VAX Common Data Dictionary Data Definition Language Reference Manual

Order Number: AA–K085D–TE

August 1988

This manual describes the VAX Common Data Dictionary Data Definition Language Utility (CDDL) and the elements of a CDDL source file.

digital equipment corporation, maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

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

The following are trademarks of Digital Equipment Corporation:

ACMS
CDD
DATATRIEVE
DEC
DECnet
DECreporter
DECUS
MicroVAX
MicroVMS
PDP
RALLY
Rdb/ELN
Rdb/VMS
ReGIS
TDMS
TEAMDATA
UNIBUS
VAX
VAXcluster
VAXinfo
VAX Information Architecture
VIDA
VMS
VT

digital™

Contents

How to Use This Manual

This manual describes how the VAX Common Data Dictionary software, also referred to in this document as CDD, allows you to enter record definitions directly into the dictionary, using the Data Definition Language Utility (CDDL).

CDD is now a subset of the VAX CDD/Plus software, also referred to in this document as CDD/Plus. Many DIGITAL products, however, continue to function using Version 3.4 or earlier of CDD.

Version 3.4 and earlier of CDD use the DMU format for dictionary definitions, but not the CDO format. This manual is for dictionary users who need to use the DMU format for dictionary definitions.

If Version 3.4 or earlier of the VAX Common Data Dictionary is installed on your system, references in this manual to the "VAX Common Data Dictionary," "Common Data Dictionary," or "CDD" refer to the VAX Common Data Dictionary installed on your system.

If VAX CDD/Plus Version 4.0 or later is installed on your system, references in this manual to the "VAX Common Data Dictionary," "Common Data Dictionary," or "CDD" refer to the DMU format dictionary.

CDD/Plus supports dictionary definitions in two distinct formats:

- DMU **format** includes dictionary definitions that can be created and manipulated with the DMU, CDDL, and CDDV utilities, and other products that do not support the new features of CDD/Plus.
- CDO **format** includes dictionary definitions that can be created and manipulated with the CDO utility, the CDD/Plus call interface, and other supporting products.

Intended Audience

The audience for this manual includes:

- The data administrator or system manager responsible for organizing the directory hierarchy and creating the record definitions to be stored in the CDD
- Programming supervisors responsible for maintaining portions of the hierarchy and the data definitions stored there
- Programmers responsible for storing new data definitions in the CDD

Before you read this manual, you should read Appendix A of the *VAX CDD/Plus User's Guide.*

Operating System Information

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

This manual consists of three chapters, five appendixes, and an index.

Chapter 1	Provides a brief overview of the hierarchical structure of the VAX Common Data Dictionary and introduces the use of the Data Definition Language Utility (CDDL).
Chapter 2	Presents complete descriptions of the elements of a CDDL source file.
Chapter 3	Presents a complete description of the CDDL compiler commands, including parameters and qualifiers.
Appendix A	Contains SOURCE.DDL, the source file for all the record definitions used as examples in this manual.
Appendix B	Contains a CDDL syntax skeleton.
Appendix C	Documents compiler error messages.
Appendix D	Documents the use of CDDL reserved words.
Appendix E	Contains changes made with Version 3.4 of CDD that were not described earlier.

Related Documents

For up-to-date references to further information on the topics covered in this manual, see the prefaces of the VAX CDD/Plus manuals in this documentation set.

Conventions

This section explains the conventions for the syntax and symbols used in this manual:

WORD	An uppercase word in a syntax format is a keyword. You must include it in the statement if the clause is used.
word	A lowercase word in a syntax format indicates a syntax element that you supply.
[ ]	Square brackets enclose optional clauses from which you can choose one or none.
{ }	Braces enclose clauses from which you must choose one alternative.
[| |]	Bars in square brackets indicate that you can choose any combination of the enclosed options, but you can use each option only once.
<RET>	This symbol indicates the RETURN key. Unless otherwise indicated, end all user input lines in examples by pressing the RETURN key.
<CTRL/x>	This symbol tells you to press the CTRL (control) key and hold it down while pressing a letter key.
<GOLD-x>	This symbol indicates that you press the GOLD key and then a specified letter key consecutively.
. . .	A horizontal ellipsis means you can repeat the previous item.
. : :	A vertical ellipsis in an example means that information not directly related to the example has been omitted.
Color	Color in examples shows user input.

References to Products

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

VAX Information Architecture documentation explaining how these products interrelate is included with VAX CDD/Plus documentation. VAX Information Architecture documentation is also available separately. Contact your DIGITAL representative.

The CDD documentation to which this manual belongs often refers to products that are part of the VAX Information Architecture by their abbreviated names:

- VAX ACMS software is referred to as ACMS.
- VAX CDD/Plus software is referred to as CDD/Plus.
- VAX CDD software is referred to as CDD.
- VAX DATATRIEVE software is referred to as DATATRIEVE.
- VAX DBMS software is referred to as VAX DBMS.
- VAX Rdb/VMS software is referred to as Rdb/VMS.
- VAX TDMS software is referred to as TDMS.
- VIDA software is referred to as VIDA.

The VAX Common Data Dictionary 1

A typical information management system today includes a combination of languages and language processors using the same data files and record definitions to perform different tasks. With information shared by various users, there is an obvious need to guarantee the accuracy and reliability of the data. Database management systems meet this need by providing central storage of and control over an organization's data.

The VAX Common Data Dictionary (CDD) performs a similar function, not for data, but for data definitions. The CDD is a central repository for data descriptions and definitions shared by VAX languages and by VAX information management processors.

1.1 The CDD Directory Hierarchy

With the CDD, you use a hierarchical directory structure to organize and arrange your data definitions. You collect related data definitions in dictionary directories in much the same way you use VAX/VMS directories to collect related files. Dictionary directories can own other directories, or they can own dictionary objects, which are the data descriptions stored in the dictionary. There are several types of CDD dictionary objects including VAX DBMS sets and realms, VAX DATATRIEVE domains and procedures, and VAX CDD record definitions.

The CDD also allows you to create special directories called subdictionary directories. Subdictionary directories function exactly like dictionary directories, but they are stored in separate physical files for security or accounting reasons. In practice, users are unaware of whether they are accessing dictionary directories or subdictionaries.

Figure 1-1 shows the hierarchical relationships in the sample dictionary that is the source of all the examples in this manual.

