes previous versions and allows you to modify only the highest visible version. If an RMS database is uncontrolled, CDO modifies the highest version unless you specify another version number.

If an RMS database element is controlled, you must reserve the database before you can issue the CHANGE DATABASE command. The SHOW DATABASE or SHOW RESERVATIONS command indicates whether this condition exists.

The ON clause moves a physical RMS database to a new location on disk. When you specify the ON clause, CDO issues a notice asking you to confirm that you want to move the database. This notice cannot be suppressed.

If the CHANGE DATABASE command succeeds, Oracle CDD/Repository moves the physical file on disk and updates the pointer to the physical file in the repository. If the CHANGE DATABASE command fails, Oracle CDD/Repository does not move the database.

Examples

1.
```
CDO>  CHANGE DATABASE DISG_FILE
cont>   DESCRIPTION "INFORMATION ON DIS SECTION EMPLOYEES"
cont>   "PERSONNEL CONTACT IS JIM SMITH"
cont>   AUDIT "JIM SMITH ACCEPTS THIS ACCOUNT 06/30/90".
```

   In this example, the CHANGE DATABASE command modifies the description clause and adds an AUDIT clause to the DISG_FILE database.

2.
```
CDO>  CHANGE DATABASE DISG_FILE
cont>   ON DISK1:[SMITH.PERSONNEL]EMP.DAT.
```

   In this example, the CHANGE DATABASE command moves the DISG_ FILE database to a new location.

CHANGE FIELD Command

Format

```
CHANGE  FIELD  field-name

        [ DESCRIPTION IS /*text*/ ]
        [ NODESCRIPTION           ]   [ AUDIT IS /*text*/ ]

        [ field-property   ]
        [ NOfield-property ]  ... .
```

Parameters

field-name
Specifies the field element you are modifying.

text
Modifies information. Within the DESCRIPTION clause, this is information documenting the field; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

field-property
Changes the value of an existing property, or adds a new property, in the field element. You specify removal with the NO keyword. See Chapter 2 for the field properties CDO provides.

Description

The CHANGE FIELD command modifies a field element by performing a change in place. CDO changes the values you specify, and other values remain the same.

If a field element is controlled, CDO freezes previous versions and allows you to modify only the highest visible version. If a field element is uncontrolled, CDO modifies the highest version unless you specify another version number.

If a field element is controlled, you must reserve the field before you can issue the CHANGE FIELD command. The SHOW FIELD or SHOW RESERVATIONS command indicates whether this condition exists.

When you change a field element that an Oracle Rdb database uses, you may need to integrate the database with the repository. CDO automatically sends a notice with the name of the database when this is necessary.

Examples

1.
```
CDO>  CHANGE FIELD POSTAL_CODE
cont>    DESCRIPTION "A 5 DIGIT POSTAL_CODE: NOTE AUDIT"
cont>    AUDIT "CHANGED TO 9 DIGIT POSTAL_CODE 6/30/90".
```

   In this example, the CHANGE FIELD command modifies the description clause and adds an audit clause to the POSTAL_CODE field element.

2.
```
CDO>  CHANGE FIELD TOTAL
cont>    NOINITIAL_VALUE.
```

   In this example, the CHANGE FIELD command removes the INITIAL_ VALUE field property with the NO keyword.

CHANGE FILE_ELEMENT Command

Format

```
CHANGE  FILE_ELEMENT  type-name  element-name

    [ DESCRIPTION IS /*text*/ ]    [ AUDIT IS /*text*/ ]
    [ NODESCRIPTION           ]

    [ property-name IS { n             } ]
    [                  { quoted-string } ]  ... .
    [ NOproperty-name                    ]

END [ FILE_ELEMENT element-name ] [ type-name ] .
```

Parameters

type-name
Specifies the type (MCS_BINARY or an MCS_BINARY subtype) of the file element you are modifying. See the *Oracle CDD/Repository Information Model* for more information on these types.

element-name
Specifies the file element you are modifying.

text
Modifies information. Within the DESCRIPTION clause, this is information documenting the file element; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command and set the character_set of the session to DEC_KANJI.

property-name
Specifies the property you are adding, changing, or removing. You specify removal with the NO keyword.

n
Modifies the value (numerical) set for a property.

quoted-string
Modifies the value (a string enclosed in quotation marks) set for a property.

CHANGE FILE_ELEMENT Command

Description

The CHANGE FILE_ELEMENT command modifies a file element by performing a change in place. CDO changes the values you specify, and other values remain the same.

Because a file element is a controlled element, CDO freezes previous versions and allows you to modify only the highest visible version.

Before you can issue the CHANGE FILE_ELEMENT command, you must reserve the file element with the RESERVE FILE_ELEMENT command.

If you add, change, or delete a property from the file element, the property you specify must be a defined or inherited property for the file element's type. See the *Oracle CDD/Repository Information Model* for a list of these properties.

Errors occur if you attempt to delete the MCS_STOREDIN property from a file element whose STORETYPE is EXTERNAL. CDO requires this property for external files.

Examples

```
CDO>  RESERVE FILE_ELEMENT MCS_BINARY PARSER_TABLES
CDO>  CHANGE FILE_ELEMENT MCS_BINARY PARSER_TABLES
cont>   DESCRIPTION IS "PARSER TABLES FOR VERSION 5.0".
cont> END MCS_BINARY.
CDO>  REPLACE FILE_ELEMENT MCS_BINARY PARSER_TABLES
```

In this example, the CHANGE FILE_ELEMENT command adds description text to the file element PARSER_TABLES.

CHANGE GENERIC Command

Format

```
CHANGE  GENERIC  type-name  element-name

    [ DESCRIPTION IS /*text*/ ]    [ AUDIT IS /*text*/ ]
    [ NODESCRIPTION           ]

    [ property-name IS { n           } ]
    [                  { quoted-string } ]  ...  .
    [ NOproperty-name                    ]

    [ DEFINE  relationship-name  relationship-mbr

    [ property-name IS { n           } ]
    [                  { quoted-string } ]  ...  .
    [ NOproperty-name                    ]

    END relationship-name DEFINE . ] ...

    [ DELETE  relationship-name  relationship-mbr . ] ...

    END [ element-name ] type-name .
```

Parameters

type-name
Specifies the type of the generic element you are modifying. This type cannot be MCS_BINARY, a subtype of MCS_BINARY, MCS_COLLECTION, MCS_ CONTEXT, or MCS_PARTITION. See the *Oracle CDD/Repository Information Model Volume I* for more information.

element-name
Specifies the generic element you are modifying.

text
Modifies information. Within the DESCRIPTION clause, this is information documenting the generic element; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

property-name
Specifies the property you are adding, changing, or removing. You specify removal with the NO keyword.

n
Modifies the value (numerical) set for a property.

quoted-string
Modifies the value (a string enclosed in quotation marks) set for a property.

relationship-name
Specifies the relationship type that you are defining or deleting in the generic element. The type must be a subtype of RELATION.

relationship-mbr
Specifies the generic element that you are defining or deleting as a member of the relationship type. The element must exist in the repository; otherwise, an error occurs.

Description

The CHANGE GENERIC command modifies a generic element by performing a change in place. CDO changes the values you specify, and other values remain the same.

If a generic element is a controlled versioned element, CDO freezes previous versions and allows you to modify only the highest visible version. If a generic element is an uncontrolled versioned element, CDO modifies the highest version unless you specify another version number.

If a generic element is controlled, you must reserve the element before you can issue the CHANGE GENERIC command. The SHOW GENERIC or SHOW RESERVATIONS command indicates whether this condition exists.

You can modify generic elements that are based on types supplied through Oracle CDD/Repository or on user-supplied (extended) types. If you do most of your work with extended types, Oracle recommends that you should work through the Oracle CDD/Repository callable interface. The CDO GENERIC commands are useful to modify and display on a spot basis, but extensibility is not supported through CDO.

If you add, change, or delete a property from the generic element, the property you specify must be a defined or inherited property for the element's type. Likewise, any relationship member you specify must be compatible with the relationship name's type. See the *Oracle CDD/Repository Information Model Volume I* for more information on valid properties and types.

If the generic element you are modifying is based on an extended type and errors occur when you attempt to add or delete a relationship, you may not have specified the processing name property as a required property for your type. The property takes a quoted string value.

Caution

Specify the MCS_processingName property, not the CDD$PROCESSING_NAME property, when you work with extended types. Otherwise, you experience performance degradation in the Oracle CDD/Repository callable interface.

The type on which the generic element definition is based determines whether an attribute is required or optional in instances of the type. If the type definition specifies that the CDD$DESCRIPTION attribute can be used in instances of the type, you can add documentation text to the generic entity definition or remove existing documentation text. You can display text entered with the DESCRIPTION clause by using the SHOW GENERIC command with the /BRIEF or /FULL qualifiers.

If the type definition specifies that the CDD$HISTORY_LIST relationship can be used in instances of the type, you can add explanatory history list entries to the generic entity definition. You can display history list entries for generic entity definitions by using the SHOW GENERIC command with the /AUDIT or /ALL qualifiers.

Examples

1.
```
CDO>  CHANGE GENERIC CDD$EXECUTABLE_IMAGE MY_PROGRAM
cont>   MCS_PROCESSINGNAME "OUR_PROGRAM".
cont> END MY_PROGRAM CDD$EXECUTABLE_IMAGE.
```

   In this example, the CHANGE GENERIC command modifies the MCS$PROCESSING_NAME (MCS_processingName property) of the generic element MY_PROGRAM. MY_PROGRAM is based on the type CDD$EXECUTABLE_IMAGE, which is supplied by Oracle CDD/Repository.

2.
```
CDO>  CHANGE GENERIC BOOK CDO_REFERENCE_MANUAL
cont>   LIBRARY_NUMBER IS "AA-KL45A-TF".
cont> END BOOK CDO_REFERENCE_MANUAL.
```

In this example, the CHANGE GENERIC command modifies the LIBRARY_NUMBER of the CDO_REFERENCE_MANUAL generic element. CDO_REFERENCE_MANUAL is based on the user-supplied type BOOK.

CHANGE PARTITION Command

Format

```
CHANGE  PARTITION partition-name

        [ DESCRIPTION IS /*text*/ ]                                  [ AUDIT IS /*text*/ ]
        [ NODESCRIPTION           ]

        [ LOOKASIDE_PARTITION IS look-partition-name ,... ]
        [ NOLOOKASIDE_PARTITION                           ]

      [ PARENT_PARTITION IS parent-partition-name ]

        [ AUTOPURGE   ]
        [ NOAUTOPURGE ] .
```

Parameters

partition-name
Specifies the partition you are modifying.

text
Modifies information. Within the DESCRIPTION clause, this is information documenting the partition; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

look-partition-name
Modifies the list of partitions that is visible through this partition. Each partition must be an existing partition.

parent-partition-name
Modifies the partition hierarchy by naming a parent (owner) for an existing parent (owner) partition.

CHANGE PARTITION Command

Description

The CHANGE PARTITION command modifies a partition by performing a change in place. CDO changes the values you specify, and other values remain the same.

Because a partition is a nonversioned element, CDO does not accept a branch designation or a version number in the partition name.

The LOOKASIDE_PARTITION clause modifies the list of partitions whose contents are visible through the partition you are modifying.

The PARENT_PARTITION clause modifies the partition hierarchy by naming an owner for an existing parent (owner) partition. This clause can be specified only once during the lifetime of the partition, in either the DEFINE PARTITION or CHANGE PARTITION command.

The AUTOPURGE and NOAUTOPURGE keywords redefine the autopurge property for the partition. The value of this property indicates whether or not CDO automatically purges intermediate versions of elements in the partition when you promote the latest version.

Examples

1.
```
CDO>  DEFINE PARTITION FIRST_BASELEVEL AUTOPURGE.
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.
CDO>  CHANGE PARTITION FRONT_END
cont>   LOOKASIDE_PARTITION IS BACK_END.
```

In this example, the CHANGE PARTITION command adds a LOOKASIDE_PARTITION clause to the FRONT_END partition element. You add this clause in a CHANGE PARTITION command, rather than in the initial DEFINE PARTITION command, because the partition named in a LOOKASIDE_PARTITION clause must be an existing partition.

2.
```
CDO>  DEFINE PARTITION FINAL_REPORT AUTOPURGE.
CDO>  DEFINE PARTITION PUBLICATION_RELEASE AUTOPURGE.
CDO>  CHANGE PARTITION FINAL_REPORT
cont>   PARENT_PARTITION IS PUBLICATION_RELEASE.
```

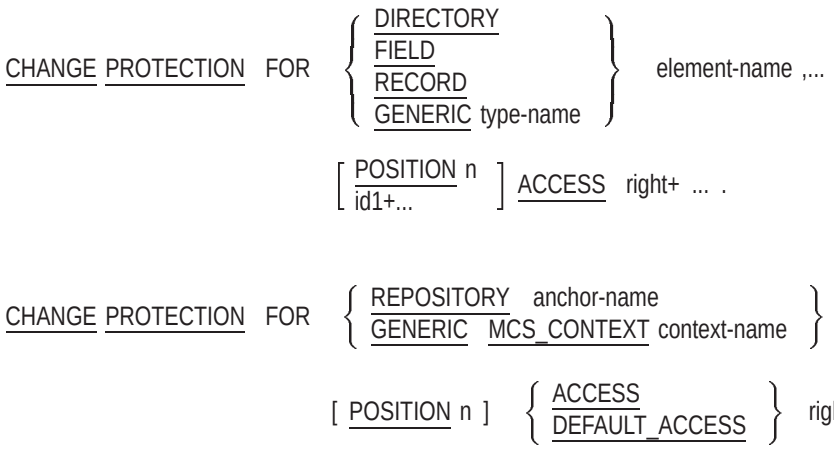
In this example, the CHANGE PARTITION command adds a PARENT_ PARTITION clause to the FINAL_REPORT partition element.

3.
```
CDO> DEFINE PARTITION FIRST_TESTBASELEVEL AUTOPURGE.
CDO> CHANGE PARTITION FIRST_TESTBASELEVEL
cont> AUDIT IS "LET'S PROMOTE ALL TEST VERSIONS" NOAUTOPURGE.
```

In this example, the CHANGE PARTITION command adds an AUDIT clause and modifies the AUTOPURGE keyword.

CHANGE PROTECTION Command

Format



Parameters

type-name
Specifies the type of the generic element whose ACE you are modifying.

element-name
Specifies the element whose ACE you are modifying. You can use wildcard characters in this name.

n
Specifies the relative position (a positive integer) in the ACL of the ACE you are modifying. If you omit this parameter and also the id1 parameter, CDO changes the first ACE by default.

id
Specifies the identifier of the ACE you are modifying. If you omit this parameter and also the n parameter, CDO changes the first ACE by default.

right
Specifies the access rights you are granting to the users specified in the ACE.

anchor-name
Specifies the anchor directory for the repository whose ACE you are modifying.

context-name
Specifies the context for which you are modifying protections.

Description

The CHANGE PROTECTION command modifies access rights for an access control list entry (ACE) in an access control list (ACL) for an element. When you specify FOR GENERIC MCS_CONTEXT or FOR REPOSITORY, this command can also add an ACE to a *default* access control list.

CHANGE PROTECTION affects a change in place. CDO changes the values you specify, and other values remain the same.

You must have CONTROL access rights to change protection for an element or a repository.

The POSITION clause identifies the ACE you are changing by its relative position within the ACL. For example, POSITION 3 indicates the third ACE in the ACL. If you specify a number greater than the number of existing ACEs, CDO changes the last ACE in the ACL.

The id parameter specifies the user or users affected by the ACE you are changing. The clause consists of one or more UIC, general, or system-specified identifiers.

If you specify more than one identifier, a user's process must hold all the identifiers before CDO grants the access rights indicated in the ACE.

The ACCESS clause specifies access rights provided by the ACE. See the DEFINE PROTECTION command for more information on access rights.

The ACCESS clause is especially useful when you need to restrict access to a context or to a repository. For example, by modifying this clause you can restrict access to a single user for OpenVMS BACKUP or VERIFY operations.

The DEFAULT_ACCESS clause is only valid for contexts (specified as GENERIC MCS_CONTEXT) or repositories. The clause specifies the default access rights for each new element you create. If a context is set, the new element receives default access rights defined for this context. If a context is not set, the new element receives the default access rights defined for the repository.

Examples

1.
```
CDO>  CHANGE PROTECTION FOR RECORD
cont> PAYROLL, PROMOTION [JONES]+INTERACTIVE
cont>   ACCESS CONTROL+READ.
```

   In this example, the CHANGE PROTECTION command affects the access rights for the PAYROLL and PROMOTION record elements. CDO locates the ACE containing [JONES]+INTERACTIVE identifiers and adds additional CONTROL and READ access rights.

2.
```
CDO>  CHANGE PROTECTION FOR FIELD
cont> EMP_DATE POSITION 3 ACCESS NOALL+READ.
```

   In this example, the CHANGE PROTECTION command affects the access rights for the EMP_DATE field element. CDO locates the third ACE in the field's ACL and removes all access rights except READ access.

3.
```
CDO> CHANGE PROTECTION FOR RECORD SALARY ACCESS NONE.
```

   In this example, the CHANGE PROTECTION command changes the first ACE in the ACL for the SALARY record element. After the command executes, the users whose identifiers match the identifiers in the first ACE will not have access to the SALARY record element.

4.
```
CDO>  CHANGE PROTECTION FOR REPOSITORY PERSONNEL
cont>   POSITION 3 DEFAULT_ACCESS READ+NOWRITE+CONTROL.
CDO>  DEFINE FIELD NEW_FIELD DATATYPE TEXT SIZE 5.
```

   In this example, the CHANGE PROTECTION command changes the default access rights for the PERSONNEL repository to READ+NOWRITE+CONTROL. If a context has not been set, CDO will then grant the newly created field, NEW_FIELD, with access rights that are equivalent to these repository default access rights.

CHANGE RECORD Command

Format

```
CHANGE  RECORD  record-name

    [ DESCRIPTION IS /*text*/ ]
    [ NODESCRIPTION           ]    [ AUDIT IS /*text*/ ]

    [ constraint-clause                  ]
    [ DELETE CONSTRAINT constr-name      ]
    [ record-property                    ]  ... .
    [ NOrecord-property                  ]

   [ DELETE name . ] ...

    [ included-name-change-clause ]
    [ local-field-clause          ]
    [ record-change-clause        ]  ...
    [ structure-change-clause     ]
    [ variants-change-clause      ]

   END  [ record-name ]  RECORD .
```

Parameters

record-name
Specifies the record element you are modifying.

text
Modifies information. Within the DESCRIPTION clause, this is information documenting the record; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

constraint-clause
Adds a condition, known as a constraint, that affects adding or modifying data in a database table (record). Supported constraint types are NOT NULL, PRIMARY KEY, FOREIGN KEY, UNIQUE, and CHECK.

Each constraint can be named and supplied with evaluation attributes DEFERRABLE or NOT DEFERRABLE. The default evaluation time for constraints in CDO is NOT DEFERRABLE (the constraint is evaluated at statement time). For more information, see the DEFINE RECORD: Constraint Clause.

constr-name
Specifies the name of a constraint.

record-property
Changes the value of an existing property or adds a new property in record, structure, variants, and variant definitions within a record element. You specify removal with the NO keyword. See Chapter 2 for the record properties CDO provides.

name
Specifies the name of a record, structure, or field that you want to delete from the record.

included-name-change-clause
Allows you to change existing field and record definitions within record elements. For more information, see the CHANGE RECORD: Included Name Change Clause.

local-field-clause
Specifies the definition of the locally defined field. For more information, see the DEFINE RECORD: Local Field Clause.

record-change-clause
Adds field, record, structure, variants, and variant definitions within an existing record definition. For more information, see the CHANGE RECORD: Record Change Clause.

structure-change-clause
Allows you to change a structure definition within a record element. For more information, see the CHANGE RECORD: Structure Change Clause.

variants-change-clause
Allows you to change a variant definition, which is a set of two or more definitions that map to the same portion of a record element. For more information, see the CHANGE RECORD: Variants Change Clause.

Description

The CHANGE RECORD command modifies a record element by performing a change in place. CDO changes the values you specify, and other values remain the same.

If a record element is controlled, CDO freezes previous versions and allows you to modify only the highest visible version. If a record element is uncontrolled, CDO modifies the highest version unless you specify another version number.

If a record element is controlled, you must reserve the record element before you can issue the CHANGE RECORD command. The SHOW RECORD or SHOW RESERVATIONS command indicates whether this condition exists.

When you change a record element that an Oracle Rdb database uses, you may need to integrate the database with the repository. CDO automatically sends a notice with the name of the database when this possibility occurs.

To remove a field, record, or structure definition from a record element, if the definition is not contained within a variant or structure definition, specify the DELETE keyword, followed by the appropriate name or clause for the type of definition you are removing.

To remove a definition from within a variant definition, use the NOVARIANTS and VARIANT keyword, followed by the DELETE clause.

To remove a definition from within a structure definition, specify the CHANGE RECORD Structure Change Clause. Specify the DELETE keyword, followed by the name of the definition you are removing.

If you are deleting a constraint, you must delete the constraint before you delete the field; they cannot be deleted simultaneously using the CHANGE RECORD command. To update the change in the database, you must integrate each change separately.

Examples

1.
```
CDO> CHANGE RECORD SUPPLIER_REC
cont> ROW_MAJOR ARRAY 1:20.
cont> END RECORD.
```

   This example uses the CHANGE RECORD command to add an array clause to a record called SUPPLIER_REC.

2.
```
CDO> CHANGE RECORD EMPLOYEE_WORK_SCHEDULE
cont> NONAME COBOL.
cont> END RECORD.
```

In this example, the CHANGE RECORD command uses the NO keyword to remove the NAME FOR COBOL record property from the EMPLOYEE_WORK_SCHEDULE record definition.

3.
```
CDO> CHANGE RECORD EMP_ADDRESS.
cont> DELETE DEPT_CODE.
cont> END RECORD.
```

In this example, the CHANGE RECORD command deletes the DEPT_CODE field definition.

4.
```
CDO> CHANGE RECORD EMP_ADDRESS.
cont> DEFINE EMP_NAME.
cont> END EMP_NAME DEFINE.
cont> END RECORD.
```

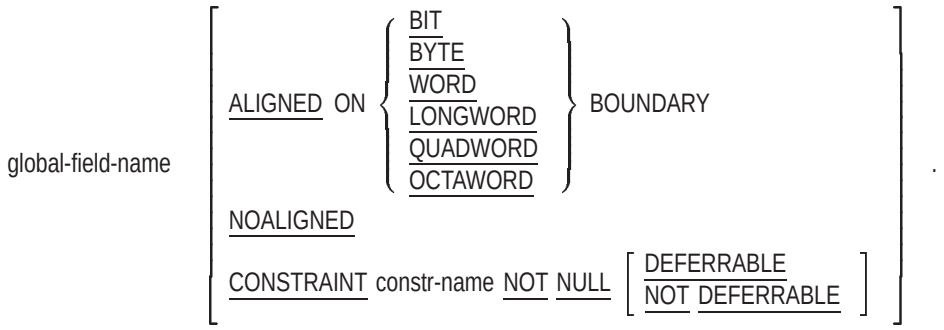
In this example, the CHANGE RECORD command adds the EMP_NAME record definition to the EMP_ADDRESS record element.

5.
```
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.
```

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. This example adds the fields WAGE_STATUS and CLASS_CODE to the record definition EMPLOYEE_REC.

CHANGE RECORD: Included Name Change Clause

Format



Parameters

global-field-name
Specifies the global field whose alignment you are creating or modifying.

constr-name
Specifies a constraint for the field.

Description

The Included Name Change Clause modifies or cancels the alignment of field or record definitions within a record element.

To modify or cancel the alignment of field or record definitions within a structure definition, specify the Structure Change Clause, then the Included Name Change Clause.

To modify or cancel the alignment of field or record definitions within a variant definition, specify the Variants Change Clause, then the Included Name Change Clause. To indicate the position of the variant, insert as many VARIANT and END VARIANT keywords as necessary, so each preceding variant is referenced.

Note

When the CHANGE RECORD command is used to change a variants or variant definition or an entity inside a variants or variant definition, you must use the Variants Change Clause to refer to each variants or variant definition that precedes the entity you are changing.

Examples

1.
```
CDO>  CHANGE RECORD PRODUCT_INVENTORY.
cont> PART_NUMBER ALIGNED ON BYTE BOUNDARY.
cont> END PRODUCT_INVENTORY RECORD.
```

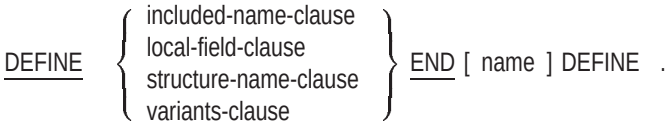
   In this example, the ALIGNED keyword in the CHANGE RECORD command realigns the PART_NUMBER field definition within the PRODUCT_INVENTORY record definition.

2.
```
CDO> CHANGE RECORD PRODUCT_INVENTORY.
cont> HOME_APPLIANCES NOALIGNED.
cont> END PRODUCT_INVENTORY RECORD.
```

   In this example, the NOALIGNED keyword in the CHANGE RECORD command cancels the explicit alignment of the HOME_APPLIANCES record definition within the PRODUCT_INVENTORY record definition.

CHANGE RECORD: Record Change Clause

Format



Parameters

included-name-clause
Includes existing field and record definitions in a record element. See DEFINE RECORD: Included Name Clause for more information.

local-field-clause
Adds the definition of a local field. See DEFINE RECORD: Local Field Clause for more information.

structure-name-clause
Adds a structure definition within a record element. For more information, see the DEFINE RECORD: Structure Name Clause.

variants-clause
Specifies a variants definition that you want to change. For more information, see the DEFINE RECORD: Variants Clause.

name
Specifies the definition you are adding or removing within a record element.

Description

Adds field, record, structure, variants, and variant definitions within an existing record definition.

You cannot remove a variant definition from a record with the Record Change Clause. Use the Variants Change Clause instead.

If you want to add a definition to a record element, but you do not want the definition to be added within an existing structure or variant definition, specify the DEFINE keyword, followed by the appropriate clause for the type of definition you are adding:

- To add a field or record definition, use the Included Name Clause or the Local Field Clause.

- To create a structure definition, use the Structure Clause.
- To create a variants definition, use the Variants Clause.

Any definition you add becomes the last definition in the record you are changing.

If you want to remove a field, record, or structure definition from a record element, and these are not contained within a structure or variant definition, specify the DELETE keyword, followed by the appropriate clause for the type of definition you are removing.

To add a definition within a structure definition, specify the Structure Change Clause. This clause contains an embedded Record Change Clause where you specify the DEFINE keyword, followed by the appropriate clause for the definition you are adding.

To remove a definition from within a structure definition, specify the Structure Change Clause. Specify the DELETE keyword, followed by the name of the definition you are removing.

To add a definition within a variant definition, specify the Variants Change Clause. This clause contains an embedded Record Change Clause where you specify the DEFINE keyword, followed by the appropriate clause for the definition you are adding. To indicate the position of the variant, insert as many VARIANT and END VARIANT keywords as necessary, so each preceding variant is referenced.

To remove a definition within a variant definition, use the NOVARIANTS or the NOVARIANT keyword of the Variants Change Clause.

Note

When the CHANGE RECORD command is used to change a variants or variant definition or an entity inside a variants or variant definition, you must use the Variants Change Clause to refer to each variants or variant definition that precedes the entity you are changing.

Examples

1.
```
CDO>  CHANGE RECORD EMP_ADDRESS.
cont>   DEFINE EMP_NAME.
cont>   END EMP_NAME DEFINE.
cont> END RECORD.
```

   With the Record Change Clause, the CHANGE RECORD command adds the EMP_NAME record definition to the EMP_ADDRESS record definition.

2.
```
CDO>  CHANGE RECORD EMP_ADDRESS.
cont>   DELETE DEPT_CODE.
cont> END RECORD.
```

   With the Record Change Clause, the CHANGE RECORD command deletes the DEPT_CODE field definition.

3.
```
CDO> DEFINE RECORD COMPANY_INVENTORY.
cont> STOCK STRUCTURE.
cont> DESCRIPTION IS /* RECORD_IDENTIFIER determines field type: */
cont> /* S = In-stock record */
cont> /* B = Back-order record */
cont> /* O = Out-of-stock record. */
cont> RECORD_IDENTIFIER.
cont> VARIANTS.
cont> IN_STOCK STRUCTURE.
cont> PRODUCT_NO.
cont> DATE_ORDERED.
cont> STATUS_CODE.
cont> QUANTITY.
cont> LOCATION.
cont> UNIT_PRICE.
cont> END IN_STOCK STRUCTURE.
cont> END VARIANT.
cont> VARIANT EXPRESSION IS "B".
cont> BACK_ORDER STRUCTURE.
cont> PRODUCT_NO.
cont> DATE_ORDERED.
cont> STATUS_CODE.
cont> QUANTITY.
cont> SUPPLIER.
cont> UNIT_PRICE.
cont> END BACK_ORDER STRUCTURE.
cont> END VARIANT.
```

```
cont> VARIANT EXPRESSION IS "O".
cont> OUT_OF_STOCK STRUCTURE.
cont> PRODUCT_NO.
cont> DATE_LAST_SOLD.
cont> END OUT_OF_STOCK STRUCTURE.
cont> END VARIANT.
cont> END VARIANTS.
cont> END STOCK STRUCTURE.
cont> END COMPANY_INVENTORY RECORD.
CDO>
CDO>  CHANGE RECORD COMPANY_INVENTORY.
cont> STOCK STRUCTURE.
cont> RECORD_IDENTIFIER.
cont> VARIANTS.
cont> VARIANT EXPRESSION IS "S".
cont> IN_STOCK STRUCTURE.
cont> DELETE STATUS_CODE.
cont> END IN_STOCK STRUCTURE.
cont> END VARIANT.
cont> VARIANT EXPRESSION IS "B".
cont> BACK_ORDER STRUCTURE.
cont> PRODUCT_NO.
cont> DATE_ORDERED.
cont> DEFINE DATE_PROMISED
cont> END DATE_PROMISED DEFINE.
cont> END BACK_ORDER STRUCTURE.
cont> END VARIANT.
cont> VARIANT EXPRESSION IS "O".
cont> OUT_OF_STOCK STRUCTURE.
cont> END OUT_OF_STOCK STRUCTURE.
cont> END VARIANT.
cont> END VARIANTS.
cont> END STOCK STRUCTURE.
cont> END COMPANY_INVENTORY RECORD.
```

This example shows how to use the Record Change Clause to define or delete entities within a structure or variant definition. The first part of the example shows the COMPANY_INVENTORY record definition. The second part of the example shows how you would use the appropriate syntax and the Record Change Clause to add and remove definitions within a structure and variant. The STATUS_CODE field is removed from the IN_STOCK structure and the DATE_PROMISED field is added to the BACK_ORDER structure.

CHANGE RECORD: Structure Change Clause

Format

```
structure-name  STRUCTURE

                [ DESCRIPTION IS /*text*/ ]
                [ NODESCRIPTION          ]

                [ record-property   ]
                [ NOrecord-property ] ... .

                [ DELETE name . ] ...

                [ included-name-change-clause ]
                [ local-field-clause          ]
                [ record-change-clause        ] ...
                [ structure-change-clause     ]
                [ variants-change-clause      ]

                END [ structure-name ]  STRUCTURE  .
```

Parameters

structure-name
Specifies the structure definition you are changing.

text
Documents the structure definition. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

record-property
Changes the value of an existing property, or adds a new property, in the structure definition. You specify removal with the NO keyword. See Chapter 2 for the record properties CDO provides.

name
Specifies the name of a record, structure, or field that you want to delete from the record.

CHANGE RECORD: Structure Change Clause

included-name-change-clause
Changes the alignment of a field or record definition within a structure definition. See CHANGE RECORD: Included Name Change Clause for more information.

local-field-clause
Changes the definition of a local field. See DEFINE RECORD: Local Field Clause for more information.

record-change-clause
Adds field, record, structure, variants, and variant definitions within a structure definition. Removes field, record, and structure definitions from within a structure definition. See CHANGE RECORD: Record Change Clause for more information.

structure-change-clause
Changes a structure definition within a structure definition. (This section describes the Structure Change Clause.)

variants-change-clause
Specifies a variant definition that you want to change or remove from a structure definition. See CHANGE RECORD: Variants Change Clause for more information.

Description

The Structure Change Clause adds or modifies field, record, structure, and variant definitions within a structure definition. It removes field, record, and structure definitions.

You cannot remove a variant definition from a structure definition with the Structure Change Clause. Use the Variants Change Clause instead.

To add a definition within a structure definition, specify the Structure Change Clause, which contains an embedded Record Change Clause. Use the DEFINE keyword of the Record Change Clause, followed by the appropriate clause for the definition you are adding.

For example, to add a field or record definition to the structure definition, specify the Included Name Clause or Local Field Clause. To add a structure definition, specify the Structure Change Clause. To add a variants definition, specify the Variants Change Clause.

The definition you add becomes the last definition in the structure definition.

To remove a definition from within a structure definition, specify the Structure Change Clause and the DELETE keyword of the embedded Record Change Clause, followed by the definition name.

Note

When you use the CHANGE RECORD command to change a variants or variant definition or an entity inside a variants or variant definition, you must use the Variants Change Clause to refer to each variants or variant definition that precedes the entity you are changing.

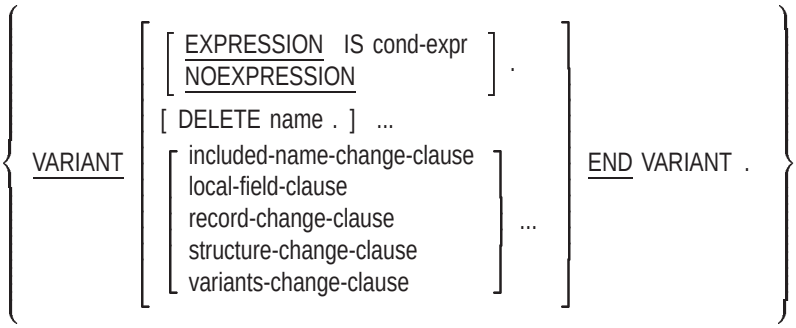
Examples

```
CDO>  CHANGE RECORD HOUSEHOLD.
cont>   DELETE ADDRESS.
cont>   DEPENDENTS STRUCTURE OCCURS 1 TO 4 TIMES
cont>     DEPENDING ON NUMBER_OF_DEPENDENTS IN HOUSEHOLD.
cont>   END DEPENDENTS STRUCTURE.
cont> END HOUSEHOLD RECORD.
```

In this example, the CHANGE RECORD command removes the ADDRESS field definition and changes the OCCURS...DEPENDING clause in the HOUSEHOLD record definition.

CHANGE RECORD: Variant Change Clause

Format



Parameters

cond-expr
Specifies an expression that represents the relationship between two value expressions. The value of a conditional expression is true, false, or null. If one definition uses an expression, each definition in the variant definition must have an expression. Each expression in the Variant Change Clause must be unique. For more information on conditional expressions, see Chapter 4.

name
Specifies the name of a record, structure, or field that you want to delete from the record.

included-name-change-clause
Changes the attribute of a field or record definition within a record element. See CHANGE RECORD: Record Change Clause for more information.

local-field-clause
Changes the definition of a local field. See DEFINE RECORD: Local Field Clause for more information.

record-change-clause
Adds field, record, structure, and variant definitions within a structure definition. Removes field, record, and structure definitions from within a structure definition. See CHANGE RECORD: Record Change Clause for more information.

structure-change-clause
Changes a structure definition within a record element. See CHANGE RECORD: Structure Change Clause for more information.

variants-change-clause
Specifies a variants definition that you want to change or remove from a record definition.

Description

The Variant Change Clause modifies or removes a variant definition within a variants definition.

When you change or add a variant definition to a record definition, you must tell CDO its position. To indicate the position of a variant definition to CDO, you use the VARIANT and END VARIANT keywords, so each preceding variant is referenced.

If you specify a conditional expression, the expression must be valid for the layered product that uses your definition. If one definition uses an expression, each definition in the variant definition must use an expression. Each expression in the Variant Change Clause must be unique.

To add a definition to a variants or variant definition, use the Variant Change Clause to specify the variants or variant you want to modify. Then use the DEFINE keyword of the Record Change Clause and the appropriate clause for the type of entity definition you are adding.

Note

You must include a structure definition for each variant contained in a CDO record if developing a new application that will use a 3GL language and DIGITAL DATATRIEVE.

For example, to add a field or record definition to the variants or variant definition, use the Included Name Change Clause. To create a structure definition in the variants or variant definition, use the Structure Clause.

To create another variant or variant definition in the variants or variant definition, use the Variant Change Clause.

To remove a field, record, or structure definition from a variants or variant definition, use the Variant Change Clause or the Variants Change Clause to specify the variants or variant definition you are changing. Then remove the

field, record, or structure definition by specifying the DELETE keyword of the Record Change Clause and the definition name.

To remove a variant definition, use the NOVARIANT keyword of the Variants Change Clause to specify the definitions you are removing.

When you add a new definition it becomes the last definition in a variant definition.

Examples

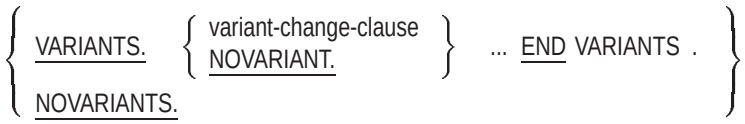
```
CDO>  CHANGE RECORD EMPLOYEE_RECORD.
cont>   VARIANTS.
cont>     VARIANT.
cont>     END VARIANT.
cont>     VARIANT.
cont>     END VARIANT.
cont>     VARIANT.
cont>       DELETE RATE.
cont>     END VARIANT.
cont>   END VARIANTS.
cont> END EMPLOYEE_REC RECORD.
```

In this example, the keyword DELETE in the CHANGE RECORD command removes the RATE field definition from the EMPLOYEE_RECORD record definition. The CHANGE RECORD command does *not* affect the other two definitions in the variant definition.

To indicate that the RATE field definition is in the third variant definition, you must use the Variants Change Clause as shown in this example. The keywords VARIANT and END VARIANT serve as placeholders for those variant definitions that you do *not* want to change.

CHANGE RECORD: Variants Change Clause

Format



Parameters

variant-change-clause
Specifies a variant definition that you want to change or remove from a record definition. See the CHANGE RECORD: Variant Change Clause for more information.

Description

The Variants Change Clause modifies or removes a variants definition within a record definition.

The NOVARIANTS clause removes a group of variant definitions from a record definition. The NOVARIANT clause removes a specific variant definition from a record definition.

To remove a variants definition, use the NOVARIANTS keyword of the Variants Change Clause to specify the definitions you are removing.

Note

When the CHANGE RECORD command is used to change a variants or variant definition or an entity inside a variants or variant definition, you must use the Variants Change Clause to refer to each variants or variant definition that precedes the entity you are changing.

Examples

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

In this example, the keyword DELETE in the CHANGE RECORD command removes the RATE field definition from the EMPLOYEE_RECORD record definition. The CHANGE RECORD command does *not* affect the other two definitions in the variant definition.

To indicate that the RATE field definition is in the third variant definition, you must use the Variants Change Clause as shown in this example. The keywords VARIANT and END VARIANT serve as placeholders for those variant definitions that you do *not* want to change.

CLEAR NOTICES Command

Format

CLEAR NOTICES [qualifier] element-name ,...

Parameters

element-name
Specifies the element whose notices you are removing. You can use wildcard characters in the element name.

Qualifiers

/CURRENT (default)
Clears notices at the element you specify.

/DOWN
Clears notices at elements owned by the element.

/UP
Clears notices at elements that own the element.

Description

The CLEAR NOTICES command removes notices that CDO has sent to an element. Use the SHOW NOTICES command to confirm that CDO has cleared notices.

Examples

```
CDO>  CLEAR NOTICES DEPT5
```

In this example, the CLEAR NOTICES command removes notices for the DEPT5 database definition.

CLOSE FILE_ELEMENT Command

Format

CLOSE FILE_ELEMENT type-name element-name

Parameters

type-name
Specifies the type (MCS_BINARY or MCS_BINARY subtype) of the file element you are closing. See the *Oracle CDD/Repository Information Model Volume I* for more information on these types.

element-name
Specifies the file element you are closing. You can substitute an asterisk (*) wildcard character for this parameter.

Description

The CLOSE FILE_ELEMENT command closes an internal file that you have previously opened. See the OPEN FILE_ELEMENT command for more information on opening a file.

Since a file element is a versioned element, CLOSE FILE_ELEMENT closes the highest visible version unless you specify another version number.

Examples

```
CDO>  CLOSE FILE_ELEMENT MCS_BINARY
cont>   PARSER_TABLES
```

In this example, CDO closes the binary file named PARSER_TABLES.

COMMIT Command

Format

COMMIT

Description

The COMMIT command ends a transaction and makes permanent any changes you made during that transaction. This command also releases all locks and closes all open streams. It affects all databases participating in the currently open transaction. See the START_TRANSACTION command description for restrictions that apply when using START_TRANSACTION . . . COMMIT stream of commands.

Restrictions

- When you delete a record, 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. Using CDO, there is no way to reuse those local fields. But, it is possible to use them through the Oracle CDD/Repository APIs. Therefore, the local fields cannot be automatically deleted at the same point in the transaction as the record.

  You must either delete the record and field in separate transactions (outside the START_TRANSACTION . . . COMMIT stream of commands) or, to accomplish this in one transaction, use the ENTER command to enter the local field, delete the record, delete the local field, and then delete the global field.
- Usually, if Oracle CDD/Repository issues any errors between the START_ TRANSACTION and COMMIT commands, it forces you to roll back the transaction. In some cases, such as in the CHANGE or DELETE commands, Oracle CDD/Repository allows you to commit the transaction. The general rules are:
  - If you receive an Oracle CDD/Repository error of E or F severity, such as a CDD-E-NODNOTFND message, you must abort the transaction.
  - If you receive a CDO error of E or F severity, such as a CDO-E-NOTFOUND message, you can continue to operate in the current transaction.

Examples

```
CDO> START_TRANSACTION.
CDO> DEFINE RECORD REC2.
cont> FLD1. END RECORD.
CDO> COMMIT
CDO> SHOW RECORD REC2
Definition of record REC2
|   Contains field          FLD1
.
.
.
```

In this example, the COMMIT command ends a session started with the START_TRANSACTION command. When you use the 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. The repository is already attached to the database and has already loaded the type definitions.

CONSTRAIN Command

Format

```
CONSTRAIN { FIELD | RECORD | GENERIC type-name } [ qualifier ] ... element-name ,...

[ DESCRIPTION IS /*text*/ ]  [ AUDIT IS /*text*/ ]
```

Parameters

type-name
Specifies the type of the generic element you are constraining.

element-name
Specifies the element you are constraining. You can substitute an asterisk (*) wildcard character for this parameter.

text
Documents the element within the DESCRIPTION clause. Adds information to the history list entry within the AUDIT clause. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

Qualifiers

/CLOSURE=TO_BOTTOM
/NOCLOSURE (default)
Specifies whether CDO constrains additional elements. Using the /CLOSURE=TO_BOTTOM qualifier constrains all children of an element that are uncontrolled.

/LOG
/NOLOG (default)
Specifies whether CDO displays text identifying each element as the element is constrained.

CONSTRAIN Command

Description

The CONSTRAIN command moves an uncontrolled element to a base partition, the first level of approval. An uncontrolled element can be one of the following:

- A field, record, or generic element that you created with Oracle CDD/Repository Version 4.0 or later
- A field, record, or generic element that you created with Oracle CDD/Repository, outside a context

Since fields, records, and generic elements are versioned elements, the CONSTRAIN command constrains the highest visible version unless you specify another version number.

Before you issue the CONSTRAIN command, you must create a partition and a context for the uncontrolled element and issue the SET CONTEXT command to this context.

The element you are constraining must be a versioned element. When you constrain one version of an element, CDO constrains all versions of the element.

An error occurs if you attempt to constrain an element that is reserved. The SHOW RESERVATIONS command indicates whether this condition exists.

An error occurs if you attempt to constrain an element that is a parent of an uncontrolled element. Constrain the parent and children simultaneously by issuing the CONSTRAIN command with the /CLOSURE=TO_BOTTOM qualifier.

After you constrain an element, you issue the ATTACH TO COMPOSITE command to explicitly attach the element to a collection, field, record, file, or generic element.

Note

The CONSTRAIN command is irreversible. A controlled element cannot be changed to an uncontrolled element. All subsequent versions of the element are controlled.

To create subsequent versions, issue the RESERVE command rather than the DEFINE command.

Examples

```
CDO>  DEFINE CONTEXT SUBSCRIPTIONS BASE_PARTITION FOURTH_QUARTER.
CDO>  SET CONTEXT SUBSCRIPTIONS
CDO>  DEFINE COLLECTION MAIL_LABEL.
CDO>  CONSTRAIN FIELD *
```

In this example, the CONSTRAIN command controls all fields in the current default directory. See the ATTACH TO COMPOSITE command for commands to attach the constrained fields to this collection.

CONVERT Command

Format

```
CONVERT { source-name ,... destination-name
        { /REPOSITORY repository-anchor-name }
```

Parameters

source-name
Specifies the Oracle Dictionary Management Utility (DMU) definition you are converting. The Oracle Dictionary Management Utility (DMU) definition must be a definition of type CDD$RECORD. You can use either a full or relative DMU path name with the CONVERT command. You can also substitute an asterisk (*) wildcard character for this parameter.

destination-name
Specifies the name that you select for the converted definition in the CDO repository. If you specify more than one DMU definition in the source name, the destination name must have a wildcard character in its name.

repository-anchor-name
Specifies the device and directory specification of the repository to be upgraded using the CONVERT/REPOSITORY command.

Qualifiers

/REPOSITORY
Specifies that the repository should be upgraded.

Description

The CONVERT command copies a DMU format definition from the DMU side of the dictionary to the CDO side of the dictionary. The CONVERT command leaves the definition in the DMU side of the dictionary.

The CONVERT/REPOSITORY command allows you to perform 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). Using this command requires SYSPRV or BYPASS privilege. Be sure you have an adequate backup of the repository before issuing this command.

Note

The CONVERT command is not the same as the CONVERT/REPOSITORY command, which allows you to perform a minor upgrade of a repository. For details on upgrading repositories using the CONVERT/REPOSITORY command, see the Upgrade_ Procedure topic in DCL level help or the instructions for upgrading a dictionary or repository provided in *Using Oracle CDD/Repository on OpenVMS Systems*.

Unless you specify a different path name, the CONVERT command copies a DMU definition into your default CDO directory.

When you convert a DMU definition, unless you specify the version number, CDO converts the highest version of the definition.

An error occurs if you specify a destination name that is the name of an existing definition in the CDO destination directory.

If you convert an Oracle Dictionary Management Utility (DMU) record definition that consists of a single field description statement, CDO converts the record definition to a CDO field definition.

When you convert an Oracle Dictionary Management Utility (DMU) format definition that includes a description clause for the definition and another description clause for a structure within the definition, only the structure description clause appears in the CDO format definition.

If you have a version of DIGITAL DATATRIEVE prior to Version 5.0 installed on your system, and you are converting a DMU definition that contains a VALID FOR DATATRIEVE IF field attribute or a COMPUTED BY DATATRIEVE field attribute clause, CDO omits these clauses from the resulting CDO definition.

If the DMU record that you are converting has a different processing name from the DMU directory name, then the resulting CDO record definition retains the old processing name. However, if both the processing and directory names are the same for the DMU definition, then the resulting CDO record definition retains the same name for both the processing and directory names.

When you convert an Oracle Dictionary Management Utility (DMU) definition, CDO creates a default ACL for it. See *Using Oracle CDD/Repository on OpenVMS Systems* for more information on default protection.

To display or manipulate a converted field within a record description with a CDO command, you must assign it a directory name. Use the ENTER command to assign this name.

When you convert an Oracle Dictionary Management Utility (DMU) definition to a CDO definition, CDO converts the DMU access rights to CDO access rights. However, because DMU protection is different from CDO protection, there is no CDO equivalent for some DMU access rights. Therefore, CDO must convert some DMU rights to the closest CDO access right.

Table 1–1 shows how CDO converts DMU access rights to the closest CDO equivalent. DMU access rights that are not listed in the table are not translated because no equivalent CDO access right is appropriate.

Table 1–1 Conversion of Oracle Dictionary Management Utility (DMU) Access Rights to CDO Access Rights

Oracle Dictionary Management Utility (DMU) Access Right	Equivalent CDO Access Right
CONTROL	CONTROL
DELETE Local or global	DELETE
DTR MODIFY	MODIFY (confirms that CHANGE access can be granted)
DTR READ	READ (confirms that SHOW access can be granted)
DTR WRITE	WRITE (confirms that DEFINE access can be granted)
SEE	SHOW
UPDATE	CHANGE + DEFINE

Examples

1.
```
CDO>  CONVERT CDD$TOP.PERSONNEL.BADGE_NUMBER BADGE_NUMBER
```

   In this example, the CONVERT command converts the DMU BADGE_ NUMBER record definition (and any embedded field definitions) to a CDO BADGE_NUMBER record definition in your default CDO directory.