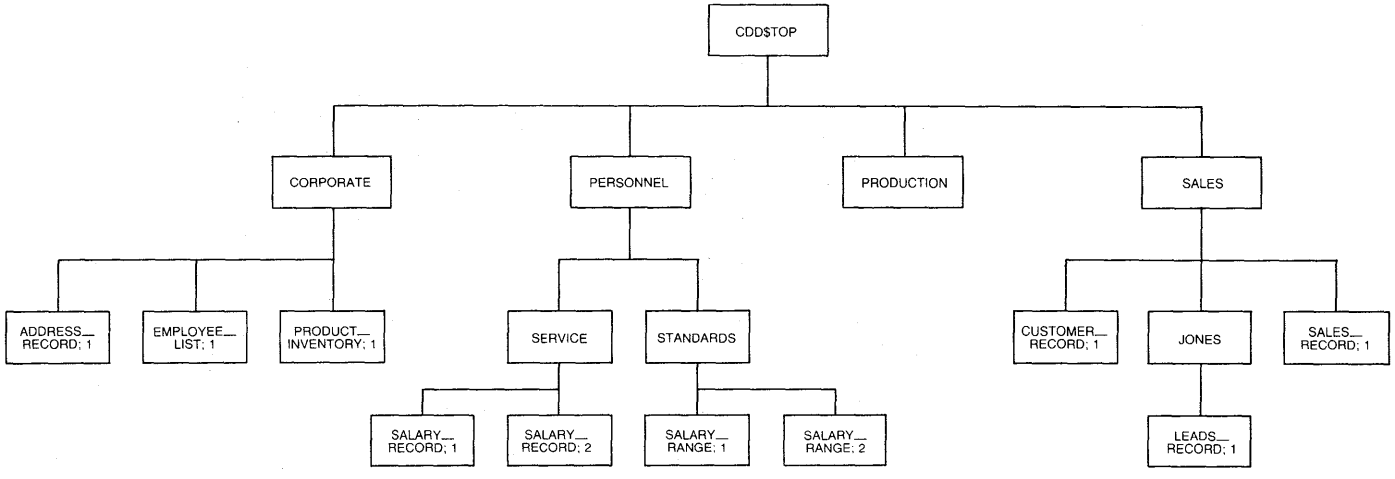


Figure 1-1: Sample Dictionary Hierarchy

As you can see from the figure, one dictionary directory, CDD$TOP, owns the entire dictionary structure. CDD$TOP is the permanently assigned name for this **root dictionary directory**. Below CDD$TOP are the directories CORPORATE, PRODUCTION, and SALES, and the PERSONNEL subdictionary. These dictionary and subdictionary directories organize the information stored in the CDD. You can use terms borrowed from family tree relationships to describe CDD hierarchical relationships:

- PRODUCTION has no children and no descendants.
- The record definitions ADDRESS_RECORD;1, EMPLOYEE_LIST;1, and PRODUCT_INVENTORY;1 are the three children of CORPORATE.
- PERSONNEL's two children are directories: SERVICE and STANDARDS. Each of these directories is the parent of two record definitions. SALARY_RECORD;1 and SALARY_RECORD;2 are children of SERVICE, and SALARY_RANGE;1 and SALARY_RANGE;2 are the children of STANDARDS.
- SALES is the parent of both a dictionary directory, JONES, and of two dictionary objects, CUSTOMER_RECORD;1 and SALES_RECORD;1. JONES, in turn, is the parent of LEADS_RECORD;1.
- LEADS_RECORD;1 is a descendant of SALES, and SALARY_RECORD;1 is a descendant of PERSONNEL.
- PERSONNEL is an ancestor of SALARY_RECORD;1, SALARY_RECORD;2, SALARY_RANGE;1, and SALARY_RANGE;2.

1.1.1 Full Path Names

To refer to a particular dictionary directory or object within the directory hierarchy, you use its dictionary path name. A full path name is the string of given names connecting CDD$TOP to the given name of the target dictionary directory or object. You separate given names from one another with periods, just as you do with VMS directories. The dictionary directory JONES, for example, has the following full path name:

```
CDD$TOP.SALES.JONES
```

Because the CDD can contain multiple versions of dictionary objects, each dictionary object has a version number associated with it. The version number is separated from the name of the object by a semicolon. For example, the full path name of version 2 of SALARY_RANGE is:

```
CDD$TOP.PERSONNEL.STANDARDS.SALARY_RANGE;2
```

You can specify the highest version of a dictionary object with an absolute version number or with no version number at all. For example, either of the following path names specifies the highest existing version of SALARY_RANGE, version 2:

```
CDD$TOP.PERSONNEL.STANDARDS.SALARY_RANGE
CDD$TOP.PERSONNEL.STANDARDS.SALARY_RANGE;2
```

1.1.2 Relative Path Names

If you define the logical name CDD$DEFAULT as the path name of a particular dictionary directory, that directory becomes your default dictionary directory. When you have defined a default directory, you can identify a dictionary directory or object by its relative path name, which includes only the names of those directories connecting the default directory to the target directory or object.

If, for example, your default dictionary directory were CDD$TOP.PERSONNEL, you could identify the dictionary object SALARY_RECORD;1 with either of the following path names:

```
CDD$TOP.PERSONNEL.SERVICE.SALARY_RECORD;1
SERVICE.SALARY_RECORD;1
```

1.2 History Lists

The CDD's optional history list feature allows you to document any operation on a dictionary directory or object. You can store documentary comments in the history list of the directory or object. You can use history lists to monitor CDD processing and to plan additions or changes to the dictionary hierarchy.

1.3 Access Control Lists

The CDD controls the security of the dictionary with access control lists associated with each dictionary directory or object. You can grant or deny nine CDD and four VAX DATATRIEVE access privileges to individual users or groups of users with CDD access control lists.

1.4 The Data Definition Language Utility (CDDL)

You can use the Data Definition Language Utility (CDDL) to define records and store them in the CDD. VAX programming languages and VAX Information Architecture products can then share these record definitions. Each language or product translates the generic definitions stored in the CDD into language- or product-specific definitions that it can use.

To enter a new data definition, or to modify an existing definition in the CDD, you first create a CDDL **source file**. This source file contains full data descriptions including field names, data types, and special attributes. Once the source file is complete, you use the CDDL compiler, either interactively or in batch mode, to insert the definitions into the CDD.

To use the CDDL compiler, you should define CDDL as a global symbol in your login command file:

```
$ CDDL:==$SYS$SYSTEM:CDDL
```

Chapter 2 of this manual describes CDDL source file statements and clauses. Chapter 3 describes CDDL compiler commands.

CDDL Source File Description 2

You create a CDDL source file by using a text editor, like VAX EDT. This chapter describes the statements and clauses you use in a CDDL source file. When you have created the source file, you can store it in the CDD by using the CDDL compiler. Chapter 3 describes CDDL compiler commands.

CDDL source files consist of:

- DEFINE and END statements, which you use to define CDD records
- Field description statements, which you use to describe the general structure of records
- Field attribute clauses, which you use to describe characteristics of fields within CDD records

Use the following syntax guidelines to help create CDDL source files:

- Each line of CDDL source can be no longer than 255 characters.
- End each field description with a period.
- You can continue field attribute clauses from one line to the next without using a continuation character.
- Within each field attribute clause, the order of keywords is important. Refer to the format diagrams for proper keyword order of specific clauses.
- Within each field description, you can specify field attribute classes in any order you please.
- In elementary field descriptions, the DATATYPE clause is the only required field attribute clause.