2.
```
CDO>  CONVERT CDD$TOP.SHIPMENTS.CUSTOMER_RECORD,
cont>  CDD$TOP.SHIPMENTS.INVENTORY_RECORD *
```

In this example, CDO converts the DMU record definitions CUSTOMER_RECORD and INVENTORY_RECORD into your CDO directory. Because the asterisk (*) wildcard character was used in the destination-name, the DMU record definitions keep the same names after the conversion.

3.
```
DEFINE RECORD CDD$TOP.HARBORMASTER.YACHTS
    DESCRIPTION IS
        /* This record contains the manufacturer, model, and
           dock number of each yacht in the harbor, along
           with the owner's name. */.
    YACHTS STRUCTURE.
        MANUFACTURER        DATATYPE IS TEXT
                            SIZE IS 30 CHARACTERS.
        MODEL               DATATYPE IS TEXT
                            SIZE IS 30 CHARACTERS.
        DOCK_NUMBER         DATATYPE IS TEXT
                            SIZE IS 2 CHARACTERS.
        NAME STRUCTURE.
            LAST_NAME       DATATYPE IS TEXT
                            SIZE IS 15 CHARACTERS.
            FIRST_NAME      DATATYPE IS TEXT
                            SIZE IS 10 CHARACTERS.
            MIDDLE_INITIAL  DATATYPE IS TEXT
                            SIZE IS 1 CHARACTER.
        END NAME STRUCTURE.
    END YACHTS STRUCTURE.
END YACHTS RECORD.
```

In this example, YACHTS is the name of a DMU record definition and also of the STRUCTURE field description statement within the record definition.

4.
```
CDO>  CONVERT YACHTS YACHTS_NEW
CDO>  SHOW RECORD/FULL YACHTS_NEW
Definition of record YACHTS_NEW
|   Contains field          MANUFACTURER
|   |   Datatype                text size is 30 characters
|   Contains field          MODEL
|   |   Datatype                text size is 30 characters
|   Contains field          DOCK_NUMBER
|   |   Datatype                text size is 2 characters
|   Contains record         NAME
|   |   Contains field          LAST_NAME
|   |   |   Datatype                text size is 15 characters
|   |   Contains field          FIRST_NAME
|   |   |   Datatype                text size is 10 characters
|   |   Contains field          MIDDLE_INITIAL
|   |   |   Datatype                text size is 1 characters
CDO>
```

When you convert the DMU record definition YACHTS to the CDO record definition YACHTS_NEW, the resulting CDO record definition has YACHTS_NEW for its directory name and processing name.

COPY Command

Format

COPY source-name ,... destination-name

Parameters

source-name
Specifies the element you are copying. The source name can be a path name, directory name, or a name with wildcard characters.

Oracle CDD/Repository does not support passwords in name strings. When you issue the COPY command, do not include your password in the name string because a CDO-E-KWSYNTAX error will occur.

destination-name
Specifies the destination to which the element will be copied. The destination name can be a path name, directory name, or a name with one wildcard character.

Description

The COPY command copies an element and the relationships it owns within the same directory, from one CDO directory to another, or from one physical repository to another.

If the element is a versioned element, and you do not specify a version number, CDO copies all versions of the element.

The COPY command preserves relationships. If you copy both a parent and child, CDO copies the relationship between them.

If you copy the parent, CDO copies the relationship from the new parent to the child. CDO also maintains the previous relationship.

If you copy the child, CDO does *not* copy the relationship.

If you substitute a wildcard character for a destination name, CDO copies the element into your current default CDO directory and keeps the same name.

If you specify only a directory name for the destination name, CDO copies the element into that CDO directory and keeps the same element name.

If you specify both a directory name and a new processing name for the destination name, CDO copies the element into that directory and gives the element the name you specified.

Oracle CDD/Repository does not support passwords in name strings.

When you issue the COPY command and include your password, you get an error message similar to the following:

```
%CDO-E-KWSYNTAX, syntax error in command line at or near
password"::DISK$[CDDPLUS]some.user
```

Table 1–2 lists the rules for using wildcard characters with the COPY command.

Table 1–2 Rules for Using Wildcard Characters With the COPY Command

If Source Name Includes	Destination Name Can Include			
	One Asterisk	More Than One Asterisk	Ellipsis	No Wildcard Characters[1]
Asterisk (*)	Yes	No	Yes	No
More Than One Asterisk	Yes	No	Yes	No
Percent (%)	Yes	No	Yes	No
Ellipsis (...)	Yes	No	Yes	No
No Wildcard Characters	Yes	No	Yes	Yes

[1]Yes = valid; No = invalid

In addition to the information in the table, the following rules also apply to the use of wildcard characters in the COPY command:

- If you use a wildcard character in the source name, you must use a wildcard character in the destination name.
- You can only use one wildcard character in a destination name.
- You can only use one ellipsis (. . .) in a source or destination name.
- You can only use multiple asterisk (*) wildcard characters in the source name.
- You can only use percent sign (%) wildcard characters in the source name.

Examples

1.
```
CDO>  COPY DISK1:[JONES.DICT]PERSONNEL.LAST_NAME
cont>   DISK2:[BOB.SHOP]WORKERS.LAST_NAME
```

   This example uses the full path name to copy the LAST_NAME field element from the DISK1:[JONES.DICT]PERSONNEL directory to the DISK2:[BOB.SHOP]WORKERS directory.

2.
```
CDO>  COPY CORPORATE.LAST_NAME, FIRST_NAME
cont>   DISK1:[JONES.DICT]PERSONNEL.*
```

   In this example, the COPY command with the asterisk (*) wildcard character copies the LAST_NAME and FIRST_NAME field elements into the [JONES.DICT]PERSONNEL directory.

3.
```
CDO>  COPY CORPORATE.ADDRESS
cont>   [JONES.DICT]PERSONNEL.EMPLOYEE_ADDRESS
```

   In this example, the COPY command copies the ADDRESS record element into the [JONES.DICT]PERSONNEL directory and gives it a new name, EMPLOYEE_ADDRESS.

4.
```
CDO>  COPY DISK1:[CORPORATE.DICT]CORP_DEFS...
cont>    DISK1:[SMITH.DICT]DEVELOPMENT...
```

   This example uses the ellipsis (...) to copy an entire subhierarchy from the CORPORATE repository (starting with the CORP_DEFS directory) into the DEVELOPMENT directory of Smith's repository.

DEFINE COLLECTION Command

Format

DEFINE COLLECTION collection-name

[DESCRIPTION IS /*text*/] [AUDIT IS /*text*/] .

Parameters

collection-name
Specifies the collection you are creating.

text
Adds information. Within the DESCRIPTION clause, this is information documenting the collection; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

Description

The DEFINE COLLECTION command creates a collection. A collection allows you to view and manipulate a group of related elements that make up a particular system or subsystem.

Because a collection is a controlled element, you use the DEFINE COLLECTION command to create the initial version of a collection. Use the RESERVE and REPLACE commands to create new versions.

Before you issue the DEFINE COLLECTION command, you must create and set a context. The SHOW CONTEXT command indicates whether you have completed these steps.

Because all elements in a collection hierarchy are children of the top collection, you can issue commands that affect the entire hierarchy with the /DESCENDANTS or /CLOSURE qualifiers. Because most elements in a collection hierarchy are also children of smaller subcollections beneath the top collection, you can also issue commands that affect only one subcollection and its children.

To create a collection hierarchy, issue the DEFINE COLLECTION command immediately after the SET CONTEXT command. SET CONTEXT implicitly sets the collection you define as the top collection in the hierarchy, provided that you did not set a top collection within the DEFINE CONTEXT command.

To extend the hierarchy beneath the top collection, you issue the following commands:

1. DEFINE COLLECTION to create the collections that participate in the hierarchy. This command attaches all collections in the first level beneath the top collection.
2. DETACH FROM COMPOSITE to detach those collections destined for lower levels in the hierarchy from the first level beneath the top collection.
3. RESERVE and ATTACH TO COMPOSITE to reserve collections in each successive level and attach their immediate children.
4. REPLACE to store in a partition the elements you have created.

Examples

1.
```
CDO>  DEFINE COLLECTION REGIONAL_SALES.
```

In this example, the DEFINE COLLECTION command creates the REGIONAL_SALES collection.

2.
```
CDO>  DEFINE PARTITION FIRST_BASELEVEL.          1
CDO>  DEFINE CONTEXT DEVELOPMENT_CONTEXT
cont>   BASE_PARTITION FIRST_BASELEVEL.
CDO>  SET CONTEXT DEVELOPMENT_CONTEXT
CDO>  DEFINE COLLECTION COMPILER_C.              2
CDO>  RESERVE COLLECTION COMPILER_C
CDO>  DEFINE COLLECTION FRONT_END.               3
CDO>  DEFINE COLLECTION BACK_END.
CDO>  DEFINE COLLECTION PARSER.

CDO>  DEFINE FILE_ELEMENT MCS_BINARY PARSER_TABLES
cont>   STORETYPE EXTERNAL
cont>   MCS_STOREDIN IS "CDD$DISK:[SMITH]PARSER_TABLES.DAT".
cont>   END FILE_ELEMENT MCS_BINARY PARSER_TABLES.
```

```
CDO>  DETACH COLLECTION PARSER FROM COMPILER_C   4
CDO>  DETACH FILE_ELEMENT PARSER_TABLES FROM
cont>   COMPILER_C
CDO>  RESERVE COLLECTION FRONT_END
CDO>  ATTACH COLLECTION PARSER TO FRONT_END      5
CDO>  RESERVE COLLECTION PARSER
CDO>  ATTACH FILE_ELEMENT PARSER_TABLES TO PARSER
CDO>  REPLACE COLLECTION /CLOSURE=TO_TOP PARSER
```

The successive DEFINE COLLECTION commands in this example participate in the creation of a collection hierarchy.

1 DEFINE PARTITION, DEFINE CONTEXT, and SET CONTEXT commands allow you to control elements.

2 DEFINE COLLECTION command creates the collection; this command also sets COMPILER_C as the top collection because the current context, DEVELOPMENT_CONTEXT, does not have a top collection defined.

3 DEFINE COLLECTION commands create subcollections FRONT_END, BACK_END, and PARSER and file element PARSER_TABLES under collection COMPILER_C.

4 DETACH commands detach PARSER and PARSE_TABLES from collection COMPILER_C.

5 ATTACH commands attach PARSER under subcollection FRONT_END and PARSE_TABLES under subcollection PARSER.

DEFINE CONTEXT Command

Format

DEFINE CONTEXT context-name

[DESCRIPTION IS /*text*/] [AUDIT IS /*text*/]

BASE_PARTITION IS partition-name

[TOP IS collection-name]

[DEFAULT_ATTACHMENT IS { SPECIFIC_VERSION | LATEST_CHECKIN | LATEST }] .

Parameters

context-name
Specifies the context you are creating.

text
Adds information. Within the DESCRIPTION clause, this is information documenting the context; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

partition-name
Specifies the base partition of the partition hierarchy.

collection-name
Specifies the top collection of the collection hierarchy.

DEFINE CONTEXT Command

Description

The DEFINE CONTEXT command creates a context. A context allows you to restrict or expand your view of the system and set the characteristics associated with your work environment.

The BASE_PARTITION clause sets the base partition property to the partition name you specify. Use the DEFINE PARTITION command to create this partition prior to issuing the DEFINE CONTEXT command.

The TOP clause sets the top collection property to the collection name you specify. Include this clause only when you are redefining the top collection property for an existing context. See the DEFINE COLLECTION command for more information on setting this property for a new context.

The DEFAULT_ATTACHMENT clause defines the default attachment property for the context. This property refers to the default behavior that occurs when you issue the UPDATE command. Choose one of the following keywords:

DEFAULT_ATTACHMENT Keyword	Behavior
LATEST	Detaches the version currently attached and attaches the lastest version, whether checked in or ghost.
LATEST_CHECKIN	Detaches the version currently attached and attaches the version most recently checked in.
SPECIFIC_VERSION	Does not detach the version currently attached.

If you do not specify the DEFAULT_ATTACHMENT clause, CDO creates the context with LATEST default attachment, by default.

Examples

```
CDO>  DEFINE CONTEXT SALES
cont>   BASE_PARTITION IS FIRST_QUARTER
cont>   DEFAULT_ATTACHMENT IS LATEST_CHECKIN.
```

In this example, the DEFINE CONTEXT command creates the SALES context. The BASE_PARTITION clause sets the base partition property to the previously defined FIRST_QUARTER partition. The keyword LATEST_ CHECKIN in the DEFAULT_ATTACHMENT clause sets the default attachment property for the context to the version most recently checked in.

See the DEFINE COLLECTION command for more information on setting the top collection property for a context.

DEFINE DATABASE Command

Format

DEFINE DATABASE database-name

[DESCRIPTION IS /*text*/] [AUDIT IS /*text*/]

USING rms-database-name ON file-name [qualifier] .

Parameters

database-name
Specifies the database element you are creating.

text
Adds information. Within the DESCRIPTION clause, this is information documenting the database; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

rms-database-name
Specifies an existing logical RMS database element. It must be the name of an existing CDD$RMS_DATABASE element.

file-name
Specifies the location on disk of the physical OpenVMS file that holds the physical RMS database. It is a character string having from 1 to 1024 characters.

Qualifiers

/EXISTING_FILE
Specifies that an RMS file exists and does not need to be created.

Description

The DEFINE DATABASE command creates a physical RMS database on disk using an RMS database element. If the command completes successfully, DEFINE DATABASE creates a CDD$DATABASE element (with the database name you specified) and a CDD$FILE element (with the OpenVMS file name you specified) in your directory.

If the RMS database element is a controlled element, you use the DEFINE DATABASE command to create the initial version of a database. Use the RESERVE and REPLACE commands to create new versions.

If the RMS database element is an uncontrolled element, you use the DEFINE DATABASE command to create both initial and new versions.

If you supply a database name that is already used for a database element in your specified directory, you will create a new version of the existing database definition.

This command allows you to create many different physical RMS databases using the same logical RMS database element. You can specify a different location on disk for each database with an OpenVMS file name.

As of Oracle CDD/Repository Version 6.1, the DEFINE DATABASE command supports unsigned numeric and ADT fields as keys in RMS databases.

If the database name does not specify a full path name, CDO creates the database definition in your current default directory. CDO attempts to translate the database name you supply to determine if it is a valid logical name. If it is a logical name and CDO cannot translate the logical name to a valid path name, the operation fails.

Examples

1.
```
CDO>  DEFINE DATABASE DISG_FILE USING EMPLOYEE_STORAGE
cont>   ON DISK1:[DISG]EMP.DAT.
```

In this example, the DEFINE DATABASE command creates the physical DISG_FILE RMS database in the OpenVMS EMP.DAT file on disk, using the logical EMPLOYEE_STORAGE RMS database element.

2.
```
CDO>  DEFINE DATABASE EMPLOYEES
cont>   AUDIT IS /* INFORMATION ON CURRENT "EMPLOYEES" */
cont>   USING EMPLOYEE_DATABANK ON DISK2:[SMITH]MORE_EMP.DATA.
```

In this example, the DEFINE DATABASE command creates the physical EMPLOYEES RMS database on disk in the OpenVMS MORE_EMP.DATA file, using the EMPLOYEE_DATABANK RMS database element.

DEFINE DIRECTORY Command

Format

DEFINE DIRECTORY directory-name .

Parameters

directory-name
Specifies the directory you are creating.

Description

The DEFINE DIRECTORY command creates a CDO repository directory.

DEFINE DIRECTORY evaluates the directory name you supply to determine if it is a logical name. If the directory name is a logical name, CDO translates it. If the translation is not a valid name for a directory, CDO does not create the directory.

CDO automatically creates any directories in the path name of the directory-name parameter that do not already exist.

Examples

1.
```
CDO>  DEFINE DIRECTORY NODE::DISK1:[BOB.DICT]TOP.
```

   In this example, the DEFINE DIRECTORY command creates a directory called TOP under the repository anchor NODE::DISK1:[BOB.DICT].

   Or, you can define the directory TOP by setting default to the repository anchor DISK1:[BOB.DICT] then issuing the DEFINE DIRECTORY command.

2.
```
CDO>  DEFINE DIRECTORY DISK1:[BOB.DICT]PERSONNEL.EMPLOYEES.BENEFITS.
```

   In this example, the DEFINE DIRECTORY command creates the BENEFITS directory three levels below the CDO repository anchor DISK1:[BOB.DICT].

   Oracle CDD/Repository will create the intermediate directories if they do not already exist.

DEFINE FIELD Command

Format

DEFINE FIELD field-name

[DESCRIPTION IS /*text*/] [AUDIT IS /*text*/]

[field-property] ... .

Parameters

field-name
Specifies the field element you are creating.

text
Adds information. Within the DESCRIPTION clause, this is information documenting the field element; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

field-property
Adds a property to the field element. See Chapter 2 for the field properties CDO provides.

Description

The DEFINE FIELD command creates a field element.

If the field element is controlled, you use the DEFINE FIELD command to create the initial version of the element. Use the RESERVE and REPLACE commands to create new versions.

If the field element is uncontrolled, use the DEFINE FIELD command to create both initial and new versions.

You can create a field element in a directory other than your default directory by specifying the appropriate path name.

If you supply a field name that is already used for a field element in your default directory, CDO creates a new version of the existing field definition.

The DEFINE FIELD command evaluates the field name you supply to determine if it is a logical name. If the field name is a logical name, CDO translates it. In some cases, the translation of the logical name for the field name may not be a valid name for a field definition, and CDO will not create the field definition. For example, if you have defined JOE as a logical name that translates to MYNODE::[RICHIE], CDO translates the symbol JOE. The following DEFINE FIELD command fails because MYNODE::[RICHIE] is not a valid field name:

```
CDO> DEFINE FIELD JOE.
%CDO-F-ERRDEFINE, error defining object
-CDD-F-NOTADIC, Does not contain an Oracle CDD/Plus dictionary:
MYNODE::
```

If this error occurs, deassign the logical name with the same name as the object, and perform the operation again. To avoid this logical name conflict, use unique names that represent the type of entity you are naming.

Examples

1.
```
CDO>  DEFINE FIELD POSTAL_CODE
cont>   DESCRIPTION IS /* A 5 DIGIT POSTAL_CODE */
cont>   AUDIT IS /* WILL BE CHANGED TO 9 DIGITS EVENTUALLY */
cont>   DATATYPE IS UNSIGNED LONGWORD
cont>   SIZE IS 5 DIGITS.
```

   In this example, the DEFINE FIELD command creates the POSTAL_ CODE field element.

2.
```
CDO>  DEFINE FIELD SEX
cont>   DATATYPE IS TEXT SIZE IS 1
cont>   VALID IF SEX = "M" OR SEX = "F".
```

   In this example, the DEFINE FIELD command creates the SEX field element. The VALID IF field property returns an error if you attempt to store a value other than M or F in the field that refers to this element.

DEFINE FILE_ELEMENT Command

Format

```
DEFINE  FILE_ELEMENT  type-name  element-name

        [ DESCRIPTION IS /*text*/ ]  [ AUDIT IS /*text*/ ]

        [ STORETYPE INTERNAL                                              ]
        [ STORETYPE EXTERNAL  MCS_STOREDIN IS quoted-string               ]

        [ MCS_IMPORTED FROM quoted-string ]
        [ property-name IS { n             } ]  ... .
                           { quoted-string }

        END  [ FILE_ELEMENT ] type-name [ element-name ] .
```

Parameters

type-name
Specifies the type (MCS_BINARY or an MCS_BINARY subtype) of the file element you are creating. See the *Oracle CDD/Repository Information Model Volume I* for more information on these types.

element-name
Specifies the file element you are creating.

text
Adds information. Within the DESCRIPTION clause, this is information documenting the file element; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

quoted-string
Sets the value (a string enclosed in quotation marks) for the property you are specifying.

property-name
Specifies the property whose value you are setting.

n
Sets the numeric value for a property.

Description

The DEFINE FILE_ELEMENT command creates a file element.

Before you issue the DEFINE FILE_ELEMENT command, you must define and set a context. The SHOW CONTEXT command indicates whether you have completed these steps.

You control file element definitions as soon as you define them. To do this, issue the following commands before you define a file element:

1. DEFINE PARTITION, which creates a partition
2. DEFINE CONTEXT, which associates this partition with a specific context
3. SET CONTEXT, which identifies this context as the current context and implicitly controls all subsequent definitions

Since a file element is a controlled element, the DEFINE FILE_ELEMENT command creates the initial version of the file element. The RESERVE and REPLACE commands create new versions.

The STORETYPE clause indicates whether or not the file is stored internally (in Oracle CDD/Repository) or externally. If you do not specify STORETYPE, the default is external.

If you add, change, or delete a property from the file element, the property you specify must be a defined or inherited property for the file element's type. See the *Oracle CDD/Repository Information Model, Volume I* for a list of these properties.

Errors occur if you do not specify the MCS_STOREDIN property for a file element whose STORETYPE is EXTERNAL. CDO requires this property for external files.

Examples

```
CDO>  DEFINE FILE_ELEMENT MCS_BINARY PARSER_TABLES
cont>   STORETYPE EXTERNAL
cont>   MCS_STOREDIN IS "CDD$DISK:[SMITH]PARSER_TABLES.DAT".
cont> END FILE_ELEMENT MCS_BINARY PARSER_TABLES.
```

In this example, the DEFINE FILE_ELEMENT command includes a STORETYPE EXTERNAL clause. CDO creates an external file element PARSER_TABLES stored in CDD$DISK:[SMITH]PARSER_TABLES.DAT.

DEFINE GENERIC Command

Format

DEFINE GENERIC type-name element-name

[DESCRIPTION IS /*text*/] [AUDIT IS /*text*/]

[property-name IS { n | quoted-string }] ... .

[RELATIONSHIPS . { RELATIONSHIP relationship-name relationship-mbr-options
[property-name IS { n | quoted-string }] ... .
END relationship-name RELATIONSHIP . } ...
END RELATIONSHIPS .]

END [GENERIC type-name element-name] .

Parameters

type-name
Specifies the type of the generic element you are defining.

element-name
Specifies the generic element you are defining.

text
Adds information. Within the DESCRIPTION clause, this is information documenting the generic element; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

property-name
Specifies a property.

n
Sets the numeric value for a property.

quoted-string
Sets the value (a string enclosed in quotation marks) for a property.

relationship-name
Specifies the relationship that you are defining for the generic element. The type must be a subtype of RELATION.

relationship-mbr-options
Allows you to specify a relationship member. This member can be an existing element in a repository, or it can be an element you create within the DEFINE GENERIC command. See DEFINE GENERIC: Relationship Member Options Clause for more information.

Description

The DEFINE GENERIC command creates a generic element definition. You can create generic elements that are based on types supplied by Oracle CDD/Repository or on user-supplied (extended) types. If you do most of your work with extended types, it is recommended that you work through the Oracle CDD/Repository callable interface. The CDO GENERIC commands are useful to modify and display on a spot basis, but extensibility is not supported through CDO.

If the generic element is a controlled versioned element, you use the DEFINE GENERIC command to create initial versions. Use the RESERVE and REPLACE commands to create new versions.

If the generic element is an uncontrolled versioned element, use the DEFINE GENERIC command to create both initial and new versions.

By default, CDO automatically assigns a directory name that is the same as the element name of the generic element that you define. However, you can assign a processing name to a generic element that is different from its directory name.

When you define a property for a generic element, the property you specify must be a defined or inherited property for the element's type. Any values you specify for the property must be compatible with the data type indicated in the property type definition. Likewise, any relationship member you specify must be compatible with the relationship name's type. See the *Oracle CDD/Repository Information Model Volume I* for more information on valid properties and members.

When you work with extended types, include the MCS_processingName property in your type definition. If you omit one of these properties, your type definition does not allow you to specify a processing name for generic elements based on it. Without a processing name, you *cannot* use the CHANGE GENERIC command to add or delete properties and relationships for a generic element.

Caution

Specify the MCS_processingName property, not the CDD$PROCESSING_NAME property, when you work with extended types. Otherwise, you experience performance degradation in the Oracle CDD/Repository callable interface.

Examples

1.
```
CDO>  DEFINE GENERIC CDD$SOURCE_MODULE INPUT_MODULE_COB
cont>   MCS_PROCESSINGNAME "INPUT_MODULE_COB".
cont> END CDD$SOURCE_MODULE INPUT_MODULE_COB.
```

   In this example, the DEFINE GENERIC command creates a generic element named INPUT_MODULE_COB based on the type CDD$SOURCE_MODULE supplied by Oracle CDD/Repository.

2.
```
CDO> DEFINE GENERIC CDD$SOURCE_MODULE OUTPUT_MODULE_COB
cont>   MCS_PROCESSINGNAME "OUTPUT_MODULE_COB".
cont> END CDD$SOURCE_MODULE OUTPUT_MODULE_COB.
```

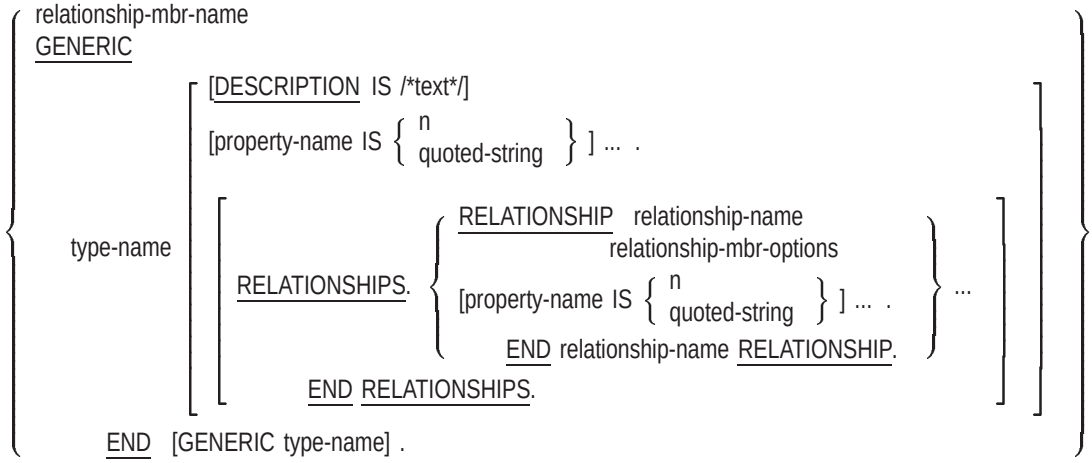
   In this example, the DEFINE GENERIC command creates a generic element named OUTPUT_MODULE_COB based on the product-supplied CDD$SOURCE_MODULE element type.

3.
```
CDO>  DEFINE GENERIC BOOK CDO_REFERENCE_MANUAL
cont>   MCS_PROCESSINGNAME IS "CDO_REFERENCE_MANUAL"
cont>   LIBRARY_NUMBER IS "AA-KL45A-TE".
cont> END BOOK CDO_REFERENCE_MANUAL.
```

   In this example, the DEFINE GENERIC command creates a generic element named CDO_REFERENCE_MANUAL based on the user-supplied type BOOK.

DEFINE GENERIC: Relationship Member Options Clause

Format



Parameters

relationship-mbr-name
Specifies an existing element that can be a valid member of the relationship type that you use.

type-name
Specifies the type of the generic element member you are defining. The type must be compatible with the relationship type.

text
Documents the generic element that you are creating as a relationship member. Within the DESCRIPTION clause, this is information documenting the member definition. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

property-name
Specifies a property.

n
Sets the numeric value for a property.

quoted-string
Sets the value (a string enclosed in quotation marks) for the property you are specifying.

relationship-name
Specifies the relationship type you are defining for the generic element member. The type must be a subtype of RELATION.

relationship-mbr-options
Specifies a relationship member. This member can be an existing element in a repository, or it can be an element you create with the DEFINE GENERIC Relationship Member Options clause.

Description

The Relationship Mbr Options clause allows you to specify a relationship member. This member can be an existing element in the repository, or it can be an element you create within the DEFINE GENERIC Relationship Member Options clause.

To specify an existing element as a relationship member, include only the element's name in the Relationship Member Options clause. To define a new element as a relationship member, specify the Generic clause within the Relationship Member Options clause.

The Generic clause does not create directory names for relationship members. Without a directory name, you cannot display elements with the DIRECTORY command or include element definitions in other definitions.

To display relationship members, issue the SHOW GENERIC command with the /FULL qualifier. To provide a directory name for a relationship member, issue the ENTER command.

If you use the Generic clause, you can nest a series of relationship members.

Examples

1.
```
CDO>  DEFINE GENERIC CDD$EXECUTABLE_IMAGE MY_PROGRAM_EXE
cont>   MCS_PROCESSINGNAME "MY_PROGRAM_EXE".
cont> RELATIONSHIPS.
cont>   RELATIONSHIP CDD$IMAGE_DERIVED_FROM
cont>     GENERIC CDD$COMPILED_MODULE
cont>     MCS_PROCESSINGNAME "INPUT_MODULE_OBJ".
cont>       RELATIONSHIPS.
cont>         RELATIONSHIP CDD$COMPILED_DERIVED_FROM
cont>           INPUT_MODULE_COB.
cont>         END CDD$COMPILED_DERIVED_FROM RELATIONSHIP.
cont>       END RELATIONSHIPS.
cont>     END CDD$COMPILED_MODULE INPUT_MODULE_OBJ.
cont>   END CDD$IMAGE_DERIVED_FROM RELATIONSHIP.
cont>   RELATIONSHIP CDD$IMAGE_DERIVED_FROM
cont>     GENERIC CDD$COMPILED_MODULE
cont>     MCS_PROCESSINGNAME "OUTPUT_MODULE_OBJ".
cont>       RELATIONSHIPS.
cont>         RELATIONSHIP CDD$COMPILED_DERIVED_FROM
cont>           OUTPUT_MODULE_COB.
cont>         END CDD$COMPILED_DERIVED_FROM RELATIONSHIP.
cont>       END RELATIONSHIPS.
cont>     END CDD$COMPILED_MODULE OUTPUT_MODULE_OBJ.
cont>   END CDD$IMAGE_DERIVED_FROM RELATIONSHIP.
cont> END RELATIONSHIPS.
cont> END CDD$EXECUTABLE_IMAGE MY_PROGRAM_EXE.
```

In this example, the following steps are performed:

a. Defines the generic entity MY_PROGRAM_EXE.

b. Specifies the processing name MY_PROGRAM_EXE for the CDD$PROCESSING_NAME attribute.

c. Defines the list of relationships that the definition MY_PROGRAM_ EXE includes.

d. Specifies a relationship type CDD$IMAGE_DERIVED_FROM, supplied by Oracle CDD/Repository, that the definition MY_PROGRAM_EXE owns.

e. The GENERIC clause creates a generic entity as a relationship member of the CDD$IMAGE_DERIVED_FROM relationship, based on the CDD$COMPILED_MODULE entity type.

f. Specifies the processing name INPUT_MODULE for the attribute type CDD$PROCESSING_NAME, supplied by Oracle CDD/Repository.

g. Begins the list of relationships that the generic entity definition INPUT_MODULE_OBJ includes.

h. Specifies the relationship type CDD$COMPILED_DERIVED_FROM, which is supplied by Oracle CDD/Repository, as a relationship owned by the generic entity INPUT_MODULE_OBJ. This relationship type specifies the generic entity INPUT_MODULE_COB (based on the entity type CDD$SOURCE_MODULE) as its relationship member.

i. Ends the relationship definition of CDD$COMPILED_DERIVED_ FROM.

j. Ends the list of relationships the generic entity definition INPUT_ MODULE_OBJ owns.

k. Ends the definition of the generic entity INPUT_MODULE_OBJ.

l. Ends the CDD$IMAGE_DERIVED_FROM relationship definition that MY_PROGRAM_EXE owns.

m. Specifies a relationship type CDD$IMAGE_DERIVED_FROM, supplied by Oracle CDD/Repository, that the generic entity definition MY_ PROGRAM_EXE owns.

n. The GENERIC clause creates a generic entity as a relationship member of the CDD$IMAGE_DERIVED_FROM relationship, based on the CDD$COMPILED_MODULE entity type.

o. Specifies the processing name OUTPUT_MODULE_OBJ for the attribute type CDD$PROCESSING_NAME, which is supplied by Oracle CDD/Repository.

p. Begins the list of relationships that the new generic entity definition OUTPUT_MODULE_OBJ includes.

q. Specifies the relationship type CDD$COMPILED_DERIVED_FROM, supplied by Oracle CDD/Repository, as a relationship owned by the generic entity OUTPUT_MODULE_OBJ. This relationship type specifies the generic entity OUTPUT_MODULE_COB (based on the entity type CDD$SOURCE_MODULE as its relationship member.

r. Ends the relationship definition of CDD$COMPILED_DERIVED_ FROM.

s. Ends the list of relationships that the generic entity definition OUTPUT_MODULE_OBJ owns.

t. Ends the definition of the generic entity definition OUTPUT_ MODULE_OBJ.

u. Ends the CDD$IMAGE_DERIVED_FROM relationship definition that MY_PROGRAM_EXE owns.

v. Ends the list of relationships the generic entity MY_PROGRAM_EXE owns.

w. Ends the definition of the generic entity MY_PROGRAM_EXE.

Because the GENERIC clause of the DEFINE GENERIC command creates the INPUT_OBJ and OUTPUT_OBJ generic entities, these entities do not have directory names. You can view their definitions only with the SHOW GENERIC/FULL command, which displays their owner (MY_PROGRAM_ EXE entity).

2.
```
CDO> SHOW GENERIC CDD$EXECUTABLE_IMAGE/FULL MY_PROGRAM_EXE
Definition of MY_PROGRAM_EXE   (Type : CDD$EXECUTABLE_IMAGE)
 |   Contains CDD$IMAGE_DERIVED_FROM
 |   |   INPUT_MODULE_OBJ                    (Type : CDD$COMPILED_MODULE)
 |   |   |   Contains CDD$COMPILED_DERIVED_FROM
 |   |   |   |   INPUT_MODULE_COB              (Type : CDD$SOURCE_MODULE)
 |   Contains CDD$IMAGE_DERIVED_FROM
 |   |   OUTPUT_MODULE_OBJ                   (Type : CDD$COMPILED_MODULE)
 |   |   |   Contains CDD$COMPILED_DERIVED_FROM
 |   |   |   |   OUTPUT_MODULE_COB             (Type : CDD$SOURCE_MODULE)
```

In this example, the DEFINE GENERIC command creates the generic element definition MY_PROGRAM_EXE, based on the type CDD$EXECUTABLE_IMAGE. The first relationship defined is the CDD$IMAGE_DERIVED_FROM relation, supplied by Oracle CDD/Repository, which in turn owns the CDD$COMPILED_DERIVED_ FROM relation, also supplied by Oracle CDD/Repository. The second relationship defined is a CDD$IMAGE_DERIVED_FROM relation, which in turn owns another CDD$COMPILED_DERIVED_FROM relation.

The first relationship links the executable image and its compiled modules. The second relationship links the compiled modules and the source module.

3.
```
CDO> DEFINE GENERIC BOOK CDD_PLUS_REFERENCE_MANUAL
cont> MCS_processingName IS "CDD_PLUS_REFERENCE_MANUAL"
cont> LIBRARY_NUMBER IS "AA-KL45A-TE".
cont> END BOOK CDD_PLUS_REFERENCE_MANUAL.
```

This example creates an entity named CDD_PLUS_REFERENCE_ MANUAL based on the BOOK protocol.

4.
```
CDO> DEFINE GENERIC LIBRARY ORACLE_LIBRARY
cont> MCS_processingName IS "ORACLE_LIBRARY"
cont> ADDRESS IS "NASHUA, NH".
cont> RELATIONSHIPS.
cont> RELATIONSHIP BOOK_IN_LIBRARY CDD_PLUS_REFERENCE_MANUAL
cont> END BOOK_IN_LIBRARY RELATIONSHIP.
cont> END RELATIONSHIPS.
cont> END LIBRARY ORACLE_LIBRARY.
```

This example creates an entity named ORACLE_LIBRARY based on the LIBRARY protocol.

5.
```
CDO> DEFINE GENERIC LIBRARY ORACLE_LIBRARY
cont> MCS_processingName IS "ORACLE_LIBRARY".
cont> RELATIONSHIPS.
cont> RELATIONSHIP BOOK_IN_LIBRARY CDO_REFERENCE_MANUAL.
cont> END BOOK_IN_LIBRARY RELATIONSHIP.
cont> RELATIONSHIPS.
cont> GENERIC BOOK USER_GUIDE
cont> MCS_processingName IS "USER_GUIDE".
cont> END RELATIONSHIPS.
cont> END LIBRARY ORACLE_LIBRARY.
```

This command creates a relationship member using the DEFINE GENERIC command.

6.
```
CDO> DEFINE GENERIC LIBRARY ORACLE_LIBRARY
cont>    MCS_processingName IS "ORACLE_LIBRARY"
cont>    ADDRESS IS "NASHUA, NH".
cont>    RELATIONSHIPS.
cont>      RELATIONSHIP BOOK_IN_LIBRARY
cont>        GENERIC BOOK
cont>        MCS_processingName IS "USER_GUIDE"
cont>    END RELATIONSHIPS.
cont> END LIBRARY ORACLE_LIBRARY.
```

This example uses the DEFINE GENERIC command to define the relationship member USER_GUIDE. Note that the keyword DEFINE and the relationship member's entity name have been omitted.

7.
```
CDO> DEFINE GENERIC LIBRARY ORACLE_LIBRARY
cont>    MCS_processingName IS "ORACLE_LIBRARY"
cont>    ADDRESS IS "NASHUA, NH".
cont>    RELATIONSHIPS.
cont>      RELATIONSHIP BOOK_IN_LIBRARY
cont>        GENERIC BOOK
cont>        MCS_processingName IS "USER_GUIDE"
cont>        LIBRARY_NUMBER IS 1.
cont>        END GENERIC.
cont>      END BOOK_IN_LIBRARY RELATIONSHIP.
cont>    END RELATIONSHIPS.
cont> END LIBRARY ORACLE_LIBRARY.
```

This example uses the DEFINE GENERIC command to define the relationship member USER_GUIDE. Note that the keyword DEFINE and the relationship member's entity name have been omitted.

8.
```
CDO> SHOW GENERIC LIBRARY ORACLE_LIBRARY
```

To view the definition of the relationship member created with the DEFINE GENERIC command in the previous example, you need to use the SHOW GENERIC command for its owner, ORACLE_LIBRARY.

9.
```
CDO>  DEFINE GENERIC CDD$RDB_DATABASE PERSONNEL DESCRIPTION IS
cont> "DEFINE RDB_DATABASE PERSONNEL, CONTAINING RECORD EMPLOYEE_REC"
cont> CDD$PROCESSING_NAME IS "PERSONNEL".
cont> RELATIONSHIPS.
cont>  RELATIONSHIP CDD$RDB_DATA_AGGREGATE
cont>    GENERIC CDD$DATA_AGGREGATE DESCRIPTION IS
cont>    "DEFINE DATA AGGREGATE EMPLOYEE_REC CONTAINING FIRST_NAME,"
cont>    "LAST_NAME, EMP_ID"
cont>    CDD$PROCESSING_NAME IS "EMPLOYEE_REC".
cont>      RELATIONSHIPS.
cont>        RELATIONSHIP CDD$DATA_AGGREGATE_CONTAINS
cont>          GENERIC CDD$DATA_ELEMENT DESCRIPTION IS
cont>          "DEFINE DATA ELEMENT FIRST_NAME = FIELD FIRST_NAME"
cont>          CDD$PROCESSING_NAME IS "FIRST_NAME"
cont>          CDD$DATA_ELEMENT_DATATYPE 14
cont>          CDD$DATA_ELEMENT_LENGTH 10.
cont>          END CDD$DATA_ELEMENT FIRST_NAME.
cont>        CDD$DATA_SEQUENCE_NUMBER IS 1.
cont>        END CDD$DATA_AGGREGATE_CONTAINS RELATIONSHIP.
cont>        RELATIONSHIP CDD$DATA_AGGREGATE_CONTAINS
cont>          GENERIC CDD$DATA_ELEMENT DESCRIPTION IS
cont>          "DEFINE DATA ELEMENT LAST_NAME = FIELD LAST_NAME"
cont>          CDD$PROCESSING_NAME IS "LAST_NAME"
cont>          CDD$DATA_ELEMENT_DATATYPE 14
cont>          CDD$DATA_ELEMENT_LENGTH 15.
cont>          END CDD$DATA_ELEMENT LAST_NAME.
cont>        CDD$DATA_SEQUENCE_NUMBER IS 2.
cont>        END CDD$DATA_AGGREGATE_CONTAINS RELATIONSHIP.
```

```
cont>        RELATIONSHIP CDD$DATA_AGGREGATE_CONTAINS
cont>          GENERIC CDD$DATA_ELEMENT DESCRIPTION IS
cont>          "DEFINE DATA ELEMENT EMP_ID = FIELD EMP_ID"
cont>          CDD$PROCESSING_NAME IS "EMP_ID"
cont>          CDD$DATA_ELEMENT_DATATYPE 4.
cont>          END CDD$DATA_ELEMENT EMP_ID.
cont>        CDD$DATA_SEQUENCE_NUMBER IS 3.
cont>        END CDD$DATA_AGGREGATE_CONTAINS RELATIONSHIP.
cont>      END RELATIONSHIPS.
cont>    END CDD$DATA_AGGREGATE EMPLOYEE_REC.
cont>  END CDD$RDB_DATA_AGGREGATE RELATIONSHIP.
cont>END RELATIONSHIPS.
cont>END CDD$RDB_DATABASE PERSONNEL.
```

This example defines an Oracle Rdb database PERSONNEL and creates a CDD$RDB_DATABASE entity PERSONNEL in the CDO repository. This database contains one record EMPLOYEE_REC, which in turn contains three fields that were defined by nesting the GENERIC clauses: FIRST_NAME, LAST_NAME, and EMP_ID. (It is recommended that you define Oracle Rdb databases in the CDO repository through the SQL. See *Oracle Rdb7 Guide to Database Design and Definition* for more information on how to use CDO with Oracle Rdb databases.)

10.
```
CDO> SHOW GENERIC CDD$RDB_DATABASE/AUDIT=ALL PERSONNEL
Definition of PERSONNEL   (Type : CDD$RDB_DATABASE)
|   |   History entered by SMITH ([CDD,SMITH])
|   |    using CDO V1.0
|   |    to CREATE definition on 15-DEC-1987 10:31:11.59
|   Contains CDD$RDB_DATA_AGGREGATE
|   |   EMPLOYEE_REC                    (Type : CDD$DATA_AGGREGATE)
|   |   |   Contains CDD$DATA_AGGREGATE_CONTAINS
|   |   |   |   FIRST_NAME                        (Type : CDD$DATA_ELEMENT)
|   |   |   Contains CDD$DATA_AGGREGATE_CONTAINS
|   |   |   |   LAST_NAME                         (Type : CDD$DATA_ELEMENT)
|   |   |   Contains CDD$DATA_AGGREGATE_CONTAINS
|   |   |   |   EMP_ID                            (Type : CDD$DATA_ELEMENT)
CDO>
```

This example displays the history list of every element owned by the PERSONNEL database by using the SHOW GENERIC command.

DEFINE KEY Command

Format

DEFINE KEY [qualifier] ... key-name key-equivalence

Parameters

key-name
Specifies the key you are defining.

key-equivalence
Specifies the character string you want processed when you press the key. Enclose the string in quotation marks to preserve spaces and lowercase characters.

Qualifiers

/ECHO (default)
/NOECHO
Specifies whether CDO displays the equivalence string on your terminal screen after you press a key. The default is ECHO, which displays the equivalence string.

You cannot use the /NOECHO qualifier with the /NOTERMINATE qualifier.

/IF_STATE=state-name
/NOIF_STATE (default)
Specifies the state that must be in effect for a key definition to work. If you omit the /IF_STATE qualifier or use the /NOIF_STATE qualifier, CDO uses the current state. The state name is an alphanumeric string. The /SET_STATE qualifier or the SET KEY command establishes the state.

/LOCK_STATE
/NOLOCK_STATE (default)
Specifies whether the state set by the /SET_STATE qualifier remains in effect until a user explicitly changes it. By default, the /SET_STATE qualifier is in effect only for the next definable key you press or the next read-terminating character that you type.

If you specify the /LOCK_STATE qualifier, you must also specify the /SET_STATE qualifier.

/PROTECTED
/NOPROTECTED (default)
Specifies whether CDO protects a key against later redefinition. The default is no protection against redefinition.

/SET_STATE=state-name
/NOSET_STATE (default)
Specifies a new state for CDO to set when you press a key; by default, CDO resets the current locked state. If you have not included this qualifier in a key definition, you can use the SET KEY command to change the current state. The state name can be any alphanumeric string.

/TERMINATE
/NOTERMINATE (default)
Specifies whether CDO immediately processes the key definition when you press the key (equivalent to typing the string and pressing the Return key).

The default is NOTERMINATE, which allows you to press other keys before CDO processes the definition. The /NOTERMINATE qualifier allows you to create key definitions that insert text into command lines, after prompts, or into other text that you are typing.

You cannot use the /NOTERMINATE qualifier with the /NOECHO qualifier.

Description

The DEFINE KEY command assigns definitions to the peripheral keys on certain terminals. These definitions can direct CDO to perform one of the following actions:

- Execute a CDO command
- Append a qualifier to a CDO command
- Append a text string to a CDO or system-level command

When you define a key to insert a text string, use the /NOTERMINATE qualifier so that you can continue typing more data after CDO inserts the string.

You should take advantage of the echo feature in most instances. With /ECHO set, CDO displays the key definition on the screen each time you press the key.

You can use the /SET_STATE qualifier to increase the number of key definitions available on your terminal keyboard. You can assign the same key any number of definitions, as long as you associate each definition with a different state. State names can contain alphanumeric characters, dollar signs, and underscores.

See the SET KEY command for information on changing keypad states.

Table 1–3 lists the keys you can define on the keyboards of different terminals.

Table 1–3 Redefineable Key Names and Terminal Designations

Key Name	VT100-series	VT200- and VT300-series
PF1	PF1	PF1
PF2	PF2	PF2
PF3	PF3	PF3
PF4	PF4	PF4
KP0, KP1, ..., KP9	0, 1, ..., 9	0, 1, ..., 9
PERIOD	.	.
COMMA	,	,
MINUS	-	-
ENTER	ENTER	ENTER
LEFT	←	←
RIGHT	→	→
E1	–	FIND
E2	–	INSERT HERE
E3	–	REMOVE
E4	–	SELECT
E5	–	PREV SCREEN
E6	–	NEXT SCREEN
HELP	–	HELP
DO	–	DO
F6, F7, ..., F20	–	F6, F7, ..., F20

Examples

```
CDO>  DEFINE KEY /TERMINATE PF3 "SHOW DEFAULT"
```

In this example, the DEFINE KEY command assigns the CDO SHOW DEFAULT command to the PF3 key. CDO executes the SHOW DEFAULT command when you press the PF3 key.

DEFINE PARTITION Command

Format

```
DEFINE  PARTITION  partition-name

        [ DESCRIPTION  IS /*text*/ ]    [ AUDIT  IS /*text*/ ]

        [ PARENT_PARTITION IS parent-partition-name
          LOOKASIDE_PARTITION IS look-partition-name ,... ]  ...

        [ AUTOPURGE
          NOAUTOPURGE ]  .
```

Parameters

partition-name
Specifies the partition you are creating.

text
Adds information. Within the DESCRIPTION clause, this is information documenting the partition; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

parent-partition-name
Specifies the parent partition, which must currently exist.

look-partition-name
Specifies a related partition that is visible through this partition. The related partition must currently exist.

Description

The DEFINE PARTITION command creates a partition. Partitions are the means by which you control elements.

When you control an element, you identify the partition, which is called the base partition, in which a public, immutable copy of this element resides. CDO provides two ways to control elements, as follows:

- On an element-by-element basis, with the CONSTRAIN command. CDO controls the element that appears within the command.
- Through a context with the DEFINE CONTEXT and SET CONTEXT commands. Once you set the context, CDO controls all subsequent elements until the context is changed.

Once an element has been controlled, you use the RESERVE and REPLACE commands to create subsequent versions. This reservation system prevents uncontrolled changes to elements.

You can link partitions together to control change in various stages of a project. Each partition then represents a higher level of approval, or completion, in the overall partition hierarchy. The PROMOTE command moves elements higher within the hierarchy.

The PARENT_PARTITION clause in the DEFINE PARTITION command creates a partition hierarchy by linking partitions in a parent-child relationship. The first, or root, partition does not have a parent partition. The second partition in the hierarchy has the first partition as its parent, and so on down the hierarchy. This clause can be specified only once during the lifetime of the partition, in either the DEFINE PARTITION or CHANGE PARTITION command.

The LOOKASIDE_PARTITION clause makes the contents of another partition visible, provided that you have read privileges for the partition. You can read, but you cannot reserve, replace, or change the contents.

The AUTOPURGE keyword ensures that CDO automatically purges intermediate versions of elements in the partition when you promote the latest version. The NOAUTOPURGE keyword prevents this automatic purging.

Examples

```
CDO>  DEFINE PARTITION FINAL_RELEASE AUTOPURGE.        1
CDO>  DEFINE PARTITION FIELDTEST_RELEASE               2
cont>   PARENT_PARTITION IS FINAL_RELEASE AUTOPURGE.
CDO>  DEFINE PARTITION SECOND_BASELEVEL
cont>   PARENT_PARTITION IS FIELDTEST_RELEASE AUTOPURGE.
CDO>  DEFINE PARTITION FIRST_BASELEVEL
cont>   PARENT_PARTITION IS SECOND_BASELEVEL AUTOPURGE.
CDO>  DEFINE PARTITION FRONT_END
cont>   PARENT_PARTITION IS FIRST_BASELEVEL AUTOPURGE.
CDO>  DEFINE PARTITION BACK_END                        3
cont>   PARENT_PARTITION IS FIRST_BASELEVEL
cont>   LOOKASIDE_PARTITION IS FRONT_END AUTOPURGE.
CDO>  CHANGE PARTITION FRONT_END                       4
cont>   LOOKASIDE_PARTITION IS BACK_END.

   .
   .
   .
CDO>  DEFINE CONTEXT BILL_CONTEXT                      5
cont>   BASE_PARTITION IS FRONT_END.
CDO>  DEFINE CONTEXT BETSY_CONTEXT
cont>   BASE_PARTITION IS BACK_END.
CDO>  DEFINE CONTEXT QA_CONTEXT
cont>   BASE_PARTITION IS FIELDTEST_RELEASE.
```

In this example, successive DEFINE PARTITION commands create a partition hierarchy.

1 The root partition is FINAL_RELEASE.

2 Each successive partition in the hierarchy is the child of the previous partition.

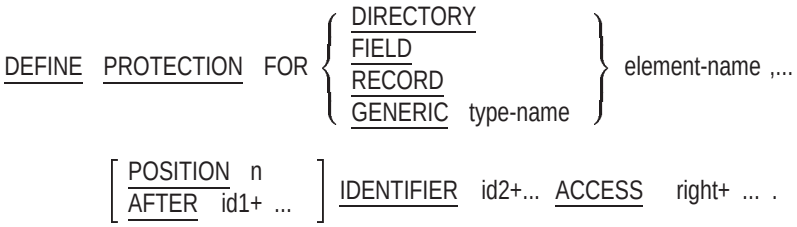
3 A partition hierarchy can include multiple children of a previous partition; LOOKASIDE_PARTITION makes the contents of FRONT_END visible to BACK_END.

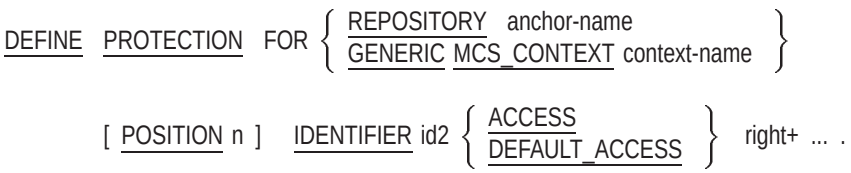
4 The CHANGE PARTITION command makes the contents of BACK_END visible to FRONT_END.

5 The base partition, or lowest visible partition, is set for three different contexts.

DEFINE PROTECTION Command

Format





Parameters

type-name
Specifies the type of the generic element whose ACE you are defining.

element-name
Specifies the element whose ACE you are defining. You can use wildcard characters in this name.

n
Specifies the relative position (a positive integer) in the ACL of the ACE you are defining. If you omit the position or the identifier, the ACE you are defining becomes the first ACE in the ACL.

id1
Specifies the identifier or identifiers of the existing ACE that will immediately precede the ACE you are defining.

id2
Specifies the identifier or identifiers of those users whose access to the element or repository you are defining in this ACE.

right
Specifies the access rights CDO grants to the users you specified in id2.

anchor-name
Specifies the anchor directory of the repository whose ACE you are defining.

context-name
Specifies the context.

Description

The DEFINE PROTECTION command adds an access control list entry (ACE) to the access control list (ACL) of an element or repository. When you specify FOR GENERIC MCS_CONTEXT or FOR REPOSITORY, this command can also add an ACE to a default access control list. To define protection, you need CONTROL access.

The ACEs in an ACL determine which users can access the element or repository and what operations each user can perform. An ACE consists of the following two parts:

- One or more identifiers that specifies a user or set of users: UIC, general, and system-defined
- A set of access rights: READ, WRITE, EXECUTE, and DELETE

The POSITION clause specifies the relative position CDO assigns your ACE in the ACL. ACEs are numbered in ascending order starting with number one. If you specify a number that is larger than the number of ACEs in the ACL, the ACE you are creating becomes the last entry in the ACL.

The AFTER clause specifies the identifiers of an existing ACE that will immediately precede the ACE that you are defining.

The IDENTIFIER clause specifies the identifiers of the user or users whose access to the element or repository you are defining in this ACE. If an ACE contains more than one identifier, a user's process must hold all the identifiers specified in the ACE to receive the access rights granted by the ACE.

The ACCESS clause specifies the rights that the ACE provides. This clause is especially useful when you need to restrict access to a context or to a repository. For example, by modifying this clause, you can restrict access to a single user for OpenVMS BACKUP or VERIFY operations.

The DEFAULT_ACCESS clause is only valid for contexts (specified as GENERIC MCS_CONTEXT) or repositories. The clause specifies the default access rights for each new element you create. If a context is set, the new element receives default access rights defined for this context. If a context is not set, the new element receives the default access rights defined for the repository.

For complete information on defining protection, see *Using Oracle CDD/Repository on OpenVMS Systems*.

Examples

1.
```
CDO>  DEFINE PROTECTION RECORD PERSONNEL
cont>   POSITION 2
cont>   IDENTIFIER [JONES,DICT]+LOCAL+INTERACTIVE
cont>   ACCESS READ+WRITE+DELETE.
```

In this example, the DEFINE PROTECTION command creates a new second ACE for the PERSONNEL record. The former second ACE becomes the new third ACE.

2.
```
CDO>  DEFINE PROTECTION FOR RECORD PERSONNEL
cont>   AFTER [JONES,DICT]+LOCAL+INTERACTIVE
cont>   IDENTIFIER [CDD,SMITH] ACCESS READ.
```

In this example, the DEFINE PROTECTION command inserts a new ACE with the identifier [CDD,SMITH] after the ACE with the [JONES,DICT]+LOCAL+INTERACTIVE identifiers.

3.
```
CDO>  DEFINE PROTECTION FOR RECORD BENEFITS.*;* POSITION 4
cont>   IDENTIFIER [PERSONNEL,*] ACCESS SHOW.
```

In this example, the DEFINE PROTECTION command creates a fourth ACE for all current records in the BENEFITS directory. This ACE does not become the default protection for definitions that are subsequently created.

4.
```
CDO>  DEFINE PROTECTION FOR RECORD PERSONNEL
cont>   IDENTIFIER [*,*] ACCESS NONE.
```

In this example, the DEFINE PROTECTION command creates an ACE that denies all access rights to all users. CDO places this ACE first in the ACL, because the user did not specify either a POSITION clause or an AFTER clause in the command. As a result, everyone (including the user who issued the command) is denied all access to the definition.

Only the owner can regain access to the definition by using either the DELETE PROTECTION or CHANGE PROTECTION command to remove or change the ACE.

5.
```
CDO>  DEFINE PROTECTION FOR REPOSITORY PERSONNEL
cont>   POSITION 2 IDENTIFIER [SYSTEM]
cont>   ACCESS READ+WRITE+DELETE+CONTROL.
```

In this example, the DEFINE PROTECTION command creates an ACE in the second position that grants the SYSTEM user READ+WRITE+DELETE+CONTROL access.

6.
```
CDO>  DEFINE PROTECTION FOR REPOSITORY PERSONNEL
cont>   POSITION 2 IDENTIFIER [*,*]
cont>   DEFAULT_ACCESS READ+WRITE.
CDO>  DEFINE FIELD NEW_FIELD DATATYPE TEXT SIZE 5.
```

In this example, the DEFINE PROTECTION command defines the default access rights for the PERSONNEL repository to READ+WRITE. If a context has not been set, CDO will grant the newly created field, NEW_FIELD, with access rights that are equivalent to the repository's default access rights.

7.
```
CDO> DEFINE PROTECTION FOR REPOSITORY CDD$REPOSITORY2
cont> POSITION 2 IDENTIFIER [*,*]
cont> ACCESS NONE.
```

In this example, the DEFINE PROTECTION command defines the access rights for the repository using a logical name for the repository name.

Protecting the Repository Anchor

Oracle CDD/Repository places a security ACL on repository anchors when a new repository is created, when a repository is moved, or when the location of the repository is changed with the CDO command VERIFY/LOCATION/FIX.

The ACL is as follows:

```
(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)
(IDENTIFIER=[*,*],OPTIONS=DEFAULT+NOPROPAGATE,ACCESS=NONE)
```

To add these ACLs to existing repository anchors on your system, you can use either one of the following methods:

- OpenVMS SET ACL/ACL command
- ACL Editor

In addition to this default protection, you should add UIC-based protection with either of the following commands:

- OpenVMS SET PROTECTION command
- OpenVMS CREATE/DIRECTORY/PROTECTION command

For more information about setting OpenVMS protection on a repository's OpenVMS anchor directory, see the OpenVMS Examples at the end of this section.

Examples

1.
```
$ SET ACL/ACL=(IDENTIFIER=CDD$SYSTEM, -
_$ ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL) -
_$ [SMITH]DIC.DIR(1)

$ SET ACL/ACL=(IDENTIFIER=[*,*],ACCESS=READ) [SMITH]DIC.DIR(1)
```

   Protect your repository anchor directory with an ACL containing the ACEs shown in the previous example. With these ACEs, only repository files can be created in a repository anchor directory.

   In this example, the SET ACL/ACL command creates an ACL for the OpenVMS anchor directory of the [SMITH.DIC] repository.

2.
```
$ SET ACL/EDIT [SMITH]DIC.DIR(1)
$ EDIT/ACL [SMITH]DIC.DIR(1)
```

   In this example, either the DCL SET ACL/EDIT command or the DCL EDIT/ACL command is used to create an OpenVMS anchor directory.

3.
```
$ SHOW ACL [SMITH]DIC.DIR(1)
element type: file, element name: CDD$DISK:[SMITH]DIC.DIR(1),
       on 27-FEB-1989 09:54:40.62
       (IDENTIFIER=CDD$SYSTEM,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
       (IDENTIFIER=[*,*],ACCESS=READ)
```

   In this example, the DCL SHOW ACL command is used to display the ACL you just created.

4.
```
$ SET PROTECTION=(S:RWED,,,) [SMITH]DIC.DIR(1)
```

   In this example, the DCL SET PROTECTION command creates UIC-based protection for the OpenVMS anchor directory [SMITH.DIC]. You should add UIC-based protection to your repository's OpenVMS anchor directory.

DEFINE RECORD Command

Format

```
DEFINE  RECORD  record-name

        [ DESCRIPTION  IS /*text*/ ]    [ AUDIT  IS /*text*/ ]

        [ record-property ] ...

        [ constraint-clause ] ...  .

        { included-name-clause
        { local-field-clause
        { structure-name-clause
        { variants-clause                } ... .

        END [ record-name ] RECORD  .
```

Parameters

record-name
Specifies the record element you are creating.

text
Adds information. Within the DESCRIPTION clause, this is information documenting the record element; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

record-property
Adds a property to the record element. See Chapter 2 for the record properties CDO provides.

constraint-clause
Specifies a condition that affects adding or modifying data to the 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 DEFINE RECORD: Constraint Clause for more information.

included-name-clause
Allows you to include existing field definitions and record definitions within record elements. See the DEFINE RECORD: Included Name Clause for more information.

local-field-clause
Allows you to create local field definitions within record elements. Describes the attributes of the local field. See the DEFINE RECORD: Local Field Clause for more information.

structure-name-clause
Creates structure definitions within record elements. See the DEFINE RECORD: Structure Name Clause for more information.

variants-clause
Creates variants definitions within record elements. See the DEFINE RECORD: Variants Clause for more information.

Description

The DEFINE RECORD command creates a record element.

If you supply a record name that is already used for a record element in your default directory, CDO creates a new version of the existing record definition.

The DEFINE RECORD command evaluates the record name you supply to determine if it is a logical name. If the record name is a logical name, CDD translates it. In some cases, the translation of the logical name for the record may not be a valid name for a record definition, and CDO will not create the record definition. For example, if you have defined JOE as a logical name that translates to MYNODE::[RICHIE], CDD translates the symbol JOE. The following DEFINE RECORD command fails because MYNODE::[RICHIE] is not a valid name:

```
CDO> DEFINE RECORD JOE.
%CDO-F-ERRDEFINE, error defining object
-CDD-F-NOTADIC, Does not contain a CDO dictionary:
MYNODE::
```

If this error occurs, deassign the logical name with the same name as the object, and perform the operation again. To avoid this logical name conflict, use unique names that represent the type of entity you are naming.

Examples

```
CDO>  DEFINE RECORD EDUCATION_RECORD.
cont>   BADGE_NUMBER.
cont>   BACHELOR_DEGREE.
cont>   MASTER_DEGREE.
cont>   DOCTORATE_DEGREE.
cont> END RECORD.
```

In this example, the DEFINE RECORD command creates the EDUCATION_RECORD record definition from four existing field definitions.

DEFINE RECORD: Constraint Clause

Format

```
CONSTRAINT  constr-name
                [ UNIQUE field-name, ...                                                    ]
                [ PRIMARY KEY field-name, ...                                               ]
                [ FOREIGN KEY field-name, ... REFERENCES record-name field-name, ...        ]
                [ CHECK (expression)                                                        ]

                [ DEFERRABLE     ]
                [ NOT DEFERRABLE ]
```

Parameters

constr-name
Specifies the name of the constraint.

field-name
Specifies the name of the field to be used in a key or a field that is unique.

record-name
Specifies the name of the record.

expression
Specifies a Boolean expression. See Chapter 4 for more information.

Description

Use to specify a condition that affects adding or modifying data to the 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.

Examples

```
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.
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
```

This example uses the CDO DEFINE RECORD command syntax to establish constraints on the PARTS record.

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.

This example assumes the PART_ID to be the primary key and the PART_NO to be a unique value across all possible parts. By not specifying whether the constraints are deferrable, the default evaluation time is accepted. In CDO, the default evaluation time for constraints is NOT DEFERRABLE. Constraints are evaluated at statement time.

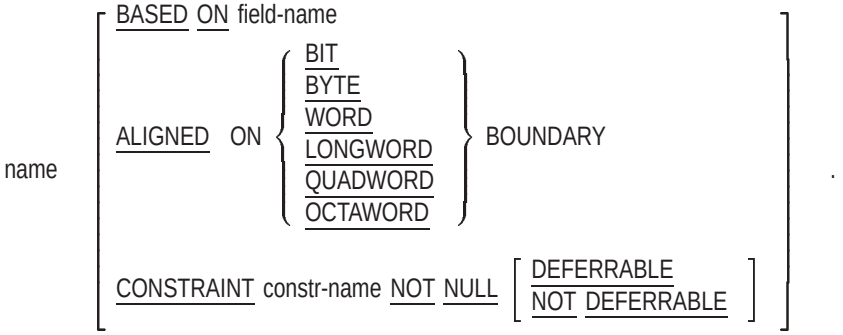
DEFINE RECORD: Constraint Clause

Using CDO, the record PARTS is defined with the following attributes:

- Primary key PARTS_PMK
- Unique constraint PARTS_UNQ
- Check constraint PART_CST
- Foreign key constraint PART_FRK

DEFINE RECORD: Included Name Clause

Format



Parameters

name
Specifies the existing field or record definition you want to include in the record element you are creating. The named field or record definition must already exist in the repository.

field-name
Specifies the name of the field to be used in a key or a field that is unique.

constr-name
Specifies the name of the constraint for the local field definition you are changing. See the DEFINE RECORD command for more information on constraints.

Description

The Included Name clause allows you to specify global field definitions and record definitions within record elements.

If you do not specify a directory name as part of the included name, CDO looks for the record or field definition in your current default directory.

DEFINE RECORD: Included Name Clause

To improve performance, some languages and language processors have alignment restrictions for data definitions. The ALIGNED clause aligns a field or record definition on a specified boundary relative to the beginning of the record you are defining. Each field or record, except BIT fields, begins by default on the first byte following the last field. BIT fields begin on the bit immediately following the last field.

The ALIGNED clause aligns fields or records within a record relative to the start of the record, not relative to virtual memory locations.

For example, if you specify LONGWORD alignment for a field, that field does not necessarily begin on a longword boundary in memory. Rather, the field begins some multiple of 32 bits beyond the start of the record. To correctly use the aligned clause, you must know the memory alignment techniques of the language you use with CDO.

Examples

1.
```
CDO>  DEFINE RECORD FULL_NAME.
cont>   LAST_NAME ALIGNED ON WORD.
cont>   FIRST_NAME ALIGNED ON WORD.
cont>   MIDDLE_INITIAL ALIGNED ON WORD.
cont> END RECORD.
```

   In this example, the DEFINE RECORD command creates the FULL_NAME record element in your default directory using existing field definitions. The keyword ALIGNED starts each field definition on a word boundary.

2.
```
CDO>  DEFINE RECORD CONTRACT.HOME_ADDRESS.
cont>   STREET_ADDRESS.
cont>   CITY.
cont>   STATE.
cont>   POSTAL_CODE.
cont> END RECORD.
```

   In this example, the DEFINE RECORD command creates the HOME_ADDRESS record element using field definitions from your default directory. Because you specify a path name, HOME_ADDRESS is created in the EMPLOYEES directory.

3.
```
CDO>  SET DEFAULT DISK1:[JONES.DICT]PERSONNEL
CDO>  DEFINE RECORD CONTRACT.WORKER_REC.
cont>   FULL_NAME.
cont>   CONTRACT.DATE_HIRED.
cont>   CONTRACT.HOURLY_WAGE.
cont>   CONTRACT.COMPLETION_DATE.
cont> END RECORD.
```

In this example, the DEFINE RECORD command creates the WORKER_ REC record element in the PERSONNEL directory using field definitions from the default PERSONNEL directory and the CONTRACT directory.

DEFINE RECORD: Local Field Clause

Format

```
local-field-name  [ DESCRIPTION IS /*text*/ ]

    [ field-property   ]
    [ NOfield-property ] ...

[                                                                    ]
[ ALIGNED ON { BIT       }                                           ]
[            { BYTE      }                                           ]
[            { WORD      } BOUNDARY                                  ]
[            { LONGWORD  }                                           ]
[            { QUADWORD  }                                           ]
[            { OCTAWORD  }                                           ]
[                                                                    ]
[ NOALIGNED                                                          ]
[                                                                    ]
[ CONSTRAINT constr-name NOT NULL [ DEFERRABLE     ]                 ]
[                                 [ NOT DEFERRABLE ]                 ]
[                                                                    ]
```

Parameters

local-field-name
Specifies the name of the locally defined field.

text
Adds information. Within the DESCRIPTION clause, this is information documenting the record element; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

field-property
Defines the characteristics of the data you store in field elements. See Chapter 2 for more information.

constr-name
Specifies the name of the constraint for the local field definition you are creating or changing. See the DEFINE RECORD command for more information on constraints.

Description

The Local Field Clause allows you to specify local field definitions and record definitions within record elements.

To improve performance, some languages and language processors have alignment restrictions for data definitions. The ALIGNED clause aligns a field or record definition on a specified boundary relative to the beginning of the record you are defining.

Each field or record, except BIT fields, begins by default on the first byte following the last field. BIT fields begin on the bit immediately following the last field.

The ALIGNED clause aligns fields or records within a record relative to the start of the record, not relative to virtual memory locations.

For example, if you specify LONGWORD alignment for a field, that field does not necessarily begin on a longword boundary in memory. Rather, the field begins some multiple of 32 bits beyond the start of the record. To correctly use the aligned clause, you must know the memory alignment techniques of the language you use with CDO.

Examples

```
CDO>  DEFINE RECORD PRODUCE.
cont> UPC_CODE DATATYPE LONGWORD 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.
```

In this example, UPC_CODE is a local field.

DEFINE RECORD: Structure Name Clause

Format

```
structure-name  STRUCTURE

    [ DESCRIPTION  IS /*text*/ ]   [ record-property ] ...

    [ ALIGNED  ON { BIT
                  | BYTE
                  | WORD
                  | LONGWORD
                  | QUADWORD
                  | OCTAWORD } BOUNDARY ]  .

    [ included-name-clause
    | local-field-clause
    | structure-name-clause
    | variants-clause ] ...

    END [ structure-name ]  STRUCTURE .
```

Parameters

structure-name
Specifies the structure you are defining.

text
Documents the structure definition. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

record-property
Adds a property to the structure definition. See Chapter 2 for a list of the valid record properties.

included-name-clause
Includes existing field and record definitions within record elements. See DEFINE RECORD: Included Name Clause for more information.

local-field-clause
Specifies the locally defined field. See DEFINE RECORD: Local Field Clause for more information.

structure-name-clause
Creates structure definitions within record elements. This section describes structure definitions.

variants-clause
Creates variants definitions within record elements. See DEFINE RECORD: Variants Clause for more information.

Description

The Structure Name Clause allows you to define a structure within a record element.

A structure definition can include both field definitions and record definitions.

Examples

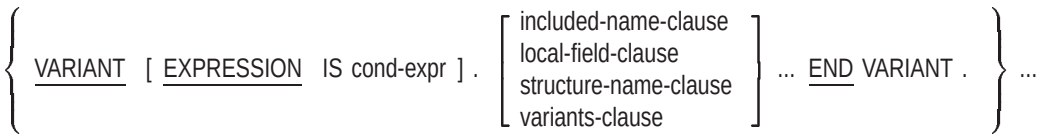
```
CDO>  DEFINE RECORD HOUSEHOLD.
cont>   ANNUAL_INCOME.
cont>   ADDRESS.
cont>   NUMBER_OF_DEPENDENTS.
cont>     DEPENDENTS STRUCTURE OCCURS 1 TO 10 TIMES
cont>       DEPENDING ON NUMBER_OF_DEPENDENTS IN HOUSEHOLD.
cont>       NAME.
cont>       AGE.
cont>       SEX.
cont>     END DEPENDENTS STRUCTURE.
cont> END HOUSEHOLD RECORD.
```

In this example, the OCCURS...DEPENDING clause in the DEPENDENTS structure specifies that the structure occurs 1 to 10 times based on the value of the NUMBER_OF_DEPENDENTS field definition in the HOUSEHOLD record element at runtime.

DEFINE RECORD: Variants Clause

Format

VARIANTS.



END VARIANTS .

Parameters

cond-expr
Specifies an expression that represents the relationship between two value expressions. The value of a conditional expression is true, false, or null.

included-name-clause
Includes existing field and record definitions within record elements. See DEFINE RECORD: Included Name Clause for more information.

local-field-clause
Specifies the locally defined field. See DEFINE RECORD: Local Field Clause for more information.

structure-name-clause
Creates structure definitions within record elements. See DEFINE RECORD: Structure Clause for more information.

variants-clause
Creates variants definitions within record elements.

Description

The Variants Clause syntax identifies a set of overlays that can be used by a COBOL REDEFINES statement or by other languages. Each variants definition can contain two or more fields, records, structures, variants, or any combination of these definitions.

Be sure that the variants definitions you define conform to the requirements of the language or language processor that accesses the record element. For example, you must include a structure definition for each variants clause contained in a CDO record if you are developing a new application that will use a 3GL language and DIGITAL DATATRIEVE.

You can specify a different data type for each definition in a variants definition.

You can create any number of variants definitions within a record element.

You can create any number of definitions within a variants definition.

If you use an expression with one variant, you must use an expression with every other variant in the variants definition.

In variant expressions, you can refer to a tag variable (field definition) whose runtime value determines which variant in a variants definition maps to the record element. The tag variable cannot be part of an array.

At runtime, the product using CDO tests the value of each Boolean expression in the variants definition to determine which definition is the current variants definition. The variants with a Boolean expression that evaluates to true is chosen.

The values that you test for in the expressions of a variants definition must conform to the following rules:

- The values being tested must be valid values for the data type of the tag variable. For example, if the data type for the tag variable is text, the value you test for must be a string.
- The range of values being tested in one expression must not overlap the range of values in any other expression.

Each variants definition begins on the same byte in the record, subject to individual alignment options. The length of the longest definition in a variants definition determines the overall length of the variants definition.

Examples

```
CDO>  DEFINE RECORD PRODUCT_INVENTORY.
cont>   FIELD_ID.
cont>     VARIANTS.
cont>       VARIANT EXPRESSION IS
cont>         FIELD_ID IN PRODUCT_INVENTORY EQ "S".
cont>         IN_STOCK STRUCTURE.
cont>           PRODUCT_NO.
cont>           DATE_ORDERED.
cont>           STATUS_CODE.
cont>           QUANTITY.
cont>           LOCATION.
cont>           UNIT_PRICE.
cont>         END IN_STOCK STRUCTURE.
cont>       END VARIANT.
cont>       VARIANT EXPRESSION IS
cont>         FIELD_ID IN PRODUCT_INVENTORY EQ "B".
cont>         BACK_ORDER STRUCTURE.
cont>           PRODUCT_NO.
cont>           DATE_ORDERED.
cont>           STATUS_CODE.
cont>           QUANTITY.
cont>           SUPPLIER.
cont>           UNIT_PRICE.
cont>         END BACK_ORDER STRUCTURE.
cont>       END VARIANT.
cont>       VARIANT EXPRESSION IS
cont>         FIELD_ID IN PRODUCT_INVENTORY EQ "O".
cont>         OUT_OF_STOCK STRUCTURE.
cont>           PRODUCT_NO.
cont>           DATE_LAST_SOLD.
cont>         END OUT_OF_STOCK STRUCTURE.
cont>       END VARIANT.
cont>     END VARIANTS.
cont> END RECORD.
```

In this example, the DEFINE RECORD command creates the PRODUCT_ INVENTORY record element, which contains a variants definition consisting of three structure definitions. Each structure definition uses an expression whose value is compared to the value of the tag variable (FIELD_ID field definition) at runtime to determine which structure definition maps to the record element.

DEFINE REPOSITORY Command

Format

DEFINE REPOSITORY anchor-name [ALTERNATE_ROOT dir-name] .

Parameters

anchor-name
Specifies the OpenVMS directory in which you are creating the repository. The directory must be empty. If you specify a directory that does not exist, CDO creates one for you in your default directory and places the repository files there. *Do not modify or delete the files created by Oracle CDD/Repository; otherwise, you will corrupt your repository.*

If you plan to provide remote access to your repository with the ALTERNATE_ROOT parameter, the device associated with the anchor name cannot be mounted through the VAX Distributed File Service (DFS). Using the ALTERNATE_ROOT parameter lets you move binary files to a top-level directory, which reduces the depth of directories created. It also allows you to move binary files to another disk, reducing I/O contention on the anchor disk.

dir-name
Specifies your top OpenVMS file directory. (Use a logical name, instead of a full node name.) The device associated with the directory can be mounted through DFS.

Description

The DEFINE REPOSITORY command creates a physical CDO repository.

Specify the OpenVMS directory where you want the repository to reside.

You can charge disk resources for your repository to a resource identifier by setting this identifier as the owner of the files DEFINE REPOSITORY creates. First, issue the DEFINE REPOSITORY command, which sets the creator as the file owner. Then, issue the CHANGE PROTECTION command. This operation requires privileges.

OpenVMS utilities, including the OpenVMS BACKUP utility, cannot directly access repository files unless you invoke them from an account with system privileges.

Restriction

Do not store any files in the OpenVMS directory that contains the repository, except the files created by Oracle CDD/Repository. Otherwise, if you decide to delete the repository later, Oracle CDD/Repository deletes all files in this directory.

Do not create a repository in your top level directory [000000].

Once a repository is defined using the ALTERNATE_ROOT parameter, the alternate root cannot be changed or moved.

Changing the alternate root means that your binary files are no longer under the repository anchor. When you back up the repository, you must synchronize the backup of all the repository files.

DEFINE REPOSITORY and Remote Access

You can issue DEFINE REPOSITORY on a local (host) machine, but not on a remote (client) machine.

To make your repository available to remote users, perform the following steps:

1. Ask your system manager to make the ALTERNATE_ROOT directory a DFS access point. This action makes the directory and subdirectories known to a DFS server.
2. Issue the DEFINE REPOSITORY command, including an ALTERNATE_ ROOT parameter. This action permanently associates the file directories with the anchor directory. You should not explicitly refer to the file directories again. For example:

   ```
   DEFINE REPOSITORY DEV1:[PROJECT.CDD]
     ALTERNATE_ROOT DEV2:[PROJECT.FILES]
   ```

   For backup purposes, you can choose to move your anchor directory to the DFS disk where you store your file directories. In this case, you specify the same logical name for both anchor and ALTERNATE_ROOT directories. For example:

   ```
   DEFINE REPOSITORY DEV1:[PROJECT.CDD]
     ALTERNATE_ROOT DEV1:[PROJECT.FILES]
   ```

To access a repository from a host machine, perform the following steps:

1. Ask your system manager to make the DFS access point available on your system. During the DFS mount, the manager identifies the access point by the ALTERNATE_ROOT logical name. For example, if the DEFINE REPOSITORY command issued at the host machine referred to ALTERNATE_ROOT DEV1:[PROJECT.FILES], the manager refers to DEV1.
2. Issue a SET DEFAULT command that includes the full node name of the anchor directory. For example:

   ```
   SET DEFAULT A_NODE::DEV1:[PROJECT.CDD]
   ```

3. Review the default protection you receive on file directories. DFS does not support remote specification of file ACLs. You must make any modifications on the host system.

Customizing the Repository Templates

When you install Oracle CDD/Repository on your system, the installation procedure creates a template repository (CDD$TEMPLATE) and a repository database directory (CDD$TEMPLATEDB). CDD$TEMPLATE contains the CDD$PROTOCOL directory, which stores all the type definitions Oracle CDD/Repository uses to create metadata.

Description

The DEFINE REPOSITORY command creates several files in the specified OpenVMS anchor directory. Oracle CDD/Repository keeps directory information in these files in the anchor directory; Oracle CDD/Repository does not store directory information with the CDO definitions in the Oracle Rdb database.

Oracle CDD/Repository creates all new CDO repositories from CDD$TEMPLATE and CDD$TEMPLATEDB. If, after defining customized types in a repository, you want to include these types in all subsequent repositories that you create, you must make them part of the template.

To do this, execute the following command procedure:

```
$ @SYS$LIBRARY:CDD_BUILD_TEMPLATE.COM -
_$ repository-anchor-dir  repository-db-anchor-dir
```

Use the repository-anchor-dir parameter to specify the repository that contains definitions of your customized types. Use the repository-db-anchor-dir parameter to specify the empty directory that will hold database files.

Then, rename the CDD$TEMPLATE and CDD$TEMPLATEDB logicals to the parameter names you specified.

After you have assigned the logical name CDD$TEMPLATE to a repository, the protocols in that repository's CDD$PROTOCOLS directory will be distributed to any new CDO repository you create. If you have extended the types supplied by Oracle CDD/Repository or if you have created your own types in a repository, you may want to assign the logical name CDD$TEMPLATE to that repository so that these types will be copied into the CDD$PROTOCOLS directory of any subsequent repositories you create. If CDD$TEMPLATE is not defined, each new repository you create will contain only types supplied by Oracle CDD/Repository.

Customizing the Repository Templates

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 the SYS$STARTUP:CDDSTRTUP.COM command 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]
```

Examples

```
CDO>  DEFINE REPOSITORY DISK1:[BOB.DICT].
```

In this example, the DEFINE REPOSITORY command creates a CDO repository in a subdirectory called [BOB.DICT]:

DEFINE RMS_DATABASE Command

Format

```
DEFINE  RMS_DATABASE  rms-database-name

        [ DESCRIPTION  IS /*text*/ ]   [ AUDIT  IS /*text*/ ]   .

        RECORD  record-name   .

        FILE_DEFINITION [ file-definition-property ] ... .

        [ AREAS .  { AREA numeric-literal  [ area-property ] ... } ... . END AREAS . ]

        [ KEYS .  { KEY numeric-literal  [ key-property ] ... } ... . END KEYS . ]


        END  [ [ rms-database-name ]  RMS_DATABASE ] .
```

Parameters

rms-database-name
Specifies the logical RMS database element you are creating.

text
Adds information. Within the DESCRIPTION clause, this is information documenting the database definition; within the AUDIT clause, it is a history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the DESCRIPTION or AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_set of the session to DEC_KANJI.

record-name
Specifies an existing record element.

file-definition-property
Defines the file and record services for a logical RMS database definition. See Chapter 3 for the file definition properties CDO provides.

area-property
Defines the area properties for a logical RMS database element. See Chapter 3 for the area properties CDO provides.

numeric-literal
Defines the number of characters or bytes in the field. See Chapter 4 for more information on numeric literals.

key-property
Defines the key properties for a logical RMS database element. See Chapter 3 for the key properties CDO provides.

Description

The DEFINE RMS_DATABASE command creates a logical RMS database element in a CDO repository.

A logical RMS database consists of only one record and file definition. However, one logical RMS database definition can be owned by many physical RMS databases, where each physical RMS database owns a different CDD$FILE element. To create a physical RMS database on a disk with the characteristics specified by the DEFINE RMS_DATABASE command, issue the DEFINE DATABASE command.

To create a valid logical RMS database element, you must specify at least one record element and a file definition property with a SEQUENTIAL file organization option.

Examples

1.
```
CDO>  DEFINE FIELD LAST_NAME DATATYPE TEXT 30.
CDO>  DEFINE FIELD FIRST_NAME DATATYPE TEXT 20.
CDO>  DEFINE FIELD EMP_ID DATATYPE UNSIGNED LONGWORD.
CDO>  DEFINE RECORD EMPLOYEE_REC.
cont>   LAST_NAME.
cont>   FIRST_NAME.
cont>   EMP_ID.
cont> END.
```

This example has three steps. It shows you how a corporation can create a logical RMS database definition that can be used by all of its divisions to maintain employee information in a physical RMS database.

The data administrator creates the EMPLOYEE_STORAGE RMS database element in the central corporate repository, using the DEFINE RMS_ DATABASE command.

2.
```
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.
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>  DEFINE DATABASE DISG_FILE USING EMPLOYEE_STORAGE
cont>   ON DISK1:[DISG]EMP.DAT.
CDO>  DEFINE DATABASE SSG_FILE USING EMPLOYEE_STORAGE
cont>   ON DISK2:[SSG]EMP.DAT.
CDO>  DEFINE DATABASE DBS_FILE USING EMPLOYEE_STORAGE
cont>   ON DISK3:[DBS]EMP.DAT.
```

Each division creates its own employee information database on disk using the DEFINE DATABASE command and the same logical RMS database element, EMPLOYEE_STORAGE, from the central corporate repository.

DELETE COLLECTION Command

Format

DELETE COLLECTION [qualifier] ... collection-name ,... .

Parameters

collection-name
Specifies the collection you are deleting.

Qualifiers

/DESCENDANTS
/NODESCENDANTS (default)
Specifies whether CDO deletes members. When you specify the /DESCENDANTS qualifier, CDO deletes all members that are not also members of additional elements outside the area defined by your top collection.

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the collection is deleted.

Description

The DELETE COLLECTION command deletes a collection and, optionally, all members of the collection.

Because a collection is a controlled element, CDO freezes previous versions and allows you to delete only the highest visible version.

If a collection is a member, you must delete its owners before you delete the collection. If the collection's immediate owner is a member of another element, you must trace the relationships back until you reach the element that has no owners and delete elements one by one, in sequence of ownership.

If you attempt to delete a collection that owns a version in a different branch, the version must be the latest version in that branch. Otherwise, an error will occur. See the *Oracle CDD/Repository Architecture Manual* for complete information on branch lines of descent.

Examples

1.
```
CDO>  DELETE COLLECTION A_COLLECTION.
```

   In this example, the DELETE COLLECTION command deletes a collection that is not a member.

2.
```
CDO>  DELETE COLLECTION REGIONAL_SALES.
CDO>  DELETE COLLECTION DISTRICT_SALES.
CDO>  DELETE COLLECTION LOCAL_AREA_SALES.
```

   In this example, the DELETE COLLECTION command deletes a collection, a subcollection, and a further subcollection in sequence of ownership.

3.
```
CDO>  DELETE COLLECTION COMPILER_C(3).
CDO>  DELETE COLLECTION COMPILER_C(2:UPDATE_BRANCH:2).
CDO>  DELETE COLLECTION COMPILER(2:UPDATE_BRANCH:1).
CDO>  DELETE COLLECTION COMPILER_C(2).
CDO>  DELETE COLLECTION COMPILER(1).
```

   In this example, successive DELETE COLLECTION commands delete the main line and branch line versions of a collection in sequence of ownership. The branch line originates from version 2 and merges back in version 3.

DELETE CONTEXT Command

Format

DELETE CONTEXT [qualifier] ... context-name ,... .

Parameters

context-name
Specifies the context you are deleting.

Qualifiers

/PARENTS
/NOPARENTS (default)
Specifies whether CDO deletes parents. If you have defined a top collection for the context, CDO cannot delete a parent that is also a parent of an element outside this collection or collection hierarchy.

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the context is deleted.

Description

The DELETE CONTEXT command deletes a context.

Because a context is a nonversioned element, CDO does not accept a branch designation or a version number in the context name.

If a context is a child, you must delete its immediate parent before you delete the context. If the context's immediate parent is a child of another element, you must trace the relationships back until you reach the element that has no parents.

If you delete your current context, CDO sets a null value for the current context before deleting the context.

An error occurs if you attempt to delete a context that has opened files or reserved elements. The SHOW CONTEXT or SHOW RESERVATIONS command indicates whether this condition exists.

Examples

```
CDO>  DELETE CONTEXT A_CONTEXT.
```

In this example, the DELETE CONTEXT command deletes the A_CONTEXT context.

DELETE DATABASE Command

Format

DELETE DATABASE [qualifier] rms-database-name .

Parameters

rms-database-name
Specifies the physical RMS database element you are deleting. You can substitute an asterisk (*) wildcard character for this parameter.

Qualifiers

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the physical RMS database element is deleted.

Description

The DELETE DATABASE command deletes the physical RMS database (CDD$FILE) from disk and its CDD$DATABASE element from the repository.

When you issue the DELETE DATABASE command, CDO prompts you to confirm that you want to proceed. You cannot suppress this prompt. If you respond Yes at the prompt, CDO deletes the highest visible version of CDD$FILE and, if you have not specified another version number, the highest visible version of CDD$DATABASE.

If CDO cannot delete the physical RMS file from disk, the DELETE DATABASE command fails, and the CDD$DATABASE and CDD$FILE elements remain in the repository.

Examples

```
CDO>  DELETE DATABASE /LOG EMP_FILE(1)
```

In this example, the DELETE DATABASE command with the /LOG qualifier confirms that CDO deleted the RMS file from disk and the RMS database element EMP_FILE from the repository.

DELETE DIRECTORY Command

Format

DELETE DIRECTORY [qualifier] directory-name ,... .

Parameters

directory-name
Specifies the repository directory you are deleting.

Qualifiers

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the directory is deleted.

Description

The DELETE DIRECTORY command deletes a CDO directory.

Unless you change directory protection, only the owner of a CDO directory or the system manager can delete a directory.

You can delete only empty directories.

CDO deletes only the last directory in a fully qualified path name. For example, if you specify a directory name of [SMITH.DICT]CORPORATE.PERSONNEL.SALARIED, CDO deletes only the SALARIED directory.

Examples

```
CDO>  DELETE DIRECTORY /LOG PROSPECTS.
%CDO-I-DIRDEL, directory PROSPECTS deleted
```

In this example, the DELETE DIRECTORY command with the /LOG qualifier confirms that CDO deleted the PROSPECTS directory.

DELETE FIELD Command

Format

DELETE FIELD [qualifier] ... field-name ,... .

Parameters

field-name
Specifies the field element you are deleting. You can substitute an asterisk (*) wildcard character for this parameter.

Qualifiers

/DESCENDANTS
/NODESCENDANTS (default)
Specifies whether CDO deletes children. When you specify the /DESCENDANTS qualifier, and your field element is controlled, CDO deletes all children that are not also children of additional elements outside the area defined by your top collection. If the field is uncontrolled, CDO deletes all children that are not also children of any other elements.

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the field element is deleted.

Description

The DELETE FIELD command deletes a field element from a repository.

If a field element is controlled, CDO freezes previous versions and allows you to delete only the highest visible version. If a field element is uncontrolled, CDO deletes the highest version unless you specify another version number.

You cannot delete a field that is a child. Delete the parent first, or include the /DESCENDANTS qualifier to delete parents and children at the same time.

Examples

```
CDO>  DELETE FIELD /DESCENDANTS ORDER_NUMBER.
```

In this example, the DELETE FIELD command with the /DESCENDANTS qualifier deletes the ORDER_NUMBER field element and its children.

DELETE FILE_ELEMENT Command

Format

DELETE FILE_ELEMENT type-name [qualifier] ... element-name ,... .

Parameters

type-name
Specifies the type (MCS_BINARY or MCS_BINARY subtype) of the file element you are deleting. See the *Oracle CDD/Repository Information Model Volume I* for information on these types.

element-name
Specifies the file element you are deleting. You can substitute an asterisk (*) wildcard character for this parameter.

Qualifiers

/DESCENDANTS
/NODESCENDANTS (default)
Specifies whether CDO deletes children. When you specify the /DESCENDANTS qualifier, CDO deletes all children that are not also children of additional elements outside the area defined by your top collection.

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the file element is deleted.

Description

The DELETE FILE_ELEMENT command deletes a file element and, optionally, all children of the file element to the bottom of the collection hierarchy.

Because a file element is a controlled element, CDO freezes all previous versions and allows you to delete only the highest visible version.

CDO cannot delete the following versions in a branch line:

- A version that is owned by a version in a different branch
- A version that owns an intermediate version in a different branch

See the *Oracle CDD/Repository Architecture Manual* for information on branch lines of descent.

Examples

1.
```
CDO>  DELETE FILE_ELEMENT MCS_BINARY PARSER_TABLES.
```

   In this example, the DELETE FILE_ELEMENT command deletes a file element definition that is based on the MCS_BINARY type.

2.
```
CDO>  DELETE FILE_ELEMENT MCS_BINARY PARSER_TABLES(3).
CDO>  DELETE FILE_ELEMENT MCS_BINARY PARSER_TABLES(2:UPDATE_BRANCH:2).
CDO>  DELETE FILE_ELEMENT MCS_BINARY PARSER_TABLES(3:UPDATE_BRANCH:1).
CDO>  DELETE FILE_ELEMENT MCS_BINARY PARSER_TABLES(2).
CDO>  DELETE FILE_ELEMENT MCS_BINARY PARSER_TABLES(1).
```

   In this example, successive DELETE FILE_ELEMENT commands delete the main line and the branch line versions of a file element in sequence of ownership. The branch line originates from version 2 and merges back in version 3.

DELETE GENERIC Command

Format

DELETE GENERIC type-name [qualifier] ... element-name ,... .

Parameters

type-name
Specifies the type of the generic element you are deleting. This type cannot be MCS_BINARY, an MCS_BINARY subtype, MCS_COLLECTION, MCS_CONTEXT, or MCS_PARTITION.

element-name
Specifies the generic element you are deleting.

Qualifiers

/DESCENDANTS
/NODESCENDANTS (default)
Specifies whether CDO deletes children. When you specify the /DESCENDANTS qualifier, and your generic element is controlled, CDO deletes all children that are not also children of additional elements outside the area defined by your top collection. If the generic element is uncontrolled, CDO deletes all children that are not also children of other elements.

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the generic element is deleted.

Description

The DELETE GENERIC command deletes a generic element. This element can be based on a type supplied by Oracle CDD/Repository or a user-supplied type.

If a generic element is a controlled versioned element, CDO freezes previous versions and allows you to modify only the highest visible version. If a generic element is an uncontrolled versioned element, CDO deletes the highest version unless you specify another version number.

If you use SQL (structured query language) to delete an Oracle Rdb database file, the corresponding CDD$DATABASE element may remain in Oracle CDD/Repository. You can use the DELETE GENERIC command to delete this element.

Examples

1.
```
CDO>  DELETE GENERIC CDD$DATABASE DEPT1.
```

   In this example, the DELETE GENERIC command deletes the DEPT1 generic element from the repository.

2.
```
CDO>  DELETE GENERIC CDD$SOURCE_MODULE
cont>   /DESCENDANTS /LOG COBOL_SOURCE.
```

   In this example, the DELETE GENERIC command with the /LOG and /DESCENDANTS qualifiers confirms that CDO has deleted the COBOL_ SOURCE generic element and children.

DELETE HISTORY Command

Format

DELETE HISTORY [qualifier] FOR { FIELD / RECORD / GENERIC type-name } element-name ,... .

Parameters

type-name
Specifies the type of the file or generic element definition whose history entries you are deleting.

element-name
Specifies the element whose history entries you are deleting. You can use wildcard characters in this parameter.

Qualifiers

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the history entries have been deleted.

Description

The DELETE HISTORY command deletes the history entries for a CDO element.

Note that the DELETE HISTORY command deletes the entire history list for a given entity. This is not a PURGE HISTORY command.

Examples

```
CDO> DELETE HISTORY FOR RECORD CAR_POOL.
```

In this example, the DELETE HISTORY command deletes the history entries for the CAR_POOL record element.

DELETE PARTITION Command

Format

DELETE PARTITION [qualifier] ... partition-name ,... .

Parameters

partition-name
Specifies the partition you are deleting.

Qualifiers

/PARENTS
/NOPARENTS (default)
Specifies whether CDO deletes parents. CDO cannot delete a parent that is also a parent of an element outside the partition or partition hierarchy. When CDO cannot delete any parents, you receive an informational notice. When CDO can delete some parents, you do not receive a notice.

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the partition is deleted.

Description

The DELETE PARTITION command deletes a partition and, optionally, all the partition's children to the bottom, or root, of the partition hierachy.

Since a partition is a nonversioned element, CDO does not accept a branch designation or a version number in partition names.

If a partition is a child, you must delete its parent or parents before you delete the partition. If the partition's immediate parent is a child of another element, you must trace the relationships back until you reach the element that has no parents.

An error occurs if you attempt to delete a partition that contains elements. Promote or delete elements from a partition prior to issuing the DELETE PARTITION command.

Examples

1.
```
CDO>  DELETE PARTITION /PARENTS FRONT_END.
```

   In this example, the DELETE PARTITION command with the /PARENTS qualifier deletes all the parent partitions of FRONT_END.

2.
```
CDO>  DELETE PARTITION /LOG REPORTS.
%CDO-I-ENTDEL, entity CDD$DISK:[SMITH.PART]REPORTS deleted
```

   In this example, the DELETE PARTITION command with the /LOG qualifier confirms that CDO deleted the partition REPORTS.

DELETE PROTECTION Command

Format

```
DELETE  PROTECTION  [ qualifier ]  FOR

    { DIRECTORY
      FIELD
      RECORD
      GENERIC  type-name }  element-name ,...  [ POSITION n
                                               id+ ... ]  [ ACCESS ]  .