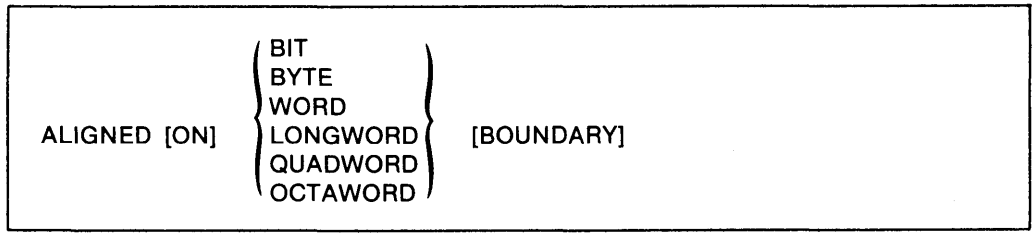
The following sections contain descriptions of the statements and clauses you use to define records in the CDDL source file. These descriptions include:

- The syntax **format** you should use for each statement or clause
- **Syntax rules** to help you define records correctly
- **Usage notes** to show you how to use each statement or clause
- **Examples** to illustrate the use of each statement or clause

2.1 ALIGNED Field Attribute Clause

Aligns a field on a specified starting boundary relative to the start of the record.

Format



Usage Notes

- To satisfy hardware requirements, some languages and language processors have field alignment restrictions for data definitions. The ALIGNED clause enables you to control the starting boundaries of fields you define.
- Each field, 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. You can modify this starting position with the ALIGNED clause.
- The ALIGNED clause aligns fields 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 the CDD.

Note

You should not use the ALIGNED clause in template records. When CDDL stores the template record, the position of an aligned field is fixed within the record and is not changed when the record is copied into another record definition. Therefore, the newly created field may not align properly in the new record definition.

Example

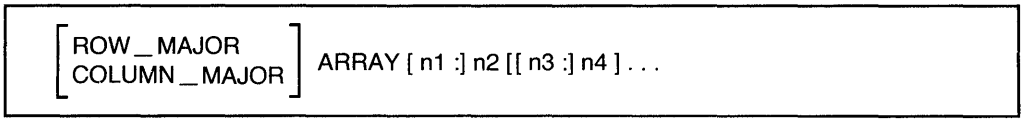
In the following example, a LONGWORD field (QUANTITY) follows a BYTE field. The PRODUCT_NO field spans 64 bits, the DATE_ORDERED field spans 64 bits, and the STATUS_CODE field spans 8 bits. The ALIGNED clause causes the three bytes following STATUS_CODE to remain empty and aligns QUANTITY exactly 160 bits beyond the start of the record.

```
IN_STOCK STRUCTURE.
    PRODUCT_NO      DATATYPE IS TEXT
                    SIZE IS 8 CHARACTERS.
    DATE_ORDERED    DATATYPE IS DATE.
    STATUS_CODE     DATATYPE IS BYTE.
    QUANTITY        DATATYPE IS LONGWORD
                    ALIGNED ON LONGWORD.
    LOCATION        ARRAY 1:4
                    DATATYPE IS TEXT
                    SIZE IS 30 CHARACTERS.
    UNIT_PRICE      DATATYPE IS LONGWORD SCALE -2.
END IN_STOCK STRUCTURE.
```

2.2 ARRAY Field Attribute Clause

Declares multidimensional arrays or one-dimensional arrays. ARRAY is most useful when the subscript's lower bound is not equal to 1.

Format



Parameters

n1, n2, n3, n4

The upper and lower bounds of the subscripts.

Syntax Rules

- Each dimension of the array is defined by a pair of *signed* integers.
- The first integer in each pair (n1, n3 . . .) specifies the subscript's lower bound. The default value for the lower bound is 1.
- The second integer in each pair (n2, n4 . . .) specifies the subscript's upper bound. The upper bound must be greater than or equal to the lower bound.

Usage Notes

- With ARRAY, each subscript has an upper and lower bound defined by a pair of signed integers.
- 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 neither ROW_MAJOR nor COLUMN_MAJOR is specified, the default is ROW_MAJOR.

Example

In the following example, the ARRAY clause declares 20 instances of SUPPLIER (from 0 to 19) where each instance is four 30-character strings.

```
SUPPLIER                        ARRAY 0:19 1:4
                                DATATYPE IS TEXT
                                SIZE IS 30 CHARACTERS.
```

2.3 BLANK WHEN ZERO Field Attribute Clause

Sets an entire field to blanks when you assign it a zero.

Format

```
BLANK WHEN ZERO
```

Usage Note

Only VAX COBOL supports this feature. All other processors ignore it.

Example

In the following example, the field NEW is initially set to blanks when it is accessed by the VAX COBOL compiler.

```
ZIP_CODE STRUCTURE.
    NEW         DATATYPE IS UNSIGNED NUMERIC
                SIZE IS 4 DIGITS
                BLANK WHEN ZERO.
    OLD         DATATYPE IS UNSIGNED NUMERIC
                SIZE IS FIVE DIGITS.
END ZIP_CODE STRUCTURE.
```

2.4 COMPUTED BY DATATRIEVE Field Attribute Clause

Supplies expressions used by VAX DATATRIEVE to calculate the values of VIRTUAL fields.

Format

```
COMPUTED BY { DTR       } AS quoted-string [quoted-string] . . .
            { DATATRIEVE }
```

Parameter

quoted-string

An expression used by VAX DATATRIEVE to calculate the field's value.

Syntax Rule

The set of quoted strings must form a valid VAX DATATRIEVE virtual expression. The CDDL compiler does not check the virtual expression for correct syntax. It is your responsibility to provide a correct virtual expression.

Usage Notes

- You must specify a VIRTUAL FIELD DATATYPE for a field using the COMPUTED BY DATATRIEVE clause.
- Only VAX DATATRIEVE can interpret VIRTUAL fields. Other processors either ignore their presence or refuse to process record descriptions containing them.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.
- You cannot specify an INITIAL VALUE field attribute for a virtual field.
- You cannot specify a CONDITION NAME field attribute for a virtual field.
- You cannot use a virtual field as a tag variable in the VARIANTS OF field description statement.

Example

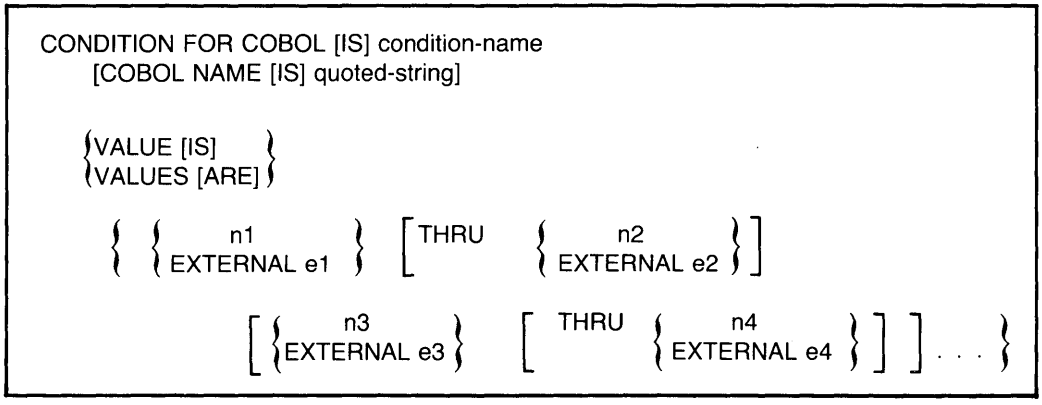
In the following example, no storage has been allocated for TOTAL_PRICE. Instead, VAX DATATRIEVE uses the virtual expression "UNIT_PRICE QUANTITY" to calculate the value of TOTAL_PRICE at run time.

```
TOTAL_PRICE             DATATYPE IS VIRTUAL FIELD
                        COMPUTED BY DATATRIEVE AS
                            "UNIT_PRICE * QUANTITY".
```

2.5 CONDITION NAME Field Attribute Clause

Enables the VAX COBOL compiler to associate one or more condition names with specific values for a field.

Format



Parameters

condition-name

The condition.

quoted-string

A COBOL name for the condition.

n1, n2, n3, n4

Field values or ranges of field values associated with the condition name.

e1, e2, e3, e4

A quoted string containing a COBOL external name. See the *VAX COBOL Language Reference Manual* for information on the legal use of external names.

Syntax Rules

- The condition name must be a string of up to 31 characters from the set A-Z, 0-9, _ , and $. The first character in the string must be a letter from A-Z, and the last character cannot be _ or $.

- Quoted strings or external names must be legal VAX COBOL names. However, the CDDL compiler does not check for correct syntax. It is your responsibility to provide a correct value expression.
- The values n1, n2, n3, and n4 can be fixed point numbers, floating point numbers, quoted strings, octal numbers, or hexadecimal numbers.
- The compiler ignores commas, but you can use them to make value specifications easier to read.

Usage Notes

- Only VAX COBOL supports this feature. Other language processors ignore the CONDITION NAME clause.
- Each CONDITION NAME clause defines one condition name. Each condition name can represent a discrete value, a range of values, or any combination of these.
- You can use the CONDITION NAME clause as many times as you wish within a field description.
- The values n1, n2, n3, and n4 must be legal values as defined by the data type declared for the field.
- The length of a literal you specify in a CONDITION NAME clause cannot exceed the length declared for the field.
- You can specify a fixed point number as the value of any field whose valid VAX COBOL data type is not DATE, TEXT, or UNSPECIFIED.
- You can specify a floating point number as the value of a field whose valid VAX COBOL data type is not DATE, TEXT, or UNSPECIFIED.
- You can specify a quoted string as the value only of a field whose valid VAX COBOL data type is DATE, TEXT, or UNSPECIFIED.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.
- You can specify an octal number as the value of any valid VAX COBOL data type. In order to specify an octal number, place single quotation marks (') around the number and precede it with %O; for example,

```
VALUE IS %O'16'
```

- You can specify a hexadecimal number as the value of any valid VAX COBOL data type. In order to specify a hexadecimal number, place single quotation marks (') around it and precede it with %X; for example,

```
VALUE IS %X'3E'
```

- Value n2 must be greater than or equal to value n1 in the field's collating sequence.
- Value n4 must be greater than or equal to value n3 in the field's collating sequence.

Example

The following example defines three valid conditions and the values where the condition is invalid, according to the value of the field RECORD_IDENTIFIER.

```
RECORD_IDENTIFIER       DATATYPE IS TEXT
                        SIZE IS 1 CHARACTER
                        CONDITION FOR COBOL IS ON_HAND
                            COBOL NAME "ON-HAND"
                            VALUE IS "S"
                        CONDITION FOR COBOL BACKORDER
                            COBOL NAME "BACKORDER"
                            VALUE IS "B"
                        CONDITION FOR COBOL OUT_OF_STOCK
                            COBOL NAME "OUT-OF-STOCK"
                            VALUE IS "O"
                        CONDITION FOR COBOL IS INVALID
                            VALUES ARE "A", "C" THRU "N",
                            "P" THRU "R", "T" THRU "Z".
```

2.6 COPY Field Description Statement

Copies the description of an existing record (called a template record) into the description of a new field (the COPY field).

Format

```
[ /* text */ ]
field-name        COPY [ FROM ] path-name [ ALIGNED clause ].
```

Parameters

text

An explanation of the field.

field-name

The name of the COPY field.

path-name

The location of the template record definition within the CDD directory hierarchy.

Syntax Rules

- You must assign a field name to a COPY field. This field name can be up to 31 characters from the set A-Z, 0-9, _, and $. The first character must be a letter from A-Z, and the last character cannot be _ or $. If you are using a terminal of the VT200 family, you can use alphabetic 8-bit characters in field names. Remember that other terminals cannot reproduce 8-bit characters.
- The path name can be a full or a relative path name, and it must be the path name of an existing CDD record description. You can specify an absolute version number with the path name.
- The COPY field description statement must terminate with a period.

Usage Notes

- The copy operation takes place when the new record is compiled, not when it is copied into a program. When you modify a template record, you should recompile any record definitions that contain COPY field descriptions copying the modified template record.
- The COPY field description statement copies a complete record, a template record, into a single field.
- The COPY field description statement copies the description of the template record as the description of the COPY field. If the first subordinate field of the template record is a BIT field, the first subordinate field of the COPY field begins on the first bit immediately following the preceding field. Otherwise, the COPY field begins on the first byte immediately following the preceding field. You can modify this starting position with the ALIGNED clause (see Section 2.1).
- If you specify an absolute version number in the path name parameter, CDDL copies the version of the template record with that version number each time you compile the record description containing the COPY field. If you do not specify a version number, CDDL copies the highest version of the template record each time you compile the record description.
- In the COPY field, the field name you assign replaces the field name copied from the field description statement of the template record.
- When the CDDL compiler compiles a record definition containing a COPY field description, it automatically makes an entry documenting the copy operation in the history list of the template record, whether or not you use the /AUDIT qualifier.