DELETE  PROTECTION  [ qualifier ]  FOR

    { REPOSITORY  anchor-name }  [ POSITION n ]  [ ACCESS
                                                   DEFAULT_ACCESS ]  .
```

Parameters

type-name
Specifies the type of file or generic element definition whose ACE or ACL you are deleting.

element-name
Specifies the element whose ACE or ACL you are deleting. You can use wildcard characters in this parameter.

n
Specifies the relative position of the ACE in the ACL that you are deleting.

id
Specifies the identifiers for the ACE you are deleting.

anchor-name
Specifies the repository anchor directory whose ACE or ACL you are deleting.

Qualifiers

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the ACE or ACL is deleted.

DELETE PROTECTION Command

Description

The DELETE PROTECTION command deletes an access control list entry (ACE) or the entire access control list (ACL) for a CDO element or repository.

You need CONTROL access to delete protection.

The POSITION clause tells CDO the relative position of the ACE to delete. ACEs are numbered starting with one. You can also delete a particular element ACE by specifying the identifier or identifiers contained in that ACE. If you omit the identifiers and the POSITION clause, CDO deletes the entire ACL.

After the DELETE PROTECTION command executes, CDO resequences the remaining ACEs in the ACL.

The default access type for all cases is ACCESS.

Examples

1.
```
CDO>  DELETE PROTECTION FOR RECORD CAR_POOL POSITION 5.
```

   In this example, the DELETE PROTECTION command deletes the fifth ACE in the ACL for the CAR_POOL record element.

2.
```
CDO>  DELETE PROTECTION FOR RECORD CAR_POOL [23,56].
```

   In this example, the DELETE PROTECTION command deletes the ACE with the identifier [23,56] for the CAR_POOL record element.

DELETE RECORD Command

Format

DELETE RECORD [qualifier] ... record-name ,... .

Parameters

record-name
Specifies the record element you are deleting.

Qualifiers

/DESCENDANTS
/NODESCENDANTS (default)
Specifies whether CDO deletes children. When you specify the /DESCENDANTS qualifier, and your record element is controlled, CDO deletes all children that are not also children of additional elements outside the area defined by your top collection. If the record element is uncontrolled, CDO deletes all children that are not also children of other elements.

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the record element is deleted.

Description

The DELETE RECORD command deletes a record element from a CDO repository.

If the record element is controlled, CDO freezes previous versions and allows you to delete only the highest visible version. If the record element is uncontrolled, CDO deletes the highest version unless you specify another version number.

Examples

```
CDO>  DELETE RECORD /DESCENDANTS CUSTOMER_ORDERS.
```

In this example, the DELETE RECORD command with the /DESCENDANTS qualifier deletes the CUSTOMER_ORDERS record element and children.

DELETE REPOSITORY Command

Format

DELETE REPOSITORY [qualifier] repository-name ,... .

Parameters

repository-name
Specifies the anchor directory of the repository you are deleting.

Qualifiers

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the repository is deleted.

Description

The DELETE REPOSITORY command deletes all the elements in a repository, any relationships between elements in the repository or elements in other repositories, and the repository itself.

Unless you change repository protection, only the owner of a repository or the system manager can delete a CDO repository.

You cannot delete a repository if it contains an element that is used by another element in a different repository.

Caution

Before you delete a repository, be sure your elements do not have relationships to other elements in other repositories. Check to make sure you have not stored user-created files in the repository.

Examples

```
CDO>  DELETE REPOSITORY DISK1:[BOB.DICT].
```

In this example, the DELETE REPOSITORY command deletes the DISK1:[BOB.DICT] repository.

DELETE RMS_DATABASE Command

Format

DELETE RMS_DATABASE [qualifier] rms-database-name .

Parameters

rms-database-name
Specifies a logical RMS database element.

Qualifiers

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the RMS database element is deleted.

Description

The DELETE RMS_DATABASE command deletes a logical RMS database element from the repository.

If the RMS database element is controlled, CDO freezes previous versions and allows you to delete only the highest visible version. If the RMS database element is uncontrolled, CDO deletes the highest visible version unless you specify another version number.

Before you can issue the DELETE RMS_DATABASE command, you must have deleted the associated physical RMS database element with the DELETE DATABASE command.

Examples

1.
```
CDO>  DELETE DATABASE DISG_FILE(2).
   .
   .
   .
CDO>  DELETE RMS_DATABASE /LOG EMPLOYEE_STORAGE.
```

In this example, the DELETE DATABASE command prompts you to confirm that you intend to delete the physical RMS database file from disk. CDO confirms this deletion. The DELETE RMS_DATABASE command with the /LOG qualifier confirms that CDO has deleted the logical RMS database element EMPLOYEE_STORAGE.

2.
```
CDO>  DELETE RMS_DATABASE EMPLOYEE_STORAGE.
%CDD-E-INUSE, element is the member of a relationship; it cannot be deleted
CDO> DELETE DATABASE DISG_FILE(2).   1
deleting file DISK1:[SMITH]EMP.DAT; proceed? [Y/N]) (N)Y   2
%CDO-I-FILEDEL, file DISK1:[SMITH]EMP.DAT; deleted
CDO> DELETE RMS_DATABASE /LOG EMPLOYEE_STORAGE. 3
```

This example shows the result when you try to delete a logical RMS database definition from the repository while a physical RMS database (on disk) is using it. When you delete the physical database (EMP.DAT), you can then delete the logical database.

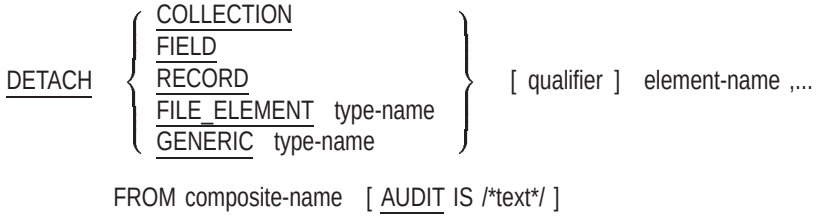
1 Delete the physical RMS database definition from the repository.

2 Type Y in acknowledgement that CDO deletes the physical RMS file from disk.

3 Delete the logical RMS database definition from the repository. The /LOG qualifier provides a confirmation of deletion.

DETACH FROM COMPOSITE Command

Format



Parameters

type-name
Specifies the type of the element you are detaching.

element-name
Specifies the name of the element from which you are detaching. You can substitute an asterisk (*) wildcard character for this parameter.

composite-name
Specifies the name of the composite from which you are detaching.

text
Adds information to the history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_ set of the session to DEC_KANJI.

Qualifiers

/LOG
/NOLOG (default)
Specifies whether CDO displays text identifying each element as the element is detached.

DETACH FROM COMPOSITE Command

Description

The DETACH FROM COMPOSITE command detaches an element from a composite. The element must be a controlled element that is not currently reserved. It also must be currently attached to the composite.

Before you issue the DETACH FROM COMPOSITE command, you must have a context set and your composite reserved. The SHOW CONTEXT and SHOW RESERVATIONS commands indicate whether these conditions exist.

You can use the DETACH FROM COMPOSITE command in conjunction with the DEFINE, RESERVE, REPLACE, and ATTACH TO COMPOSITE commands to link collections into collection hierarchies. See the DEFINE COLLECTION command for an example of a collection hierarchy.

You can also use the DETACH FROM COMPOSITE and ATTACH TO COMPOSITE commands to move between lines of descent. See the *Oracle CDD/Repository Architecture Manual* for more information.

Examples

1.
```
CDO>  RESERVE COLLECTION SALES_EACH_PRODUCT
CDO>  ATTACH FIELD PART_NUMBER TO SALES_EACH_PRODUCT
CDO>  REPLACE COLLECTION SALES_EACH_PRODUCT
   .
   .
   .
CDO>  RESERVE COLLECTION SALES_EACH_PRODUCT
CDO>  DETACH FIELD PART_NUMBER FROM SALES_EACH_PRODUCT
CDO>  REPLACE COLLECTION SALES_EACH_PRODUCT
```

In this example, the DETACH command detaches an element from a collection.

2.
```
CDO>  RESERVE COLLECTION EMPLOYEE_RECORDS
CDO>  DETACH FIELD FIRST_NAME(2:BRANCH:2) FROM EMPLOYEE_RECORDS
CDO>  ATTACH FIELD FIRST_NAME(2) TO EMPLOYEE_RECORDS
CDO>  REPLACE COLLECTION EMPLOYEE_RECORDS
```

In this example, the DETACH FROM COMPOSITE command detaches a version in the branch line of descent, FIRST_NAME(2:BRANCH:2), from the EMPLOYEE_RECORDS collection. After you attach a version in the main line, FIRST_NAME(2), to EMPLOYEE_RECORDS you can create new versions in the main line, instead of in the branch line where you had been working.

DIRECTORY Command

Format

DIRECTORY [qualifier] ... [element-name] ,...

Parameters

element-name
Specifies the element or elements to be listed. This name can specify a CDO element, a CDO directory, or a logical name.

Qualifiers

/BEFORE=time
/NOBEFORE (default)
Selects only those elements dated before the specified time. If you do not specify a time with the /BEFORE qualifier, CDO uses /BEFORE=TODAY qualifier as the value.

/BRIEF (default)
Displays the element names and their types. This is the default display.

/FORMAT
/NOFORMAT (default)
Displays the text specifying whether the element is in CDO or DMU format (in a CDO repository or a DMU dictionary).

/FULL
Displays the name, type, parent, approximate size, and access control information (protection) for each repository element. If an element appears under an alternate name in a distributed repository, these names appear also.

/SINCE=time
/NOSINCE (default)
Displays only those elements dated at or after the specified time. If you do not specify a time with the /SINCE qualifier, CDO uses /SINCE=TODAY as the value.

/TYPE=(type-name,...)
/NOTYPE (default)
Displays only elements of the specified type or types. You *cannot* use wildcard characters in the type name.

DIRECTORY Command

Description

The DIRECTORY command lists an element or elements in one or more repository directories.

Restriction

If you have a context set, CDO limits the display to those elements that are visible to your context. If you do not have a context set, this restriction does not apply.

CDO only displays the names of those elements to which you have READ access.

With the /BEFORE and /SINCE qualifiers, you can specify any of the following for a time:

- Absolute time
- Combination of absolute and delta times
- A keyword such as TODAY or YESTERDAY

See the OpenVMS documentation for more information about specifying times.

Examples

1. `CDO>  DIRECTORY /SINCE=YESTERDAY`

   In this example, the DIRECTORY command will display the names, versions, and types of all elements CDO has created since 00:00 (midnight) of the previous day in the current default directory.

2.
```
CDO>  DIRECTORY /FULL DISK1:[JONES.DICT]REVIEW.*
 Directory DISK1:[JONES.DICT]REVIEW
FIRST_NAME(1)                                    FIELD Created: 8-MAY-1995
13:33:28.95   Revised:  8-MAY-1995 13:33:32.99 Owner: [VDD,JONES] Access
Cntrl List: (IDENTIFIER=[VDD,JONES],ACCESS=READ+WRITE+MODIFY+
ERASE+SHOW+DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[VDD,CDDCDD],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[*,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+OPERATOR+ADMIN)
Size: 2277       dictionary_format: CDO format
FULL_ADDRESS(1)                                 RECORD Created: 8-MAY-1995
14:09:51.19   Revised:  8-MAY-1995 14:09:51.19 Owner: [VDD,JONES] Access
Cntrl List: (IDENTIFIER=[VDD,JONES],ACCESS=READ+WRITE+MODIFY+
ERASE+SHOW+DEFINE+CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[VDD,CDDCDD],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+DEFINE+
CHANGE+DELETE+CONTROL+OPERATOR+ADMINISTRATOR)
(IDENTIFIER=[*,*],ACCESS=READ+WRITE+MODIFY+ERASE+SHOW+OPERATOR+ ADMIN)
Size: 1337       dictionary_format: CDO format
   .
   .
   .
```

In this example, the DIRECTORY command with the /FULL qualifier will display the name, type, parent, and protection information for elements in the REVIEW directory and in any distributed repositories.

EDIT Command

Format

EDIT { FIELD / RECORD } [element-name]

Parameters

element-name
Specifies the field or record element you are creating.

Description

The EDIT command invokes the CDO editor, which allows you to create uncontrolled field and record definitions.

If you do not specify an element name, CDO prompts you for one after you enter the editor.

See *Using Oracle CDD/Repository on OpenVMS Systems* for more information on the CDO editor.

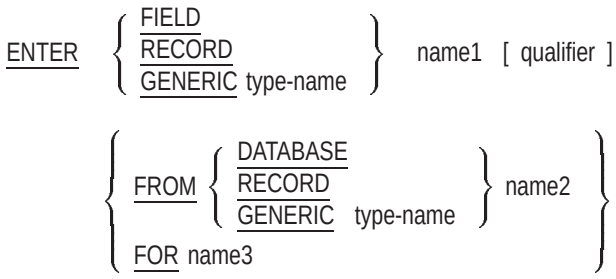
Examples

```
CDO>  EDIT FIELD
```

In this example, the EDIT command invokes the CDO editor to create an uncontrolled field element.

ENTER Command

Format



Parameters

type-name
Specifies the type of a generic element.

name1
Specifies the processing name of the field, record, or generic element. You can use an asterisk (*) wildcard character to indicate all element names or a specific element name. Afer the ENTER command executes, CDO creates a directory name that is the same as the processing name.

name2
Specifies the database, record, or generic element that owns name1.

name3
Specifies an additional directory name you are designating for the field, record, or generic element.

Qualifiers

/RDB_METADATA
/NORDB_METADATA (default)
Specifies whether CDO enters a directory name for Oracle Rdb system relations. The /NORDB_METADATA qualifier is the default. This qualifier is synonymous with the SHOW /[NO]SYSTEM qualifier.

ENTER Command

Description

The ENTER command assigns a directory name or an additional directory name to a field, record, or generic element.

The FROM clause assigns a directory name to an element that does not have a directory name. For example, field and record elements within an Oracle Rdb database definition may not have directory names.

Without a directory name, the 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 element without a directory name in an Oracle Rdb global field definition.

The FOR clause assigns an additional directory name to an element that has a directory name. This functionality allows you to give an element different names on a local node and a remote node.

You must issue the ENTER command with the FOR clause for an element before you can reserve that element in a distributed environment. See *Using Oracle CDD/Repository on OpenVMS Systems* for information on reserving elements in a distributed environment.

CDO enters directory names in your currect default directory. The ENTER command fails if an element in that directory has a directory name that is the same as the processing name you specify.

Restriction

The ENTER command does not apply a default ACL to the object being entered. Therefore, if the object did not have an ACL prior to being entered, it will not have one after being entered in the directory system. Setting the desired ACLs is left to the discretion of the user.

You cannot issue the ENTER command to enter fields or records within a variant of CDO record definitions that were converted from DMU definitions. Field definitions and record definitions that exist only within the context of a variant cannot be given directory names.

Use this command to assign directory names to field definitions or structure definitions within record definitions converted from DMU. When an object (such as a field definition) has a directory name, that object can be included in other definitions (for example, field definitions with directory names can be used as Oracle Rdb global fields).

Examples

1.
```
CDO>  ENTER FIELD PART_NUMBER FROM DATABASE PARTS
```

   In this example, the ENTER command enters a PART_NUMBER directory name for the PART_NUMBER field element from the database element PARTS.

2.
```
CDO>  ENTER FIELD SALARY_CLASS FOR WAGE_CLASS
```

   In this example, the ENTER command assigns SALARY_CLASS as an alternative directory name for the WAGE_CLASS field element.

3.
```
CDO>  ENTER GENERIC MCS_COLLECTION CORP_DATA_DEFS
con>   FOR CORPORATE:CORP_DATA_DEFS
CDO>  RESERVE GENERIC MCS_COLLECTION CORP_DATA_DEFS
```

   In this example, the ENTER command assigns the alternative directory name CORP_DATA_DEFS on a local node for the collection CORPORATE:CORP_DATA_DEFS.

EXIT Command

Format

EXIT

Description

The EXIT command ends a CDO session.

You can also exit from CDO by pressing Ctrl/Z.

Examples

```
CDO>  EXIT
$
```

In this example, the EXIT command exits you from CDO and returns you to the OpenVMS DCL prompt.

EXTRACT Command

Format

EXTRACT { FIELD / RECORD } element-name ,... [qualifier]

Parameters

element-name
Specifies the field or record element that you want to display in the DEFINE command format or ANSI C language format. You can use wildcard characters in this parameter.

Qualifiers

/LANGUAGE=CC
/LANGUAGE=CDO (default)
Use the /LANGUAGE qualifier to generate data defintions in one of two formats; either the default DEFINE command format or the ANSI-standard syntax for the C programming language. Valid options are:

- CC

  Specifies that the EXTRACT command converts the record to ANSI C language syntax. Each record that is converted to ANSI-standard syntax will include a comment statement that lists the original Oracle CDD/Repository data type information for each field in the record. For example:

  ```
  char  field1;    /*Text*/
  ```

- CDO

  Specifies that the EXTRACT command displays one or more repository elements in the format of the DEFINE command. The CDO option is the default.

EXTRACT Command

Description

The EXTRACT command displays one or more repository elements in the specified format. You can choose the DEFINE command format or the ANSI-standard syntax for the C programming language. By displaying an element in the DEFINE command format, the EXTRACT command makes it easier for you to create new versions of an uncontrolled element. By displaying an element in the ANSI C programming language format, you can use the definition when building applications.

You can capture the output of the EXTRACT command in a file by issuing the SET OUTPUT command as the preceding command. For the DEFINE command, edit and execute the command file with the @ command. For the ANSI C programming language format, edit the output file to remove the EXTRACT command and then include the file in an application.

If a field has character set attributes, you can display them using the SHOW and EXTRACT commands; in addition, you can use the SHOW command to display size information of a field in both character-based size and octet-based size. See the description of SET CHARACTER_SET command and the DATATYPE field property for more information.

Examples

1.
```
CDO> DEFINE RECORD FULL_NAME.
cont> FIRST_NAME.
cont> MIDDLE.
cont> LAST_NAME.
cont> END RECORD.
   .
   .
   .
CDO> DEFINE RECORD HOME_ADDRESS.
cont> STREET_ADDRESS.
cont> CITY.
cont> STATE.
cont> POSTAL_CODE.
cont> END RECORD.
   .
   .
   .
CDO> DEFINE FIELD BADGE DATATYPE IS UNSIGNED LONGWORD SIZE IS 5 DIGITS.
CDO> DEFINE RECORD EMPLOYEE_REC_ONE.
cont> FULL_NAME.
cont> HOME_ADDRESS.
cont> BADGE.
cont> END RECORD.
```

   This example shows the definition of records used in the following examples.

2.
```
CDO> EXTRACT RECORD EMPLOYEE_REC_ONE /LANGUAGE=CC

  struct employee_rec_one
  {
    struct   {
      char  first_name[20];     /* Text */
      char  middle;     /* Text */
      char  last_name[20];      /* Text */
    } full_name;
    struct   {
      char  street_address[30]; /* Text */
      char  city[20];   /* Text */
      char  state[2];   /* Text */
      unsigned long  postal_code;       /* Unsigned Longword */
    } home_address;
    unsigned long  badge;       /* Unsigned Longword */
  };
```

   This example shows the EXTRACT command specifying the /LANGUAGE=CC qualifier.

3.
```
CDO> EXTRACT RECORD EMPLOYEE_REC_ONE /LANGUAGE=CDO
Define record CDDRTEST:[CDDR_TEST.userid.TEST_REP]MY_DIR.EMPLOYEE_REC_ONE
    .
    CDDRTEST:[CDDR_TEST.userid.TEST_REP]MY_DIR.FULL_NAME(1).
    CDDRTEST:[CDDR_TEST.userid.TEST_REP]MY_DIR.HOME_ADDRESS(1).
    CDDRTEST:[CDDR_TEST.userid.TEST_REP]MY_DIR.BADGE(1)
    .
End record.
```

   This example shows the EXTRACT command specifying the default /LANGUAGE=CDO qualifier.

FETCH Command

Format

FETCH FILE_ELEMENT type-name element-name TO file-name

Parameters

type-name
Specifies the type of file element you are fetching. Valid types are MCS_BINARY or one of the binary subtypes. The binary subtypes are:

- MCS_ANALYSIS_DATA_FILE
- MCS_BINARY_TOOL
- MCS_C_SOURCE_FILE
- MCS_DIAGNOSTIC_FILE
- MCS_EXECUTABLE_FILE
- MCS_LISTING_FILE
- MCS_LOG_FILE
- MCS_OBJECT_FILE
- MCS_TEXT
- MCS_TEXT_TOOL

element-name
Specifies the file element you are fetching.

file-name
Specifies the name of an OpenVMS file that will be created by the FETCH command. You may include a device and directory with the name.

Description

The FETCH command copies the contents of a file element to the OpenVMS file specified on the command line. You must use the SET CONTEXT command to define a current context before using the FETCH command.

The OpenVMS file created by the FETCH command is available for both read and write access. It has no further connection to the file element from which it was copied or to the repository. You are responsible for maintaining the OpenVMS file.

Examples

```
CDO> DEFINE PARTITION FIRST_BASELEVEL.
CDO> DEFINE CONTEXT DEVELOPMENT_CONTEXT
cont> BASE_PARTITION FIRST_BASELEVEL.
CDO> SET CONTEXT DEVELOPMENT_CONTEXT
CDO> DEFINE COLLECTION CLIENT.
CDO> RESERVE COLLECTION CLIENT
CDO> DEFINE FILE_ELEMENT MCS_TEXT_TOOL CLIENT_BUILD
cont> STORETYPE INTERNAL. END.
CDO> RESERVE FILE_ELEMENT MCS_TEXT_TOOL CLIENT_BUILD
CDO> FETCH FILE_ELEMENT MCS_TEXT_TOOL CLIENT_BUILD(1)
cont> TO BLD$:BUILD_CLIENT.COM
CDO> REPLACE FILE_ELEMENT MCS_TEXT_TOOL CLIENT_BUILD
```

In certain circumstances, you may want to use the FETCH command instead of the OPEN and CLOSE commands. This example shows how to use the FETCH command to access the contents of an earlier version of a file element that is currently reserved.

HELP Command

Format

HELP [topic [subtopic] ...]

Parameters

topic
Specifies the topic about which you are requesting information. If you specify an incomplete topic name, CDO displays information on all topics starting with that string. If you specify wildcard characters, CDO displays information on all matching strings.

If you do not specify a topic, CDO displays a list of all available topics, followed by a request for a topic.

subtopic
Provides more information, such as parameters and qualifiers, associated with this command. A subtopic can also be a description and examples of a command.

Description

The HELP command provides help on CDO commands and concepts.

Type Ctrl/Z to exit from help and return to the CDO prompt.

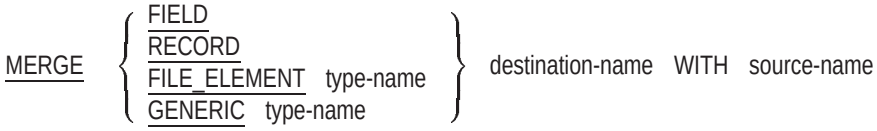
Examples

```
CDO>  HELP DEFINE COLLECTION
```

In this example, the HELP command requests information on the DEFINE COLLECTION command.

MERGE Command

Format



Parameters

type-name
Specifies the type of the file or generic element you are merging.

destination-name
Specifies the name of the merged version.

source-name
Specifies the name of the branch line version to be merged.

Description

The MERGE command creates a merge relationship between a branch line version and a new version on the originating line of descent. Both versions must be controlled.

If the versions are of type MCS_TEXT, CDO also performs a physical merge that incorporates the highest versions on both lines of descent. If the versions are of another type, you must perform the physical merge yourself with an editor appropriate to your system. If you are familiar with the Oracle CDD/Repository callable interface, you can write a method to perform the physical merging.

When you issue the MERGE command, your collection must include the version on the originating line, rather than the version on the branch line. If your collection includes the branch line version, issue the DETACH FROM COMPOSITE and ATTACH TO COMPOSITE commands to attach to the highest version on the originating line. You must reserve this version before you issue the MERGE command.

Examples

```
CDO>  RESERVE FILE_ELEMENT MCS_TEXT JULY_REPORT(2)
CDO>  MERGE FILE_ELEMENT MCS_TEXT JULY_REPORT(3)
cont>   WITH JULY_REPORT(1:DRAFT:2)
CDO>  REPLACE FILE_ELEMENT MCS_TEXT JULY_REPORT(3)
```

In this example, the MERGE command creates a relationship between JULY_REPORT(1:DRAFT:2) on the branch line and JULY_REPORT(3) on the main line. Because JULY_REPORT is of MCS_TEXT type, the MERGE command also physically merges the contents of JULY_REPORT(1:DRAFT:2) and JULY_REPORT(2) in JULY_REPORT(3).

MOVE REPOSITORY Command

Format

MOVE REPOSITORY [qualifier] anchor-name1 TO anchor-name2 .

Parameters

anchor-name1
Specifies the full path name of the anchor directory that contains the repository files you are moving.

anchor-name2
Specifies the full path name of the anchor directory to which you are moving the repository files. This must be an empty directory.

Qualifiers

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the repository files are successfully moved to the new directory.

Description

The MOVE REPOSITORY command physically moves repository files from one system directory to another system directory.

If the MOVE REPOSITORY command is successful, the repository files no longer exist in the original directory.

As part of a successful MOVE REPOSITORY command, CDO updates all references to this repository from other repositories to point to the new location. If CDO cannot update any of the references, the MOVE REPOSITORY command fails.

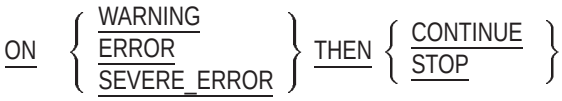
Examples

```
CDO>  MOVE REPOSITORY DISK1:[SMITH.DICT] TO
cont>   DISK1:[TAYLOR.REP].
```

In this example, the MOVE REPOSITORY command moves the repository contents to a different anchor directory.

ON Command

Format



Description

The ON command specifies an action for CDO to perform when it encounters an error condition during the execution of a command procedure.

The following keywords allow you to specify error conditions of increasing severity:

- WARNING
- ERROR
- SEVERE_ERROR

Table 1–4 describes error handling when you specify CONTINUE.

Table 1–4 Error Handling if Action is CONTINUE

Error Condition Specified	Action Taken on Error
WARNING	Command procedure continues on warnings only; it stops for errors or severe errors.
ERROR	Command procedure continues on warnings and errors; it stops for severe errors only.
SEVERE_ERROR	Command procedure continues on warnings, errors, and severe errors.

Table 1–5 describes error handling when you specify STOP.

Table 1–5 Error Handling if Action is STOP

Error Condition Specified	Action Taken on Error
WARNING	Command procedure stops on warnings, errors, or severe errors.
ERROR	Command procedure stops on errors or severe errors; it continues on warnings only.
SEVERE_ERROR	Command procedure stops on severe errors; it continues on warnings and errors.

By default, errors cause CDO to stop execution if you do not specify an ON command within a command procedure. When CDO stops a command procedure, it returns you to the CDO prompt.

If you nest command procedures and the ON command is executed, CDO returns you to the CDO prompt, instead of the previous command procedure.

Examples

```
CDO>  @MY_PROCEDURE
SET VERIFY
ON SEVERE_ERROR THEN CONTINUE
DEFINE FIELD INVALID FIELD NAME
DEFINE FIELD INVALID FIELD NAME
                     ^
%CDO-E-KWSYNTAX, syntax error in command line at or near FIELD
DATATYPE IS TEXT IS 7.
^
%CDO-E-KWSYNTAX, syntax error incommand line at or near DATATYPE
DEFINE FIELD CORRECT_NAME_FIELD
DATATYPE IS TEXT SIZE IS 7.
@RECORD.CDO
DEFINE RECORD VALID_RECORD.
CORRECT_NAME_FIELD.
END RECORD.
CDO>
```

In this example, the command procedure specifies ON SEVERE_ERROR THEN CONTINUE. Because the command procedure encounters only an ERROR condition, it continues to execute and defines the CORRECT_NAME_ FIELD field element and the VALID_RECORD record element.

OPEN FILE_ELEMENT Command

Format

OPEN FILE_ELEMENT type-name element-name

Parameters

type-name
Specifies the type (MCS_BINARY or MCS_BINARY subtype) of the file element you are opening. See the *Oracle CDD/Repository Information Model Volume I* for information on these types.

element-name
Specifies the file element you are opening. You can substitute an asterisk (*) wildcard character for this parameter.

Description

The OPEN command opens an existing internal file.

Before you issue the OPEN command, you will most likely want to reserve your file. Reserving the file allows you read/write access. If you do not reserve the file, you have read-only access.

Once you have issued the OPEN command, exit from CDO. Set default to the repository files directory that contains the file, and use the commands appropriate for your system to examine or edit it.

After you have finished with the file, return to CDO and issue the CLOSE FILE_ELEMENT command at the CDO prompt.

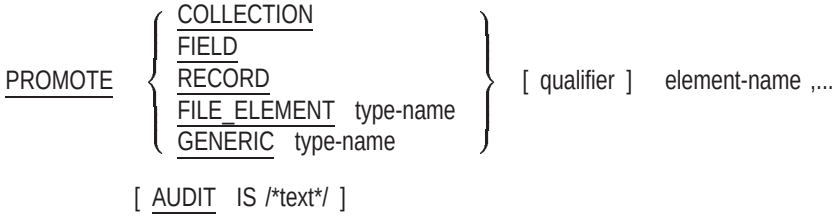
Examples

```
CDO>  RESERVE FILE_ELEMENT MCS_TEXT JULY_REPORT
CDO>  OPEN FILE_ELEMENT MCS_TEXT JULY_REPORT
CDO>  EXIT
```

In this example, CDO opens the text file named JULY_REPORT for editing.

PROMOTE Command

Format



Parameters

type-name
Specifies the type of the file or generic element you are promoting.

element-name
Specifies the element you are promoting.

text
Adds information to the history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_ set of the session to DEC_KANJI.

Qualifiers

/CLOSURE=TO_BOTTOM
/NOCLOSURE (default)
Specifies whether CDO promotes additional elements. Specifying the /CLOSURE=TO_BOTTOM qualifier promotes all children of an element that reside in the same partition.

Description

The PROMOTE command moves a controlled element to the next-higher approval level.

You cannot promote an element more than one approval level at a time.

You cannot promote more than one version of an element at a time. If you do not include a version number, PROMOTE promotes the highest visible version.

An error occurs if you attempt to promote an element that is reserved. The SHOW RESERVATIONS command indicates whether this condition exists.

An error occurs if you attempt to promote an element that is a parent of another element in the same partition.

An error occurs if you attempt to promote an element that is a parent of another element in a partition that is not on the path between the destination partition and the root partition.

An error occurs if you attempt to promote an element from the root partition. The root partition is the highest partition in the partition hierarchy.

When you promote a version to higher levels of approval, the value of the autopurge property determines whether CDO deletes intermediate versions of the element in the source partition. You set the autopurge property for a partition in the DEFINE PARTITION command.

When you promote a version to higher levels of approval, you can still access previous versions at lower levels. However, you must explicitly promote any changes you make to the element at lower levels to see this change reflected in the higher level versions.

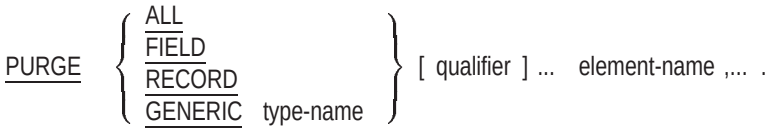
Examples

```
CDO>  PROMOTE RECORD /CLOSURE=TO_BOTTOM SUBSCRIBER
```

In this example, the PROMOTE command promotes the record SUBSCRIBER and all children.

PURGE Command

Format



Parameters

type-name
Specifies the type of the generic element you are purging.

element-name
Specifies the element that you are purging. You can use wildcard characters in this parameter.

Qualifiers

/DESCENDANTS
/NODESCENDANTS (default)
Specifies whether CDO also purges children. When you specify the /DESCENDANTS qualifier, and your element is controlled, CDO deletes all children that are not also children of additional elements outside the area defined by your top collection. If the element is uncontrolled, CDO deletes all children that are not also children of any other elements.

/LOG
/NOLOG (default)
Specifies whether CDO displays text confirming that the element was purged.

Description

The PURGE command deletes all but the first and last version of an element. You cannot delete the first version, and intermediate versions are not purged if a branch line descends from them, or if they are children in a dependency relationship.

You can only purge one line of descent at a time. Purge branch lines before main lines.

You must specify a name of an element, or if you are using the PURGE ALL command, use the asterisk (*) wildcard character.

Examples

1.
```
CDO>  PURGE RECORD REGION.INVENTORY.PART.
```

   In this example, the PURGE command deletes all but the first and highest numbered versions of the REGION.INVENTORY.PART record element.

2.
```
CDO>  PURGE ALL GREF*.
```

   In this example, the PURGE command deletes all but the first and highest versions of each element that begins with the letters GREF.

3.
```
CDO> PURGE FIELD FIRST_NAME.
   .
   .
   .
CDO> PURGE RECORD /DESCENDANTS FULL_NAME.
```

   In this example, the first PURGE command fails because the FIRST_ NAME field element is a child of the FULL_NAME record element. By purging the descendants of FULL_NAME, you can accomplish the desired purge.

REMOVE Command

Format

Parameters

type-name
Specifies the type of the generic element.

directory-name
Specifies the directory name you are removing.

Description

The REMOVE command allows you to remove a directory name for an element that has multiple directory names in a CDO repository or repositories. REMOVE affects all versions of an element.

If you issued the ENTER command to reserve elements in a distributed environment, you must issue the REMOVE command after replacing these elements. The REMOVE command deletes the directory name you entered for your element, so other distributed users cannot access or change it by mistake. See the *Using Oracle CDD/Repository on OpenVMS Systems* for information on reserving elements in a distributed environment.

If the name you specify is the only directory name for an element or the element does not have a parent, CDO records an error and does not remove the directory name.

Examples

1.
```
CDO>  REMOVE FIELD POSTAL_CODE
```

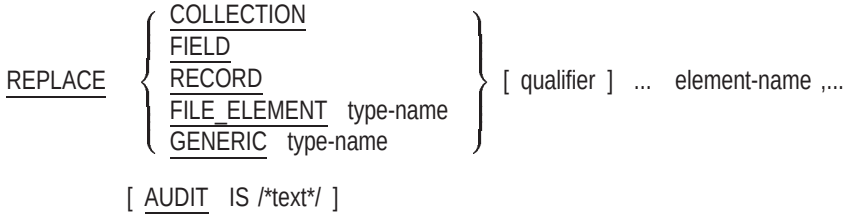
   In this example, the REMOVE command deletes the POSTAL_CODE alternate directory name from the repository.

2.
```
CDO>  REPLACE COLLECTION CORP_DATA_DEFS
CDO>  REMOVE COLLECTION CORP_DATA_DEFS
```

   In this example, the REMOVE command deletes the CORP_DATA_DEFS alternate directory name on a local node.

REPLACE Command

Format



Parameters

type-name
Specifies the type of the file or generic element you are replacing.

element-name
Specifies the element you are replacing. You can substitute an asterisk (*) wildcard character for this parameter unless you are working in the CDD$METADATA collection, where all wildcards are not allowed.

text
Adds information to the history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_ set of the session to DEC_KANJI.

Qualifiers

/BRANCH=branchname
/NOBRANCH (default)
Specifies whether CDO creates a version on a new branch line or on an existing line of descent. The BRANCH option you specify in the REPLACE command overrides any BRANCH option you specified in the RESERVE command.

/CLOSURE=keyword
/NOCLOSURE (default)
Specifies whether CDO replaces additional elements. A CLOSURE operation fails if any element is a child of an element outside the area defined by the CLOSURE keyword.

The /CLOSURE qualifier takes one of the following keywords:

CLOSURE Keyword	Behavior
TO_BOTH	Replaces the element specified and all parents and children.
TO_BOTTOM	Replaces the element specified and all children.
TO_TOP	Replaces the element specified and all parents.

If you specify TO_BOTH or TO_TOP, CDO ignores parents of the top collection.

/LOG
/NOLOG (default)
Specifies whether CDO displays text identifying each element as the element is replaced.

Description

The REPLACE command checks in a version of an element that you previously checked out with the RESERVE command.

You must have a context set to issue the REPLACE command.

The REPLACE command converts the ghost copy reserved to your context into a new version stored in a base partition. This new version is accessible to anyone whose context is set to that base partition. To create subsequent versions of a controlled element, use the RESERVE and REPLACE commands, rather than the DEFINE command.

An error occurs if you issue the /BRANCH qualifier with a branch name already in use.

If you issue the REPLACE command with branch information specified in the element name, do not include a /CLOSURE qualifier. The /CLOSURE qualifier will add this branch name to the name of every element you replace.

If you are issuing the REPLACE command in a distributed environment, you must issue the REMOVE command after issuing the RESERVE command.

If you decide to discard the changes you have made to your working copy, use the UNRESERVE command to cancel your reservation and destroy your copy.

If you decide to merge a branch line that you have created back into the main line of descent, use the MERGE command.

Examples

1.
```
CDO>  RESERVE FIELD /CLOSURE=TO_TOP FIRST_NAME
CDO>  CHANGE FIELD FIRST_NAME
cont>   DESCRIPTION IS "SPELL OUT THOSE INITIALS!"
```

   In this example, the REPLACE command replaces the FIRST_NAME field element and all parents to the top of the collection hierarchy.

2.
```
CDO>  RESERVE FIELD /CLOSURE=TO_TOP /BRANCH=AUDITOR PRODUCT_NUMBER
CDO>  CHANGE FIELD PRODUCT_NUMBER(1:AUDITOR:1)
cont>   AUDIT IS "THESE VERSIONS SUBMITTED FOR AUDIT".
CDO>  REPLACE FIELD PRODUCT_NUMBER(1:AUDITOR:1)
CDO>  REPLACE RECORD /CLOSURE=TO_TOP PRODUCTS
```

   In this example, the initial REPLACE command replaces the branch version of the PRODUCT_NUMBER field element; a second REPLACE command replaces those elements on the path from PRODUCT to the top collection.

3.
```
CDO>  RESERVE FIELD /CLOSURE=TO_TOP PRODUCT_NUMBER(1:AUDITOR:1)
CDO>  CHANGE FIELD PRODUCT_NUMBER(1:AUDITOR:2)
cont>   AUDIT IS "SEVEN OBSOLETE PRODUCT NUMBERS".
CDO>  REPLACE FIELD PRODUCT_NUMBER(1:AUDITOR:2)
CDO>  REPLACE RECORD /CLOSURE=TO_TOP PRODUCTS
```

   In this example, the REPLACE command replaces the new second version in the AUDITOR branch line.

4.
```
CDO>  REPLACE FIELD /BRANCH=QA PRODUCT_NUMBER(1:AUDITOR:2)
```

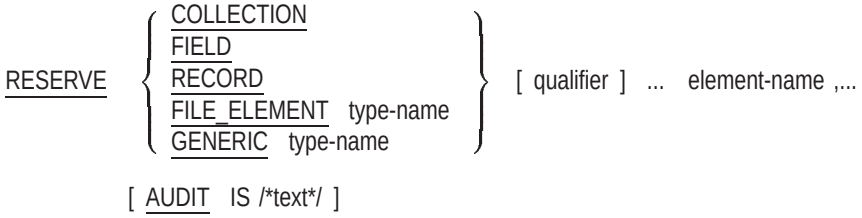
   By substituting this command for REPLACE FIELD PRODUCT_ NUMBER(1:AUDITOR:2) in the previous example, you can change the branch name from AUDITOR to QA.

5.
```
CDO>  REPLACE FIELD /BRANCH PRODUCT_NUMBER(1:AUDITOR:2)
```

   By substituting this command for REPLACE FIELD PRODUCT_ NUMBER(1:AUDITOR:2) in the third example, you can reverse the creation of the branch line and replace PRODUCT_NUMBER on the main line.

RESERVE Command

Format



Parameters

type-name
Specifies the type of the file or generic element you are reserving.

element-name
Specifies the element you are reserving. You can substitute an asterisk (*) wildcard character for this parameter. If the element is uncontrolled, you must reserve the highest version.

text
Adds information to the history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_ set of the session to DEC_KANJI.

Qualifiers

/BRANCH=branchname
/NOBRANCH (default)
Specifies whether CDO creates a version on a new branch line or on an existing line of descent. The element must be controlled to use the /BRANCH qualifier.

/CLOSURE=keyword
/NOCLOSURE (default)
Specifies whether CDO reserves additional elements. A CLOSURE operation fails if any element is a child of an element outside the area defined by the CLOSURE keyword.

CLOSURE takes one of the following keywords:

CLOSURE Keyword	Behavior
TO_BOTH	Reserves the element specified and all owners and members.
TO_BOTTOM	Reserves the element specified and all members.
TO_CLOSURE	Reserves the element specified, all owners, and any element under the top collection that depends on the element specified.
TO_TOP	Reserves the element specified and all owners.

In most cases, you can specify the TO_BOTH, TO_TOP, or TO_BOTTOM keywords. The TO_CLOSURE keyword is useful when you are working with the source and derived files common in system building applications.

If you specify TO_BOTH, TO_CLOSURE, or TO_TOP, CDO ignores owners of the top collection.

/LOG
/NOLOG (default)
Specifies whether CDO displays text identifying each element as the element is reserved.

/OUTPUT (default)
/NOOUTPUT
The /NOOUTPUT qualifier lets you reserve a FILE_ELEMENT of type MCS_BINARY without copying the file into the current context directory. This capability is useful for using the repository to manage binary files that are superseded each time they are reserved. It lets you reserve the file element without incurrring the processing time to create the binary file in the context directory.

If you use the /NOOUTPUT qualifier in a RESERVE command, you must update the context directory with the latest binary file to be replaced. If you do not update the current context directory with a new file, the REPLACE command will fail. When the REPLACE command fails, CDO displays an error message containing the full directory specification of the reserved file that CDO was attempting to replace. Refer to this error message and place a new copy of the specified file in the context directory.

If you reserve a file with the /NOOUTPUT qualifier, CDO does not create the file in the context directory. If you manually place a file in the context directory and then issue the UNRESERVE command, the reserved file is unreserved and any copies of the file in the context directory are deleted. This occurs even if you manually superseded any files in the context directory.

Note

If you use the VERIFY/FIX command on a repository, any files reserved with the /NOOUTPUT qualifier are created in the context directory, because the VERIFY command cannot find reserved files. This performance cost has always been associated with the VERIFY/FIX command when it cannot find reserved files.

Description

The RESERVE command checks out a version of a controlled element. A controlled element is one of the following:

- An element you created using Oracle CDD/Repository inside a context
- An element you have controlled with the CONSTRAIN command

You must have a context set to issue the RESERVE command.

If you are issuing the RESERVE command in a distributed environment, you must issue the ENTER command before issuing the RESERVE command.

The RESERVE command creates a copy of the immutable version that is stored in the base partition associated with your context. This copy is called a ghost; it is reserved to your context and you can modify it.

In general, the ghost has a version number that is one number higher than that of the original version in the partition. For example, if you reserve PRODUCT(1), the ghost of this version is called PRODUCT(2).

If, however, you specify the creation of a parallel line of development (or branch), the ghost becomes the first version in that branch line. For example, if you reserve PRODUCT(2) with a /BRANCH=AUDITOR qualifier, the ghost copy of this version is called PRODUCT(2:AUDITOR:1).

If you specify the /BRANCH qualifier, you can reserve any version of a controlled element in a line of descent that is available for reservation. If an element is uncontrolled, an error occurs if you attempt to reserve any version but the latest version in a line of descent.

An error occurs if you issue the /BRANCH qualifier with a branch name already in use.

An error occurs if you attempt to reserve a child without previously reserving its owners. Use the /CLOSURE qualifier to reserve as many elements as necessary.

An error occurs if you attempt to reserve a version of an uncontrolled element that has already been reserved. An element can have only one outstanding reservation.

When you finish modifying your working copy of a version, you use the REPLACE command to check in the new version to the partition or the repository.

If you decide to discard the changes you have made to your working copy, use the UNRESERVE command to cancel your reservation and destroy your copy.

If you decide to merge a branch line that you have created back into the main line of descent, use the MERGE command.

When reserve is invoked with a branch name specified, the new ghost version is created with a name that incorporates the branch name and is properly linked to the element from which the branch line originates.

If the target of a reserve notice is involved in one or more correspondence relationships, those relationships may be propagated to the new version.

Examples

1.
```
CDO>  RESERVE FIELD /CLOSURE=TO_TOP FIRST_NAME
```

   In this example, the RESERVE command with the /CLOSURE=TO_TOP qualifier reserves the FIRST_NAME field element and all owners to the top collection.

2.
```
CDO>  RESERVE FIELD /CLOSURE=TO_TOP /BRANCH=AUDITOR PRODUCT_NUMBER
```

   In this example, the RESERVE command with the /CLOSURE=TO_TOP qualifier reserves all elements on the path between the top collection and field PRODUCT_NUMBER; the /BRANCH qualifier creates a branch line AUDITOR descending from PRODUCT_NUMBER.

3.
```
CDO>  RESERVE FIELD /CLOSURE=TO_TOP PRODUCT_NUMBER(1:AUDITOR:1)
```

   In this example, the RESERVE command reserves the first version in the AUDITOR branch line.

4.
```
SYSTEM collection
   SOURCE_FILES collection
      FIRST_FILE.C
      SECOND_FILE.C
      INC.H
   DERIVED_FILES collection
      FIRST_FILE.OBJ
      SECOND_FILE.OBJ
      IMAGE_FILE.EXE
```

   In this example, the code shows a collection hierarchy with SYSTEM defined as the top collection. The dependencies in SYSTEM are as follows:

   - IMAGE_FILE.EXE depends on FIRST_FILE.OBJ and SECOND_ FILE.OBJ.
   - FIRST_FILE.OBJ depends on FIRST_FILE.C and INC.H.
   - SECOND_FILE.OBJ depends on SECOND_FILE.C and INC.H.

5.
```
CDO>  RESERVE FILE_ELEMENT MCS_BINARY /CLOSURE=TO_CLOSURE INC.H
```

   In this example, the RESERVE command reserves the following elements:

   - Element INC.H
   - Owners of the element specified, SOURCE_FILES and SYSTEM
   - Elements that directly or indirectly depend on the element, FIRST_ FILE.OBJ, SECOND_FILE.OBJ, IMAGE_FILE.EXE

6.
```
CDO> RESERVE FILE_ELEMENT MCS_BINARY /NOOUTPUT "/ISAM_FILE.DAT"
```

   In this example, ISAM_FILE.DAT is reserved using the /NOOUTPUT qualifier, but the file is not copied into the current context directory.

ROLLBACK Command

Format

ROLLBACK

Description

The ROLLBACK command terminates a transaction and undoes all changes that have been made to the database since the program's most recent START_TRANSACTION command. The ROLLBACK command also releases all locks, closes all open streams, and releases all readied relations. It affects all databases participating in the currently open transaction. The ROLLBACK command implicitly performs the END_STREAM statement.

Restrictions

- When you delete a record, 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. Using CDO, there is no way to reuse those local fields. It is possible to use local fields through the Oracle CDD/Repository APIs. Therefore, the local fields cannot be automatically deleted at the same point in the transaction as the record.

  You must either delete the record and field in separate transactions (outside the START_TRANSACTION . . . COMMIT stream of commands) or, to accomplish this in one transaction, use the ENTER command to enter the local field, delete the record, delete the local field, and then delete the global field.

- Usually, if Oracle CDD/Repository issues any errors between the START_TRANSACTION and COMMIT commands, it forces you to roll back the transaction. In some cases, such as in the CHANGE or DELETE commands, Oracle CDD/Repository allows you to commit the transaction. The general rules are:

  – If you receive an Oracle CDD/Repository error of E or F severity, such as a CDD-E-NODNOTFND message, you must abort the transaction.

  – If you receive a CDO error of E or F severity, such as CDO-E-NOTFOUND, you can continue to operate in the current transaction.

Examples

```
CDO>  START_TRANSACTION
CDO> DEFINE RECORD REC2.
cont> FLD1. END RECORD.
   .
   .
   .
CDO> SHOW RECORD REC2
Definition of record REC2 |   Contains field          FLD1
CDO> ROLLBACK
CDO> SHOW RECORD REC2
%CDO-E-ERRSHOW, error displaying an object
-CDO-E-NOTFOUND, entity REC2 not found in dictionary
```

In this example, a record is defined within a transaction, but because the transaction is terminated using the ROLLBACK command, the record is not defined.

SET CHARACTER_SET Command

Format

SET CHARACTER_SET character-set

Parameters

character-set
Specifies the type of characters to be used during the current CDO session. Table 1–6 lists the valid character set names.

Table 1–6 Valid Character Set Names

Character Set Type	Character Set	Description
MCS	DEC_MCS	A set of international alphanumeric characters
Kanji+ASCII	DEC_KANJI	Japanese characters as defined by the JIS X0208:1990 standard, Narrow Katakana characters, as defined by the JIS X0201:1976 standard, and ASCII characters

Description

Specifies the valid characters that you can use for an element name, the initial value field property, and in comments. You must set the character-set parameter to DEC_KANJI when you use Japanese with Oracle CDD/Repository.

If you omit the SET CHARACTER SET command, Oracle CDD/Repository references the equivalence name of the CDD$CHARACTER_SET logical as the character set for the session. If this logical is not assigned, the default character set is DEC_MCS.

Restriction

DEC_KANJI is not available through the CDO editor.

Oracle CDD/Repository accepts a maximum field and record name length of 31 octets (31 characters for ASCII/MCS characters; 15 characters for Kanji and Narrow Katakana).

Oracle CDD/Repository accepts the following characters as the field and record name when the character-set parameter is DEC_KANJI:

- Kanji (Japanese characters as defined by the JIS X0208:1990 standard)
- Katakana (Japanese phonetic alphabet, Narrow Katakana, as defined by the JIS X0201:1976 standard)
- ASCII characters (A-Z, a-z, _, $)

Examples

```
CDO>  SET CHARACTER_SET DEC_KANJI
```

In this example, the character-set parameter is DEC_KANJI to support Japanese characters during the CDO session.

SET CONTEXT Command

Format

SET CONTEXT [context-name] [AUDIT IS /*text*/]

Parameters

context-name
Specifies the context you are setting. If you omit this parameter, CDO sets the current context to null.

text
Adds information to the history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_ set of the session to DEC_KANJI.

Description

The SET CONTEXT command defines a context as the current context and implicitly controls all versioned elements that you define while the context is set.

A context is a nonversioned element. Do not include a version number in the context name.

If the top collection associated with the context is undefined, the SET CONTEXT command takes the element named in the next command as the top collection for the context. An error occurs if the next command is not a DEFINE COLLECTION command.

The context set remains the current context until you issue another SET CONTEXT command, you delete the context, or you end the CDO session.

As an alternative to the SET CONTEXT command, you can define the CDD$CONTEXT logical name. Once set, this context becomes the current context each time you access the repository. For example:

```
$  DEFINE CDD$CONTEXT "cdd$disk:[smith.rep]test_context"
```

Examples

```
CDO>  DEFINE PARTITION FIRST_BASELEVEL.
CDO>  DEFINE CONTEXT DEVELOPMENT_CONTEXT.
cont>   BASE_PARTITION FIRST_BASELEVEL.
CDO>  SET CONTEXT DEVELOPMENT_CONTEXT
CDO>  DEFINE COLLECTION COMPILER_C.            1
CDO>  RESERVE COLLECTION COMPILER_C
CDO>  DEFINE COLLECTION FRONT_END.
CDO>  DEFINE COLLECTION BACK_END.
CDO>  REPLACE COLLECTION COMPILER_C.
CDO>  SET CONTEXT                              2
CDO>  DEFINE RECORD ISSUES.
```

In this example, the SET CONTEXT command sets the current context and implicitly defines a collection as the top collection. Subsequent definitions will be implicitly controlled.

1 The DEFINE COLLECTION command sets the top collection for the current context. All definitions made within the current context are attached to the top collection. FRONT_END and BACK_END are attached to COMPILER_C.

2 The SET CONTEXT command sets the current context to a null value. ISSUES is uncontrolled and unattached because it is defined outside a context.

SET DEFAULT Command

Format

SET DEFAULT path-name

Parameters

path-name
Specifies a default repository directory. You cannot use wildcard characters in this parameter.

Description

The SET DEFAULT command establishes the default repository directory for the current CDO session.

You can use a logical name that translates to a search list as the path name in the SET DEFAULT command. After setting the default repository area, commands that directly affect elements, such as CHANGE, DEFINE, or DELETE, only operate on the first occurrence of the element in the search list. However, the DIRECTORY command searches through all the repository areas specified in the search list.

Examples

```
CDO>  SET DEFAULT DISK:1[SMITH.DATA]REVIEW
```

In this example, the SET DEFAULT command sets the default directory to the REVIEW directory in the DISK1:[SMITH.DATA] anchor repository.

SET KEY Command

Format

SET KEY qualifier

Qualifiers

/STATE=key-state
Specifies the key state to be set.

Description

The SET KEY command sets the current key state. See the DEFINE KEY command description for information on the key states.

Examples

```
CDO>  SET KEY/STATE=ONE
```

In this example, the SET KEY command sets the key state to ONE.

SET OUTPUT Command

Format

SET OUTPUT [file-spec]

Parameters

file-spec
Specifies the file to which CDO sends the output from CDO commands.

Description

The SET OUTPUT command defines where CDO sends the output from CDO during a session.

If you specify a file with the SET OUTPUT command, CDO sends output to the default output location (SYS$OUTPUT) for your current process and to the specified file.

If you specify SET OUTPUT without a file specification, CDO sends output only to the default output location.

The SET OUTPUT command stays in effect until you change it with another SET OUTPUT command.

Examples

```
CDO>  SET OUTPUT
```

In this example, the SET OUTPUT command captures the output from a CDO session and sends it to the default output location for your process.

SET VERIFY Command

Format

SET { VERIFY / NOVERIFY }

Description

The SET VERIFY command causes CDO to display all commands in a command procedure before executing them. The initial setting is off (NOVERIFY).

To override this default, you can issue the SET VERIFY command at the CDO prompt before you process the command procedure. SET VERIFY then remains in effect until you issue a SET NOVERIFY command.

Alternatively, you can insert the SET VERIFY command as the first command within your command procedure. SET VERIFY then remains in effect until the command procedure finishes executing.

Examples

```
CDO>  SET VERIFY
CDO>  @ON.CDO
```

In this example, the SET VERIFY command causes CDO to display all commands in the ON.CDO command procedure as they execute.

SHOW ALL Command

Format

SHOW ALL [qualifier]

Qualifiers

/ALL
Displays system-specified properties, such as time of creation, and user-specified properties, such as data type, for each element in the directory and for all children.

/AUDIT
Displays history list entries for each element. AUDIT=ALL displays history list entries for each element and all children.

/BRIEF (default)
Displays user-specified properties for each element, and provides the names of direct children.

/FULL
Displays the user-specified properties for the element and for all children.

Description

The SHOW ALL command displays a complete or partial list of properties for all visible elements in the default directory.

If you have your default directory set to a directory in the compatibility repository, the SHOW ALL command displays DMU record definitions of the CDD$RECORD type, but cannot display other definitions that may be stored in your DMU repository. Some examples of DMU definitions that CDO cannot display include the following:

- ACMS application, menu, task group, and task definitions
- DATATRIEVE domain, plot, table and view definitions, and procedures
- DBMS schema, subschemas, security schemas, and storage schemas
- Oracle Rdb relation, constraint, index, view, and field definitions
- TDMS form, request, and request library definitions

When you display definitions from the compatibility repository, CDO displays DMU definitions in CDO format.

You must have read access to an element for CDO to display information on that element.

CDO displays type definitions only if your default directory is set to the CDD$PROTOCOLS directory.

Examples

```
CDO>  SHOW ALL
```

In this example, because no qualifier is specified, the SHOW command displays default BRIEF information. This information includes the user-specified properties for each element and the names of direct children.

SHOW CHARACTER_SET Command

Format

SHOW CHARACTER_SET

Description

Displays the character set of the current CDO session.

You must set the character-set parameter to DEC_KANJI to use Japanese characters with Oracle CDD/Repository. Use the SET CHARACTER_SET command to specify the character set for a CDO session.

If the setting has not been specified using the SET CHARACTER_SET command, Oracle CDD/Repository references the equivalence name of the CDD$CHARACTER_SET logical name. If this logical name is not assigned, the default character set is DEC_MCS.

Examples

```
CDO>  SHOW CHARACTER_SET
Session Character_set is DEC_MCS
```

In this example the current character set setting is DEC_MCS, which supports a set of international alphanumeric characters.

SHOW COLLECTION Command

Format

SHOW COLLECTION [qualifier] collection-name ,...

Parameters

collection-name
Specifies the collection whose properties you are displaying.

Qualifiers

/ALL
Displays system-specified properties, such as time of creation, and user-specified properties, such as description, for the collection and for all children.

/AUDIT
Displays history list entries for the collection.

/BRIEF (default)
Displays user-specified properties for the collection.

/FULL
Displays user-specified properties for the collection and for all children.

Description

The SHOW COLLECTION command displays a complete or partial list of properties for the collection or collections specified.

If you do not specify a version number for the collection, CDO displays the properties of the highest visible version.

Examples

```
CDO>  SHOW COLLECTION MAMMALS
```

In this example, because no qualifier is specified, the SHOW COLLECTION command displays default BRIEF information. This information includes the user-specified properties for the MAMMALS collection and the names of direct children.

SHOW CONTEXT Command

Format

SHOW CONTEXT [qualifier] [context-name] ,...

Parameters

context-name
Specifies the context whose properties you are displaying. If you omit this parameter, CDO displays the name of the current context.

Qualifiers

/ALL
Displays system-specified properties, such as time of creation, and user-specified properties, such as description, for the context and for all children.

/AUDIT
Displays history list entries for the context.

/BRIEF (default)
Displays user-specified properties for the context, and provides the names of direct children.

/FULL
Displays user-specified properties for the context and for all children.

Description

The SHOW CONTEXT command displays a complete or partial list of properties for the context or contexts you specify.

Because a context is a nonversioned element, CDO does not accept a branch designation or a version number in the context name.

Examples

```
CDO>  SHOW CONTEXT ANIMAL_KINGDOM
```

In this example, because no qualifier is specified, the SHOW CONTEXT command displays default BRIEF information. This information includes the user-specified properties for the ANIMAL_KINGDOM context and the names of direct children.

SHOW DATABASE Command

Format

SHOW DATABASE [qualifier] [database-name]

Parameters

database-name
Specifies the physical database whose properties you are displaying. By default, if you do not specify a full path name for the database, CDO displays all physical database elements in your default directory.

Qualifiers

/ALL
For an RMS database, displays the database name and description, record definition, file organization, fully qualified path name, and system-specified properties.

For an Oracle Rdb database, displays the database name, file name, and fully qualified path name, as well as the system-specified properties.

/AUDIT
Displays the history list entries for the database definition. The /AUDIT=ALL qualifier displays the history list for all elements owned by the database.

/BRIEF (default)
For an RMS database, displays the database name and description, record name, fully qualified path name, and file organization properties.

For an Oracle Rdb database, displays the database name, file name, and fully qualified path name.

/FULL
For an RMS database, displays the database name and description, record definition properties, file organization properties, and fully qualified path name.

For an Oracle Rdb database, displays the database name, file name, and fully qualified path name.

Description

The SHOW DATABASE command displays a complete or partial list of properties for the database elements specified.

If you do not specify a version number for a database element, CDO displays the highest visible version.

When you use the SHOW DATABASE command to display an Oracle Rdb database element, CDO shows only the database name, file name, and the fully qualified path name. Use the SHOW GENERIC command with the /FULL qualifier or use SQL (structured query language) to view the complete definition of an Oracle Rdb database.

Examples

```
CDO>  SHOW DATABASE DEPT5
```

In this example, because no qualifier is specified, the SHOW DATABASE command displays default BRIEF information. This information includes the DEPT5 Oracle Rdb database name, file name, and fully qualified path name.

SHOW DEFAULT Command

Format

SHOW DEFAULT

Description

The SHOW DEFAULT command displays the current default CDO repository directory.

If you set your default directory to a logical name that translates to a search list and you issue the SHOW DEFAULT command, CDO displays the names of the repository areas in the same order as they appear in the search list.

Examples

1.
```
CDO>  SET DEFAULT MY_DICT
CDO>  SHOW DEFAULT
```

   In this example, the SHOW DEFAULT command displays the names of the local and remote repository areas specified by the logical name MY_DICT.

2.
```
CDO>  SHOW DEFAULT
```

   In this example, the SHOW DEFAULT command displays the current default CDO directory.

SHOW FIELD Command

Format

SHOW FIELD [qualifier] ... [field-name] ,...

[FROM DATABASE database-name]

Parameters

field-name
Specifies the field element whose properties you are displaying.

When you use the FROM DATABASE clause, specify only one field name for each SHOW FIELD command.

Specify an asterisk (*) wildcard character for the entire field-name parameter only. If you use a wildcard character as part of the field-name parameter, an error occurs.

database-name
Specifies the Oracle Rdb database that contains the field. CDO requires this parameter for fields from an Oracle Rdb database. CDO accepts wildcard characters in the database name.

Qualifiers

/ALL
Displays system-specified properties, such as time of creation, and user-specified properties, such as description, for the field name and for all children.

/AUDIT
Displays history list entries for the field name. The /AUDIT=ALL qualifier displays the history list entries for the field name and for all children. Do not use this qualifier if the field name you specify is from an Oracle Rdb database definition.

/BRIEF (default)
Displays user-specified properties for the field name, and provides the names of direct children.

/FULL
Displays user-specified properties for the field name and for all children.

/SYSTEM
/NOSYSTEM (default)
Specifies whether CDO displays Oracle Rdb system relations.

/RDB_METADATA
/NORDB_METADATA (default)
Specifies whether CDO displays Oracle Rdb system relations. This qualifier is synonymous with the /SYSTEM qualifier.

Description

The SHOW FIELD command displays a complete or partial list of properties for the field names you specify, provided you have read privileges.

If you do not specify a full path name (or the FROM DATABASE clause) for the field name, CDO searches your current default directory for the field name. If you do not specify a field name, CDO displays the properties of all field names in your default directory.

If you do not specify a version number for a field name, CDO displays the properties of the highest visible version.

Note

If you make incompatible changes to the CDD$DATA_ELEMENT type, supplied by Oracle CDD/Repository, the SHOW FIELD command may not display those properties whose data types you have modified.

If a field has character set attributes, you can display them using the SHOW and EXTRACT commands; in addition, you can use the SHOW command to display size information of a field in both character-based size and octet-based size. See the descriptions of SET CHARACTER_SET command and the DATATYPE field property for more information.

Examples

1.
```
CDO>  SHOW FIELD CORPORATE_ZIPCODE FROM DATABASE DEPT3
```

   In this example, because no qualifier is specified, the SHOW FIELD command displays default BRIEF information. This information includes the user-specified properties for the CORPORATE_ZIPCODE field name and the names of direct children.

2.
```
CDO> DEFINE FIELD FULL_NAME
cont> DATATYPE TEXT CHARACTER_SET KANJI
cont> SIZE 2 CHARACTERS.
CDO> SHOW FIELD FULL_NAME
Definition of field FULL_NAME
|   Datatype              text size is 2 characters (4 Octets)
|   Character_set         KANJI
```

This example defines and shows the field FULL_NAME.

SHOW FILE_ELEMENT Command

Format

SHOW FILE_ELEMENT type-name [qualifier] element-name ,...

Parameters

type-name
Specifies the type (MCS_BINARY or MCS_BINARY subtype) of the file element you are displaying. See the *Oracle CDD/Repository Information Model* for information on these types.

element-name
Specifies the file element whose properties you are displaying.

Qualifiers

/ALL
Displays system-specified properties, such as time of creation, and user-specified properties, such as description, for the file element and for all children.

/AUDIT
Displays history list entries for the file element.

/BRIEF (default)
Displays user-specified properties for the file element, and provides the names of direct children.

/FULL
Displays user-specified properties for the file element and for all children.

Description

The SHOW FILE_ELEMENT command displays a complete or partial list of properties for the file element or elements you specify.

If you do not specify a version number for a file element, CDO displays the properties of the highest visible version.

Examples

```
CDO>  SHOW FILE_ELEMENT MCS_TEXT CAT
```

In this example, because no qualifier is specified, the SHOW command displays default BRIEF information. This information includes the user-specified properties for the CAT file element and the names of direct children.

SHOW GENERIC Command

Format

SHOW GENERIC type-name [qualifier] [element-name] ,...

[FROM DATABASE database-name]

Parameters

type-name
Specifies the type of the generic element whose properties you are displaying. This type cannot be MCS_BINARY, a subtype of MCS_BINARY, MCS_ COLLECTION, MCS_CONTEXT, or MCS_PARTITION. See the *Oracle CDD/Repository Informatin Model* for information on these types.

element-name
Specifies the generic element whose properties you are displaying.

When you use the FROM DATABASE clause, specify only one element name for each SHOW GENERIC command.

Specify an asterisk (*) wildcard character for the entire element-name parameter only. If you use a wildcard character as part of the element-name, an error occurs.

database-name
Specifies the Oracle Rdb database that contains the element. CDO requires this parameter for elements from an Oracle Rdb database. You can include wildcard characters in the database name.

Qualifiers

/ALL
Displays system-specified properties, such as time of creation, and user-specified properties, such as description, for the generic element and for all children.

/AUDIT
Displays history list entries for the generic element. The /AUDIT=ALL qualifier displays the history list for all children. CDO displays history list entries only if you specified the CDD$HISTORY_LIST relationship as part of this generic element's type definition.

/BRIEF (default)
Displays user-specified properties for the generic element, and provides the names of direct children.

/FULL
Displays user-specified properties for the generic element and for all children. CDO displays description text only if you specified the CDD$DESCRIPTION property as part of this generic element's type definition.

Description

The SHOW GENERIC command displays a complete or partial list of properties for the generic element or elements you specify.

If you specify the type name but not an element name, the SHOW GENERIC command displays all elements of the specified type in your default directory, provided that you have privilege to display them.

You must have read access to all components of the generic element for CDO to display those components.

If you do not specify a version number for a generic element, CDO displays the highest visible version.

You can display a field definition by specifying CDD$DATA_ELEMENT as the type name and the name of the field definition as the element name.

You can display a record definition by specifying CDD$DATA_AGGREGATE as the type name and the name of the record definition as the element name.

You can display indexes or constraints in an Oracle Rdb database element by specifying CDD$INDEX or CDD$CONSTRAINT as the type name and the name of the index or constraint as the element name. Remember to include the FROM DATABASE clause.

Examples

```
CDO>  SHOW GENERIC BOOK REFERENCE_MANUAL
```

In this example, because no qualifier is specified, the SHOW command displays default BRIEF information. This information includes user-specified properties for the REFERENCE_MANUAL element and the names of direct children.

SHOW KEY Command

Format

SHOW KEY [qualifier] ... [key-name]

Parameters

key-name
Specifies the key whose properties you are displaying.

Qualifiers

/ALL
Displays all key definitions in a key state. You cannot use the /ALL qualifier if you specify one or more key names.

/BRIEF (default)
Displays the key definition and state.

/DIRECTORY
/NODIRECTORY (default)
Displays the names of all states for which you have defined keys. If you specify the /DIRECTORY qualifier, you cannot specify any other SHOW KEY qualifiers.

/FULL
/NOFULL (default)
Displays all qualifiers for the key definition you specify. Specifying the /NOFULL qualifier gives the same results as the /BRIEF qualifier.

/STATE=key-state
/NOSTATE (default)
Displays key definitions for the state you specify. The /NOSTATE qualifier displays key definitions for the current state.

Description

The SHOW KEY command displays a complete or partial list of properties for the key you specify.

If you do not specify a key name, CDO displays the definition for all keys.

You use the DEFINE KEY command to create key definitions.

Examples

```
CDO>  SHOW KEY PF3
```

In this example, because no qualifier is specified, the SHOW command displays default BRIEF information. This information includes the key definition and state for the PF3 key.

SHOW NOTICES Command

Format

SHOW NOTICES element-name ,...

Parameters

element-name
Specifies the element whose notices you are displaying.

Description

The SHOW NOTICES command displays the notices at the element or elements you specify. CDO sends notices to elements when you:

- Change an element and the change affects other elements. For example, if you change the name of a field element in a database, the database element may need to be integrated.
- Change an element and the change affects the parent. For example, if you delete the name of a record element in a database, the database element needs to be integrated.
- Create a new version of an element. For example, if you create a new record element that appears in a program, the program needs to be recompiled.

If you issue the SHOW NOTICES command for an element that does not have notices, CDO informs you that the element does not have notices.

You can display new version notices by using the SHOW NOTICES command at any of the member's parents that have a CDD$NOTICE_ACTION property value of SUCCESS or SIGNAL.

You can display notices that the CHANGE command generates by using the SHOW NOTICES command at any of the member's parents that have a CDD$NOTICE_ACTION property value of SIGNAL.

CDO sends notices when you either change a member of a relationship with the CHANGE command or you define a new version of the member.

The three types of notices that definitions receive and the meanings of these notices are:

- CDD$K_POSSIBLY_INVALID

A definition used by this definition has changed. This change might affect this definition. This notice indicates the name of the definition that changed.

- CDD$K_INVALID

  A definition used by this definition changed or was removed from the repository. This definition is invalid. If a changed definition initiated the notice, the notice supplies the name of the definition. If a deleted definition initiated the notice, the notice does not supply a definition name.

- CDD$K_NEW_VERSION

  A new version of a definition used by this definition was created. The notice supplies the name of the definition that has the new version.

- CDD$K_CHILD_USAGE

  A relationship from one of the definitions used by this element to one of its children was changed. The notice indicates the owner of the changed relationship. You cannot generate this type of notice using CDO. Only programs using the Oracle CDD/Repository callable interface can cause this notice. You can, however, read this type of notice using CDO.

Examples

1.
```
CDO>  SHOW NOTICES EMPLOYEE_REC
   .
   .
   .
CDO>  CLEAR NOTICES EMPLOYEE_REC
```

In this example, the SHOW NOTICES command displays notices sent to the EMPLOYEE_REC record. You can clear any notices by issuing the CLEAR NOTICES command.

2.
```
CDO>  CHANGE FIELD FLD_A
cont>   DATATYPE IS SIGNED LONGWORD.
%CDO-I-DBMBR, database DISK1:[SMITH.DICT]MY_RDB_DB(1) may need to be
  integrated
CDO>  SHOW NOTICES MY_RDB_DB
DISK1:[SMITH.DICT]MY_RDB_DB(1) is possibly invalid, triggered by entity
DISK1:[SMITH.DICT]FLD_A(1)
CDO>
```

In this example, CDO sends a notice that a database might require integration as a consequence of the CHANGE command.

You can use the SHOW NOTICES command to display this notice at the MY_RDB_DB database definition.

3.
```
CDO>  DEFINE FIELD FLD_B
cont> DATATYPE SIGNED LONGWORD.
CDO>  SHOW NOTICES REC_B
DISK1:[SMITH.DICT]RDB_REC_B(1) uses an entity which has new versions,
triggered by CDD$DATA_ELEMENT DISK1:[SMITH.DICT]FLD_B(1)
CDO>  SHOW NOTICES MY_RDB_DB
DISK1:[SMITH.DICT]MY_RDB_DB(1) uses an entity which has new versions,
triggered by CDD$DATA_ELEMENT DISK1:[SMITH.DICT]FLD_B(1)
DISK1:[SMITH.DICT]MY_RDB_DB(1) is possibly invalid, triggered by
CDD$DATA_ELEMENT DISK1:[SMITH.DICT]FLD_A(1)
```

When you create a new version of the FLD_B field definition by using the DEFINE FIELD command, CDO sends new version notices to the parents of FLD_B. The following set of examples shows this sequence of events:

1. The DEFINE FIELD command creates a new version of the FLD_B field definition.
2. The first SHOW NOTICES command shows that FLD_B's immediate parent, REC_B record definition received the new version notice when CDO created the new version of FLD_B.
3. The second SHOW NOTICES command shows two notices at the MY_RDB_DB CDD$DATABASE definition. The CHANGE command sends one notice on behalf of the FLD_A field definition from the previous example, and the DEFINE FIELD command sends a new version notice on behalf of the new version of the FLD_B field definition.

4.
```
CDO> CLEAR NOTICES MY_RDB_DB
CDO>  SHOW NOTICES MY_RDB_DB
%CDO-I-NONOTICES, DISK1:[SMITH.DICT]MY_RDB_DB(1) has no notices
```

To clear the notices at MY_RDB_DB, use the CLEAR NOTICES command.

To verify that you cleared the notices at MY_RDB_DB, use the SHOW NOTICES command. If you ask to see the notices at a definition without notices, CDO responds that there are no notices.

SHOW PARTITION Command

Format

SHOW PARTITION [qualifier] partition-name ,...

Parameters

partition-name
Specifies the partition whose properties you are displaying.

Qualifiers

/ALL
Displays system-specified properties, such as time of creation, and user-specified properties, such as description, for the partition and for all children.

/AUDIT
Displays history list entries for the partition.

/BRIEF (default)
Displays user-specified properties for the partition, and provides the names of direct children.

/FULL
Displays user-specified properties for the partition and for all children.

Description

The SHOW PARTITION command displays a complete or partial list of properties for the partition or partitions specified.

Because a partition is a nonversioned element, CDO does not accept a branch designation or a version number in partition names.

Examples

```
CDO>  SHOW PARTITION INITIAL_PROPOSAL
```

In this example, because no qualifier is specified, the SHOW PARTITION command displays default BRIEF information. This information includes the user-specified properties for the INITIAL_PROPOSAL partition and the names of direct children.

SHOW PRIVILEGES Command

Format

SHOW PRIVILEGES FOR { DIRECTORY | FIELD | RECORD } element-name ,...

SHOW PRIVILEGES FOR GENERIC type-name

SHOW PRIVILEGES FOR REPOSITORY anchor-name

Parameters

element-name
Specifies the element whose access rights you are displaying.

type-name
Specifies the type of the generic element.

anchor-name
Specifies the anchor directory of the repository for which you want to display privileges.

Description

The SHOW PRIVILEGES command displays the access rights for the elements you specify.

To display your privileges for a type, use the SHOW PROTOCOL command.

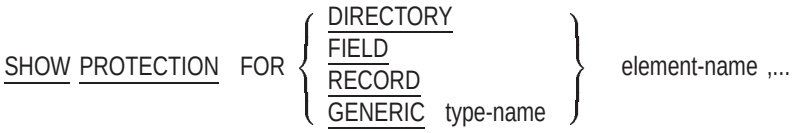
Examples

```
CDO>  SHOW PRIVILEGES FOR FIELD CURRENT_SALARY
```

In this example, the SHOW PRIVILEGES command displays your access rights to the CURRENT_SALARY field element.

SHOW PROTECTION Command

Format



SHOW PROTECTION FOR REPOSITORY anchor-name

Parameters

type-name
Specifies the type of the generic element.

element-name
Specifies the element whose ACL you are displaying.

anchor-name
Specifies the repository anchor whose ACL you are displaying.

Description

The SHOW PROTECTION command displays the access control list (ACL) for the element you specify. When you specify FOR GENERIC MCS_CONTEXT or FOR REPOSITORY, SHOW PROTECTION also displays the default access control list.

To display the access control list for a type, you can also use the SHOW PROTOCOL command.

Examples

1. `CDO>  SHOW PROTECTION FOR FIELD CURRENT_SALARY`

   In this example, the SHOW PROTECTION command displays the access control list for the CURRENT_SALARY field definition.

2. `CDO> SHOW PROTECTION FOR REPOSITORY CDD$REPOSITORY2`

   In this example, Oracle CDD/Repository translates the logical name for the repository.

SHOW PROTOCOL Command

Format

SHOW PROTOCOL [qualifier] type-name ,...

Parameters

type-name
Specifies the type whose properties you are displaying.

Qualifiers

/ALL
Displays all the possible relationships the type can own, as well as the access control list for the type.

/AUDIT
Displays the history list entries for the type. Specifying the /AUDIT=ALL qualifier displays the history list for all the relationships that the type owns.

/FULL
Displays all the possible relationships this type can own.

/BRIEF (default)
Displays the relationships most commonly owned by this type.

Description

The SHOW PROTOCOL command displays a complete or partial list of properties for the type or types you specify.

CDO looks for types in the CDD$PROTOCOLS directory below your current default anchor. It is not necessary to set default to the CDD$PROTOCOLS directory to display types.

CDO uses relationships to pass notices among repository definitions. Every relationship protocol in the repository has the CDD$NOTICE_ACTION property, whose value controls the passing of notices. You can use the CDO SHOW PROTOCOL command to display the value of the CDD$NOTICE_ ACTION property.

The CDD$NOTICE_ACTION property can have one of three values:

- SUCCESS

When you change a member of the relationship with the CHANGE command and the CDD$NOTICE_ACTION property value is SUCCESS, CDO does not send notices to the owner of the relationship. Instead, CDO sends notices to the owner's parents, provided that the parents are members of a relationship with the CDD$NOTICE_ACTION attribute value of SIGNAL.

CDO continues to send these notices until the member you are changing has no more parents or until CDO encounters the CDD$NOTICE_ACTION property value BLOCK.

When you define a new version of a member and the CDD$NOTICE_ACTION attribute value is SUCCESS, CDO sends notices to both the owner of the relationship and the owner's parents, until CDO encounters the CDD$NOTICE_ACTION property value of BLOCK or until there are no more parents.

- SIGNAL

  When you change the member with the CHANGE command or when you define a new version of the member and the CDD$NOTICE_ACTION attribute value is SIGNAL, CDO sends a notice to both the owner and the parents of a relationship.

  When you define a new version of a member, CDO also sends notices to the parents until CDO encounters the CDD$NOTICE_ACTION property value of BLOCK or until there are no more parents.

- BLOCK

  When the CDD$NOTICE_ACTION property value is BLOCK, CDO does not forward a notice from the member to the owner. CDO blocks the notice regardless of whether you are changing the member with the CHANGE command or defining new versions of the member.

Examples

1. `CDO>  SHOW PROTOCOL CDD$DATA_AGGREGATE_CONTAINS`

   In this example, because no qualifier is specified, the SHOW PROTOCOL command displays default BRIEF information. This information includes the user-specified properties for the CDD$DATA_AGGREGATE_CONTAINS type and the names of direct children.

2.
```
CDO> SHOW PROTOCOL CDD$DATABASE_SCHEMA
Definition of protocol CDD$DATABASE_SCHEMA    (Type :MCS_RELATION_TYPE)
 |   MCS_rdbRelation                 CDD$$O_DATABASE_REL
 |   MCS_groupingRelation            NO_GROUPING
 |   MCS_noticeAction                SIGNAL
 |   CDD$OBJECT_KIND                 CDD$RELATIONSHIP
 |   MCS_protocolMajor               1
 |   MCS_protocolMinor               0
 |   MCS_tag                         2818865
 |   MCS_createdDate                 14-MAR-1994 09:13:32.42
 |   CDD$MODIFIED_TIME               14-MAR-1994 09:13:32.42
 |   MCS_instantiable                1
 |   MCS_pattern
 |   MCS_status                      0
 |   MCS_freezeTime                  14-MAR-1994 09:13:32.42
 |   MCS_controlled                  1
 |   MCS_allowConcurrent             1
 |   MCS_HAS_PROPERTY
 |   |   MCS_attachment              (Type : MCS_PROPERTY_TYPE)
 |   |   MCS_inherited               (Type : MCS_PROPERTY_TYPE)
 |   |   CDD$PROTOCOL_TAG            (Type : MCS_PROPERTY_TYPE)
 |   |   MCS_containsDatabases       (Type : MCS_PROPERTY_TYPE)
 |   |   MCS_relOwner                (Type : MCS_PROPERTY_TYPE)
 |   |   MCS_relMember               (Type : MCS_PROPERTY_TYPE)
 |   |   MCS_databaseElement         (Type : MCS_PROPERTY_TYPE)
 |   |   MCS_elementType             (Type : MCS_PROPERTY_TYPE)
 |   |   MCS_allElementTypes         (Type : MCS_PROPERTY_TYPE)
 |   MCS_HAS_RELATION
 |   |   CDD$DATABASE                (Type : MCS_ELEMENT_TYPE)
 |   |   CDD$DATABASE                (Type : MCS_ELEMENT_TYPE)
 |   MCS_RELATION_MEMBER
 |   |   CDD$DATA_AGGREGATE          (Type : MCS_ELEMENT_TYPE)
 |   |   CDD$RDB_DATABASE            (Type : MCS_ELEMENT_TYPE)
 |   |   DBM$SCHEMA                  (Type : MCS_ELEMENT_TYPE)
 |   |   CDD$RMS_DATABASE            (Type : MCS_ELEMENT_TYPE)
 |   |   CDD$DATA_AGGREGATE          (Type : MCS_ELEMENT_TYPE)
 |   |   CDD$RDB_DATABASE            (Type : MCS_ELEMENT_TYPE)
 |   |   DBM$SCHEMA                  (Type : MCS_ELEMENT_TYPE)
 |   |   CDD$RMS_DATABASE            (Type : MCS_ELEMENT_TYPE)
 |   MCS_OBJECT_VALIDATION
 |   |   CDD$RELATION_VAL            (Type : MCS_VALIDATION)
 |   |   CDD$ELEMENT_VAL             (Type : MCS_VALIDATION)
```

In this example, the SHOW PROTOCOL command displays the CDD$NOTICE_ACTION property of SIGNAL for the CDD$DATABASE_ SCHEMA relationship.

SHOW RECORD Command

Format

SHOW RECORD [qualifier] ... [record-name] ,...

[FROM DATABASE database-name]

Parameters

record-name
Specifies the record, relationship, or view whose properties you are displaying.

When you use the FROM DATABASE clause, specify only one record name for each SHOW RECORD command.

Specify an asterisk (*) wildcard character for the entire record-name parameter only. If you use a wildcard character as part of the record name, an error occurs.

database-name
Specifies the Oracle Rdb database that contains the record. CDO requires this parameter for records from an Oracle Rdb database. You can include wildcard characters in the database name.

Qualifiers

/ALL
Displays system-specified properties, such as time of creation, and user-specified properties, such as description, for the record element and for all children.

/AUDIT
Displays history list entries for the record element. Specifying the /AUDIT=ALL qualifier displays the history list entries for the record element and for all children. Do not use this qualifier if the record you specify is from an Oracle Rdb database definition.

/BRIEF (default)
Displays user-specified properties for the record element, and provides the names of direct children.

/FULL
Displays user-specified properties for the record element and for all children.

/SYSTEM
/NOSYSTEM (default)
Specifies whether CDO displays Oracle Rdb system relations.

/RDB_METADATA
/NORDB_METADATA (default)
Specifies whether CDO displays Oracle Rdb system relations. This qualifier is synonymous with the /SYSTEM qualifier.

Description

The SHOW RECORD command displays a complete or partial list of properties for the record elements you specify, provided you have read privilege.

If you do not specify a full path name (or the FROM DATABASE clause), CDO searches your current default directory for the record name. If you do not specify a record name, CDO displays the properties of all record elements in your default directory.

If you do not specify a version number for a record element, CDO displays the properties of the highest visible version.

If you do not have read privilege for a record and for each component part of that record, CDO will not display the record name or properties.

Examples

```
CDO>  SHOW RECORD ADDRESS_RECORD
cont>   FROM DATABASE SUBSCRIPTIONS
```

In this example, because no qualifier is specified, the SHOW RECORD command displays default BRIEF information. This information includes the user-specified properties for the ADDRESS_RECORD record element and the names of direct children.

SHOW REPOSITORIES Command

Format

SHOW REPOSITORIES [qualifier]

Qualifiers

/FULL
Displays the elements within other repositories being referenced by your repository.

Description

The SHOW REPOSITORIES command scans your repository for dependencies on elements in other repositories. The command displays the name of these other repositories and, if you specify the /FULL qualifier, the names of the elements.

Example

```
CDO>  SHOW REPOSITORIES
```

In this example, because no qualifier is specified, the SHOW REPOSITORIES command displays the names of the repositories that your repository references.

SHOW RESERVATIONS Command

Format

SHOW RESERVATIONS [qualifier]

Qualifiers

/ALL
Displays all the reserved elements, their types, and the contexts reserving them throughout the repository.

/BRIEF (default)
Displays reserved elements and their types in your current context.

Description

The SHOW RESERVATIONS command displays the reserved elements for your current context or for the entire repository.

An error occurs if a context has not been set.

Examples

```
CDO>  SET CONTEXT PERSONNEL
CDO>  RESERVE COLLECTION OFFICERS
CDO>  SHOW RESERVATIONS
   .
   .
   .
CDO>  REPLACE COLLECTION OFFICERS
CDO>  SHOW RESERVATIONS
   .
   .
   .
```

In this example, the first SHOW RESERVATIONS command displays the reserved elements, their types, and the reserving context. The second SHOW RESERVATIONS command displays an informational notice indicating that no elements are reserved.

SHOW RMS_DATABASE Command

Format

SHOW RMS_DATABASE [qualifier] [rms-database-name]

Parameters

rms-database-name
Specifies the logical RMS database element whose properties you are displaying.

If you do not specify an RMS database name, CDO displays all the RMS database definitions in the repository.

Qualifiers

/ALL
Displays system-specified properties, such as time of creation, and user-specified properties, such as description, for the RMS database element and all children.

/AUDIT
Displays history list entries for the RMS database element. Specifying the /AUDIT=ALL qualifier displays the history list for the database element and for all children.

/BRIEF (default)
Displays the file organization properties for the RMS database element.

/FULL
Displays the file organization properties, the record definition, and the description for the logical RMS database element.

Description

The SHOW RMS_DATABASE command displays a complete or partial list of properties for the RMS database element you specify.

If you do not specify a version number for an RMS database element, CDO displays the highest visible version.

When you use SHOW RMS_DATABASE or SHOW RMS_DATABASE/FULL to display an RMS database element with a NULL_VALUE property, CDO displays the null value as a decimal value.

Examples

```
CDO>  SHOW RMS_DATABASE EMPLOYEE_STORAGE
```

In this example, because no qualifier is specified, the SHOW RMS_DATABASE command displays default BRIEF information. This information includes the user-specified properties for the EMPLOYEE_STORAGE RMS database definition and the names of direct children.

SHOW UNUSED Command

Format

SHOW UNUSED [qualifier] ... element-name ,...

Parameters

element-name
Specifies the element whose relationships you are investigating.

Qualifiers

/TYPE=[(type-name)] ,...

/FULL
Displays all owners and members of the element.

Description

The SHOW UNUSED command determines whether an element has owners or members. The element must have a directory name.

If the element does not have owners or members, the name and type of the element is displayed. If the element has owners or members, CDO sends an informational notice. If you receive an informational notice, issue the SHOW USES and SHOW USED_BY commands to identify these owners or members.

Examples

1. `CDO> SHOW UNUSED EMPLOYEE_DB`

   In this example, because no qualifier is specified, the SHOW UNUSED command displays default BRIEF information. This information includes the names of immediate owners or members of the EMPLOYEE_DB database element.

2. `CDO> SHOW UNUSED /FULL FIELD_A(2)`

   In this example, the SHOW UNUSED command with the /FULL qualifier displays all owners or members of the FIELD_A(2) field element.

3. CDO> SHOW UNUSED /FULL /TYPE=(FIELD) EMPLOYEE_REC

   In this example, the SHOW UNUSED command with the /FULL and /TYPE qualifiers displays all owners or members of EMPLOYEE_REC that are fields.

SHOW USED_BY Command

Format

SHOW USED_BY [qualifier] ... element-name ,...

Parameters

element-name
Specifies the element whose children you are displaying.

Qualifiers

/TYPE=[(type-name)] ,...
Displays only children of the type you specify. This type must be the keyword RECORD or FIELD or a type name that is valid for this command. Valid types include instances of ELEMENT_TYPE or RELATIONSHIP_TYPE; they do not include instances of PROPERTY_TYPE.

/BRIEF (default)
Displays the name, type, and relationship type of immediate children.

/FULL
Displays the name, type, and relationship type of all children.

Description

The SHOW USED_BY command displays the children of the element you specify. The display includes the properties of children and the values associated with these properties.

The SHOW USED_BY command lists children if they have either directory or processing names. CDO looks first for the directory name and displays it if one exists. If a directory name does not exist, CDO then looks for and displays the processing name.

A definition can only exist in the repository without a directory or processing name if it has a relationship to an owner that has a directory name. CDO displays a name unspecified message in this case.

To display Oracle Rdb database definitions, specify the name of the database as the element-name parameter and use the /FULL qualifier. Because CDO displays the full path name of each repository element, you can use the SHOW USED_BY command to determine where each instance of a particular element occurs and how it relates to other elements.

Examples

1.
```
CDO>  SHOW USED_BY EMPLOYEE_DB
```

   In this example, because no qualifier is specified, the SHOW USED_BY command displays the default information (/BRIEF). This information includes the names of immediate children of the EMPLOYEE_DB database element.

2.
```
CDO>  SHOW USED_BY /FULL FIELD_A(2)
```

   In this example, the SHOW USED_BY command with the /FULL qualifier displays all children of the FIELD_A(2) field element.

3.
```
CDO>  SHOW USED_BY /FULL /TYPE=(FIELD) EMPLOYEE_REC
```

   In this example, the SHOW USED_BY command with the /FULL and /TYPE qualifiers displays all children of EMPLOYEE_REC that are fields.

SHOW USES Command

Format

SHOW USES [qualifier] ... element-name ,...

Parameters

element-name
Specifies the element whose owners you are displaying.

Qualifiers

/TYPE=[(type-name)] ,...
Displays only owners of the type you specify. This type must be the keyword RECORD or FIELD or a type name that is valid for this command. Valid types include instances of ELEMENT_TYPE or RELATIONSHIP_TYPE; they do not include instances of PROPERTY_TYPE.

/BRIEF (default)
Displays the name, type, and relationship type of immediate owners.

/FULL
Displays the name, type, and relationship type of all owners.

Description

The SHOW USES command displays the owners of the element you specify. The display includes the properties of owners and the values associated with these properties.

The SHOW USES command lists owners if they have either directory names or processing names. CDO looks first for the directory name and displays it if one exists. If a directory name does not exist, CDO then looks for and displays the processing name.

An element can only exist in the repository without a directory or processing name if it has a relationship to an owner with a directory name. CDO displays a name unspecified message in this case.

You can use the SHOW USES command to display the names of elements that receive new version messages if you create a new version of the element you specify.

Examples

1. `CDO> SHOW USES EMPLOYEE_DB`

   In this example, because no qualifier is specified, the SHOW USES command displays the default information (/BRIEF). This information includes the names of immediate owners of the EMPLOYEE_DB database element.

2. `CDO> SHOW USES /FULL FIELD_A(2)`

   In this example, the SHOW USES command with the /FULL qualifier displays all owners of the FIELD_A(2) field element.

3. `CDO> SHOW USES /FULL /TYPE=(RECORD) EMPLOYEE_NAME`

   In this example, the SHOW USES command with the /FULL and /TYPE qualifiers displays all owners of EMPLOYEE_NAME that are records.

SHOW VERSION Command

Format

SHOW VERSION

Description

The SHOW VERSION command displays the following information about one or more respositories that you have invoked and to which the CDO session is currently attached:

- Version number of Oracle CDD/Repository that is installed on your system
- Major and minor version numbers of each repository
- Fully qualified name of each repository
- Version numbers of Oracle CDD/Repository with which each repository is compatible

If you issue the SHOW VERSION command and you have not invoked a repository during the CDO session, CDO displays only the version of Oracle CDD/Repository that is currently installed.

Commands such as DEFINE FIELD or SHOW FIELD will invoke a repository; however, the SET DEFAULT or SHOW DEFAULT commands will not. If you issue a DIRECTORY command and the repository contains elements, the session will invoke a repository.

Examples

1.
```
CDO>  SHOW DEFAULT
CDO>  SHOW VERSION
Installed version of Oracle CDD/Repository is V7.0.1
```

   In this example, the SHOW DEFAULT command does not invoke a repository. Therefore, the SHOW VERSION command displays only the version of Oracle CDD/Repository that is currently installed.

2.
```
CDO>  SHOW FIELD
Definition of field A1
|   Datatype                text size is 1 characters
|   Row_major array         1:4
Definition of field A1_KEY
|   Datatype                text size is 1 characters
   .
   .
   .
CDO> SHOW VERSION
Installed version of Oracle CDD/Repository is V7.0.1
Attached to repository
CDD$R0:[SMITH.REPOS3]
Repository Version V6.1 / V7.0.1
Internal Major Version 26
Internal Minor Version 3
```

In this example, the SHOW FIELD command invokes a repository. Consequently, the SHOW VERSION command displays information about the repository to which it is attached.

SHOW WHAT_IF Command

Format

SHOW WHAT_IF [qualifier] ... element-name ,...

Parameters

element-name
Specifies the element you are considering changing.

Qualifiers

/TYPE=[(type-name)] ,...
Displays those owners of the type you specify that could possibly receive an invalid notice if you perform a change in location with the CHANGE command. This type must be the keyword RECORD or FIELD or a type name that is valid for this command. Valid types include instances of ELEMENT_TYPE or RELATIONSHIP_TYPE; they do not include instances of PROPERTY_TYPE.

/BRIEF (default)
Displays the name, type, and relationship type of immediate owners that receive a possibly invalid notice if you perform a change in location with the CHANGE command.

/FULL
Displays the name, type, and relationship type of all owners that receive a possibly invalid notice if you perform a change in location with the CHANGE command.

Description

The SHOW WHAT_IF command displays the owners that are affected if the element you specify is modified by the CHANGE command. For the owners to be displayed, the relationship between owner and member must have an associated CDD$NOTICE_ACTION property value of SIGNAL.

Owners with this property value generally represent an object outside the repository, such as a database. Each owner receives a possibly invalid warning if you issue the CHANGE command for the specified element.

To determine the CDD$NOTICE_ACTION property value of a relationship, use the SHOW PROTOCOL command.

SHOW WHAT_IF Command

The SHOW WHAT_IF command lists owners if they have either directory or processing names. CDO looks first for the directory name and displays it if one exists. If a directory name does not exist, CDO then looks for and displays the processing name.

An element can only exist in the repository without either a directory or processing name if it has a relationship to an element that has a directory name. CDO displays the name unspecified message in this case.

Examples

1. `CDO>  SHOW WHAT_IF EMPLOYEE_DB`

   In this example, because no qualifier is specified, the SHOW WHAT_IF command displays the default information (/BRIEF). This information includes the names of immediate owners that receive a possibly invalid notice if you issue the CHANGE command for EMPLOYEE_DB.

2. `CDO>  SHOW WHAT_IF /FULL FIELD_A(2)`

   In this example, the SHOW WHAT_IF command with the /FULL qualifier displays all owners that receive a possibly invalid notice if you issue the CHANGE command for FIELD_A(2).

3. `CDO>  SHOW WHAT_IF /FULL /TYPE=(CDD$DATABASE) EMPLOYEE_REC`

   In this example, the SHOW WHAT_IF command with the /FULL and /TYPE qualifiers displays all owners of the CDD$DATABASE type that receive a possibly invalid notice if you issue the CHANGE command for EMPLOYEE_REC.

SPAWN Command

Format

SPAWN [qualifier] ... [command-string]

Parameters

command-string
Specifies an OpenVMS DCL command you want to perform in the context of the subprocess the SPAWN command creates. After the subprocess executes this command string, DCL returns control to your CDO process. A command string cannot exceed 132 characters.

Qualifiers

/INPUT=file-spec
Specifies an OpenVMS file containing one or more DCL commands that DCL executes in the spawned subprocess. Once DCL finishes processing your input file, DCL terminates the subprocess and returns you to the CDO prompt.

/OUTPUT=file-spec
Requests that the output from the subprocess be written to the OpenVMS file you specify.

/WAIT (default)
/NOWAIT
Specifies whether the system waits until DCL completes a subprocess before allowing more commands to be issued in the parent process (the process in which you are running CDO).

The /WAIT qualifier does not return you to the parent process until the command string you specify completes execution, or you log out of the created subprocess. You can also use the ATTACH command to return to the parent process.

The /NOWAIT qualifier allows you to issue new commands while a subprocess is running. Use the /NOWAIT qualifier interactively. This directs output from the subprocess to a file so only one process at a time uses your terminal. Otherwise, the only way to distinguish one process from another is by the prompt. The CDO prompt indicates the parent process; the DCL prompt (normally a dollar sign) indicates the subprocess.

If you specify the /NOWAIT qualifier and your input device is a terminal, control characters such as Ctrl/T or Ctrl/Y affect all subprocesses sharing the input device. For example, Ctrl/Y interrupts all such subprocesses.

Description

The SPAWN command creates a subprocess of the current CDO process.

Examples

1.
```
CDO>  SPAWN SHOW TIME
17-FEB-1997 16:28:29
CDO>
```

   In this example, the SPAWN command creates a subprocess to execute the DCL command SHOW TIME. After the SHOW TIME command completes executing, DCL returns control to the parent CDO process.

2.
```
CDO>  SPAWN
$ LOGOUT
CDO>
```

   In this example, the SPAWN command creates a subprocess at the DCL prompt. To return to the CDO process, type LOGOUT at the DCL prompt.

3.
```
CDO> SPAWN RUN SQL$
SQL>
```

   In this example, the SPAWN command creates a subprocess to run interactive SQL.

START_TRANSACTION Command

Format

START_TRANSACTION

Description

The START_TRANSACTION command initiates a group of commands that Oracle CDD/Repository executes as a unit. A transaction ends with a COMMIT or ROLLBACK command. The COMMIT command causes all commands to execute, while the ROLLBACK command causes no commands to execute.

Restrictions

- When you delete a record, 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. Using CDO, there is no way to reuse those local fields. It is possible to use local fields through the Oracle CDD/Repository APIs. Therefore, the local fields cannot be automatically deleted at the same point in the transaction as the record.

  You must either delete the record and field in separate transactions (outside the START_TRANSACTION . . . COMMIT stream of commands) or, to accomplish this in one transaction, use ENTER to enter the local field, delete the record, delete the local field, and then delete the global field.

- Usually, if Oracle CDD/Repository issues any errors between the START_TRANSACTION and COMMIT commands, it forces you to roll back the transaction. In some cases, such as in the CHANGE or DELETE commands, Oracle CDD/Repository allows you to commit the transaction. The general rules are:

  – If you receive an Oracle CDD/Repository error of E or F severity, such as a CDD-E-NODNOTFND message, you must abort the transaction.

  – If you receive a CDO error of E or F severity, such as CDO-E-NOTFOUND, you can continue to operate in the current transaction.

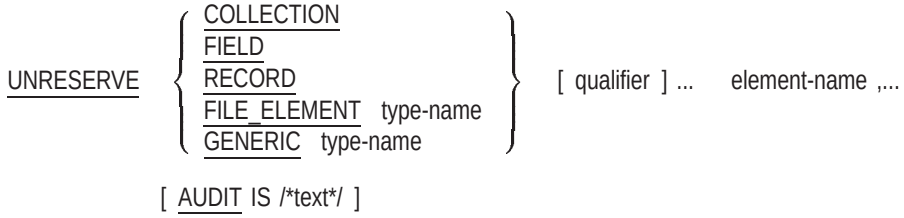
Examples