Example

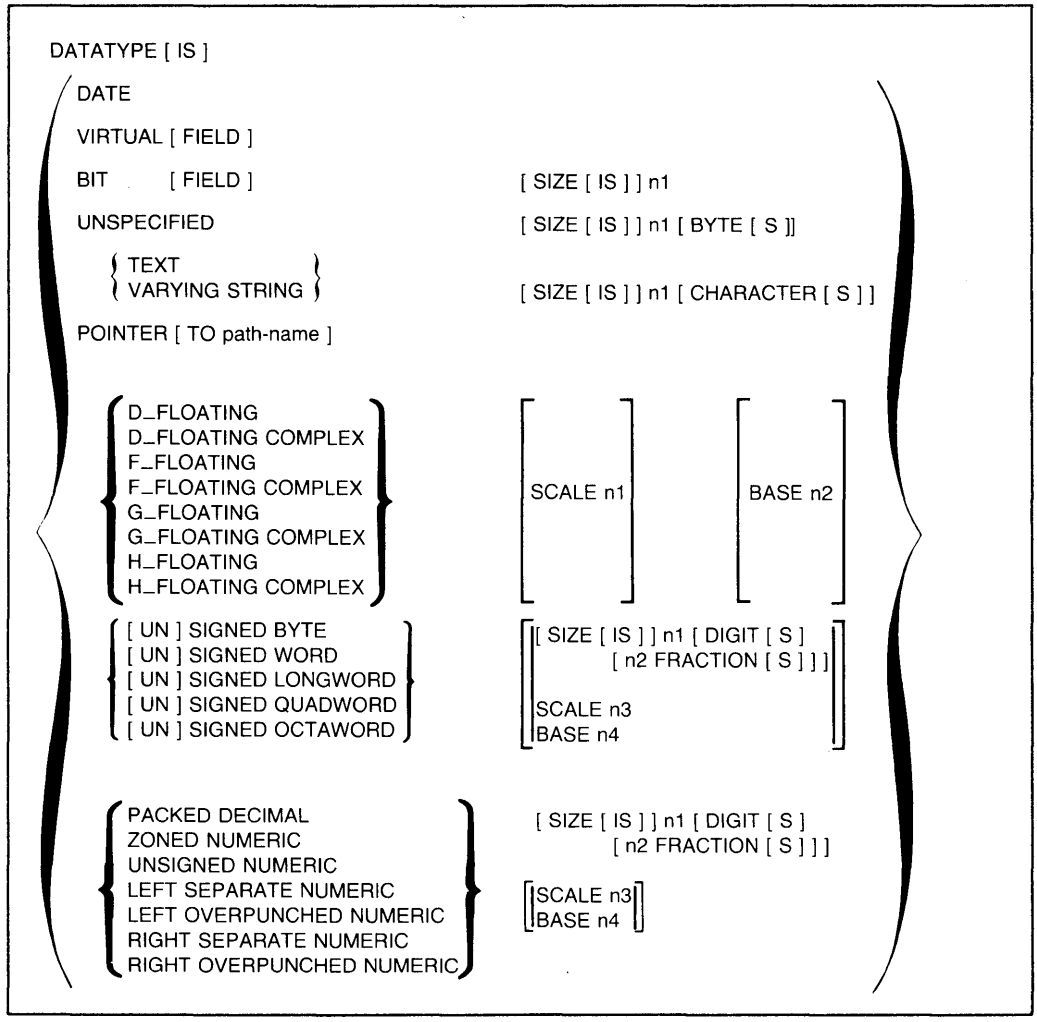
CDD$TOP.CORPORATE.ADDRESS_RECORD defines the standard format for addresses. You can copy this field description into any new record requiring an address field.

```
ADDRESS                     COPY FROM
                            CDD$TOP.CORPORATE.ADDRESS_RECORD.
```

2.7 DATATYPE Field Attribute Clause

Declares the type and size of a field. Some valid CDD data types may not be supported by the languages or language processors you will be using with the CDD. It is your responsibility to make certain that the records you define are valid for the language processors you use.

Format



Syntax Rules

- DATE specifies that the field is a 64-bit VAX standard absolute date data type.
- VIRTUAL FIELD specifies that the field is a VAX DATATRIEVE virtual field. No space is allocated for virtual fields in a record. The COMPUTED BY DATATRIEVE clause determines the value of a virtual field at run time. A STRUCTURE field cannot contain the VIRTUAL FIELD datatype. See Section 2.4.
- BIT specifies that the field is a bit string. Indicate the number of bits in the field with an unsigned integer (n1).
- UNSPECIFIED declares that the field is a sequence of 8-bit unsigned bytes. Indicate the number of bytes in the field with an unsigned integer (n1).
- TEXT specifies that the field is a sequence of 8-bit ASCII bytes. Indicate the number of characters in the field with an unsigned integer (n1). CDDL accepts CHARACTER as a synonym for TEXT.
- VARYING STRING specifies that the field is a PL/I or PASCAL varying string. Indicate the number of characters in the field with an unsigned integer (n1). CDDL accepts VARYING TEXT as a synonym for VARYING STRING.
- POINTER specifies that the field contains the address of another field or record definition. In PL/I, for example, POINTER fields are used 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 path-name] lets you connect a POINTER to a structure.
- Floating point data types:
  - You can specify a SCALE as an implied exponent. The signed integer (n1) must be in the range -128 to 127. The SCALE specification indicates the number of places to shift the decimal point when the field is evaluated. Negative n1 indicates a shift of n1 places to the left, and positive n1 indicates a shift of n1 places to the right.
  - You can also specify the radix, or BASE, with an unsigned integer (n2). The BASE indicates the number system to be used when the field is evaluated. The default BASE is 10.
  - D_FLOATING specifies that the field is a 64-bit floating point number with precision to approximately 16 decimal digits.

- D_FLOATING COMPLEX specifies that the field consists of two 64-bit floating complex numbers, one for the real component and one for the imaginary. CDDL accepts D_FLOATING_COMPLEX as a synonym for D_FLOATING COMPLEX.
- F_FLOATING specifies that the field is a 32-bit floating point number with precision to approximately seven decimal digits.
- F_FLOATING COMPLEX specifies that the field consists of two 32-bit floating complex numbers, one for the real component and one for the imaginary. CDDL accepts FLOATING_COMPLEX, FLOATING COMPLEX, and F_FLOATING_COMPLEX as synonyms for F_FLOATING COMPLEX.
- G_FLOATING specifies that the field is an extended range 64-bit floating point number with precision to approximately 15 decimal digits.
- G_FLOATING COMPLEX specifies that the field consists of two extended range 64-bit floating complex numbers, one for the real component and one for the imaginary. CDDL accepts G_FLOATING_COMPLEX as a synonym for G_FLOATING COMPLEX.
- H_FLOATING specifies that the field is an extended range 128-bit floating point number with precision to approximately 33 decimal digits.
- H_FLOATING COMPLEX specifies that the field consists of two extended range 128-bit floating complex numbers, one for the real component and one for the imaginary. CDDL accepts H_FLOATING_COMPLEX as a synonym for H_FLOATING COMPLEX.

• Fixed point data types:

- You can declare the total number of DIGITS (n1) and the number of those digits that are FRACTIONS (n2). The number of digits must be greater than 0 and less than 32. The number of fractions must not be greater than the number of digits. The default number of fractions is 0.
- You can specify a SCALE as an implied exponent. The signed integer (n1) must be in the range -128 to 127. The SCALE specification indicates the number of places to shift the decimal point when the field is evaluated. Negative n1 indicates a shift of n1 places to the left, and positive n1 indicates a shift of n1 places to the right.

- You can also specify the radix, or BASE, with an unsigned integer (n2). The BASE indicates the number system to be used when the field is evaluated. The default BASE is 10.
- BYTE specifies that the field is an 8-bit byte. The BYTE can be SIGNED or UNSIGNED. If there is no sign specification, UNSIGNED is the default.
- WORD specifies that the field is a 16-bit word. The field can be SIGNED or UNSIGNED. If no sign is specified, UNSIGNED is the default.
- LONGWORD specifies that the field is a 32-bit longword. The LONGWORD can be SIGNED or UNSIGNED. If no sign is specified, UNSIGNED is the default.
- QUADWORD specifies that the field is a 64-bit quadword field. The field can be SIGNED or UNSIGNED. If no sign is specified, UNSIGNED is the default.
- OCTAWORD specifies that the field is a 128-bit octaword field. The field can be SIGNED or UNSIGNED. If no sign is specified, UNSIGNED is the default.