```
CDO> START_TRANSACTION.
CDO> DEFINE RECORD REC2.
cont> FLD1. END RECORD.
CDO> COMMIT
CDO> SHOW RECORD REC2
Definition of record REC2
|   Contains field          FLD1
   .
   .
   .
```

In this example, the COMMIT command ends a session started with the START_TRANSACTION command. When you use the 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. The repository is already attached to the database and has already loaded the type definitions. The objects REC2 and FLD1 are retrieved from memory instead of from disk.

UNRESERVE Command

Format



Parameters

type-name
Specifies the type of the file or generic element you are unreserving.

element-name
Specifies the element you are unreserving. You can substitute an asterisk (*) wildcard character for this parameter.

text
Adds information to the history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_ set of the session to DEC_KANJI.

Qualifiers

/CLOSURE=keyword
/NOCLOSURE (default)
Specifies whether CDO unreserves additional elements. An unreserve operation fails if any element is a child of an element outside the area defined by the /CLOSURE qualifier.

The /CLOSURE qualifier takes one of the following keywords:

CLOSURE Keyword	Behavior
TO_BOTH	Unreserves the element specified and all parents and children.
TO_BOTTOM	Unreserves the element specified and all children.
TO_TOP	Unreserves the element specified and all parents.

If you specify TO_BOTH or TO_TOP, CDO ignores parents above the top collection.

/LOG
/NOLOG (default)
Specifies whether CDO displays text identifying each element as the element is unreserved.

Description

The UNRESERVE command cancels the reservation previously placed on a version of an element. This operation deletes the ghost version of the element and discards any changes you made while you reserved the version.

Examples

```
CDO>  RESERVE COLLECTION /CLOSURE=TO_BOTTOM PRODUCT_INVENTORY
CDO>  DEFINE FIELD PRODUCT_NUMBER
cont>   DATATYPE BIT SIZE 5.
CDO>  DEFINE FIELD PRODUCT_DESCRIPTION
cont>   DATATYPE TEXT SIZE 50.
CDO>  UNRESERVE COLLECTION /CLOSURE=TO_BOTTOM PRODUCT_INVENTORY
```

In this example, the UNRESERVE command cancels the PRODUCT_ INVENTORY reservation and deletes the two fields that were defined.

UPDATE Command

Format

UPDATE COMPOSITE [qualifier] composite-name

[AUDIT IS /*text*/]

Parameters

composite-name
Specifies the collection, record, or field you are updating.

text
Adds information to the history list entry. Valid delimiters are /* */ or double quotation marks (" ").

You can use Japanese to document comments in the AUDIT clause for a field. To do this, use the SET CHARACTER_SET command, and set the character_ set of the session to DEC_KANJI.

Qualifiers

/CLOSURE=TO_BOTTOM
/NOCLOSURE (default)
Specifies whether CDO updates additional elements. Specifying the /CLOSURE=TO_BOTTOM qualifier updates all children of a reserved element, unless the element is a child of an element outside the area defined by the CLOSURE keyword.

Description

The UPDATE command allows you to attach more recent versions of elements to your own collections, records, or fields. In this way, you can access the work of others in your working group.

Before you issue the UPDATE command, you must reserve the elements you wish to update. The SHOW RESERVATIONS command indicates whether this condition exists.

When you issue the UPDATE command, the action that occurs depends on the keyword you specified in the DEFAULT_ATTACHMENT clause of the DEFINE CONTEXT or the CHANGE CONTEXT command for your current context. The following table lists the keywords and behaviors associated with them:

DEFAULT_ATTACHMENT Keyword	Behavior
LATEST	Detaches the version currently attached and attaches the latest version, whether checked in or ghost. This keyword is the default attachment method.
LATEST_CHECKIN	Detaches the version currently attached and attaches the version most recently checked in.
SPECIFIC_VERSION	Does not detach the version currently attached.

The UPDATE command cannot attach a ghost version created with another context. The UPDATE command also cannot attach a checked-in version unless you have write privilege for the partition where the version resides.

Examples

1.
```
CDO>  DEFINE CONTEXT BUILD_SYSTEM_CONTEXT
cont>   BASE_PARTITION IS FIRST_BASELEVEL  TOP IS COMPILER_C
cont>   DEFAULT_ATTACHMENT IS LATEST_CHECKIN.
   .
   .
   .
CDO>  UPDATE COLLECTION COMPILER_C
```

In this example, the UPDATE command refers to the DEFAULT_ ATTACHMENT keyword (LATEST_CHECKIN) for the version to attach. UPDATE then detaches the currently attached version of the COMPILER_C collection and attaches the version most recently replaced.

2.
```
CDO>  DEFINE CONTEXT WRITE_CONTEXT
cont>   BASE_PARTITION IS FIRST_DRAFT TOP IS REFERENCE_MANUAL
cont>   DEFAULT_ATTACHMENT IS SPECIFIC_VERSION.
   .
   .
   .
CDO>  CHANGE CONTEXT WRITER_CONTEXT
cont>   DESCRIPTION IS "CHANGING DEFAULT_ATTACHMENT"
cont>   "TO PICK UP AL'S CHAPTERS"
cont>   DEFAULT_ATTACHMENT IS LATEST.
CDO>  UPDATE COLLECTION REFERENCE_MANUAL
```

In this example, the UPDATE command refers to the DEFAULT_ ATTACHMENT keyword (LATEST) for the version to attach. UPDATE then detaches the currently attached version of the REFERENCE_ MANUAL collection and attaches the latest version, whether checked in or ghost.

VERIFY Command

Format

VERIFY [qualifier] ... anchor-name ,...

Parameters

anchor-name
Specifies the anchor of the repository you are verifying.

Qualifiers

/ALL
Performs all the verification options, except for the REBUILD_DIRECTORY and COMPRESS options.

The /ALL qualifier includes the /NOFIX qualifier by default. If you want to use the /FIX qualifier as the default to the VERIFY/ALL command, define the CDD$VERIFY_ALL_FIX logical name to be any value. Define the CDD$VERIFY_ALL_FIX logical name at the process level or higher.

If you specify the VERIFY/ALL command without specifying a /FIX or /NOFIX qualifier, and if you have not defined the CDD$VERIFY_ALL_FIX logical name, an informational error message, CDO-I-VF_ALL_NOFIX, will display and the VERIFY command will continue using the default /NOFIX qualifier.

/COMPRESS
/NOCOMPRESS (default)
Specifies whether CDO compresses the CDD$DATABASE.SNP file to its original size at the time you created the repository. If you specify the /COMPRESS qualifier, you cannot include any other qualifier in your command.

/DIRECTORY
/NODIRECTORY (default)
Specifies whether CDO checks all directory names against a stored definition.

When you also specify the /FIX qualifier, the /DIRECTORY qualifier removes directory names that do not refer to any stored definition.

/EXTERNAL_REFERENCES
/NOEXTERNAL_REFERENCES (default)
Specifies whether CDO checks all relationships where either the owner or member is outside the repository you specified.

When you also specify the /FIX qualifier, the /EXTERNAL_REFERENCES qualifier fixes the errors it detects.

/FIX
/NOFIX (default)
Specifies whether CDO corrects errors found by the other qualifiers you specify.

/LOCATION
/NOLOCATION (default)
Specifies whether CDO checks that the repository is in the correct directory and is correctly referenced by other repositories on the system.

When you also specify the /FIX qualifier, the /LOCATION qualifier fixes the errors it detects.

/LOG
/NOLOG (default)
Specifies whether CDO sends informational and error text to the default output location for your system. Specifying the /NOLOG qualifier displays only error text.

/ORPHANS
/NOORPHANS (default)
Specifies whether CDO searches for definitions with no directory names and no owners. The /ORPHANS qualifier also checks relationships to ensure that they have valid owners and members.

When you also specify the /FIX qualifier, it places homeless definitions in a directory called CDD$ORPHANS. CDO creates directory names for them while the VERIFY command executes. The /FIX qualifier deletes relationships without valid owners and members.

Caution

If you specify the /ORPHANS qualifier and the /LOG qualifier, CDO generates text for every element that is not an orphan. This could potentially be a very large number.

/REBUILD_DIRECTORY
/NOREBUILD_DIRECTORY (default)
Specifies whether CDO checks that the repository is in the correct directory and is correctly referenced by other repositories on the system, then deletes and re-creates all directory entries for the repository anchor you specify. If

there are no directories to delete and re-create, the /REBUILD_DIRECTORY qualifier builds a directory for the repository.

Use the /REBUILD_DIRECTORY qualifier to recover a corrupted repository system only when corruption is so severe that all other qualifiers fail. The /REBUILD_DIRECTORY qualifier requires SYSPRV or BYPASS privilege.

Note

Use system backup utilities to back up your repository before using the /FIX qualifier or the /REBUILD_DIRECTORY qualifier. See *Using Oracle CDD/Repository on OpenVMS Systems* for information on performing a backup operation.

Description

The VERIFY command determines whether a repository is structurally correct. This command requires read access. If you include the /FIX qualifier, you may also need write access. The /FIX qualifier requires SYSPRV or BYPASS privilege.

The /COMPRESS and /REBUILD_DIRECTORY qualifiers correct the errors they encounter; all other qualifiers require the /FIX qualifier to correct errors. The /ALL qualifier corrects any errors it finds if the logical name CDD$VERIFY_ALL_FIX has been defined.

The /COMPRESS qualifier requires the following:

- You must have SYSPRV privilege. Otherwise, CDO displays a no privilege error.
- You must issue the VERIFY command with the /COMPRESS qualifier as the first CDO command in a CDO session. Otherwise, CDO reports a conflict error with other users.
- You must be the only user of the repository when you issue the VERIFY /COMPRESS command. Otherwise, CDO reports an Oracle Rdb lock conflict error.

If you are working with remote repositories, issue the VERIFY/EXTERNAL_REFERENCES command by itself before you issue the VERIFY/EXTERNAL_REFERENCES command with the /FIX qualifier. If a remote device is not mounted, the VERIFY/EXTERNAL_REFERENCES command returns an error that the /FIX qualifier attempts to correct. Until a device is mounted, the command cannot complete.

Use the VERIFY/LOCATION command if you issued the OpenVMS DCL COPY command to copy a repository or change the name of your anchor directory. Use the /LOCATION qualifier with the /EXTERNAL_REFERENCES qualifier if you have other repositories that reference the repository you are verifying.

Examples

```
CDO>  VERIFY /LOCATION /FIX [SMITH.REP]
```

In this example, the VERIFY /LOCATION /FIX command checks that the [SMITH.REP] repository is in the correct directory and is correctly referenced by other repositories on the system. This command also corrects any errors detected.

Part II

CDO Parameters

This part provides additional information on CDO properties, expressions, and edit strings that can be used within commands.

2
Field and Record Properties

Field and record properties define the characteristics of the data you store in field and record elements. You can remove a field or record property by adding the NO keyword to the property name. For example, NOARRAY removes the ARRAY property.

Not all languages or language processors support all CDO properties. Those properties that are not supported are ignored.

ARRAY Field or Record Property

Format

```
[ ROW_MAJOR    ]
[ COLUMN_MAJOR ] ARRAY  { [ n1: ] n2 } ...
```

Parameters

n1
Specifies the lower bound of the subscript. Replace n1 with a signed integer or a value expression that translates to a signed integer. The default value is 1.

n2
Specifies the upper bound of the subscript. Replace n2 with a signed integer or a value expression that translates to a signed integer. This value is greater than or equal to n1.

Description

The ARRAY property defines a single- or multidimensional array in a field or record element.

In multidimensional arrays, ROW_MAJOR declares the rightmost subscript to be the fastest varying. COLUMN_MAJOR declares the leftmost subscript to be the fastest varying.

If you do not specify either ROW_MAJOR or COLUMN_MAJOR, the default is ROW_MAJOR.

Examples

1.
```
CDO>  DEFINE FIELD SUPPLIER
cont>   ARRAY 0:19 1:4
cont>   DATATYPE IS TEXT
cont>   SIZE IS 30 CHARACTERS.
```

In this example, the DEFINE RECORD command includes an ARRAY property that declares 20 instances of the SUPPLIER field element (from 0 to 19). Each instance is four 30-character strings.

2.
```
CDO>  DEFINE RECORD SUPPLIER_REC
cont>   ROW_MAJOR ARRAY 1:20.
cont> END RECORD.
```

In this example, the DEFINE RECORD command includes an ARRAY property that creates the SUPPLIER_REC record element as an array.

3.
```
CDO>  CHANGE RECORD SUPPLIER_REC.
cont>   NOARRAY.
cont> END RECORD.
```

In this example, the CHANGE RECORD command includes a NOARRAY property that removes the ARRAY property from the SUPPLIER_REC record element.

BASED ON Field Property

Format

BASED ON field-name

Parameters

field-name
Specifies the field name on whose properties you are basing a new field element.

Description

The BASED ON field property bases the properties of a new field element on one that already exists.

You must have privilege to read a field element to be able to base other elements upon it.

You can use BASED ON field properties to define several fields related to a base field and to each other.

You can use the BASED ON field property to give individual names to field elements that share the same properties. This allows you to uniquely refer to these field elements in record elements.

If you want the new field to have additional properties not found in the base field, you can specify the additional properties in the DEFINE FIELD or CHANGE FIELD command.

Examples

1.
```
CDO>  DEFINE FIELD SUPERVISOR_BADGE_NUMBER
cont>   BASED ON BADGE_NUMBER
cont>   VALID IF SUPERVISOR_BADGE_NUMBER > 500.
```

In this example, the DEFINE FIELD command bases SUPERVISOR_ BADGE_NUMBER on the BADGE_NUMBER field element. The VALID IF property is an additional property that is unique to SUPERVISOR_ BADGE_NUMBER.

2.
```
CDO>  DEFINE FIELD MANAGER_BADGE_NUMBER
cont>   BASED ON SUPERVISOR_BADGE_NUMBER
cont>   VALID IF MANAGER_BADGE_NUMBER > 1000.
```

In this example, the DEFINE FIELD command bases a second field element on the element created in the previous example. The VALID IF property explicitly defined for the new element overrides the property included in the previous element.

3.
```
CDO>  DEFINE FIELD SUPERVISOR_SSN
cont>   BASED ON SSN.
```

In this example, the DEFINE FIELD command creates a new element from a standard element (SSN). When you use the BASED ON property to give different names to field elements that share the same properties, you base the new elements on a field element that does not change frequently.

4.
```
CDO>  DEFINE FIELD MANAGER_SSN
cont>   BASED ON SSN
cont>   QUERY_HEADER IS "MANAGER SSN".
CDO>  CHANGE FIELD MANAGER_SSN
cont>   NOBASED ON.
```

In this example, the NOBASED ON keyword removes the BASED ON property, but does not remove the QUERY_HEADER property, from the MANAGER_SSN field element. Because all other MANAGER_ SSN properties were based on SSN, you must define new properties for MANAGER_SSN, unless the QUERY_HEADER property is adequate.

COLLATING_SEQUENCE Field Property

Format

COLLATING_SEQUENCE IS text-string

Parameters

text-string
Specifies a sequence name that was previously defined in RDO or SQL.

Description

The COLLATING_SEQUENCE field property refers to a collating sequence that you have defined in RDO or SQL. The DEFINE FIELD and CHANGE FIELD commands accept the COLLATING_SEQUENCE syntax. The CHANGE FIELD command accepts a NOCOLLATING_SEQUENCE keyword that deletes the collating sequence; the SHOW FIELD and EXTRACT FIELD commands process the attributes.

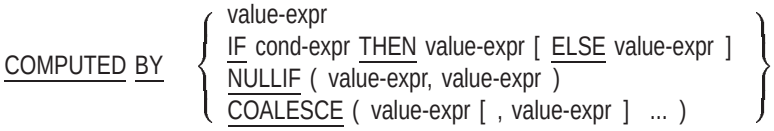
Examples

```
CDO>  COLLATING_SEQUENCE IS "French"
```

In this example, the COLLATING_SEQUENCE field property sets the collating sequence to French.

COMPUTED BY Field Property

Format



Parameters

value-expr
Specifies an expression a product can use to calculate a field's value. See Chapter 4 for more information on value expressions.

cond-expr
Specifies an expression that represents the relationship between two value expressions. See Chapter 4 for more information on conditional expressions.

Description

The COMPUTED BY field property evaluates an expression, allowing a product that uses CDO to determine the value of a field at runtime.

The expression must be a valid CDO expression. CDO checks the expression for correct syntax and field references.

The product must be able to interpret the CDO expression.

When you specify a conditional expression in the COMPUTED BY field property, you can define a field that is equivalent to a COBOL level 88 condition. The computed by expression must be in one of the following forms:

- if [name EQ literal1] THEN 1 ELSE 0
- if [(name GE literal1 AND name LE literal2) OR (name GE literal3 AND name LE literal4)]... THEN 1 ELSE 0

Use NULL IF to substitute NULL when two value expressions are equal.

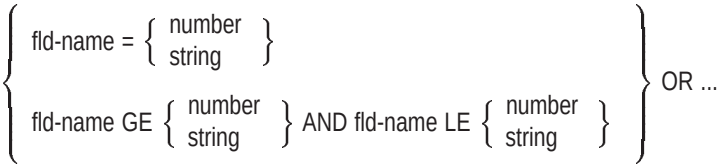
Use COALESCE to return the first non-NULL value from a series of value expressions.

COMPUTED BY Field Property

There is a limited subset of valid COMPUTED BY fields that are acceptable in COBOL syntax for inclusion through the COPY FROM DICTIONARY clause. They have the following format:

```
COMPUTED BY IF expression THEN 1 ELSE 0
```

Where expression is:



For example, the following COMPUTED BY fields are defined:

```
DEFINE FIELD Y_TRUE
     COMPUTED BY IF (Y = "TRUE") THEN 1 ELSE 0.
DEFINE FIELD Z_NULL
    COMPUTED BY IF (Z = 0) THEN 1 ELSE 0.
DEFINE FIELD W_ALPHABETIC COMPUTED BY
    IF ((W GE "A") AND (W LE "Z")) OR ((W GE "a") AND (W LE "z"))
    THEN 1 ELSE 0.
DEFINE FIELD X_3_DIGITS
    COMPUTED BY IF (X GE 100) AND (X LE 999) THEN 1 ELSE 0.
```

They are translated as the following COBOL level 88 conditions:

```
02 Y ...
   88 Y_TRUE VALUE IS "TRUE".
02 Z ...
   88 Z_NULL VALUE IS 0.
02 W ...
   88 W_ALPHABETIC VALUES ARE "A" THRU "Z", "a" THRU "z".
02 X ...
 88 X_3_DIGITS VALUES ARE 100 THRU 999.
```

Restriction

The COMPUTED BY field property can reference only one field. The fld-name parameter must be the same field name in all instances. When included in COBOL, the COMPUTED BY field will be translated as a level 88 condition associated with the field that was referenced.

Examples

1.
```
CDO>  DEFINE FIELD SUBTOTAL_PRICE
cont>   COMPUTED BY UNIT_PRICE * QUANTITY.
```

   In this example, the DEFINE FIELD command includes the COMPUTED BY property to calculate a value for the SUBTOTAL_PRICE field element. The value is computed by multiplying UNIT_PRICE by QUANTITY.

2.
```
CDO>  DEFINE FIELD TOTAL_PRICE
cont>   COMPUTED BY UNIT_PRICE (3) * 10.
```

   In this example, the DEFINE FIELD command includes a COMPUTED BY property to calculate a value for the TOTAL_PRICE field element. The value is calculated by multiplying the value in the third instance of the UNIT_PRICE field element by 10.

3.
```
CDO>  CHANGE FIELD TOTAL_PRICE
cont>   NOCOMPUTED BY.
```

   In this example, the CHANGE FIELD command includes the NOCOMPUTED BY keywords to remove the COMPUTED BY property from the TOTAL_PRICE field element.

4.
```
CDO>  DEFINE FIELD C
cont>   DATATYPE SIGNED WORD.
CDO>  DEFINE FIELD C_ONE
cont>   COMPUTED BY IF C EQ 1 THEN 1 ELSE 0.
CDO>  DEFINE FIELD C_FIVE_TEN
cont>   NAME FOR COBOL IS C_5_10
cont>   COMPUTED BY IF C GE 5 AND C LE 10 THEN 1 ELSE 0.
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.
CDO> DEFINE RECORD COB88.
cont> C.
cont> C_ONE.
cont> C_FIVE_TEN.
cont> C_OTHER.
cont> END RECORD.
```

   In this example, the DEFINE FIELD commands include COMPUTED BY properties that contain conditional and value expressions. These expressions are related to the value of the C field element, as follows:

   - The C_ONE field element takes the value of one (if C evaluates to one) or zero.

- The C_FIVE_TEN field element takes the value of one (if C evaluates to a value between five and ten) or zero.
- The C_OTHER field element takes the value of one (if C evaluates to a value between two and four or if C evaluates to a value between eleven and twenty) or zero.

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

This example shows COBOL syntax for the record containing level 88 definitions.

CURRENCY_SIGN Field Property

Format

CURRENCY_SIGN IS quoted-string

Parameters

quoted-string
Specifies the character that displays as a currency sign.

Description

The CURRENCY_SIGN field property indicates how a product using CDO displays the currency sign of a field element. Only DIGITAL DECforms supports the CURRENCY_SIGN field property.

You can specify only one CURRENCY_SIGN property for a field element.

Examples

1.
```
CDO>  DEFINE FIELD PRICE
cont>   DATATYPE IS LONGWORD
cont>   EDIT_STRING IS 999999
cont>   CURRENCY_SIGN IS "¥".
```

In this example, the DEFINE FIELD command creates the PRICE field element with the yen symbol as the currency sign.

2.
```
CDO>  CHANGE FIELD PRICE
cont>   NOCURRENCY_SIGN.
```

In this example, the NOCURRENCY_SIGN keyword removes the CURRENCY_SIGN property from the PRICE field element.

DATATYPE Field Property

Format

```
{ ALPHABETIC SIZE IS numeric-literal case CHARACTERS
  [ ALIGNED | UNALIGNED ] BIT SIZE IS numeric-literal
  date-time-dtypes
  decimal-string-dtypes
  fixed-point-dtypes
  floating-point-dtypes
  POINTER [ TO name [ IN name ] ... ]
  REAL
  SEGMENTED STRING [ SEGMENT_LENGTH IS numeric-literal BYTES ] [ SEGMENT_TYPE IS string-type ]
  TEXT CHARACTER_SET character-set-name SIZE IS numeric-literal case CHARACTERS
  UNSPECIFIED SIZE IS numeric-literal BYTE
  VARYING STRING CHARACTER_SET character-set-name SIZE IS numeric-literal case CHARACTERS }
```

Parameters

numeric-literal
Specifies the number of characters or bytes in the field being defined. See Chapter 4 for more information on numeric literals.

case

```
{ CASE_INSENSITIVE
  LOWERCASE
  UPPERCASE }
```

Specifies whether the characters in a string data type are uppercase, lowercase, or mixed case. The default is CASE_INSENSITIVE.

date-time-dtypes
Specifies a date-time data type for a field. See DATATYPE Field Property: Date-Time Data Types for more information.

decimal-string-dtypes
Specifies a decimal string data type for a field. See DATATYPE Field Property: Decimal String Data Types for more information.

fixed-point-dtypes
Specifies a fixed point data type for a field. See DATATYPE Field Property: Fixed Point Data Types for more information.

floating-point-dtypes
Specifies a floating point data type for a field. See DATATYPE Field Property: Floating Point Data Types for more information.

name
Specifies the structure used to provide a path to an element.

string-type
Specifies a numeric or character string literal that contains the name of the segment type. See Chapter 4 for more information on literals.

character-set-name
Table 2–1 shows the valid character-set-names.

Table 2–1 Valid Character Set Name Values for Character Set Attributes

CHARACTER_ SET Attribute	character-set-name	Description
MCS	DEC_MCS	A set of international alphanumeric characters
Kanji+ASCII	DEC_KANJI	Japanese characters as defined by the JIS X0208:1990 standard, Narrow Katakana characters as defined by the JIS X0201:1976 standard, and ASCII characters
Kanji	KANJI	Japanese characters as defined by the JIS X0208:1990 standard and user-defined characters
Katakana	KATAKANA	Narrow Katakana characters as defined JIS X0201:1976 standard

Oracle CDD/Repository does not have a default character set attribute; Oracle CDD/Repository stores the character set attribute that you specify. The default character set that is used for a field in Oracle CDD/Repository, and how an unspecified character set is handled by other products, depends on each product. See the documentation for the product in which you intend to use the character set attribute.

DATATYPE Field Property

Description

The DATATYPE field property defines the type and size of a field. Some valid CDO data types are not supported by all languages or language processors. Consult the documentation for your product.

The case you specify for characters with the ALPHABETIC, TEXT, and VARYING STRING data types must be valid for your product.

The following list provides information on valid data types:

- ALPHABETIC specifies that the field is a sequence of 8-bit ASCII bytes. You cannot use non-alphabetic characters with this data type.
- BIT specifies that the field is a bit string. The optional UNALIGNED keyword specifies that the string is not aligned. The optional ALIGNED keyword specifies that the string is aligned on a byte boundary. If no alignment keyword is specified, the default is ALIGNED.
- POINTER specifies that the field contains the address of another field or record element. PL/I, for example, uses POINTER fields to access based variables and buffers allocated by the system. Although PL/I does not associate POINTER fields with a specified record structure, other languages do; the optional TO name lets you connect a POINTER to a structure. The optional IN name lets you connect a POINTER to a structure in a structure.
- REAL specifies that the field is a 32-bit floating point number with precision to approximately seven decimal digits. VAX BASIC uses REAL as an optional alternative to the floating-point data type.
- SEGMENTED STRING specifies that the field will contain a pointer to a sequential file with a segmented internal structure.

  The maximum size of a string segment is 64K bytes. In a segmented string, you can store large amounts of text, long strings of binary input from a data collecting device, or graphic data.

  Oracle Rdb databases support this data type. Its SEGMENT_LENGTH component corresponds to RDB$LENGTH and its SEGMENT_TYPE component corresponds to RDB$VALUE. A numeric-literal must follow a SEGMENT_TYPE. The following table lists the valid values for SEGMENT_TYPE.

Table 2–2 Values for SEGMENT_TYPE

Value	Meaning
0	The contents of the segmented string are unspecified.
1	The segmented string contains text.
2	The segmented string contains Binary Language Representation statements.
Greater than 2	Reserved for use by Oracle.
Less than 0	Reserved for use by customers.

See the *Oracle Rdb7 SQL Reference Manual* for more information about segmented strings.

- TEXT specifies that the field is a sequence of 8-bit ASCII bytes.

  When you define the TEXT data type field property, CDO accepts two units of size for the field:

  – CHARACTERS

  – OCTETS

  To specify a character-based field size, use the CHARACTERS unit. To specify octet-based field size, use the OCTETS unit. For a field with a single octet character set attribute, such as DEC_MCS, KATAKANA and so on, one character corresponds to one octet. On the other hand, for fields with multiple-octet character set attributes, such as Kanji, the field size is changed depending on the unit. The default is CHARACTERS.

  When you specify a field size using CHARACTERS, CDO translates the correct length of octets and stores the field size in octets. When OCTETS is specified, CDO ensures that the valid field size in CHARACTERS is translated.

  Table 2–3 shows the number of octets used for one character in each character set.

Table 2–3 Number of Octets Used for One Character in Each Character Set

Character Set	M+Number of Octets Used for One Character	Number of Octets Translated in CDO
MCS	1 octet	1 octet

(continued on next page)

Table 2–3 (Cont.) Number of Octets Used for One Character in Each Character Set

Character Set	M+Number of Octets Used for One Character	Number of Octets Translated in CDO
Katakana	1 octet	1 octet
Kanji	2 octets	2 octets
Kanji+ASCII	1 octet for ASCII; 2 octets for Kanji	2 octets

If CDO cannot translate a valid field size in characters, an error occurs. For example, when you try to define a field with the Kanji character set attribute, and you specify a size of odd octets, CDO returns an error because it cannot identify a valid field size in characters.

- UNSPECIFIED declares that the field is a sequence of 8-bit unsigned bytes.
- VARYING STRING specifies that the field is a PL/I or PASCAL varying string.

See the information about the field values CHARACTERS and OCTETS, described under the TEXT data type.

Examples

1.
```
CDO>  DEFINE FIELD BEST_SELLER
cont>   DATATYPE IS TEXT 40.
```

In this example, the DEFINE FIELD command creates the BEST_SELLER field element with data type TEXT. The numeric literal limits BEST_ SELLER to 40 characters.

2.
```
CDO>  CHANGE FIELD BEST_SELLER
cont>   NODATATYPE.
```

In this example, the CHANGE FIELD command includes a NODATATYPE keyword that removes the DATATYPE field property from the BEST_ SELLER field element.

3.
```
CDO> DEFINE FIELD CUSTOMER
cont> DATATYPE IS TEXT CHARACTER_SET IS KANJI
cont> SIZE IS 20 CHARACTERS.
```

In this example, a new field is defined and the character-set-name of KANJI is specified.

4.
```
CDO> DEFINE FIELD FULL_NAME
cont> DATATYPE TEXT CHARACTER_SET IS KANJI
cont> SIZE IS 20 CHARACTERS.
```

In this example, KANJI is specified as the character set attribute, and CHARACTERS is specified as the unit of size for the FULL_NAME field. In this case, FULL_NAME will be defined with a size of 40 octets.

5.
```
CDO> DEFINE FIELD FULL_NAME
cont> DATATYPE TEXT CHARACTER_SET IS KANJI
cont> SIZE IS 20 OCTETS.
```

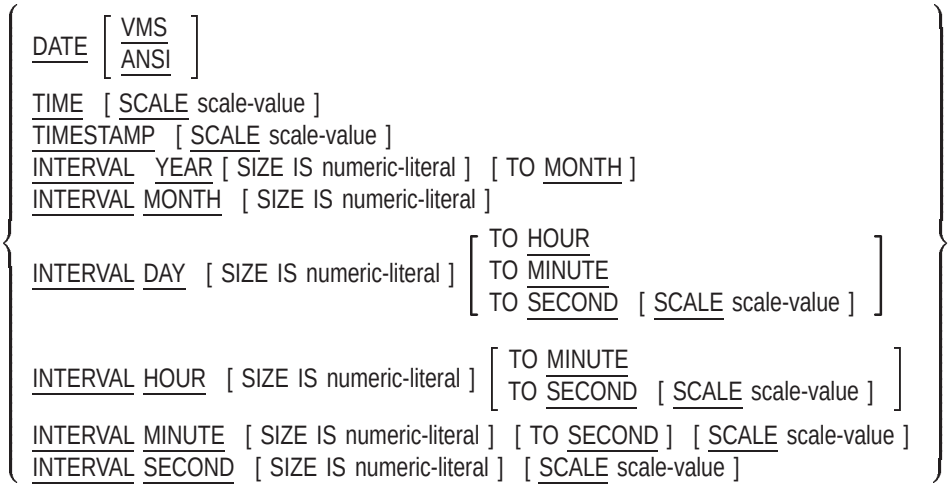
In this example, KANJI is the character set attribute, and OCTETS is the size of the FULL_NAME field. In this case, FULL_NAME will be defined with a size of 20 octets.

6.
```
CDO> DEFINE FIELD FULL_NAME
cont> DATATYPE TEXT CHARACTER_SET IS KANJI
cont> SIZE IS 20.
```

In this example, CHARACTERS is used as a default unit of size. KANJI is specified as the character-set-name of the field. In this case, FULL_NAME will be defined with a size of 40 octets.

DATATYPE Field Property: Date-Time Data Types

Format



Parameters

scale-value
The default value for SCALE is -2 with the exception of the TIME keyword which has a default value of 0.

numeric-literal
Specifies the number of digits allowed in the field. This number is greater than 0 and less than 32.

Description

The default value for SIZE is 2 and the valid range is between 2 and 9. See the *Oracle Rdb7 SQL Reference Manual* for more information on different date-time data types.

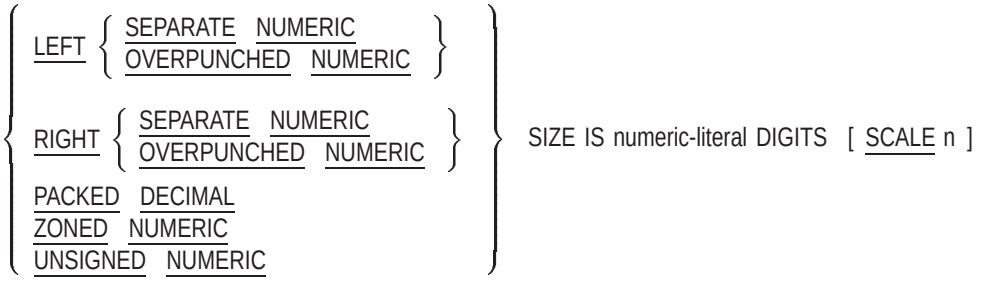
Examples

```
CDO>  DEFINE FIELD SAMPLE_FLD DATATYPE DATE ANSI.
CDO> DEFINE FIELD SAMPLE_FLD DATATYPE TIME SCALE -2.
CDO> DEFINE FIELD SAMPLE_FLD DATATYPE INTERVAL YEAR.
CDO> DEFINE FIELD SAMPLE_FLD DATATYPE INTERVAL YEAR TO MONTH.
CDO> DEFINE FIELD SAMPLE_FLD DATATYPE INTERVAL DAY
cont> SIZE 3 TO SECOND SCALE -2.
CDO> DEFINE FIELD SAMPLE_FLD DATATYPE INTERVAL HOUR
cont> SIZE 7 TO SECOND SCALE -2.
CDO> DEFINE FIELD SAMPLE_FLD
cont> DATATYPE INTERVAL MINUTE TO SECOND SCALE -2.
CDO> DEFINE FIELD SAMPLE_FLD
cont> DATATYPE INTERVAL SECOND SIZE 4 SCALE -2.
```

This example shows the definition of fields with date-time data types.

DATATYPE Field Property: Decimal String Data Types

Format



Parameters

numeric-literal
Specifies the number of digits allowed in the field. This number is greater than 0 and less than 32.

n
Specifies an implied exponent. The n value indicates the number of places the decimal point shifts when evaluating the field. This number is a signed integer in the range -128 to 127.

Description

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

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

The following list explains the characteristics of each decimal string data type:

- UNSIGNED NUMERIC specifies an unsigned numeric ASCII string. You must include the keyword UNSIGNED.
- LEFT SEPARATE NUMERIC specifies a signed numeric ASCII string. The leftmost byte contains the sign.
- LEFT OVERPUNCHED NUMERIC specifies a signed numeric ASCII string. The sign and the leftmost digit occupy the same byte.

- RIGHT SEPARATE NUMERIC specifies a signed numeric ASCII string. The rightmost byte contains the sign.
- RIGHT OVERPUNCHED NUMERIC specifies a signed numeric ASCII string. The sign and the rightmost digit occupy the same byte.
- ZONED NUMERIC specifies the VAX ZONED NUMERIC type. This signed numeric ASCII string is similar to the RIGHT OVERPUNCHED NUMERIC, but the sign codes differ.
- PACKED DECIMAL specifies a signed numeric ASCII string. Two digits occupy each byte, and the low half of the last byte is reserved for the sign.

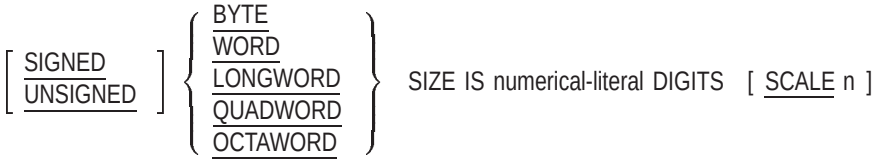
Examples

```
CDO>  DEFINE FIELD ACCOUNT_BALANCE
cont>   DATATYPE IS PACKED DECIMAL.
```

In this example, the DEFINE FIELD command creates the ACCOUNT_ BALANCE field element with a PACKED DECIMAL data type.

DATATYPE Field Property: Fixed-Point Data Types

Format



Parameters

numeric-literal
Specifies the number of digits allowed in the field. This number is greater than 0 and less than 32. The default is UNSIGNED.

n
Specifies an implied exponent. The n value indicates the number of places the decimal point shifts when evaluating the field. This number is a signed integer in the range -128 to 127.

Description

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

Fixed-point numbers of the data type SIGNED are stored in two's complement form. Values range from $-2^{(n-1)}$ to $2^{(n-1)} - 1$, where *n* is equal to the number of bits in the data type.

Fixed-point numbers of the data type UNSIGNED range from 0 to $2^n - 1$. Table 2–4 shows the fixed-point data types.

Table 2–4 Fixed-Point Data Types

Data Type	Length	Unsigned	Signed
BYTE	8 bits	0 to 255	−128 to 127

(continued on next page)

Table 2–4 (Cont.) Fixed-Point Data Types

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

Examples

```
CDO>  DEFINE FIELD NEW_MEMBERS
cont>   DATATYPE IS UNSIGNED LONGWORD 3.
```

In this example, the DEFINE FIELD command creates the NEW_MEMBERS field element with the UNSIGNED LONGWORD data type.

DATATYPE Field Property: Floating-Point Data Types

Format

{ D_FLOATING | F_FLOATING | G_FLOATING | H_FLOATING } [COMPLEX] [SCALE n]

Parameters

n
Specifies an implied exponent. The n value indicates the number of places the decimal point shifts when evaluating the field. This number is a signed integer in the range -128 to 127.

Description

Floating-point data types represent approximations to quantities in a scientific notation consisting of a signed exponent and a mantissa. The floating-point data types are shown in Table 2–5.

Table 2–5 Floating-Point Data Types

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

Complex numbers specify ordered pairs of floating-point data types, representing the real and imaginary components of a number.

Complex numbers are shown in Table 2–6.

Table 2–6 Complex Numbers

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

Examples

```
CDO>  DEFINE FIELD STANDARD_DEVIATION
cont>   DATATYPE IS H_FLOATING.
```

In this example, the DEFINE FIELD command creates the STANDARD_ DEVIATION field element with the H_FLOATING data type.

DECIMAL_POINT Field Property

Format

DECIMAL_POINT IS quoted-string

Parameters

quoted-string
Specifies the character displayed as a decimal point.

Description

The DECIMAL_POINT field property indicates how to display the decimal point of a field element. Only DIGITAL DECforms supports the DECIMAL_POINT field property.

You can specify only one DECIMAL_POINT property for a field element.

Examples

1.
```
CDO>  DEFINE FIELD PRICE
cont>   DATATYPE IS LONGWORD
cont>   EDIT_STRING IS 999999
cont>   CURRENCY_SIGN IS "£"
cont>   DECIMAL_POINT IS ",".
```

In this example, the DEFINE FIELD command creates the PRICE field element that displays a comma for the decimal point.

2.
```
CDO>  CHANGE FIELD PRICE
cont>   NODECIMAL_POINT.
```

In this example, the NODECIMAL_POINT keyword removes the DECIMAL_POINT property from the PRICE field element.

DEFAULT_VALUE FOR SQL Field Property

Format

[NO]DEFAULT_VALUE FOR SQL IS value-expr

Parameters

value-expr
Specifies an expression a product can use to calculate a field's value. See Chapter 4 for more information on value expressions.

Description

The DEFINE FIELD, CHANGE FIELD, and EDIT FIELD commands accept the DEFAULT_VALUE FOR SQL syntax.

The CHANGE FIELD command accepts the NODEFAULT_VALUE FOR SQL keyword that deletes the default value for SQL.

The SHOW FIELD and EXTRACT FIELD commands process the attributes.

Examples

1.
```
CDO>  DEFINE FIELD AMOUNT
cont> DATATYPE TEXT 5
cont> DEFAULT_VALUE FOR SQL IS "-----".
```

   This example shows the definition of the AMOUNT field with a default value for SQL of dashes.

DISPLAY_SCALE Field Property

Format

DISPLAY_SCALE IS n

Parameters

n
Specifies a signed integer indicating the number of places to shift the decimal point. A negative integer moves the decimal point to the left. A positive integer moves the decimal point to the right.

Description

The DISPLAY_SCALE field property indicates how to shift the decimal point when displaying the value of a field element. Only DIGITAL DECforms supports the DISPLAY_SCALE field property.

You can specify only one DISPLAY_SCALE property for a field element.

Examples

1.
```
CDO>  DEFINE FIELD AMOUNT
cont>   DATATYPE IS LONGWORD
cont>   EDIT_STRING IS 9999.99
cont>   INPUT_EDIT_STRING IS 9999.99
cont>   DISPLAY_SCALE -2.
```

In this example, the DEFINE FIELD command creates the AMOUNT field element with a decimal point shifted two places to the left.

2.
```
CDO>  CHANGE FIELD AMOUNT
cont>   NODISPLAY_SCALE.
```

In this example, the NODISPLAY_SCALE keyword removes the DISPLAY_SCALE property from the AMOUNT field element.

EDIT_STRING Field Property

Format

Parameters

edit-string
Specifies an edit string. See Chapter 5 for detailed information on edit strings.

Description

The EDIT_STRING field property indicates how to display the value of a field element.

You can specify a CDO generic edit string or a language-specific edit string for the following languages:

- COBOL
- DATATRIEVE
- PL/I
- RPG

When you specify a language-specific edit string for a field element that already contains a generic edit string, the language-specific edit string overrides the existing generic edit string.

You should create a language-specific edit string when:

- One or more characters in the generic edit string cannot be translated into valid edit string characters for a language that uses the generic edit string. Table 5–1 shows how CDO translates characters in a generic edit string for COBOL, DIGITAL DATATRIEVE, PL/I, and RPG.
- A language that uses the generic edit string does not support the data type of the field element that contains the generic edit string.

If your programs fail to compile due to edit string or data type errors, the language may not support the generic edit string. If this is the case, you should create language-specific edit strings to exclude this language from accessing the generic edit string.

Examples

1.
```
CDO>  DEFINE FIELD TRANS_DATE
cont>   DATATYPE IS DATE
cont>   EDIT_STRING IS NN"/"DD"/"YY.
```

   In this example, the DEFINE FIELD command creates the TRANS_DATE field element, which displays as a series of three, two-digit numbers in a month/day/year format.

2.
```
CDO>  CHANGE FIELD TRANS_DATE
cont>   NOEDIT_STRING.
CDO>  CHANGE FIELD COBOL_TRANS_DATE
cont>   NOCOBOL EDIT_STRING.
```

   In this example, the NOEDIT_STRING keywords remove the generic EDIT_STRING property from the TRANS_DATE field element. The NOCOBOL EDIT_STRING keywords remove the COBOL-specific EDIT_ STRING property.

FILLER Field Property

Format

FILLER

Description

The FILLER field property creates an unnamed field element. Unnamed field elements are similar to FILLER fields in COBOL. You can use them to format print records or to reserve space in a record for future additions.

When you specify the FILLER property, CDO creates the field element without a processing name.

Examples

```
CDO>  DEFINE FIELD BLANKS
cont>   DATATYPE IS TEXT 30 FILLER.
```

In this example, the DEFINE FIELD command includes a FILLER property that suppresses the BLANKS processing name.

GENERIC Field Property

Format

GENERIC type-name IS { quoted-string | n }

Parameters

type-name
Specifies the user-defined type of the property you are adding.

quoted-string
Specifies the value (a string enclosed in quotation marks) for this property.

n
Specifies the value (numerical) for this property.

Description

The GENERIC field property creates a generic field property. You specify generic field properties only if you have made changes to the field type (CDD$DATA_ELEMENT) supplied by Oracle CDD/Repository, and the changes require generic field properties.

You can specify the NOGENERIC keyword to remove a generic field property only if the changes you have made to CDD$DATA_ELEMENT indicate that this property is optional.

Examples

```
CDO>  CHANGE FIELD TEST_FIELD
cont>   NOGENERIC MY_ATTRIBUTE.
```

In this example, the NOGENERIC keyword in the CHANGE FIELD command removes the MY_ATTRIBUTE generic field property from the TEST_FIELD field element.

HELP_TEXT Field Property

Format

HELP_TEXT IS quoted-string

Parameters

quoted-string
Specifies the text you want the product to display when this field element is active and an operator presses the Help key.

Description

The HELP_TEXT field property tells a product to display user-supplied help text for the current element. Only DIGITAL DECforms supports the HELP_TEXT field property.

You can define only one HELP_TEXT property for a field element.

When you enter a quoted string that extends beyond one line, let the string wrap to the next line. Do not enclose each line of the quoted string with quotation marks. Do not press the Return key until you have closed the quoted string.

Examples

1.
```
CDO>  DEFINE FIELD EMPLOYEE_STATUS
cont>   DATATYPE IS TEXT SIZE IS 5
cont>   HELP_TEXT IS " Enter:  C for currently employed,
cont>   R for retired, D for dismissed,
cont>   or MLOA for medical leave of absence. ".
```

In this example, the DEFINE FIELD command includes a HELP_TEXT property that defines help text for the EMPLOYEE_STATUS field element.

Note

When you enter a quoted string that extends beyond one line, let the string wrap to the next line. Do not enclose each line of the quoted string with quotation marks. Do not press the Return key until you close the quoted string.

2.
```
CDO>  CHANGE FIELD EMPLOYEE_STATUS
cont>   NOHELP_TEXT.
```

   In this example, the NOHELP_TEXT keyword removes the HELP_TEXT property from the EMPLOYEE_STATUS field element.

INITIAL_VALUE Field Property

Format

INITIAL_VALUE IS value-expr

Parameters

value-expr
Specifies an expression a product can use to calculate a field's value. See Chapter 4 for more information on value expressions.

Description

The INITIAL_VALUE field property declares a field's value when the product first allocates the field. The expression you specify must be a valid expression for the product evaluating it.

The value of the expression must fit into the space allocated for the field.

You can specify a complex number for the INITIAL_VALUE property of a field if the field's data type is F_FLOATING COMPLEX, D_FLOATING COMPLEX, G_FLOATING COMPLEX, or H_FLOATING COMPLEX.

You can specify a fixed-point number for the INITIAL_VALUE property of any field whose data type is not DATE, TEXT, UNSPECIFIED, or VARYING STRING.

You can specify a floating-point number for the INITIAL_VALUE property of a field whose data type is not DATE, TEXT, UNSPECIFIED, or VARYING STRING.

You can use Japanese in an INITIAL_VALUE field property and to document comments (DESCRIPTION and AUDIT clauses) for a field. To do this set the character set of a session to DEC_KANJI; otherwise, the information may not display correctly. See the SET CHARACTER_SET command to set the character_set of a session.

Examples

1.
```
CDO>  DEFINE FIELD AMOUNT
cont>   DATATYPE IS UNSIGNED NUMERIC
cont>   SIZE IS 8 DIGITS
cont>   INITIAL_VALUE IS 0.
```

In this example, the DEFINE FIELD command assigns 0 as the initial value to the AMOUNT field element.

2.
```
CDO>  CHANGE FIELD AMOUNT
cont>   NOINITIAL_VALUE.
```

In this example, the NOINITIAL_VALUE keyword removes the INITIAL_VALUE property from the AMOUNT field element.

INPUT_VALUE Field Property

Format

INPUT_VALUE IS { OPTIONAL | REQUIRED }

Description

The INPUT_VALUE field property indicates if a field requires input data (REQUIRED) or can be empty (OPTIONAL).

Only DIGITAL DECforms supports the INPUT_VALUE field property.

Examples

1.
```
CDO>  DEFINE FIELD PRICE
cont>   DATATYPE IS LONGWORD
cont>   INPUT_VALUE IS REQUIRED.
```

In this example, the DEFINE FIELD command includes an INPUT_ VALUE property that requires at least one input character for the PRICE field element.

2.
```
CDO>  CHANGE FIELD PRICE
cont>   NOINPUT_VALUE.
```

In this example, the NOINPUT_VALUE keyword removes the INPUT_ VALUE property from the PRICE field element.

JUSTIFIED Field Property

Format

JUSTIFIED { CENTER | DECIMAL | LEFT | RIGHT }

Description

The JUSTIFIED field property indicates how to fill the storage space allocated to a field element.

- JUSTIFIED CENTER centers a TEXT field.
- JUSTIFIED DECIMAL right-justifies the whole part of a number to the left of a decimal point and left-justifies the fractional part of the number to the right of a decimal point. DIGITAL DECforms provides interactive decimal justification that appears as a user types numeric data.
- JUSTIFIED LEFT truncates or fills a TEXT field against the left margin. This is the default value.
- JUSTIFIED RIGHT truncates or fills a TEXT field against the right margin.

Only DIGITAL DECforms supports JUSTIFIED DECIMAL. All other products ignore it. JUSTIFIED DECIMAL requires a decimal string or floating-point data type.

Only COBOL and DIGITAL DECforms support the JUSTIFIED RIGHT option. Other language processors ignore it. COBOL displays as much of the right portion of a JUSTIFIED RIGHT string as possible. If this adjustment leaves storage space to the left of the string, COBOL fills this space with blanks.

Use the JUSTIFIED field property only with fields that have the following data types:

- TEXT
- UNSPECIFIED
- Decimal string
- Fixed-point
- Floating-point

Examples

1.
```
CDO>  DEFINE FIELD STREET
cont>   DATATYPE IS TEXT 15
cont>   NAME FOR COBOL IS C_STREET
cont>   JUSTIFIED RIGHT.
```

   In this example, the DEFINE FIELD command allocates space for 15 right-justified text characters in the STREET field element.

2.
```
thsonian Avenue
15 Maple Street
ΔΔΔ6 Oak Street
```

   In this continuation of the previous example, COBOL displays the strings "137 Smithsonian Avenue," "15 Maple Street", and "6 Oak Street" in the STREET field element. The deltas represent blanks that COBOL enters to fill the 15 characters allocated for the STREET field element. In this example, 15 characters appear on a line; however, the individual line lengths vary due to the different character sizes.

3.
```
CDO>  CHANGE FIELD STREET
cont>   NOJUSTIFIED.
```

   In this example, the NOJUSTIFIED keyword removes the JUSTIFIED RIGHT property from the STREET field element.

MISSING_VALUE Field Property

Format

MISSING_VALUE IS value-expr

Parameters

value-expr
Specifies an expression a product can use to calculate a field's value. See Chapter 4 for more information on value expressions.

Description

The MISSING_VALUE field property specifies a value to use if a field has not been assigned a meaningful value. See the DIGITAL DATATRIEVE or Oracle Rdb documentation for more information on how those products interpret the MISSING_VALUE field property.

The expression you specify must be a valid expression for the product evaluating it.

Products using CDO ignore field elements that contain the MISSING_VALUE field property when performing statistical operations.

Examples

1.
```
CDO>  DEFINE FIELD PRICE
cont>  DATATYPE IS SIGNED LONGWORD
cont>  MISSING_VALUE IS 0.
```

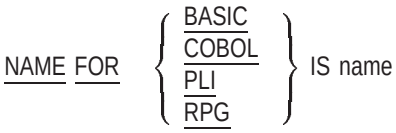
In this example, the DEFINE FIELD command includes a MISSING_ VALUE field property that can assign a value of 0 to a null or missing value.

2.
```
CDO>  CHANGE FIELD PRICE
cont>   NOMISSING_VALUE.
```

In this example, the NOMISSING_VALUE keyword removes the MISSING_VALUE property from the PRICE field element.

NAME Field or Record Property

Format



Parameters

name
Specifies a language-specific name for a field or record element.

Description

The NAME property declares a language-specific name for a field or record element.

This name must be a valid name for the specified language or language processor. CDO does not check the validity of the name that you specify.

Once you have assigned a language-specific name to an element, the specific language no longer recognizes the element's original name.

You can assign only one language-specific name per language to an element.

Caution

Be careful when you use the NAME field property because it allows you to assign completely different names to the same field or record element. Choose a language-specific name that is similar to the element's directory or processing name to avoid confusion.

Examples

```
CDO>  DEFINE FIELD ORDER_NUMBER
cont>   DATATYPE IS UNSIGNED NUMERIC
cont>   SIZE IS 10 DIGITS
cont>   NAME FOR COBOL IS ORDER-NUMBER.
```

In this example, the NAME property assigns a language-specific processing name to the ORDER_NUMBER field used by COBOL.

OCCURS Field Property

Format

OCCURS n TIMES

[INDEXED BY index-name [, index-name] ...]

Parameters

n
Specifies the number of occurrences of the array. This number is greater than zero.

index-name
Specifies the field element that functions as an index.

Description

The OCCURS field property declares one or more fixed-length, one-dimensional arrays.

The n value represents the upper bound of the array; the lower bound is always 1. Use the ARRAY field property to specify a lower bound other than 1.

Examples

1.
```
CDO>  DEFINE FIELD MULTIPLE
cont>   OCCURS 3 TIMES
cont>   DATATYPE IS SIGNED LONGWORD.
```

In this example, the OCCURS field property in the DEFINE FIELD command creates the MULTIPLE field element, which occurs 3 times.

2.
```
CDO>  CHANGE FIELD MULTIPLE
cont>   NOOCCURS.
```

In this example, the NOOCCURS keyword removes the OCCURS property from the MULTIPLE field element.

OCCURS ... DEPENDING Record Property

Format

```
OCCURS  n1 { TO n2 TIMES DEPENDING  ON name1 [ IN name2 ] ... }
           { TIMES                                            }

        [ INDEXED BY index-name ]
```

Parameters

n1, n2
Specifies the range for the number of occurrences. The value of n1 is greater than or equal to zero. The value of n2 is greater than or equal to n1.

name1
Specifies the field element whose value determines the actual number of occurrences.

name2
Specifies the element that contains the depending field. This element is often a record.

index-name
Specifies the field element that functions as an index.

Description

The OCCURS ... DEPENDING record property declares a variable-length, one-dimensional array.

The actual number of occurrences varies according to the value of the name1 field element.

An error occurs if name2 does not include the name1 field, or if name2 specifies an array record element.

You can repeat name2 as many times as necessary to identify the particular instance of name1. For example, name2 can repeat to identify a field element within nested record elements:

```
CDO>  DEFINE RECORD RETIREMENT_CHECKS OCCURS 1 TO 2 TIMES
cont>   DEPENDING ON ID_NUMBER IN EMPLOYEES
cont>   IN MEDICALLY_RETIRED_EMPLOYEES IN RETIRED_EMPLOYEES.
cont>   END RECORD.
```

Examples

1.
```
CDO>  DEFINE RECORD VACATION_PAY OCCURS 1 TO 2 TIMES
cont>   DEPENDING ON EXCESS_VACATION IN EMPLOYEE_BENEFITS.
cont>   EMP_SSN.
cont>   WEEKLY_SALARY.
cont>   END RECORD.
```

   In this example, the OCCURS ... DEPENDING record property specifies the number of times the VACATION_PAY record element occurs. The number of occurrences is based on the run-time value of the tag variable field element, EXCESS_VACATION, which is part of the EMPLOYEE_ BENEFITS record element.

2.
```
CDO>  CHANGE RECORD VACATION_PAY
cont>   NOOCCURS.
cont>   END RECORD.
```

   In this example, the keyword NOOCCURS in the CHANGE RECORD command removes the OCCURS ... DEPENDING record property from the VACATION_PAY record element.

3.
```
CDO> DEFINE FIELD MESSAGE_TABLE_IDX DATATYPE IS LONGWORD.
CDO> DEFINE FIELD MESSAGE_TEXT DATATYPE IS TEXT SIZE IS
cont> 256 CHARACTERS.
CDO> DEFINE RECORD MESSAGE_TABLE.
cont> MESSAGE_STRUCT STRUCTURE OCCURS 10 TIMES INDEXED BY
cont> MESSAGE_TABLE_IDX.
cont> MESSAGE_TEXT.
cont> END MESSAGE STRUCTURE.
cont> END MESSAGE_TABLE RECORD.
CDO> DEFINE RECORD MESSAGE_TABLE2.
cont> MESSAGE_STRUCT STRUCTURE OCCURS 1 TO 10 TIMES
cont> DEPENDING ON MESSAGE_TEXT
cont> INDEXED BY MESSAGE_TABLE_IDX.
cont> MESSAGE_TEXT.
cont> END MESSAGE STRUCTURE.
cont> END MESSAGE_TABLE2 RECORD.
CDO> SHOW RECORD MESSAGE_TABLE/FULL

Definition of record MESSAGE_TABLE
 |   Contains record          MESSAGE_STRUCT
 |   |   Occurs               10 indexed by MESSAGE_TABLE_IDX
 |   |   Contains field       MESSAGE_TEXT
 |   |   |   Datatype         text size is 256 characters
CDO> SHOW RECORD MESSAGE_TABLE2/FULL
```

```
Definition of record MESSAGE_TABLE2
 |   Contains record          MESSAGE_STRUCT
 |   |   Occurs               1 to 10  depending on MESSAGE_TEXT
indexed by MESSAGE_TABLE_IDX
 |   |   Contains field       MESSAGE_TEXT
 |   |   |   Datatype         text size is 256 characters
```

In this example, the MESSAGE_TABLE record contains an INDEXED BY clause. The name of the index field must already be defined.

QUERY_HEADER Field Property

Format

QUERY_HEADER IS quoted-string ,...

Parameters

quoted-string
Specifies the label you are using as a column heading.

Description

The QUERY_HEADER field property creates a column heading for use in printouts and reports.

The quoted string must be a valid column heading for the product that uses it. CDO accepts a text string of any length as a query header.

Examples

1.
```
CDO>  DEFINE FIELD TOTAL_PRICE
cont>   DATATYPE IS UNSIGNED LONGWORD
cont>   COMPUTED BY UNIT_PRICE * QUANTITY
cont>   QUERY_HEADER IS "TOTAL PRICE".
```

In this example, the QUERY_HEADER field property in the DEFINE FIELD command creates the TOTAL PRICE column heading for the TOTAL_PRICE field element.

2.
```
CDO>  CHANGE FIELD TOTAL_PRICE
cont>   QUERY_HEADER IS "TOTAL".
```

In this example, the CHANGE FIELD command changes the column heading in the TOTAL_PRICE field to TOTAL.

3.
```
CDO>  CHANGE FIELD TOTAL_PRICE
cont>   NOQUERY_HEADER.
```

In this example, the NOQUERY_HEADER keyword removes the QUERY_HEADER property from the TOTAL_PRICE field element.

QUERY_NAME Field Property

Format

QUERY_NAME IS { quoted-string / query-name }

Parameters

quoted-string
Specifies a string that is enclosed by quotation marks. DIGITAL DATATRIEVE only uses the string itself (not the quotation marks that enclose it) as the query name.

query-name
Specifies a string that is not enclosed by quotation marks.

Description

The QUERY_NAME field property provides an alternate name for a field element. Only DIGITAL DATATRIEVE supports this property.

CDO accepts query names of up to 256 characters in length. Except for the quotation mark ("), comma (,), apostrophe ('), and embedded blanks, any characters in the Digital Multinational Character Set are valid in query names.

Make sure the query name you specify is valid for the product that uses it.

CDO does not check whether the string you specified for the query name is valid.

When you assign a query name to a field element, products that support the QUERY_NAME field property can refer to the field element either by its query name or its processing name.

Examples

```
CDO>  DEFINE FIELD TOTAL_PRICE
cont>   DATATYPE IS UNSIGNED LONGWORD
cont>   COMPUTED BY UNIT_PRICE * QUANTITY
cont>   QUERY_NAME IS "TP".
```

In this example, the QUERY_NAME field property in the DEFINE FIELD command specifies the TP alternate name for the TOTAL_PRICE field element.

VALID IF Field Property

Format

VALID IF cond-expr

Parameters

cond-expr
Specifies an expression that forms the validation condition. See Chapter 4 for more information on value expressions.

Description

The VALID IF field property checks values assigned to a field to ensure that they are in the acceptable range for the field.

The expression you specify must be a valid expression for the product evaluating it.

Examples

1.
```
CDO>  DEFINE FIELD AMOUNT_OWED
cont>   DATATYPE IS UNSIGNED WORD
cont>   VALID IF AMOUNT_OWED > 0.
```

   In this example, the VALID IF field property in the DEFINE FIELD command specifies a range of valid values for the AMOUNT_OWED field element.

2.
```
CDO>  CHANGE FIELD AMOUNT_OWED
cont>   NOVALID IF.
```

   In this example, the NOVALID IF keywords remove the VALID IF property from the AMOUNT_OWED field element.

3

File Definition, Area, and Key Properties

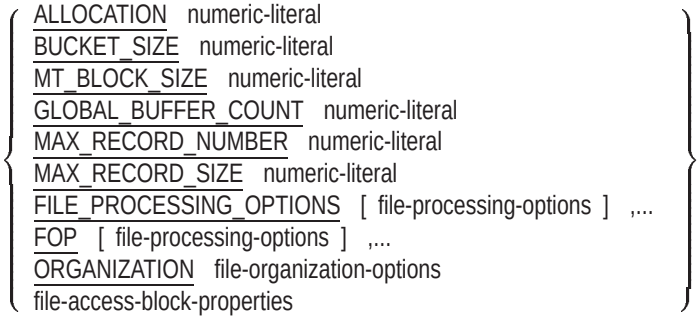
The file definition, area, and key properties define the physical characteristics of a Record Management Services (RMS) database file.

In most cases, the properties correspond to the RMS file and allocation control field blocks (FABs or XABs). CDO does not provide properties corresponding to the RMS record access block attributes (RABs).

See the OpenVMS documentation on RMS for detailed descriptions and the values accepted for each property listed in this chapter.

File Definition Properties

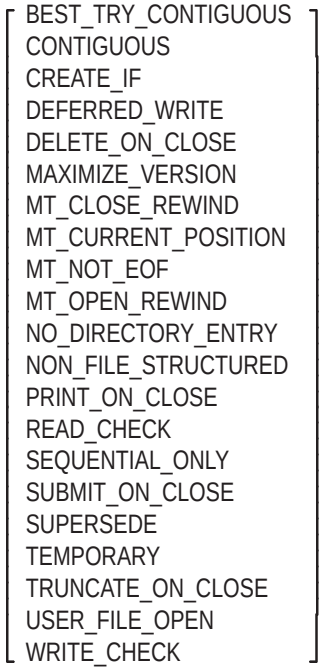
Format



Parameters

numeric-literal
Specifies a positive integer.

file-processing-options

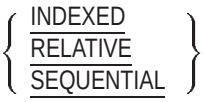


Define optional file operations for a program. The file processing options fall into the following seven functional categories:

- Allocation and extension options
- File disposition options
- File name parsing modifiers
- Magnetic tape processing options
- Nonstandard processing options
- Performance options
- Reliability options

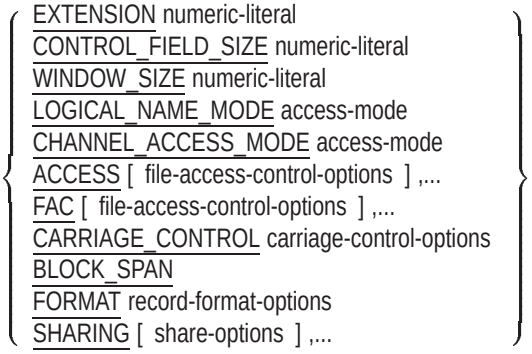
File Definition Properties

file-organization-options



Define the organization of the file. The default value for the file organization option is SEQUENTIAL.

file-access-block-attributes

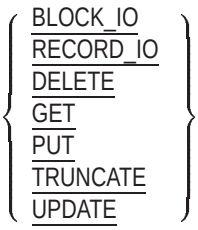


Define file access characteristics and certain run-time options.

access-mode

Defines an access mode for a channel. NONE is the default value, which is interpreted by RMS as the executive mode.

file-access-control-options



Define the type of record access operations that a program can perform.

carriage-control-options

{ CARRIAGE_RETURN | FORTRAN | PRINT }

Define record control information for a record. CARRIAGE_RETURN is the default value for most OpenVMS programs.

record-format-options

{ FIXED | STREAM | STREAM_CR | STREAM_LF | UNDEFINED | VARIABLE | VFC }

Define the format for all records in a file.

share-options

{ DELETE | GET | MULTISTREAM | PROHIBIT | PUT | UPDATE | USER_INTERLOCK }

Define the type of record operations that a program can perform when sharing access to a file with other programs.

Description

File definition properties define file characteristics and certain run-time options for a logical RMS database element.

For a description and valid values for a particular keyword, see the OpenVMS documentation on RMS.

For a mapping of CDO keywords to symbolics, see Appendix A.

File Definition Properties

Examples

```
CDO>  DEFINE RMS_DATABASE EMPLOYEE_INFO DESCRIPTION IS "INFORMATION ON"
cont>   "CURRENT EMPLOYEE".
cont>   RECORD EMPLOYEE_REC.
cont>   FILE_DEFINITION
cont>     ORGANIZATION INDEXED
cont>     CHANNEL_ACCESS_MODE SUPER
cont>     CARRIAGE_CONTROL CARRIAGE_RETURN
cont>     ACCESS  RECORD_IO
cont>     FILE_PROCESSING_OPTIONS BEST_TRY_CONTIGUOUS
cont>     FORMAT VARIABLE
cont>     SHARING GET, USER_INTERLOCK.
cont>   AREAS.
cont>     AREA 0
cont>       ALLOCATE 1000
cont>       BUCKET_SIZE 10
cont>       EXTENSION 100
cont>       CONTIGUOUS.
cont>     AREA 1
cont>       ALLOCATE 1000
cont>       BUCKET_SIZE 1
cont>       EXTENSION 100
cont>       BEST_TRY_CONTIGUOUS.
cont>     AREA 2
cont>       ALLOCATE 1000
cont>       BUCKET_SIZE 1
cont>       EXTENSION 100
cont>       BEST_TRY_CONTIGUOUS.
cont>   END AREAS.

cont>   KEYS.
cont>     KEY 0
cont>       DATA_AREA 0
cont>       INDEX_AREA 0
cont>       SEGMENT EMP_ID IN EMPLOYEE_REC.
cont>     KEY 1
cont>       DUPLICATES
cont>       DATA_AREA 1
cont>       INDEX_AREA 2
cont>       SEGMENT LAST_NAME IN EMPLOYEE_REC.
cont>   END KEYS.
cont> END EMPLOYEE_INFO RMS_DATABASE.
```

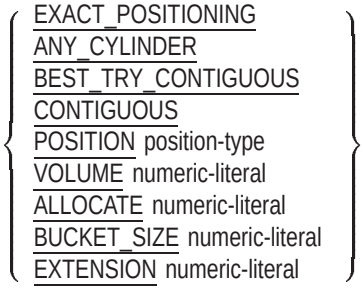
This example is the complete logical RMS database definition for EMPLOYEE_INFO. Seven file definition properties appear in the DEFINE RMS_DATABASE command:

- The option for file organization is INDEXED.
- The channel access mode is SUPER.

- The carriage control option is CARRIAGE_RETURN, although it is not necessary to code this because CARRIAGE_RETURN is the default.
- The type of record access operation this program can perform is RECORD_IO.
- An optional file processing operation is BEST_TRY_CONTIGUOUS.
- Record format for this definition is VARIABLE.
- The types of record operations that this program can perform when sharing access to a file with other programs are GET and USER_INTERLOCK.

Area Properties

Format



Parameters

position-type
Defines the alignment of an allocated area. The default alignment value is NONE. For more information, see Table A–5

numeric-literal
Specifies a positive integer.

Description

Area properties provide additional control over file or area space allocation on disk devices to optimize performance.

You usually include areas using indexed files. A file can contain up to 255 areas. Define areas in numerical order, beginning with 0.

For a description and valid values for a particular keyword, see the OpenVMS documentation on RMS.

For a mapping of CDO keywords to symbolics, see Appendix A.

Examples

1.
```
CDO>  DEFINE RMS_DATABASE EMPLOYEE_INFO DESCRIPTION IS "INFORMATION ON"
cont> "CURRENT EMPLOYEE".
cont>   RECORD EMPLOYEE_REC.
   .
   .
   .
cont>   AREAS.
cont>     AREA 0
cont>       ALLOCATE 1000
cont>       BUCKET_SIZE 10
cont>       EXTENSION 100
cont>       CONTIGUOUS.
cont>     AREA 1
cont>       ALLOCATE 1000
cont>       BUCKET_SIZE 1
cont>       EXTENSION 100
cont>       BEST_TRY_CONTIGUOUS.
cont>     AREA 2
cont>       ALLOCATE 1000
cont>       BUCKET_SIZE 1
cont>       EXTENSION 100
cont>       BEST_TRY_CONTIGUOUS.
cont>   END AREAS.
   .
   .
   .
cont> END EMPLOYEE_INFO RMS_DATABASE.
CDO>
```

This example shows the syntax for defining the ALLOCATE, BUCKET_SIZE, EXTENSION, CONTIGUOUS, and BEST_TRY_CONTIGUOUS properties for three areas in the EMPLOYEE_INFO RMS database element.

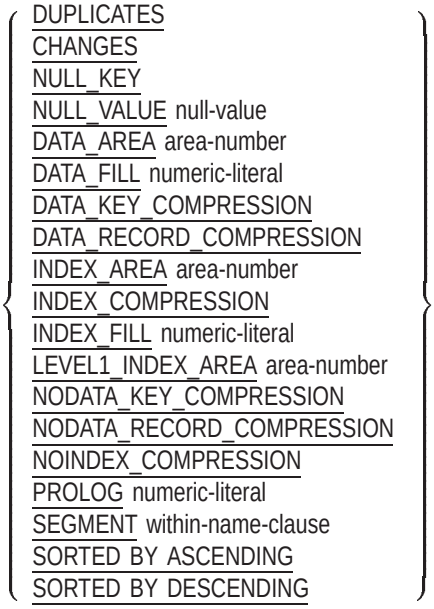
Area Properties

2.
```
CDO>  DEFINE RMS_DATABASE MORE_EMPLOYEE_INFO
cont>   "DESCRIPTION IS " DATA ON CURRENT EMPLOYEES ".
cont>   RECORD EMPLOYEE_REC.
cont>     FILE_DEFINITION
cont>     AREA 1
cont>     POSITION NONE.
   .
   .
   .
cont> END MORE_EMPLOYEE_INFO RMS_DATABASE.
CDO>
```

This example shows the syntax that defines the NONE position type option for the area property POSITION in the MORE_EMPLOYEE_INFO RMS database element.

Key Properties

Format



Parameters

null-value
Specifies a null key value for an alternate index. This value is either an integer or a single character within quotation marks.

area-number
Specifies a positive integer that corresponds to the number of a previously defined area.

numeric-literal
Specifies a positive integer. See the OpenVMS documentation on RMS for the valid values for a particular keyword. For a mapping of CDO keywords to the corresponding RMS symbolic field offset, see Appendix A.

Key Properties

within-name-clause

field-name { IN record-name } ...

Specifies the field or record names that are part of a segmented key. A segmented key allows you to define a key that accesses noncontiguous fields in a record. The record name is the same name that defines the data structure of the RMS database in the DEFINE RMS_DATABASE command.

Description

Key properties define the characteristics of a key in an indexed file. You define keys in numerical order, starting with 0. By default, Key 0 always has a value of DATA_AREA 0.

A segmented key allows you to define a key that accesses non-contiguous fields in a record. You can specify from 2 to 8 segments for each segmented key in an indexed file. For example:

- A field in a record
- A field in a record that is in a record
- A field in a structure in a record

You must specify a segment for each key you define. A segmented key can contain up to 8 segments.

The field element or elements specified by the keyword SEGMENT determine the data type of the key. A key with only 1 segment can point to a field that has any of the following RMS data types:

DSC$K_DTYPE_B
DSC$K_DTYPE_BU
DSC$K_DTYPE_W
DSC$K_DTYPE_WU
DSC$K_DTYPE_L
DSC$K_DTYPE_LU
DSC$K_DTYPE_Q
DSC$K_DTYPE_QU
DSC$K_DTYPE_P
DSC$K_DTYPE_T

See the OpenVMS documentation on RMS for more information on valid RMS data types.

Each segment of a key can contain up to 255 characters. The sum of all characters for all key segments must not exceed 255 characters.

When defining a segment, you cannot use a field that is in the variant portion of a record.

If you specify NULL_VALUE without specifying NULL_KEY, CDO automatically specifies NULL_KEY. If you specify NULL_KEY without specifying NULL_VALUE, the default value for NULL_VALUE is 0. When you display a RMS database element with a NULL_VALUE property, the null value appears as a decimal value.

For a description and valid values for a particular keyword, see the OpenVMS documentation on RMS.

For a mapping of CDO keywords to symbolics, see Appendix A.

Examples

```
CDO>  DEFINE RMS_DATABASE EMPLOYEE_INFO DESCRIPTION IS "INFORMATION ON"
cont> "CURRENT EMPLOYEE".
cont>   RECORD EMPLOYEE_REC.
   .
   .
   .
cont>   KEYS.
cont>     KEY 0
cont>       DATA_AREA 0
cont>       INDEX_AREA 0
cont>       SEGMENT EMP_ID IN EMPLOYEE_REC.
cont>     KEY 1
cont>       DUPLICATES
cont>       DATA_AREA 1
cont>       INDEX_AREA 2
cont>       SEGMENT LAST_NAME IN EMPLOYEE_REC.
cont>   END KEYS.
cont> END EMPLOYEE_INFO RMS_DATABASE.
CDO>
```

This example shows the syntax for defining the DATA_AREA, INDEX_AREA, SEGMENT, and DUPLICATES key properties in the EMPLOYEE_INFO RMS database element.

4
Expressions

CDO provides the following expressions:

- Value expressions—to calculate a value
- Conditional expressions—to represent a relationship between values
- Record selection expressions (RSE)—to state a condition for processing

CDO stores expressions in a generic format, not as text, so that many products and applications can share the same expression. The product using the CDO expression calculates the value at run time.

Precedence Ordering

The following list shows the order in which Oracle CDD/Repository interprets symbols used in an expression:

1. (symbols)

   Any field contained in parentheses.

2. * /

   Multiplication and division symbols.

3. +

   Addition and subtraction symbols.

4. < > <= >= = <>

   Relational operators. See Relational Operators for more information.

5. NOT

   Logical operator.

6. AND

   Logical operator.

7. OR

   Logical operator.

Table 4–1 shows equivalent symbols for the relational operators shown in the Precedence Ordering within Expressions table.

Table 4–1 Relational Operators Equivalent Symbols

Relational Operator	Equivalent Symbol	Meaning
<	LT	Less than
>	GT	Greater than
<=	LE	Less than or equal to
>=	GE	Greater than or equal to
=	EQ	Equal to
<>	NE	Not equal to

In general, CDO evaluates expressions from left to right. When an expression contains parentheses or operators, CDO evaluates these operations first. The

following list shows the order of precedence for these symbols, from highest to lowest:

- A symbol or symbols within parentheses
- Multiplication or division symbols
- Addition or subtraction symbols
- Relational operators: LT, GT, LE, GE, EQ, NE
- NOT
- AND
- OR

In the following example, the order of precedence determines that the first expression evaluates to 3, while the second expression evaluates to 8.

```
(6 + 12)/6 = 3
6 +  12/6 = 8
```

In the following expression, CDO evaluates X as a value between 2 and 4 or 11 and 20.

```
IF (X GE 2 AND X LE 4) OR (X GE 11 AND X LE 20) THEN 1 ELSE 0
```

Value Expressions

Format

{ arithmetic-expr
builtin-expr
case-expr
char-string-literal
concatenated-expr
external-literal
field-or-record-expr
first-from-expr
numeric-literal
statistical-expr }

Parameters

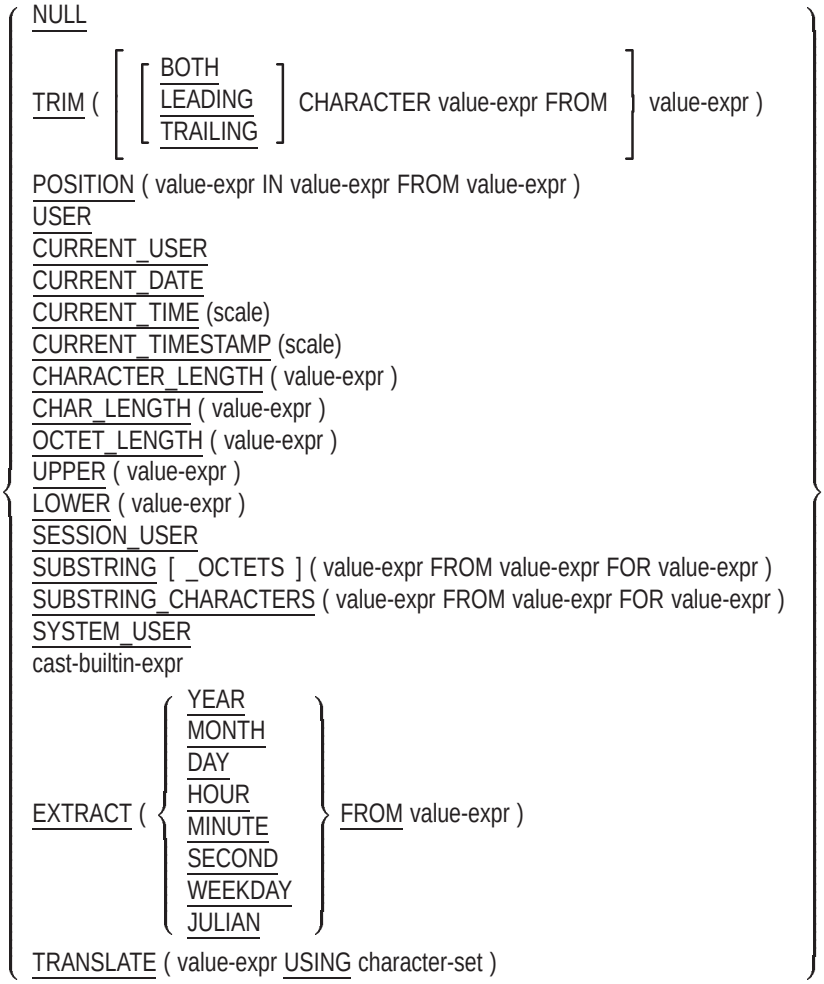
arithmetic-expr

value-expr { + - * / } value-expr

An arithmetic expression combines value expressions and arithmetic operators. When you use an arithmetic expression in a value expression, the product using the CDO expression calculates the value associated with the expression and uses that value when executing the statement. Therefore, an arithmetic expression must be reducible to a value.

The value expression, value-expr, is a symbol or a string of symbols used to calculate a value.

builtin-expr

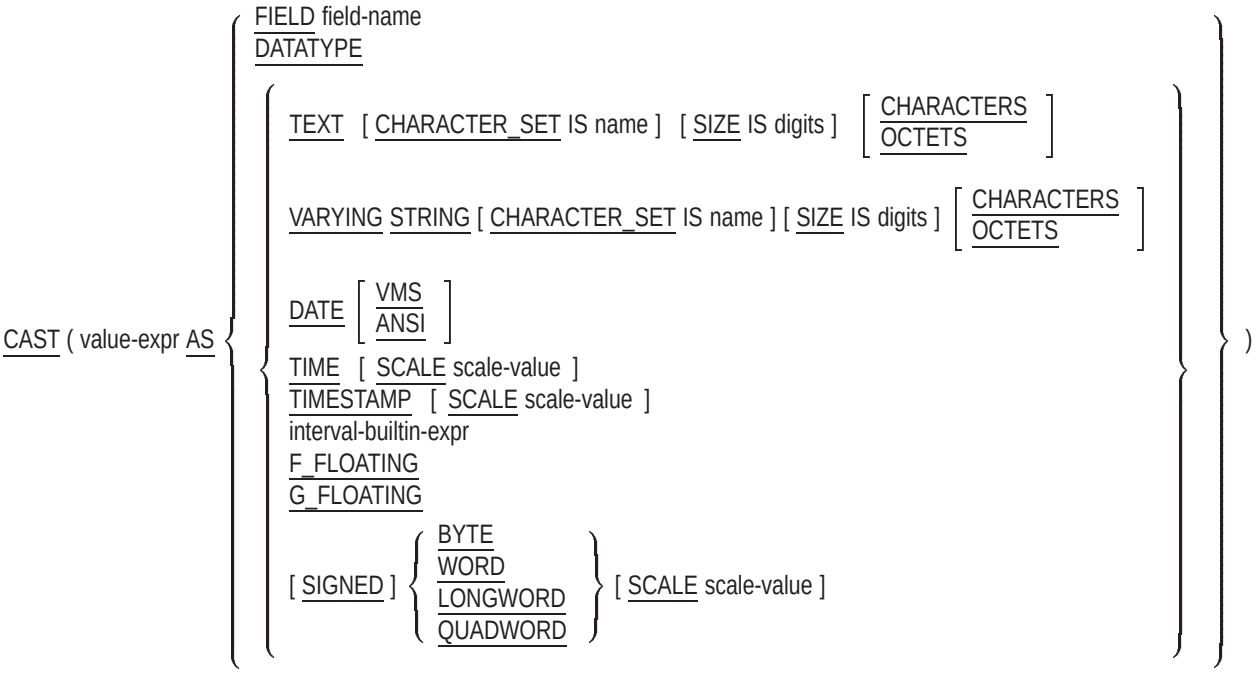


Calculate values based on specified value expressions. See Table 4–4 for descriptions.

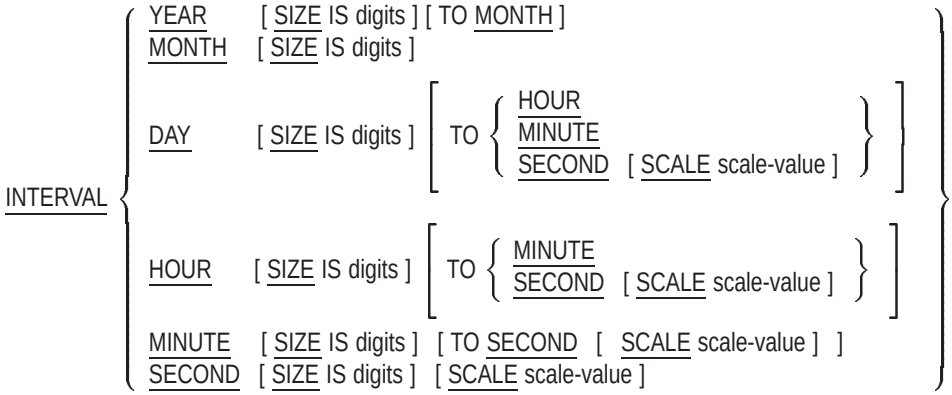
See the *Oracle Rdb7 SQL Reference Manual* for the character set types.

Value Expressions

cast-builtin-expr



interval-builtin-expr



Converts a value expression to another data type.

Restriction

The CAST builtin expression requires a space between the letter T in CAST and the open parenthesis character of the value expression; otherwise, a syntax error occurs.

case-expr

CASE value-expr { WHEN value-expr THEN value-expr } ... [ELSE value-expr] END

Matches two value expressions for equality. This clause is identical to the SQL SIMPLE CASE expression.

char-string-literal

Specifies a string of printable characters. See Value Expressions: Character String Literals for more information.

concatenated-expr

value-expr { | ^ } value-expr ...

Combines two value expressions by joining the second expression to the end of the first expression.

You can combine value expressions of any kind, including numeric expressions, string expressions, and literals.

The vertical bar (|) specifies a value through combining one or more value expressions. The circumflex character (^) specifies a value through combining one or more value expressions using SQL concatenation rules.

external-literal

EXTERNAL quoted-string

Specifies the name of an external literal. Defines the equivalent of the COBOL initial value (VALUE IS EXTERNAL clause) and level 88 condition values (VALUES ARE EXTERNAL clause).

field-or-record-expr

{ dir-name
{ name IN } ... context-var }

Specifies the name of a field or a record in a database by directory name, or by field or record name and context variable. A context variable is a temporary

name you associate with a record. You define a context variable in a record selection expression (RSE). You specify a context variable only when you use the name IN parameter of the field or record expression syntax.

For example, once you define E as the context variable for the EMPLOYEES relation, LAST_NAME IN E is a value expression that refers to a value from the LAST_NAME field of EMPLOYEES. Use name IN only in an expression with a context variable.

first-from-expr

FIRST value-expr FROM rse

Specifies a value by forming a record stream (as indicated by a record selection expression). If at least one record matches the RSE, the values stored in the first record of the record stream are used to evaluate the value expression.

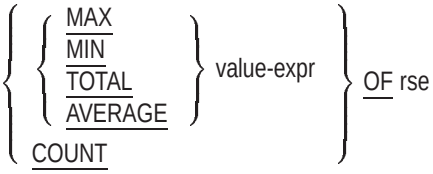
The FIRST FROM expression can perform the equivalent of a table lookup when you are sure that the value you want to find in a relation is unique.

The value expression, value-expr, is a symbol or a string of symbols used to calculate a value. The rse parameter specifies a clause that products use at run time to include specific records for processing.

numeric-literal

Specifies a value that can be expressed as a decimal number. See Value Expressions: Numeric Literals for more information.

statistical-expr



Specifies a value by forming a record stream (as indicated by a record selection expression), and evaluating its value expression against every record in the record stream. Statistical expressions are sometimes called aggregate expressions because they calculate a single value for a collection of records. When you use a statistical expression (except for COUNT), you specify a value expression and an RSE. A layered product evaluates the value expression for each record in the record stream formed by the RSE. Then the product calculates a single value based on the results of the first step.

The COUNT expression differs from the other statistical operators because it operates on the record stream defined by the RSE, rather than on values in that record stream. It returns the number of records in the record stream. In the following expression, the number of employees working in New Hampshire is returned.

```
COUNT OF E IN EMPLOYEES WITH STATE IN E = "NH"
```

Description

A value expression returns a value that can be a string, a number, or a null value.

Table 4–2 describes the operators used in arithmetic expressions.

Table 4–2 Arithmetic Operators

Symbol	Function
+	Addition
–	Subtraction
*	Multiplication
/	Division

Table 4–3 describes the operators used in statistical expressions.

Table 4–3 Statistical Operators

Function	Value of Function
AVERAGE	The average of the values specified by the value expression for all records specified by the RSE. The value expression must be a numeric data type.
COUNT	The number of records in the stream specified by the RSE.
MAX	The largest of the values specified by the value expression for all records specified by the RSE.
MIN	The smallest of the values specified by the value expression for all records specified by the RSE.

(continued on next page)

Table 4–3 (Cont.) Statistical Operators

Function	Value of Function
TOTAL	The sum of the values specified by the value expression for all records specified by the RSE. The value expression must be a numeric data type.

Table 4–4 describes the built-in function names and values. See the *Oracle Rdb7 SQL Reference Manual* for more details on the use and restrictions for using SQL built-in functions.

Table 4–4 Built-in Function Description

Name	Description
NULL	Specifies a null value.
TRIM	Removes leading or trailing characters from any character value expression. Note: The CHARACTER keyword is required in CDO.
POSITION	Searches for a string in a character value expression.
USER	Specifies the user name of the current process.
CURRENT_USER	Returns the current active user name for a request.
CURRENT_DATE	Returns a DATE data type value containing year, month, and day for today's date.
CURRENT_TIME	Returns a TIME data type value containing hours, minutes, and seconds for the current time. You can specify a fractional precision between 0 and 2 for the seconds returned by CURRENT_TIME. The fractional seconds precision is a number that designates the number of digits returned in the field. The fractional precision is the negative of the value specified in the SCALE clause. The CURRENT_TIME keyword and the left parenthesis for the fractional precision must be separated by a space. Otherwise, CDO interprets it as the name of an element with a version of the value specified in the fractional precision.

(continued on next page)

Table 4–4 (Cont.) Built-in Function Description

Name	Description
CURRENT_ TIMESTAMP	Returns a TIMESTAMP data type value containing year, month, and day for today's date and hours, minutes, and seconds for the current time. You can specify a fractional precision between 0 and 2 for the seconds returned by CURRENT_TIMESTAMP. The fractional seconds precision is a number that designates the number of digits returned in the field. The fractional precision is the negative of the value specified in the SCALE clause. The CURRENT_ TIMESTAMP keyword and the left parenthesis for the fractional precision must be separated by a space. Otherwise, CDO interprets it as the name of an element with a version of the value specified in the fractional precision.
CHARACTER_ LENGTH	Calculates the length of a value expression of any data type. You can use CHAR_LENGTH as an alternative for CHARACTER_LENGTH.
OCTET_LENGTH	Calculates the length, in octets, of a value expression of any data type.
UPPER	Converts all lowercase characters in a value expression to uppercase characters.
LOWER	Converts all uppercase characters in a value expression to lowercase characters.
SESSION_USER	Returns the current active session user name.
SUBSTRING	Returns portions of character value expressions.
SYSTEM_USER	Returns the user name of the login process at the time of the database attachment.
CAST	Converts a value expression to another data type.
EXTRACT	Returns a single date-time field expressed as an integer from a field defined with a data type of DATE, TIME, TIMESTAMP, or INTERVAL.

(continued on next page)

Table 4–4 (Cont.) Built-in Function Description

Name	Description
TRANSLATE	Translates a character value expression from one character set to another compatible character set, such as RDB$KANJI to Kanji.

Examples

1.
```
(8 + 14) / 2 - 4
```

   In this example, the arithmetic expression evaluates as 7.

2.
```
DEFINE FIELD NAME
COMPUTED BY FIRST_NAME | ' ' | MIDDLE_INITIAL | ' ' | LAST_NAME.
```

   The output is:

```
JOHN Q PUBLIC
```

   In this example, the concatenated expression combines three fields into the NAME field definition. The space between each pair of quotation marks appears in the output of the NAME field.

3.
```
COUNT OF E IN EMPLOYEES WITH
LAST_NAME IN FULL_NAME IN E = "SMITH".
```

   In this example, the FIELD or RECORD expression specifies the LAST_NAME field in the FULL_NAME record in the EMPLOYEES relation.

4.
```
FIRST SALARY IN E FROM E IN EMPLOYEES
WITH LOCATION IN E = "BUILDING_A"
```

   In this example, the FIRST FROM expression finds the salary of the first employee who works in BUILDING_A.

5.
```
AVERAGE SALARY_AMOUNT IN CS OF CS IN SALARY WITH SALARY_AMOUNT IN CS GT
50000
```

   In this example, the AVERAGE statistical expression uses the RSE to form a stream of records where the SALARY_AMOUNT field is greater than 50,000. The average of the values is calculated by the product reading the expression.

6. `MAX SALARY_AMOUNT IN CS OF SAL IN CURRENT_SALARY WITH SALARY IN SAL = MAX`

   This example shows how to use the MAX expression to find the highest paid employee in the company.

7. `MIN SALARY_AMOUNT IN CS OF SAL IN CURRENT_SALARY WITH SALARY IN SAL = MIN`

   In this example, the MIN expression finds the lowest paid employee in the company.

8. `TOTAL SALARY_AMOUNT IN CS OF CS IN CURRENT_SALARY`

   The TOTAL expression finds the total annual payroll of the company.

9. `8 + 7`

   This example shows an arithmetic expression that adds two numeric literals.

10. `8 + 14 / 2 - 4`

    This is an example of an arithmetic expression that is evaluated as 11.

11. `8 + 14 / (2 - 4)`

    In this example, the arithmetic expression is evaluated as 1.

Value Expressions: Character String Literals

A character string literal is a string of printable characters. The maximum length of a character string is 65,536 characters. The printable characters are:

- Uppercase alphabetic characters (A–Z)
- Lowercase alphabetic characters (a–z)
- Numerals (0–9)
- The following special characters:

```
! @ # $ % ^ & * ( ) - _ = + ‘ ~
[ ] { } ; : ’ " \ | / ? > < . ,
```

- Any other characters that are part of the Digital Multinational character set
- Japanese characters: Kanji, as defined by the JIS X0208:1990 standard, and Narrow Katakana, as defined by the JIS X0201:1976 standard

You must enclose a character string literal in a pair of either single or double quotation marks. Table 4–5 shows the valid use of quotation marks in character string literals.

Table 4–5 Quotation Marks in Character String Literals

Character String Value Expression	Value
"JONES"	JONES
’JONES’	JONES
"JONES’	[invalid]
""""""	"""
"""""	[invalid]
’My name is "Lefty".’	My name is "Lefty".
’My "handle" is "Lefty".’	My ’handle’ is "Lefty".

CDO usually treats uppercase and lowercase forms of the same letter as the same character. However, it preserves the case distinction when doing comparisons of character strings; for example, NAME = "JONES" and NAME = "Jones" yield different results.

- Begin and end a character string literal with the same type of quotation mark.

- To include a quotation mark of one type in a character string literal, enclose the literal in quotation marks of the other type. For example, to include double quotation marks in a character string literal, enclose the character string in single quotation marks.
- If a quotation mark appears in a character string literal enclosed by quotation marks of the same type, use two consecutive quotation marks for every one you want to include in the literal. This technique is necessary if you want to include quotation marks of both types in a single quoted string.

Examples

```
E IN EMPLOYEES WITH LAST_NAME IN E = "Toliver"
```

In this example, the expression specifies the character string literal Toliver.

Value Expressions: Numeric Literals

You can use a literal as a value expression. A literal is either a character string or a numeric literal.

Numeric literals can take the following forms:

- A decimal string consisting of digits and an optional decimal point. The maximum length, not counting the decimal point, is 19 digits.
- A decimal number in scientific notation (E-format), consisting of a decimal string mantissa and a signed integer exponent, separated by the letter D (for double), E (for E-format) or Q (for H_floating).

CDO allows you to use unary plus and minus signs in numeric literals. Numeric literals must start and end with a numeral and cannot include hexadecimal digits. Numeric literals in E notation cannot include embedded spaces.

The following expressions are valid numeric literals:

```
+123
-3.49
0.3338889909
6.03 E+23
```

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

A period at the end of a data definition command line terminates the command; therefore, you cannot use a decimal point to terminate a number if you want to include more data definition clauses in the statement.

If you want to include more data definition clauses, include a zero after the decimal point, or place the value expression in parentheses:

```
COMPUTED BY X * 2.0
COMPUTED BY (X * 2.)
```

Examples

```
S IN SALARY_HISTORY WITH SALARY_AMOUNT IN S > 40000
```

In this example, the expression specifies the numeric literal 40000.

Conditional Expressions

Format

```
{ value-expr1 [CASE_SENSITIVE] operator value-expr2
  condition-clause
  containing-clause
  matching-clause
  missing-clause
  starting-with-clause }
```

Parameters

value-expr1
value-expr2
Specifies a value. A value expression can consist of any of the following: character string literals, numeric literals, or arithmetic, concatenated, or statistical expressions. If either value expression in a condition evaluates to null, the entire condition evaluates to null.

operator
Specifies a mathematical relational operator. See the mathematical relational operators in Table 4–7.

condition-clause

```
field-expr { ALPHABETIC
             ALPHABETIC_LOWER
             ALPHABETIC_UPPER
             EMPTY_FIELD
             FULL_FIELD
             NUMERIC
             NOT ALPHABETIC
             NOT ALPHABETIC_LOWER
             NOT ALPHABETIC_UPPER
             NOT EMPTY_FIELD
             NOT FULL_FIELD
             NOT NUMERIC }
```

Specifies whether a field expression satisfies the specified condition.

Conditional Expressions

The product using CDO evaluates a condition clause as true if the field expression satisfies the condition specified. The field expression specifies the name of a field in the database, consisting of a field name and a directory name or context variable.

When you use the keyword NOT, the product using CDO evaluates the clause as true if the field expression does not satisfy this condition.

containing-clause

value-expr1 CASE_SENSITIVE { CONTAINING | NOT CONTAINING } value-expr2

Specifies whether a value expression contains a second value expression. This operation is not case sensitive unless you specify the CASE_SENSITIVE keyword.

When you use the keyword NOT, the product using CDO evaluates the clause as true if the first string expression does not contain the string that the second string expression specifies.

matching-clause

value-expr CASE_SENSITIVE { MATCHING | NOT MATCHING } match-expr

Specifies a relational clause that tests for substring matches. By using wildcard characters, you can specify the position of the substring. This operation is not case sensitive.

The product using CDO evaluates a MATCHING clause as true if match expression, the second expression, matches a substring of the first expression. Specify the match expression in quotation marks.

When you use the keyword NOT, the product using CDO evaluates the clause as true if the second expression does not match a substring of the first value expression.

missing-clause

{ field-expr | record-expr } { MISSING | NOT MISSING }

Specifies whether a field or record expression is null. The product using CDO evaluates a MISSING clause as true if the record or field expression is null.

Specifies the name of a field or record in the database, consisting of a directory name or a field or record name and a context variable.

When you use the keyword NOT, the product using CDO evaluates the clause as true if the record or field expression is not null.

starting-with-clause

value-expr1 CASE_SENSITIVE { STARTING WITH | NOT STARTING WITH } value-expr2

Specifies whether the first characters of a value expression match the characters of a second value expression. This operation is case sensitive.

The product using CDO evaluates a STARTING WITH clause as true if the first characters of the first string expression match the characters in the second string expression.

When you use the keyword NOT, the product using CDO evaluates the clause as true if the first string does not contain the string that the second string expression specifies.

If either value expression in a condition evaluates to null, the condition evaluates to null.

Description

A conditional expression, sometimes called a Boolean expression, represents the relationship between two value expressions. A conditional expression returns a value of true, false, or null (missing).

Conditional expressions consist of value expressions and relational or logical operators.

You can use conditional expressions in CDO as objects for the WITH clause or VALID IF clause of the record selection expression or the VALID IF clause in field definitions.

Relational Operators

Relational operators specify the relationship of value expressions and perform the following kinds of operations:

- Compare a value with a range
- Match a pattern
- Test for missing fields

Description

CDO uses mathematical relational operators and pattern testing relational operators in its conditional expressions.

Mathematical relational operators are symbols that allow you to compare values. Pattern testing relational operators are keywords that allow you to test for a pattern of values. Unlike the mathematical relational operators, each pattern testing relational operator has its own unique syntax.

Table 4–6 lists the pattern testing relational operators.

Table 4–6 Pattern Testing Relational Operators

Clause	Relational Operation
BETWEEN	True if the first value expression is less than or equal to the second value expression and greater than or equal to the third value expression.
CONTAINING	True if the string specified by the second string expression is found within the string specified by the first string expression. CONTAINING is not case sensitive.

(continued on next page)

Table 4–6 (Cont.) Pattern Testing Relational Operators

Clause	Relational Operation
MATCHING	True if the second value expression matches a substring of the first value expression. MATCHING is not case sensitive. It uses the following wildcard characters: • Asterisk (*)— Matches any string of zero or more characters • Percent sign (%)—Matches any single character in that position
STARTING WITH	True if the first characters of the first string expression match the second string expression. STARTING WITH is case sensitive.

The logical operators AND, OR, and NOT let you compare two or more conditional expressions and optionally reverse the value of a conditional expression. The result of using a logical operator is another conditional expression.

Table 4–7 lists the mathematical relational operators. These operators allow you to compare values. In all cases, if either value expression in a conditional expression is null, the value of the entire condition is null.

Table 4–7 Mathematical Relational Operators

Permitted Symbols	Relational Operation
EQ or =	True if the two value expressions are equal.
NE or <>	True if the two value expressions are not equal.
GT or >	True if the first value expression is greater than the second.
GE or >=	True if the first value expression is greater than or equal to the second.
LT or <	True if the first value expression is less than the second.
LE or <=	True if the first value expression is less than or equal to the second.

Relational Operators

Use either the alphabetic symbol or the mathematical symbol from the Permitted Symbols column, but do not use both when you specify a relational operator.

See the documentation for the languages and products that use the repository to determine how that product evaluates character string literals. In some cases, character string literals are compared according to the ASCII collating sequence. Under ASCII, lowercase letters have a greater value than uppercase letters, and the letters near the beginning of the alphabet have a lesser value than those near the end.

For products that compare character string literals according to the ASCII collating sequence, the following statements are true:

- a > A
- a < z
- A < Z

To determine how CDO conditional expressions linked by logical operators are evaluated, see the documentation for the product that will be evaluating the conditional expression.

See the documentation for languages and products that use the repository to determine how they evaluate character string literals.

Caution

The NOT operator applies to conditional expressions. Do not use the NOT operator and an equal sign instead of the NE or <> relational operators. The following statement is not valid:

```
WITH SALARY_AMOUNT IN S NOT = 30000
```

Use one of the following alternatives:

```
WITH NOT (SALARY_AMOUNT IN S = 30000)
WITH SALARY_AMOUNT IN S NE 30000
WITH SALARY_AMOUNT IN S <> 30000
```

Examples

1.
```
LAST_NAME CONTAINING "ith"
LAST_NAME NOT CONTAINING "son"
```

   In this example, if LAST_NAME has the string ith, CDO evaluates the CONTAINING clause as true; if LAST_NAME does not contain the string son, CDO evaluates the CONTAINING clause as true.

2.
```
SALARY_AMOUNT IN SH > 50000
```

   In this example, the conditional expression is true if the value in the SALARY_AMOUNT field is greater than 50,000.

3.
```
NOT SALARY_AMOUNT IN SH < 50000
```

   In this example, the conditional expression is true if the value in the SALARY_AMOUNT field is less than 50,000.

4.
```
DEFINE FIELD SEX
VALID IF (SEX CASE_SENSITIVE EQ "M") OR (SEX CASE_SENSITIVE EQ "F").
```

   In this example, the DEFINE FIELD uses a case sensitive relational operator in the VALID IF clause to test whether the code to be entered in the field SEX is M or F. The conditional expression is true if the value for the field SEX is M or F (not m or f).

5.
```
LAST_NAME MATCHING "*ON"
```

   In this example, the conditional expression is true if the field LAST_NAME has ON as the last two letters. You can use this expression to find all records with LAST_NAME fields satisfying this condition.

6.
```
LAST_NAME IN FULL_NAME IN E MISSING
```

   In this example, the conditional expression is true if the LAST_NAME field in the FULL_NAME record of the EMPLOYEES relation is missing.

7.
```
LAST_NAME IN FULL_NAME IN E ALPHABETIC
```

   In this example, CDO evaluates the field expression as true when the LAST_NAME field from the FULL_NAME record of the EMPLOYEES relation is alphabetic.

8. `SALARY_AMOUNT NOT MISSING`

   In this example, the conditional expression is true if the SALARY_ AMOUNT field has a value that is not null.

9. `SALARY_AMOUNT MATCHING "4*"`

   This example shows the matching clause used with numeric data types. In this example, the matching clause finds all the salaries that start with the number 4.

10. `SALARY_AMOUNT BETWEEN 40000 AND 49999`

    This example finds all salaries in a range by using the BETWEEN clause.

Record Selection Expression (RSE)

Format

[first-clause] relation-clause [cross-clause] [with-clause] [reduced-clause] [sort-clause]

Parameters

first-clause

FIRST value-expr

Specifies how many records are in the record stream formed by the record selection expression (RSE). The value expression, value-expr, is a symbol or string of symbols used to calculate a value. The value expression in a FIRST clause must either be a positive number or a value expression that evaluates to a positive integer. The record stream cannot contain more records than the number specified by the value expression.

relation-clause

context-var IN relation-name

Declares context variables for a record stream or a loop. The context variable specifies a temporary name that identifies the record stream to the product evaluating the clause. You then use the context variable to refer to fields from that relation. The relation name specifies the relation from which CDO will take the records in the record stream.

cross-clause

{ CROSS relation-clause } ...

Allows you to combine records from two or more record streams. You join these records in combinations based on the relationship between the values of fields in each record stream. This combination is called a relational join.

The relation clause declares context variables for a record stream or loop.

with-clause

WITH cond-expr

Allows you to specify conditions that must be true for CDO to include a record in a record stream. You specify any conditional expression in this clause.

Record Selection Expression (RSE)

The record becomes part of a record stream only when its values satisfy the conditions you specified in the conditional expression (that is, only when the conditional expression is true). If the conditional expression evaluates to false or missing for a record, that record is not included in the record stream.

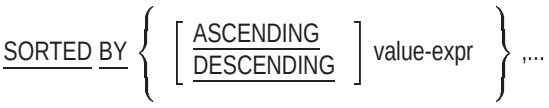
reduced-clause

REDUCED TO value-expr ,...

Allows you to eliminate duplicate values for fields in a record stream and to group the records in a relation according to unique field values. However, only using the REDUCED clause does not guarantee the sort order within groups and the results are unpredictable. To ensure specific order, use the SORTED BY clause.

The value expression, value-expr, specifies a symbol or string of symbols used to calculate a value.

sort-clause



Allows you to sort the records in the record stream by the values of specific fields. The value expression, or sort key, determines the order in which CDO returns records. The default for an initial sort key is ASCENDING. The default for subsequent keys is the specification for the initial key.

The value expression, value-expr, specifies the value to sort by; this value is called the sort key.

Description

A record selection expression (RSE) is a clause that products use at run time to include specific records for processing. The RSE defines the conditions that individual records must meet before CDO includes them in a record stream.

Examples

1.
```
FIRST 5 C IN CURRENT_SALARY
SORTED BY DESCENDING SALARY_AMOUNT IN C
```

   You can use FIRST and SORTED BY clauses to find the maximum values for a field. In this example, the FIRST clause finds the five highest paid employees.

2. `E IN EMPLOYEES`

   In this example, the RELATION clause retrieves all records from the EMPLOYEES relation.

3. `COUNT OF E IN EMPLOYEES WITH STATE IN E = "NY"`

   In this example, the RELATION clause declares E as the context variable for the stream of records from the EMPLOYEES relation.

4. ```
   E IN EMPLOYEES CROSS JH IN JOB_HISTORY
   WITH EMP_ID IN E = EMP_ID IN JH
   ```

   In this example, the CROSS clause finds all employees for whom data is stored in the JOB_HISTORY relation.