• Decimal string data types:

- You must declare the total number of DIGITS (n1) in the field. You can also declare which of those digits are FRACTIONS (n2). The number of digits must be greater than 0 and less than 32. The number of fractions must not be greater than the number of digits. The default number of fractions is 0.
- You can specify a SCALE as an implied exponent. The signed integer (n1) must be in the range -128 to 127. The SCALE specification indicates the number of places to shift the decimal point when the field is evaluated. Negative n1 indicates a shift of n1 places to the left, and positive n1 indicates a shift of n1 places to the right.
- You can also specify the radix, or BASE, with an unsigned integer (n2). The BASE indicates the number system to be used when the field is evaluated. The default BASE is 10.
- PACKED DECIMAL specifies that the field is a packed decimal numeric field. CDDL accepts PACKED NUMERIC as a synonym for PACKED DECIMAL.
- UNSIGNED NUMERIC specifies that the field is an unsigned decimal string. You must use the UNSIGNED keyword.

- ZONED NUMERIC specifies that the field is a VAX ZONED NUMERIC data type. CDDL accepts SIGNED NUMERIC as a synonym for ZONED NUMERIC.
- LEFT SEPARATE NUMERIC specifies that the field is a left separate signed numeric decimal string. CDDL accepts [SIGNED] NUMERIC LEFT SEPARATE as a synonym for LEFT SEPARATE NUMERIC.
- LEFT OVERPUNCHED NUMERIC specifies that the field is a left overpunched signed numeric decimal string. CDDL accepts [SIGNED] NUMERIC LEFT OVERPUNCHED as a synonym for LEFT OVERPUNCHED NUMERIC.
- RIGHT SEPARATE NUMERIC specifies that the field is a right separate signed numeric decimal string. CDDL accepts [SIGNED] NUMERIC RIGHT SEPARATE as a synonym for RIGHT SEPARATE NUMERIC.
- RIGHT OVERPUNCHED NUMERIC specifies that the field is a right overpunched signed numeric decimal string. CDDL accepts [SIGNED] NUMERIC RIGHT OVERPUNCHED as a synonym for RIGHT OVERPUNCHED NUMERIC.

- CDDL accepts TYPE as a synonym for DATATYPE, but the compiler issues a warning when you use TYPE.

Usage Notes

- The SCALE specification and the FRACTIONS specification have a similar function. They both indicate how many digits in a field are to the right of the decimal point. There are, however, two important differences:
  - The number of FRACTIONS you can specify is limited to the number of DIGITS you declare for the data type. With SCALE, there is no such limitation.
  - With SCALE, you can shift the decimal point to the right and to the left, but with FRACTIONS, you can shift the decimal point only to the left.
- Use a positive integer in the SCALE specification to move the decimal point to the right.
- Use a negative integer in the SCALE specification to move the decimal point to the left.

Example

The STRUCTURE field BACK_ORDER contains examples of valid CDDL data declarations.

```
BACK_ORDER STRUCTURE.
    PRODUCT_NO      DATATYPE IS TEXT
                    SIZE IS 8 CHARACTERS.
    DATE_ORDERED    DATATYPE IS DATE.
    STATUS_CODE     DATATYPE IS BYTE.
    QUANTITY        DATATYPE IS LONGWORD
                    ALIGNED ON LONGWORD.
    SUPPLIER        ARRAY 1:4
                    DATATYPE IS TEXT
                    SIZE IS 30 CHARACTERS.
    UNIT_PRICE      DATATYPE IS LONGWORD SCALE -2.
END BACK_ORDER STRUCTURE.
```

2.8 DEFAULT_VALUE Field Attribute Clause

Sets a default value for a field accessed by VAX DATATRIEVE.

Format

```
DEFAULT_VALUE FOR { DTR       } [ IS ] { fixed-point-number }
                  { DATATRIEVE }       { quoted-string      }
```

Parameters

fixed-point-number
quoted-string

A VAX DATATRIEVE expression that is the default value.

Syntax Rules

- The quoted string or fixed point number must be a valid VAX DATATRIEVE expression. However, the CDDL compiler does not check for correct syntax. It is your responsibility to provide a correct default value expression.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

Usage Notes

- The DEFAULT_VALUE clause, used with VAX DATATRIEVE, serves much the same purpose as the INITIAL_VALUE clause. The main difference between the two is that the INITIAL_VALUE clause assigns a value to a field when the field is first allocated, usually at compile time, but the DEFAULT_VALUE clause assigns a value to a field each time the record is stored without an explicit value assigned to it.
- Only VAX DATATRIEVE supports the DEFAULT_VALUE clause. Other language processors ignore it.

Example

The following example assigns a DEFAULT_VALUE of 0 to the field TOTAL_PRICE.

```
TOTAL_PRICE                 DATATYPE IS VIRTUAL FIELD
                            COMPUTED BY DATATRIEVE AS
                                "UNIT_PRICE * QUANTITY"
                            DEFAULT_VALUE FOR DATATRIEVE IS 0.
```

2.9 DEFINE and END Statements

The DEFINE statement begins each record definition in the source file. The END statement terminates each record definition.

Format

```
DEFINE RECORD path-name

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

    field-description-statement

END { path-name  } [ RECORD ].
    { given-name }
```

Parameters

path-name

The location of the new record definition within the directory hierarchy.

text

Explanatory text describing the record definition.

field-description-statement

The field description for the entire record. This record field can be an elementary, STRUCTURE, COPY, or VARIANTS field.

given-name

The new record definition's name.

Syntax Rules

- You must terminate the DEFINE statement and the END statement with periods.
- The path name can be a full or a relative path specification. The last given name in the path name is the name you assign to the new record definition. You can specify an absolute version number with the path name.

- The given name of the new record definition is a string of up to 31 characters from the set A-Z, 0-9, _, and $. The first character must be a letter from A-Z, and the last character must not be _ or $. If you are using a terminal of the VT200 family, you can use 8-bit alphabetic characters in path names. Remember that other terminals cannot reproduce 8-bit characters.
- Use the optional DESCRIPTION clause to include text in the CDD to document the record definition. You must use the keyword DESCRIPTION and the text delimiters /* and */ to insert one or more lines of text describing the record into the DEFINE statement. See Section 2.10.
- The field description statement defines the whole record. You can use an elementary, STRUCTURE, COPY, or VARIANTS field description statement, but you can use only one field description statement to define a record. STRUCTURE, COPY, and VARIANT field description statements are themselves subdivided and defined by subordinate field description statements. See Sections 2.6, 2.24, and 2.26.
- If you specify a path name in the END clause, it must match the path name in the corresponding DEFINE clause.
- If you specify a given name in the END clause, it must be the given name of the new data definition and match the final given name in the path specification of the corresponding DEFINE clause.

Usage Notes

- If the path name contains dictionary directories that do not exist, the CDD automatically creates them.
- You can include passwords in the DEFINE statement path name, but this is not recommended because passwords included in the source can be displayed with the DMU LIST/ITEM=SOURCE command. Instead, use the /PATH qualifier with the compile command if passwords are required in the path name. See Chapter 3.
- The only way to assign a password to a record definition is to use the DMU SET PROTECTION or DMU SET PROTECTION/EDIT command after you compile the source file. You cannot assign a password to a record definition by including it in the DEFINE statement.