5. `E IN EMPLOYEES CROSS J IN JOBS`

   In this example, the CROSS clause retrieves information on all employees and their job descriptions.

6. `E IN EMPLOYEES WITH JOB_CODE IN E = "R"`

   In this example, the WITH clause returns all employees whose JOB_CODE equals R.

7. `REDUCED TO JOB_CODE IN J`

   In this example, the REDUCED clause lists all active job codes once.

8. `EMPLOYEES SORTED BY EMPLOYEE_ID IN E`

   In this clause, the SORTED BY clause sorts EMPLOYEES by EMPLOYEE_ID.

9. ```
   SORTED BY DESCENDING STATUS_CODE IN E
   ASCENDING LAST_NAME IN E, EMPLOYEE_ID IN E
   ```

   In this example, the SORTED BY clause sorts first by STATUS_CODE in descending order. Within each STATUS_CODE group, SORTED BY sorts by LAST_NAME in ascending order. Finally, within groups of employees with the same last name, SORTED BY sorts by EMPLOYEE_ID. The order for this last sort is also ascending, because it adopts the order from the previous sort key.

5
CDO Edit Strings

If a CDO field element contains an edit string, CDO performs an automatic translation of the CDO edit string characters for the following languages that support edit strings:

- COBOL
- DATATRIEVE
- PL/I
- RPG

Table 5–1 shows how CDO translates edit string characters for COBOL picture clause characters, DIGITAL DATATRIEVE edit string characters, PL/I picture clause characters, and RPG edit word characters.

The following four symbols are used in Table 5–1:

- <n>—Not supported; if the CDO character appears in a CDO edit string, no picture clause or edit string is generated for the language.
- <i>—Ignored; the CDO character is ignored for the language. However, if the CDO character appears in an edit string with other characters that can be translated, CDO will perform the appropriate translation for the other edit string characters.
- <pc>—The CDO character has the same value as the previous character in the edit string.
- <**>—Characters appearing after the CDO character in the edit string are not translated for the language.

Table 5–1 Translation of CDO Edit Strings for Languages and Products

Character Type	CDO Character or String	COBOL PICTURE	DTR EDIT	PL/I PICTURE	RPG EDIT WORD
Alphabetic	A	A	A	<n>	<n>
Alphanumeric	T	X	T	<n>	<n>
	X	X	X	<n>	<n>
Comma	,	,	,	,	,
Date, Day, and Time	D	<n>	D	<n>	<n>
	H	<n>	<i>	<n>	<n>
	J	<n>	J	<n>	<n>
	M	<n>	M	<n>	<n>
	N	<n>	N	<n>	<n>
	P	<n>	<i>	<n>	<n>
	Q	<n>	<i>	<n>	<n>
	R	<n>	<i>	<n>	<n>
	W	<n>	W	<n>	<n>
	Y	<n>	Y	<n>	<n>
	%	<n>	<i>	<n>	<n>
	*	<n>	<i>	<n>	<n>
Decimal point	.	.	.	.	.
Digit	F	<n>	<n>	<n>	<n>
	7	<n>	<n>	<n>	<n>
	9	9	9	9	blank
Encoded sign	C	-9	-9	R	<n>
	G	+9	+9	T	<n>
	K	+9	+9	I	<n>
Exponent	E	<n>	E	<n>	<n>
Floating	S	+	+	S	–

(continued on next page)

Table 5–1 (Cont.) Translation of CDO Edit Strings for Languages and Products

Character Type	CDO Character or String	COBOL PICTURE	DTR EDIT	PL/I PICTURE	RPG EDIT WORD
	Z"string"	See Table 5–2.	See Table 5–2.	See Table 5–2.	See Table 5–2.
	–	–	–	–	–
	+	+	+	+	blank
	$	$	$	$	$
	\	<pc>	<pc>	<pc>	<pc>
Literal	"string"	See Table 5–3.	See Table 5–3.	See Table 5–3.	See Table 5–3.
Logical	B	9	9	9	blank
Lowercase	L	<i>	<i>	<i>	<n>
Minus literal	&"string"	See Table 5–4.	See Table 5–4.	See Table 5–4.	See Table 5–4.
Minus parentheses	(( ))	–	(( ))	–	–
Missing separator	?	<i><**>	?	<i><**>	<i><**>
Repeat count	x(n)	x(n)	x(n)	(n)x	x repeats n times.
Uppercase	U	<i>	<i>	<i>	<n>

5.1 Chapter Organization

The rest of this chapter provides descriptions and examples of the CDO edit string characters. To find a description of a particular edit string character, locate the character in Table 5–1 and determine its character type from the first column of the table.

Character type descriptions appear in alphabetical order, and descriptions of individual edit string characters appear in alphabetical order within the character type descriptions.

The following sections include examples of CDO edit strings. Each example is one of two types:

- The example shows a sample CDO edit string, the translation of the CDO edit string that CDO provides to the language or product using the string, the field value of the edit string, and the edited value of the string.
- The example shows a sample CDO edit string, the field value of the edit string, and the edited value that would be produced if a language or product supported the string.

Use Table 5–1 to determine how CDO translates CDO edit string characters into COBOL picture clause characters, DIGITAL DATATRIEVE edit string characters, PL/I picture clause characters, and RPG edit word characters.

5.2 Alphabetic Character

The edit string character **A** (uppercase letter A) is replaced by an alphabetic character from the field's content. The action taken when a character in the field is not alphabetic is language dependent. See the documentation for each language to determine the action taken when a digit or nonalphabetic character occurs.

CDO Edit String:	AAAA
DTR Edit String:	AAAA
Field Value:	WXYZ
Edited Value:	WXYZ

You can use the repeat count edit string character to indicate that you want to repeat the edit string character A a certain number of times. The following two edit strings are equivalent:

```
AAAA
A(4)
```

5.3 Alphanumeric Character

The two alphanumeric edit string characters are T and X.

5.3.1 T: Long Text Character

The edit string character **T** (uppercase letter T) allows you to display any characters from a field's content on one or more lines. The primary use of the T edit string is to print fields containing large amounts of text. The number of Ts in the edit string indicates the maximum number of characters to be printed on one line. For example, the edit string TTTTT indicates that a line of output will contain no more than five characters.

If the field contains more characters than specified in the edit string, DIGITAL DATATRIEVE prints as many full words on the line as possible. (A word in this sense is a string of characters delimited by a space.) DIGITAL DATATRIEVE then prints the remaining characters on the following lines, if necessary. DIGITAL DATATRIEVE does not print out trailing spaces when you use a T edit string.

DIGITAL DATATRIEVE prints only full words. It does not divide words unless a single word is longer than the maximum number of characters specified by the edit string. In that case, DIGITAL DATATRIEVE truncates some characters and prints them on the next line.

CDO Edit String:	`TTTTT`
DTR Edit String:	`TTTTT`
Field value:	`1234567890`
Edited Value:	`12345` `67890`

You can use the repeat count edit string character to indicate that you want to repeat the edit string character T a certain number of times. The following two edit strings are equivalent:

```
TTTTT
T(5)
```

5.3.2 X: Any Character

The edit string character **X** (uppercase letter X) displays any character from the field's content.

CDO Edit String:	`XXXXXXXXXX`
DTR Edit String:	`XXXXXXXXXX`
Field Value:	`fj32dj%^*I`
Edited Value:	`fj32dj%^*I`

You can use the repeat count edit string character to indicate that you want to repeat the edit string character X a certain number of times. The following two edit strings are equivalent:

```
XXXXXXXXXX
X(10)
```

5.4 Comma Character

In fields with nonnumeric values, the edit string **,** character (a comma) inserts a comma into the edited value.

In fields with numeric values, the edit string comma character inserts a comma or suppresses a leading zero in the edited value.

The following example shows how the comma character is inserted into the edited value when the edit string has a nonnumeric value.

CDO Edit String:	AA,AA
DTR Edit String:	AA,AA
Field Value:	ohno
Edited Value:	oh,no

The following example shows how a comma character in a CDO edit string causes a comma to be inserted in the edited value.

CDO Edit String:	$$,$$9.99
DTR Edit String:	$$,$$9.99
Field Value:	1234.56
Edited Value:	$1,234.56

The following example shows what happens when a comma character is included in an edit string with numeric values, but the comma is not required in the edited value:

CDO Edit String:	$$,$$9.99
DTR Edit String:	$$,$$9.99
Field Value:	12.34
Edited Value:	$12.34

5.5 Date, Day, and Time Characters

The edit string characters described in this section are used to specify the output format of fields containing date, day, and time information. See Section 5.11 for an explanation of how products and languages handle characters enclosed by double quotation marks in CDO edit strings.

5.5.1 D: Day Number Character

The edit string character **D** (uppercase letter D) displays a digit of the day within a month. You should repeat this character twice within an edit string.

CDO Edit String:	`NN"/"DD"/"YYYY`
DTR Edit String:	`NN/DD/YYYY`
Field Value:	`May 4, 1996`
Edited Value:	`05/04/1996`

5.5.2 H: Twelve-Hour Mode Character

The edit string character **H** (uppercase letter H) displays one digit of the hour, in 12-hour mode, in the edited value. You should repeat this character twice within an edit string.

Do not use this character in edit strings containing the R (24-hour mode) character.

CDO Edit String:	`HH":"PP" "%%`
Field Value:	`11:30 a.m.`
Edited Value:	`11:30 AM`

5.5.3 J: Julian Digit Character

The edit string character **J** (uppercase letter J) displays a digit of the Julian date in the edited value. You should repeat this character three times within an edit string.

CDO Edit String:	`YYYY"/"JJJ`
DTR Edit String:	`YYYY/JJJ`
Field Value:	`June 4, 1980`
Edited Value:	`1980/156`

5.5.4 M: Month Name Character

The edit string character **M** (uppercase letter M) displays a letter of the month name in the edited value.

CDO Edit String:	`MMM" "DD" "YYYY`
DTR Edit String:	`MMMBDDBYYYY`
Field Value:	`May 4, 1980`
Edited Value:	`MAY 04 1980`

In the following example, the CDO edit string contains five M characters. If the month name in the field value contains more than five characters, only the first five characters of the month name are displayed in the edited value.

CDO Edit String:	`MMMMM" "DD" "YYYY`
DTR Edit String:	`MMMMMBDDBYYYY`
Field Value:	`December 4, 1987`
Edited Value:	`DECEM 04 1987`

If the month name in the field value contains fewer than five characters, trailing blanks are displayed after the month name in the edited value.

CDO Edit String:	`MMMMM" "DD" "YYYY`
DTR Edit String:	`MMMMMBDDBYYYY`
Field Value:	`May 4, 1980`
Edited Value:	`MAY 04 1980`

5.5.5 N: Month Number Character

The edit string character **N** (uppercase letter N) displays a digit for the number of the month in the edited value. You should repeat this character twice within an edit string.

CDO Edit String:	`NN"/"DD"/"YYYY`
DTR Edit String:	`NN/DD/YYYY`
Field Value:	`May 4, 1985`
Edited Value:	`05/04/1985`

5.5.6 P: Minute Character

The edit string character **P** (uppercase letter P) displays one digit for the number of minutes in a time value in the edited field. You should repeat this character twice within an edit string.

CDO Edit String:	`HH":"PP" "%%`
Field Value:	`11:30 a.m.`
Edited Value:	`11:30 AM`

5.5.7 Q: Second Character

The edit string character **Q** (uppercase letter Q) displays one digit for the number of seconds in a time value in the edited value. You should repeat this character twice within an edit string.

CDO Edit String:	`PP":"QQ"."**`
Field Value:	`23 minutes 13.56 seconds`
Edited Value:	`23:13.56`

5.5.8 R: Twenty-Four Hour Mode Character

The edit string character **R** (uppercase letter R) displays one digit of the hour, in 24-hour mode, in the edited value. You should repeat this character twice within an edit string.

Do not use this character in edit strings containing the H character (12-Hour mode) character.

CDO Edit String:	`RR":"PP`
Field Value:	`2:30 p.m.`
Edited Value:	`14:30`

5.5.9 W: Weekday Name Character

The edit string character **W** (uppercase letter W) displays a letter from the day of week in a time value into the edited value.

CDO Edit String:	`WWWWWWWWW`
DTR Edit String:	`WWWWWWWWW`
Field Value:	`June 3, 1987`
Edited Value:	`WEDNESDAY`

You can use the repeat count edit string character to indicate that you want to repeat the edit string character W a certain number of times. The following two edit strings are equivalent.

```
WWWWWWWWW
W(9)
```

5.5.10 Y: Year Character

The edit string character **Y** (uppercase letter Y) displays a digit of the year in a time value into the edited value. You should repeat this character either two or four times in an edit string.

CDO Edit String:	`MMM" "DD" "YY`
DTR Edit String:	`MMMBDDBYY`
Field Value:	`May 4, '85`
Edited Value:	`MAY 04 85`

The CDO edit string can also contain four Y characters.

CDO Edit String:	`MMM" "DD" "YYYY`
DTR Edit String:	`MMMBDDBYYYY`
Field Value:	`May 4, 1985`
Edited Value:	`MAY 04 1985`

5.5.11 % : AM/PM Character

The edit string character **%** (a percent sign) displays a character from one of the strings AM or PM in the edited value. You should repeat this character twice within an edit string.

The edit string character % is most useful when used with the edit string character H (twelve-hour mode character) and when placed at the end of the edit string.

CDO Edit String:	`HH":"PP" "%%`
Field Value:	`11:30 a.m.`
Edited Value:	`11:30 AM`

5.5.12 * : Fraction Second Character

The edit string character ***** (an asterisk) displays a value for fractions of a second within a time field in the edited value. Repeat this character twice within an edit string to denote hundredths of a second.

CDO Edit String:	`MM":"QQ"."**`
Field Value:	`23 minutes 13.56 seconds`
Edited Value:	`23:13.56`

5.6 Decimal Point Character

The edit string decimal point character **.** (a period) inserts a period into the edited value. You can use this character only once within an edit string for numeric fields.

CDO Edit String:	`99.99`
DTR Edit String:	`99.99`
Field Value:	`2813E-2`
Edited Value:	`28.13`

5.7 Digit Characters

You can represent hexadecimal (F), octal (7), and decimal (9) digits in edit strings.

5.7.1 F: Hexadecimal Digit Character

The edit string character **F** (uppercase letter F) displays one hexadecimal digit in the edited value.

Do not use the hexadecimal digital character (F) within an edit string containing the octal character (7) or the decimal character (9).

CDO Edit String:	`FFF`
Field Value:	`32`
Edited Value:	`020`

5.7.2 7: Octal Digit Character

The edit string character **7** (the number seven) displays one octal digit in the edited value.

Do not use the octal digit character (7) within an edit string containing the hexadecimal character (F) or the decimal character (9).

CDO Edit String:	`777`
Field Value:	`32`
Edited Value:	`040`

5.7.3 9: Decimal Digit Character

The edit string character **9** (the number nine) displays one decimal digit in the edited value.

Do not use the decimal digit character (9) within an edit string containing the hexadecimal character (F) or the octal character (7).

CDO Edit String:	`999`
DTR Edit String:	`999`
Field Value:	`613`
Edited Value:	`613`

5.8 Encoded Sign Characters

The encoded sign edit string characters are the encoded minus edit string character (C), the encoded sign edit string character (G), and the encoded plus edit string character (K).

5.8.1 C: Encoded Minus Character

If the field's value is negative, the encoded minus character **C** (uppercase letter C) overwrites the next digit with a minus sign (-), then moves the encoded digit to the edit string.

If the field's value is positive or zero, this character moves the next digit into the edited string.

Use this character only at the beginning or end of a string.

An edit string can contain only one character designating a sign.

Do not use this character within an edit string that contains other characters designating a sign.

See the reference manual for the language or product that will interpret the edit string to determine how the language or product interprets encoded sign characters.

CDO Edit String:	`C99`
PL/I Picture Clause:	`R99`
Field Value:	`-456`
Edited Value:	`M56`

5.8.2 G: Encoded Sign Character

If the field's value is positive, the encoded sign character **G** (uppercase letter G) overwrites the next digit with a plus sign (+), then moves the encoded digit to the edit string.

If the field's value is negative, the encoded sign character (G) overwrites the next digit with a minus sign (-), then moves the encoded digit to the edit string.

If the field's value is zero, the action of this character depends on the language.

Use this character only at the beginning or end of a string.

An edit string can contain only one character designating a sign.

Do not use this character within an edit string that contains other characters designating a sign.

See the reference manual for the language or product that will interpret the edit string to determine how the language or product interprets encoded sign characters.

CDO Edit String:	`G99`
PL/I Picture Clause:	`T99`
Field Value:	`+123`
Edited Value:	`A23`

5.8.3 K: Encoded Plus Character

If the field's value is positive, the encoded plus character **K** (uppercase letter K) overwrites the next digit with a plus sign (+), then moves the encoded plus digit to the edit string.

If the field's value is negative or zero, the encoded plus character (K) moves the next digit into the edited string.

Use this character only at the beginning or end of a string.

An edit string can contain only one character designating a sign.

Do not use this character within an edit string that contains other characters designating a sign.

See the reference manual for the language or product that will interpret the edit string to determine how the language or product interprets encoded plus characters.

CDO Edit String:	K99
PL/I Picture Clause:	I99
Field Value:	+123
Edited Value:	A23

5.9 Exponent Character

The edit string character **E** (uppercase letter E) divides an edit string into two parts for floating-point or scientific notation. The first part is the characteristic and mantissa edit string and the second part is the exponent edit string.

CDO Edit String:	S99ES99
DTR Edit String:	+99E+99
Field Value:	1200
Edited Value:	+12E+02

5.10 Floating Characters

There are six edit string floating characters: S, Z, -, +, $, and \.

5.10.1 S: Floating Sign Character

If you use the edit string character **S** (uppercase letter S) only once within an edit string, one of the following results can occur:

- If the field's value is positive, the character S inserts a plus sign (+) into the edited value.

CDO Edit String:	S9
DTR Edit String:	+9
Field Value:	6
Edited Value:	+6

- If the field's value is negative, the floating sign character inserts a minus sign (-) into the edited value.

  CDO Edit String: `S9`
  DTR Edit String: `+9`
  Field Value: `-8`
  Edited Value: `-8`

- If the field's value is zero, the effect of the floating sign character depends on the language reading the field.

  CDO Edit String: `S9`
  DTR Edit String: `+9`
  Field Value: `0`
  Edited Value: `+0`

  If you use more than one floating sign character at the beginning of an edit string, the S character suppresses leading zeros before inserting a plus sign (+) or minus sign (-) into the edited value.

  CDO Edit String: `SS99`
  DTR Edit String: `++99`
  Field Value: `0054`
  Edited Value: `+54`

You cannot use the S character within an edit string that contains another character designating a sign.

5.10.2 Z: Floating Zero Replace Character

If the digit within the field's value is zero, the floating zero replace character **Z** (uppercase letter Z) displays a literal value instead of the zero digit in the edited string.

If the digit within the field's value is not zero, the floating zero replace character displays the digit.

CDO Edit String: `Z" "99`
DTR Edit String: `Z99`
Field Value: `25`
Edited Value: `25`

The following example also shows the floating zero replace character.

CDO Edit String: `Z"*"99`
DTR Edit String: `*99`
Field Value: `25`
Edited Value: `*25`

Table 5–2 shows how CDO translates individual CDO characters in floating zero replace edit strings for other languages.

Table 5–2 Translation of Characters in Floating Zero Replace Edit Strings

Z String Format	CDO Character in String	COBOL PICTURE	DTR EDIT	PL/I PICTURE	RPG EDIT WORD
Z"string"	blank	Z	Z	Z or Y	0
	*	*	*	*	*
	any other character	*	*	*	*

See the PL/I documentation for an explanation of when the CDO blank character is translated to Z and when it is translated to Y.

You cannot use the Z character within an edit string that contains another character designating a sign.

5.10.3 - : Floating Minus Character

If you use the edit string floating minus character, a minus sign (-), only once within an edit string, one of the following results can occur:

- If the field's value is positive, the floating minus character inserts a blank into the edited value.
- If the field's value is negative, the floating minus character inserts a minus sign into the edited value.
- If the field's value is zero, the effect of the floating minus character depends on the language reading the field. If you use this character a number of times at the beginning of an edit string, the character suppresses leading zeros before inserting a blank or minus sign into the edited value.

You cannot use the minus character within an edit string that contains another character designating a sign.

For a nonnumeric field, this character moves a minus sign into the edited value.

CDO Edit String:	`---9`
DTR Edit String:	`---9`
Field Value:	`-54`
Edited Value:	`-54`

5.10.4 + : Floating Plus Character

If you use the edit string floating plus character, a plus sign (**+**), only once within an edit string, one of the following results can occur:

- If the field's value is positive, the floating plus character inserts a plus sign into the edited value.
- If the field's value is negative, the floating plus character inserts a blank into the edited value.
- If the field's value is zero, the effect of the floating plus character depends on the language reading the field. If you use this character a number of times at the beginning of an edit string, the character suppresses leading zeros before inserting a blank or plus sign into the edited value.

You cannot use this character within an edit string that contains another character designating a sign.

CDO Edit String:	`+++9`
DTR Edit String:	`+++9`
Field Value:	`54`
Edited Value:	`+54`

5.10.5 $: Floating Currency Character

If you use the floating currency character only once, it inserts the dollar sign (**$**) in the next character position in the edited value.

If you supply the floating currency character more than once at the beginning of an edit string, it suppresses leading zeros. To suppress up to four leading zeros, supply the character four times at the beginning of the edit string.

The floating currency character inserts the currency sign to the left of the first printed digit of the edited value.

CDO Edit String:	`$,$$$.99`

DTR Edit String:	`$,$$$.99`
Field Value:	`157.86`
Edited Value:	`$157.86`

5.10.6 \ : Floating Blank Character

The floating blank character, a backslash (\), suppresses blanks from an edited value. If the value of the character is a blank, the edited value excludes it.

CDO Edit String:	`MMM\\\\\,YYYY`
Field Value:	`June 15, 1982`
Edited Value:	`JUNE,1982`

5.11 Literal Characters

A pair of double quotation marks (" ") are the edit string literal characters. Any character string enclosed by double quotation marks in an edit string is inserted into the edited value.

CDO Edit String:	`99" ""Hours"`
DTR Edit String:	`99B'Hours'`
Field Value:	`40`
Edited Value:	`40 Hours`

Table 5–3 shows how CDO translates CDO literal edit strings for other languages. In the table, <i> means that the language does not support the CDO character, and the character is ignored for the language.

If the character appears in an edit string with other characters that can be translated, CDO will perform the appropriate translation for the other edit string characters. If CDO characters other than a blank, 0, /, or % occur in a string, CDO passes those characters to DIGITAL DATATRIEVE and RPG without performing a translation.

If a CDO edit string consists of only a 0, /, %, or a blank, then CDO performs the appropriate translation as shown in the first four lines of the table. The last line of the table shows how CDO translates any other edit string for each language. For example, CDO translates the CDO edit string "sample string" to 'sample string' for DIGITAL DATATRIEVE.

Note that the initial blank character in the CDO edit string does not become a B in DIGITAL DATATRIEVE except when the blank is the only character in the edit string. CDO translates characters from the first four lines of the table differently depending on whether or not they are the only characters in an edit string.

Table 5–3 Translation of CDO Literal Edit Strings

Literal String Format	CDO String	COBOL PICTURE	DTR EDIT	PL/I PICTURE	RPG EDIT WORD
"string"	Blank	B	B	B	&
	0	0	0	<i>	0
	/	/	/	/	/
	%	<i>	%	<i>	%
	Any other string	<i>	'string'	<i>	string

5.12 Logical Character

The edit string character **B** (uppercase letter B) is the logical character. It displays logical fields as either TRUE or FALSE.

CDO Edit String: `BBBBB`
Field Value: `0`
Edited Value: `FALSE`

5.13 Lowercase Character

The edit string character **L** (uppercase letter L) prints any remaining alphabetic characters in a field's value in lowercase.

CDO Edit String: `LMMM" "DD`
Field Value: `November 12th`
Edited Value: `nov 12`

5.14 Minus Literal Character

The minus literal character **&** (an ampersand) replaces a negative sign in a field's value with a literal you supply. If the field's value is positive, this character moves a blank to the string instead of the literal you supply.

Do not use this character within an edit string that contains another character designating a sign.

CDO Edit String: `99&"CR"`

DTR Edit String: 99CR
Field Value: -15
Edited Value: 15CR

Table 5–4 shows how CDO translates CDO minus literal edit strings for other languages. If CDO characters other than CR or DB appear in a string, CDO translates the string as a hyphen (-) for COBOL, DIGITAL DATATRIEVE, and PL/I. CDO passes any CDO character to RPG without performing a translation.

Table 5–4 Translation of CDO Minus Literal Edit Strings

Minus Literal String Format	CDO String	COBOL PICTURE	DTR EDIT	PL/I PICTURE	RPG EDIT WORD
&"string"	CR	CR	CR	CR	CR
	DB	DB	DB	DB	DB
	Any other string	-	-	-	string

5.15 Minus Parentheses Character

The edit string character **(( ))** (double opening and closing parentheses characters) is the minus parentheses character. It encloses negative values in parentheses.

CDO Edit String: ((999))
DTR Edit String: ((999))
Field Value: -678
Edited Value: (678)

5.16 Missing Separator Character

The edit string character **?** (a question mark) is the missing separator character. If the field has a missing value attribute or relationship, this character separates two edit strings. If the field value is not the missing value, the first edit string controls the output of the field. If the contents of the field contain the missing value, the second edit string controls the output of the field.

CDO Edit String:	`999?"Unknown"`
DTR Edit String:	`999?'Unknown'`
Field Value:	`missing value`
Edited Value:	`Unknown`

5.17 Repeat Count Character

The edit string character **x(n)** is the repeat count character. Replace x with the character that you want to repeat in an edit string. Replace n with a positive integer that indicates the number of times you want to repeat the character designated by x.

CDO Edit String:	`W(9)`
DTR Edit String:	`W(9)`
Field Value:	`June 3, 1987`
Edited Value:	`WEDNESDAY`

5.18 Uppercase Character

The edit string character **U** (uppercase letter U) prints any remaining alphabetic characters in a field's value in uppercase.

The following example shows a CDO edit string containing the U character, the field value, and the edited value that would be produced if a product supported these characters:

CDO Edit String:	`UA(20)`
Field Value:	`Jones`
Edited Value:	`JONES`

5.19 Japanese Edit Strings

Oracle CDD/Repository supports edit string syntax that provides characters for DIGITAL VAX COBOL to handle Japanese edit strings in the PICTURE clause. Japanese edit string support is available in Oracle CDD/Repository Version 6.1 and later. When you specify N or B for an edit string, enclose it between square brackets ([ ]).

Do not use these edit string characters for Oracle CDD/Repository field definitions that will be used from DIGITAL DEC COBOL.

A

Mapping of Keywords with the DEFINE_RMS_DATABASE Command

The tables in this appendix show the mapping of the different keywords used when creating a logical RMS database in a CDO dictionary.

Table A–1 shows the mapping of File Definition Properties keywords to symbolic field offsets.

Table A–1 Mapping of Keywords to Symbolic Field Offsets

Keyword	Symbolic Field Offset
Area Properties	
ALLOCATE	XAB$L_ALQ
ANY_CYLINDER	XAB$V_ONC
BEST_TRY_CONTIGUOUS	XAB$V_CBT[1]
BUCKET_SIZE	XAB$B_BKZ[2]
CONTIGUOUS	XAB$V_CTG[3]
EXACT_POSITIONING	XAB$V_HRD
EXTENSION	XAB$W_DEQ[4]
POSITION	XAB$B_ALN
VOLUME	XAB$W_VOL

[1]The BEST_TRY_CONTIGUOUS *area* property overrides the BEST_TRY_CONTIGUOUS *file processing* property when you use both in the same definition.

[2]The BUCKET_SIZE *area* property overrides the BUCKET_SIZE *file definition* property when you use both in the same definition.

[3]The CONTIGUOUS *area* property overrides the CONTIGUOUS *file processing* property when you use both in the same definition.

[4]The EXTENSION *area* property overrides the EXTENSION *field* property when you use both in the same definition.

(continued on next page)

Table A–1 (Cont.) Mapping of Keywords to Symbolic Field Offsets

Keyword	Symbolic Field Offset
File Access Block Properties	
ACCESS	FAB$B_FAC
BLOCK_SPAN	FAB$V_BLK
CARRIAGE_CONTROL	FAB$B_RAT
CHANNEL_ACCESS_MODE	FAB$V_CHAN_MODE
CONTROL_FIELD_SIZE	FAB$B_FSZ
EXTENSION	FAB$W_DEQ[5]
FAC	FAB$B_FAC
FORMAT	FAB$B_RFM
LOGICAL_NAME_MODE	FAB$V_LNM_MODE
SHARING	FAB$B_SHR
WINDOW_SIZE	FAB$B_RTV
File Definition Properties	
ALLOCATION	FAB$L_ALQ
BUCKET_SIZE	FAB$B_BKS[6]
FILE_PROCESSING_OPTIONS	FAB$L_FOP
FOP	FAB$L_FOP
GLOBAL_BUFFER_COUNT	FAB$W_GBC
MAX_RECORD_NUMBER	FAB$L_MRN
MAX_RECORD_SIZE	FAB$W_MRS
MT_BLOCK_SIZE	FAB$W_BLS
ORGANIZATION	FAB$B_ORG
Key Properties	
ASCENDING	XAB$B_DTP

[5]The EXTENSION *area* property overrides the EXTENSION *file access block* property when you use both in the same definition.

[6]The BUCKET_SIZE *area* property overrides the BUCKET_SIZE *file definition* property when you use both in the same definition.

(continued on next page)

Table A–1 (Cont.) Mapping of Keywords to Symbolic Field Offsets

Keyword	Symbolic Field Offset
Key Properties	
CHANGES	XAB$V_CHG
DATA_AREA	XAB$B_DAN
DATA_FILL	XAB$W_DFL
DATA_KEY_COMPRESSION	XAB$V_KEY_NCMPR
DATA_RECORD_COMPRESSION	XAB$V_DAT_NCMPR
DESCENDING	XAB$B_DTP
DUPLICATES	XAB$V_DUP
INDEX_AREA	XAB$B_IAN
INDEX_COMPRESSION	XAB$V_IDX_NCMPR
INDEX_FILL	XAB$W_IFL
LEVEL1_INDEX_AREA	XAB$B_LAN
NODATA_KEY_COMPRESSION	XAB$V_KEY_NCMPR
NODATA_RECORD_COMPRESSION	XAB$V_DAT_NCMPR
NOINDEX_COMPRESSION	XAB$V_IDX_NCMPR
NULL_KEY	XAB$V_NUL
NULL_VALUE	XAB$B_NUL
PROLOG	XAB$B_PROLOG

Table A–2 shows the mapping of keywords to symbolic constants.

Table A–2 Mapping of Keywords to Symbolic Constants

Keyword	Symbolic Constant	Constant Type
Access Mode		
EXECUTIVE	PSL$C_EXEC	FAB$V_CHAN_MODE
NONE (default)	0	FAB$V_CHAN_MODE
SUPER	PSL$C_SUPER	FAB$V_CHAN_MODE
USER	PSL$C_USER	FAB$V_CHAN_MODE

(continued on next page)

Table A–2 (Cont.) Mapping of Keywords to Symbolic Constants

Keyword	Symbolic Constant	Constant Type
	File Access Control	
BLOCK_IO	FAB$V_BIO	FAB$B_FAC
DELETE	FAB$V_DEL	FAB$B_FAC
GET	FAB$V_GET	FAB$B_FAC
PUT	FAB$V_PUT	FAB$B_FAC
RECORD_IO	FAB$V_BRO	FAB$B_FAC
TRUNCATE	FAB$V_TRN	FAB$B_FAC
UPDATE	FAB$V_UPD	FAB$B_FAC
	File Organization	
INDEXED	FAB$C_INX	FAB$B_ORG
RELATIVE	FAB$C_REL	FAB$B_ORG
SEQUENTIAL (default)	FAB$C_SEQ	FAB$B_ORG
	Position Type	
CYLINDER	XAB$C_CYL	XAB$B_ALN
FILE_ID	XAB$C_RFI	XAB$B_ALN
LOGICAL	XAB$C_LBN	XAB$B_ALN
NONE (default)	XAB$C_ANY	XAB$B_ALN
VIRTUAL	XAB$C_VBN	XAB$B_ALN

(continued on next page)

Table A–2 (Cont.) Mapping of Keywords to Symbolic Constants

Keyword	Symbolic Constant	Constant Type
	Record Format	
FIXED	FAB$C_FIX	FAB$B_RFM
STREAM	FAB$C_STM	FAB$B_RFM
STREAM_CR	FAB$C_STMCR	FAB$B_RFM
STREAM_LF	FAB$C_STMLF	FAB$B_RFM
UNDEFINED	FAB$C_UDF	FAB$B_RFM
VARIABLE	FAB$C_VAR	FAB$B_RFM
VFC	FAB$C_VFC	FAB$B_RFM

Table A–3 shows the mapping of keywords to symbolic bit offsets.

Table A–3 Mapping of Keywords to Symbolic Bit Offsets

Keyword	Symbolic Bit Offset	Symbolic Bit Offset Type
	Carriage Control Options	
CARRIAGE_RETURN (default)	FAB$V_CR	FAB$B_RAT
FORTRAN	FAB$V_FTN	FAB$B_RAT
PRINT	FAB$V_PRN	FAB$B_RAT
	File Processing Options—Allocation and Extension	
BEST_TRY_CONTIGUOUS	FAB$V_CBT	RMS[1]
CONTIGUOUS	FAB$V_CTG	RMS[2]
TRUNCATE_ON_CLOSE	FAB$V_TEF	RMS
	File Processing Options—Disposition	
DELETE_ON_CLOSE	FAB$V_DLT	RMS

[1]The BEST_TRY_CONTIGUOUS *area* property overrides the BEST_TRY_CONTIGUOUS *file processing* property when you use both in the same definition.

[2]The CONTIGUOUS *area* property overrides the CONTIGUOUS *file processing* property when you use both in the same definition.

(continued on next page)

Table A–3 (Cont.) Mapping of Keywords to Symbolic Bit Offsets

Keyword	Symbolic Bit Offset	Symbolic Bit Offset Type
File Processing Options—Disposition		
PRINT_ON_CLOSE	FAB$V_SPL	RMS
SUBMIT_ON_CLOSE	FAB$V_SCF	RMS
TEMPORARY	FAB$V_TMD	RMS
NO_DIRECTORY_ENTRY	FAB$V_TMP	RMS
File Processing Options—Name Parsing Modifiers		
CREATE_IF	FAB$V_CIF	RMS
MAXIMIZE_VERSION	FAB$V_MXV	RMS
SUPERSEDE	FAB$V_SUP	RMS
File Processing Options—Magnetic Tape		
MT_NOT_EOF	FAB$V_NEF	RMS
MT_CURRENT_POSITION	FAB$V_POS	RMS
MT_CLOSE_REWIND	FAB$V_RWC	RMS
MT_OPEN_REWIND	FAB$V_RWO	RMS
File Processing Options—Nonstandard		
NON_FILE_STRUCTURED	FAB$V_NFS	RMS
USER_FILE_OPEN	FAB$V_UFO	RMS
File Processing Options—Performance		
DEFERRED_WRITE	FAB$V_DFW	RMS
SEQUENTIAL_ONLY	FAB$V_SQO	RMS
File Processing Options—Reliability		
READ_CHECK	FAB$V_RCK	RMS
WRITE_CHECK	FAB$V_WCK	RMS

(continued on next page)

Table A–3 (Cont.) Mapping of Keywords to Symbolic Bit Offsets

Keyword	Symbolic Bit Offset	Symbolic Bit Offset Type
	Share Options	
DELETE	FAB$V_SHRDEL	FAB$B_SHR
GET	FAB$V_SHRGET	FAB$B_SHR
MULTISTREAM	FAB$V_MSE	FAB$B_SHR
PROHIBIT	FAB$V_NIL	FAB$B_SHR
PUT	FAB$V_SHRPUT	FAB$B_SHR
UPDATE	FAB$V_SHRUPD	FAB$B_SHR
USER_INTERLOCK	FAB$V_UPI	FAB$B_SHR

Table A–4 and Table A–5 show the mapping of area properties keywords to symbolic field offsets and constants.

Table A–4 Mapping of CDO Area Properties to RMS Symbolic Field Offsets

CDO Property	RMS Symbolic Field Offset
ALLOCATE	XAB$L_ALQ
ANY_CYLINDER	XAB$V_ONC
BEST_TRY_CONTIGUOUS	XAB$V_CBT[1]
BUCKET_SIZE	XAB$B_BKZ[2]
CONTIGUOUS	XAB$V_CTG[3]
EXACT_POSITIONING	XAB$V_HRD
EXTENSION	XAB$W_DEQ[4]
POSITION	XAB$B_ALN
VOLUME	XAB$W_VOL

[1]The BEST_TRY_CONTIGUOUS *area* property overrides the BEST_TRY_CONTIGUOUS *field* property when you use both in the same definition.

[2]The BUCKET_SIZE *area* property overrides the BUCKET_SIZE *field* property when you use both in the same definition.

[3]The CONTIGUOUS *area* property overrides the CONTIGUOUS *field* property when you use both in the same definition.

[4]The EXTENSION *area* property overrides the EXTENSION *field* property when you use both in the same definition.

Table A–5 Mapping of CDO Position Type Options to XAB$B_ALN Symbolic Constants

CDO Option	Symbolic Constant
CYLINDER	XAB$C_CYL
FILE_ID	XAB$C_RFI
LOGICAL	XAB$C_LBN
NONE (default)	XAB$C_ANY
VIRTUAL	XAB$C_VBN

Table A–6 shows the mapping of key properties keywords to symbolic field offsets.

Table A–6 Mapping of CDO Key Properties to RMS Symbolic Field Offsets

CDO Property	RMS Symbolic Field Offset
ASCENDING	XAB$B_DTP
DESCENDING	XAB$B_DTP
DUPLICATES	XAB$V_DUP
CHANGES	XAB$V_CHG
NULL_KEY	XAB$V_NUL
NULL_VALUE	XAB$B_NUL
DATA_AREA	XAB$B_DAN
DATA_FILL	XAB$W_DFL
DATA_KEY_COMPRESSION	XAB$V_KEY_NCMPR
DATA_RECORD_COMPRESSION	XAB$V_DAT_NCMPR
INDEX_AREA	XAB$B_IAN
INDEX_COMPRESSION	XAB$V_IDX_NCMPR
INDEX_FILL	XAB$W_IFL
LEVEL1_INDEX_AREA	XAB$B_LAN
NODATA_KEY_COMPRESSION	XAB$V_KEY_NCMPR
NODATA_RECORD_COMPRESSION	XAB$V_DAT_NCMPR
NOINDEX_COMPRESSION	XAB$V_IDX_NCMPR
PROLOG	XAB$B_PROLOG

B

Repository Logical Names Table

Table B–1 describes the logical names that are associated with Oracle CDD/Repository.

Table B–1 Oracle CDD/Repository Logical Names

Logical Name	Mode	Table	Purpose
CDD$CHARACTER_SET	Executive	System	Specifies the character set used for Japanese input during a CDO session.
CDD$COMPATIBILITY	Executive	System	Specifies OpenVMS anchor directory for CDO repository portion of compatibility repository.
CDD$CONTEXT	User	Process or Job	Specifies the current context.

(continued on next page)

Table B–1 (Cont.) Oracle CDD/Repository Logical Names

Logical Name	Mode	Table	Purpose
CDD$DEBUG_FLAGS	User	Process or Job	Specifies values for setting of debugger instructions. Valid values are: B - Dump all buffers. C - Dump Oracle CDD/Repository-Oracle Rdb change buffers. D - Display a message when a deadlock occurs. I - Dump inheritance information. M - Display the names of all methods called by the method dispatcher. R - Dump reservation and replacement information. T - Write out an informational line indicating when a transaction is started and when it is committed or rolled back.
CDD$DEFAULT	User	Process or Job	Specifies the default repository directory.[1]
CDD$DICTIONARY	Executive	System	Specifies OpenVMS directory location of DMU root repository file. See the Oracle Dictionary Management Utility (DMU) Documentation Kit for more information on DMU.
CDD$EXTENSIONS	Executive	System	Specifies the location of the Oracle CDD/Repository extensions directory.

[1]If you enter a utility that deals with the repository, Oracle CDD/Repository automatically places you in the directory specified by the CDD$DEFAULT logical name. If you create a definition in a utility that deals with the repository and you do not specify a directory as part of the path name, Oracle CDD/Repository automatically places the definition in the directory specified by the CDD$DEFAULT logical name. Use the SET DEFAULT command in any utility that deals with the repository to change the default repository directory.

(continued on next page)

Table B–1 (Cont.) Oracle CDD/Repository Logical Names

Logical Name	Mode	Table	Purpose
CDD$MAX_OBJECTS_IN_MEMORY	User	Process or Job	Specifies the number of repository objects Oracle CDD/Repository caches in memory. The default is 10,000.
CDD$SMGSHR	Executive	System	Specifies the Japanese input method for the CDO interface when using the Japanese screen management library (SMG) with the Japanese OpenVMS operating system. The equivalence name is SYS$LIBRARY:JSY$SMGSHR.
CDD$SMG_CHARACTER_SET	Executive	System	Required to invoke Japanese SMG for CDO. If the character set input is Japanese Kanji, the equivalence name is KANJI; if the character set is the Japanese phonetic alphabet, Narrow Katakana, the equivalence name is SDK.
CDD$TEMPLATE	Executive	System	Specifies the OpenVMS directory for new CDO repositories.
CDD$TEMPLATEDB	Executive	System	Specifies the OpenVMS directory for new multifile databases.
CDD$TOP	User	Process or Job	Specifies the first part of a path name. If you work in a directory many levels down, you can set CDD$TOP a repository directory that CDD/Repository treats as the topmost directory of interest.[2]
CDD$VERSION_LIMIT	User	Process	Specifies maximum number of versions of a data entity in a DMU repository.

[2]If you enter a utility that deals with the repository, Oracle CDD/Repository automatically places you in the directory specified by the CDD$DEFAULT logical name. If you create a definition in a utility that deals with the repository and you do not specify a directory as part of the path name, Oracle CDD/Repository automatically places the definition in the directory specified by CDD$DEFAULT. Use the SET DEFAULT command in any utility that deals with the repository to change the default repository directory.

(continued on next page)

Table B–1 (Cont.) Oracle CDD/Repository Logical Names

Logical Name	Mode	Table	Purpose
CDD$WAIT	User	Process	Specifies whether a command will wait until the DMU repository is unlocked from another user's operation.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S