Example

The following record description defines the dictionary object SALARY_RECORD in the sample CDD hierarchy. The field description statement for the record is SALARY STRUCTURE.

```
DEFINE RECORD CDD$TOP.PERSONNEL.SERVICE.SALARY_RECORD.
    SALARY STRUCTURE.
        EMPLOYEE_ID             DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 9 DIGITS.
        PAY STRUCTURE.
            JOB_CLASS           DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 3 DIGITS.
            INCR_LEVEL          DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 1 DIGIT.
            WEEKLY_SALARY       DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 6 DIGITS 2 FRACTIONS.
        END PAY STRUCTURE.
    END SALARY STRUCTURE.
END SALARY_RECORD RECORD.
```

2.10 DESCRIPTION Clause

Inserts text documenting record and field definitions into the CDD.

Format

Within the DEFINE statement:

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

Preceding field description statements:

```
/* text */
```

Syntax Rules

- You must use the keyword DESCRIPTION and the text delimiters /* and */ to insert one or more lines of text describing the record into the DEFINE statement.
- You can also include text to describe individual fields within the record definition. Use the text delimiters /* and */ without the keyword DESCRIPTION, and place the text immediately before the field description statement of the field you want to document.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL descriptions.

Usage Notes

- This clause provides a convenient way to associate text with a record description. Proper use of this capability provides a tool for documenting the sources of data and for describing the field structure of a record. DESCRIPTION text can also aid in establishing record definition conventions and standards.
- You can also include comments in a CDDL source file by using the exclamation point (!) as a comment delimiter. Using the DESCRIPTION clause is preferable to including comment lines in the source file because DESCRIPTION text is stored directly in the CDD. CDDL/COPY_LIST copies DESCRIPTION clauses from template records, but does not copy comments preceded by an exclamation point. Also, you can display record-level DESCRIPTION text with DMU's

list/ITEM = DESCRIPTION command. DMU's EXTRACT/RECORD command extracts record- and field-level DESCRIPTION text but does not extract comments preceded by an exclamation point.

Example

The following example contains DESCRIPTION text of the record definition as a whole, and of the elementary field ID.

```
DEFINE RECORD CDD$TOP.CORPORATE.EMPLOYEE_LIST
    DESCRIPTION /* This record contains the employee master list,
                   and it is the source from which employee fields
                   in other record descriptions are copied. */.
    EMPLOYEE STRUCTURE.
        /* An employee's ID number is his or her social
        security number. */
        ID                          DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 9 DIGITS.
        NAME STRUCTURE.
            LAST_NAME               DATATYPE IS TEXT
                                    SIZE IS 15 CHARACTERS.
            FIRST_NAME              DATATYPE IS TEXT
                                    SIZE IS 10 CHARACTERS.
            MIDDLE_INITIAL          DATATYPE IS TEXT
                                    SIZE IS 1 CHARACTER.
        END NAME STRUCTURE.
        ADDRESS                     COPY FROM
                                    CDD$TOP.CORPORATE.ADDRESS_RECORD.
        DEPT_CODE                   DATATYPE IS UNSIGNED NUMERIC
                                    SIZE IS 3 DIGITS.
    END EMPLOYEE STRUCTURE.
END EMPLOYEE_LIST RECORD.
```

2.11 EDIT_CODE Field Attribute Clause

Provides a code that VAX RPG II follows when printing a field's value.

Format

```
EDIT_CODE FOR RPG [ IS ] quoted-string
```

Parameter

quoted-string

A VAX RPG II edit code or modifier.

Syntax Rules

- The quoted string must be a valid VAX RPG II edit code or modifier. However, the CDDL compiler does not check the quoted string for correct syntax.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

Usage Note

Only VAX RPG II supports the EDIT_CODE clause. Other language processors ignore it.

Example

In the following example, the edit code for the field ORDNUM is 3. Because the attribute's value is 3, a VAX RPG II program will suppress zeros when printing ORDNUM.

```
TRANSACTION STRUCTURE.
    ORDNUM          DATATYPE IS NUMERIC RIGHT OVERPUNCHED
                    SIZE IS 8 DIGITS
                    EDIT_CODE FOR RPG IS "3".
    AMOUNT          DATATYPE IS NUMERIC RIGHT OVERPUNCHED
                    SIZE IS 8 DIGITS 2 FRACTIONS
                    EDIT_WORD FOR RPG IS "$0   ,   .  CR".
END TRANSACTION STRUCTURE.
```

2.12 EDIT_STRING Field Attribute Clause

Provides a format that VAX DATATRIEVE follows when displaying a field's value.

Format

```
EDIT_STRING FOR { DTR       } [ IS ] quoted-string.
                { DATATRIEVE }
```

Parameter

quoted-string

A VAX DATATRIEVE edit string.

Syntax Rules

- The edit string must be a valid VAX DATATRIEVE expression. However, the CDDL compiler does not check the quoted string for correct syntax.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

Usage Note

Only VAX DATATRIEVE supports the EDIT_STRING clause. Other language processors ignore it.

Example

In the following example, VAX DATATRIEVE displays the TRANS_DATE field as a series of three 2-digit numbers in the format month/day/year.

```
TRANSACTION STRUCTURE      OCCURS 1 TO 99 TIMES
                           DEPENDING ON TRANSACTION_COUNT.
    TRANS_DATE             DATATYPE IS DATE
                           EDIT_STRING FOR DATATRIEVE
                           IS "MM/DD/YY".
    ORDER_NUMBER           DATATYPE IS UNSIGNED NUMERIC
                           SIZE IS 10 DIGITS.
    AMOUNT                 DATATYPE IS UNSIGNED NUMERIC
                           SIZE IS 8 DIGITS 2 FRACTIONS
                           INITIAL VALUE IS 0.
END TRANSACTION STRUCTURE.
```

2.13 EDIT_WORD Field Attribute Clause

Provides a format that VAX RPG II follows when printing a field's value.

Format

```
EDIT_WORD FOR RPG [ IS ] quoted-string
```

Parameter

quoted-string

A VAX RPG II edit word.

Syntax Rules

- The quoted string must be a valid VAX RPG II edit word. However, the CDDL compiler does not check the quoted string for correct syntax.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

Usage Note

Only VAX RPG II supports the EDIT_WORD clause. Other language processors ignore it.

Example

In the following example, the EDIT_WORD attribute in the AMOUNT field specifies a monetary format:

```
TRANSACTION STRUCTURE.
    ORDNUM          DATATYPE IS NUMERIC RIGHT OVERPUNCHED
                    SIZE IS 8 DIGITS
                    EDIT_CODE FOR RPG IS "3".
    AMOUNT          DATATYPE IS NUMERIC RIGHT OVERPUNCHED
                    SIZE IS 8 DIGITS 2 FRACTIONS
                    EDIT_WORD FOR RPG IS "$0   ,   .  CR".
END TRANSACTION STRUCTURE.
```

2.14 Elementary Field Description Statement

Defines the characteristics of a field that is not subdivided into other fields.

Format

```
[ /* text */ ]

{ field-name | * } field-attribute [ field-attribute ] . . . .
```

Parameters

text

Explanatory text describing the field.

field-name

The field's name.

field-attribute

The field's characteristics, including data type.

Syntax Rules

- The field name you assign can be up to 31 characters from the set A-Z, 0-9, _, and $. The first character must be a letter from A-Z, and the last character cannot be _ or $. If you are using a terminal of the VT200 family, you can use 8-bit alphabetic characters in field names. Remember that other terminals cannot reproduce 8-bit characters.
- If you use an asterisk (*) instead of a field name, you create an unnamed field.
- You must include the DATATYPE clause among the selected field attributes.
- You must terminate the elementary field description statement with a period.

Usage Notes

- Unnamed fields 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.

Elementary

- Each field, except BIT fields, begins on the first byte following the preceding field. BIT fields begin on the bit immediately following the preceding field. You can modify this starting position with the ALIGNED clause. See Section 2.1.

Example

NAME and ACCOUNT_NUMBER in the example below are elementary fields.

```
CUSTOMER STRUCTURE.
    NAME                    DATATYPE IS TEXT
                            SIZE IS 30 CHARACTERS.
    ACCOUNT_NUMBER          DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 7 CHARACTERS.
END CUSTOMER STRUCTURE.
```

2.15 INITIAL_VALUE Field Attribute Clause

Declares a field's value when CDDL first allocates the field.

Format

```
INITIAL_VALUE [IS] { complex-number
                   | fixed-point-number
                   | floating-point-number
                   | quoted-string
                   | hex-number
                   | octal-number
                   | EXTERNAL quoted-string }
```

Usage Notes

- The value of the literal must fit into the space allocated for the field.
- You can specify a complex number as the INITIAL_VALUE only of a field whose data type is F_FLOATING COMPLEX, D_FLOATING COMPLEX, G_FLOATING COMPLEX, or H_FLOATING COMPLEX.
- You can specify a fixed point number as the INITIAL_VALUE of any field whose data type is not DATE, TEXT, UNSPECIFIED, VARYING STRING, or VIRTUAL FIELD.
- You can specify a floating point number as the INITIAL_VALUE of a field whose data type is not DATE, TEXT, UNSPECIFIED, VARYING STRING, or VIRTUAL FIELD.
- You can specify a quoted string as the INITIAL_VALUE only of a field whose data type is DATE, TEXT, UNSPECIFIED, or VARYING STRING.
- The quoted-string in the EXTERNAL subclause contains a legal VAX COBOL external name. See the *VAX COBOL Language Reference Manual* for legal EXTERNAL datatypes.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

- You can specify a hexadecimal number as the INITIAL_VALUE of a field with any data type except VIRTUAL FIELD. In order to specify a hexadecimal number, place single quotation marks (') around the number and precede it with %X; for example,

```
INITIAL_VALUE IS %X'3E'
```

- You can specify an octal number as the INITIAL_VALUE of a field with any data type except VIRTUAL FIELD. In order to specify an octal number, place single quotation marks (') around the number and precede it with %O; for example,

```
INITIAL_VALUE IS %O'16'
```

- A VIRTUAL FIELD cannot have an INITIAL_VALUE clause.
- Language processors that do not support the INITIAL_VALUE clause ignore it.
- If the base is not ten and scale is not zero, you can specify initial values only in hexadecimal or octal. Furthermore, before you translate the initial value to hexadecimal or octal, you should multiply it by the base raised to the value of the scale. For example, to specify an initial value of 1 for a field with base 2 and scale 5, first multiply the value by 2 raised to the fifth, yielding 32. Then convert 32 to its hexadecimal or octal equivalent, and store that value.

Example

The following data declaration gives the field AMOUNT an INITIAL_VALUE of 0.

```
TRANSACTION STRUCTURE      OCCURS 1 TO 99 TIMES
                           DEPENDING ON TRANSACTION_COUNT.
    TRANS_DATE             DATATYPE IS DATE.
    ORDER_NUMBER           DATATYPE IS UNSIGNED NUMERIC
                           SIZE IS 10 DIGITS.
    AMOUNT                 DATATYPE IS UNSIGNED NUMERIC
                           SIZE IS 8 DIGITS 2 FRACTIONS
                           INITIAL_VALUE IS 0.
END TRANSACTION STRUCTURE.
```

2.16 JUSTIFIED RIGHT Field Attribute Clause

Truncates or fills a TEXT or UNSPECIFIED field from the left instead of from the right.

Format

```
JUSTIFIED RIGHT
```

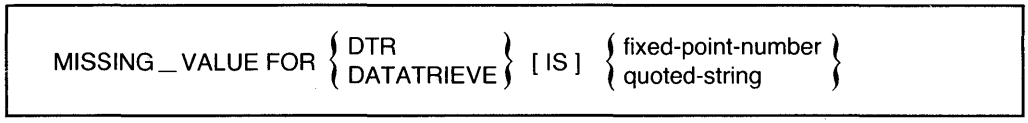
Usage Notes

- Only VAX COBOL supports the JUSTIFIED RIGHT clause. Other language processors ignore it.
- Use this clause only on fields whose data type is TEXT or UNSPECIFIED.

2.17 MISSING_VALUE Field Attribute Clause

Specifies a value to indicate that a field has never been assigned a meaningful value.

Format



Parameters

fixed-point-number
quoted-string

The VAX DATATRIEVE missing value.

Syntax Rules

- The quoted string or fixed point number must be a valid VAX DATATRIEVE expression for the field. The CDDL compiler does not check for correct syntax.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

Usage Notes

- Only VAX DATATRIEVE supports the MISSING_VALUE clause. Other language processors ignore it.
- VAX DATATRIEVE treats a field containing a MISSING_VALUE as a special case; for example, VAX DATATRIEVE ignores fields containing a MISSING_VALUE when it performs statistical operations.

Example

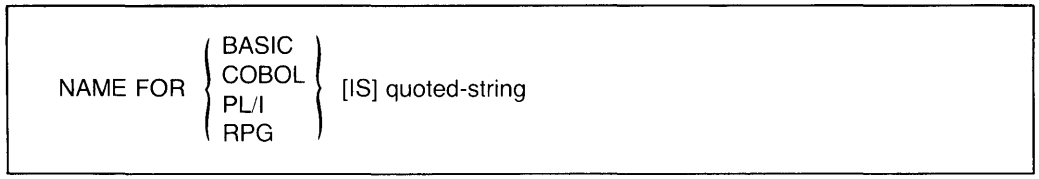
The following example assigns a missing value of 0 to the field PRICE. A PRICE of 0 indicates to VAX DATATRIEVE that the value for PRICE is missing; VAX DATATRIEVE ignores records with PRICE equal to 0 when it performs operations involving the PRICE field.

```
PRICE                          DATATYPE IS UNSIGNED NUMERIC
                               SIZE IS 8 DIGITS 2 FRACTIONS
                               MISSING_VALUE FOR DATATRIEVE IS 0.
```

2.18 NAME Field Attribute Clause

Declares a facility-specific name for a field. The specified language or language processor then recognizes only this name when you refer to the field.

Format



Parameter

quoted-string

The facility-specific field name.

Syntax Rules

- The quoted string must be a legal name for the specified language or language processor. The CDDL does not check the quoted string for validity or correct syntax.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

Usage Notes

- You can use this clause only if you have specified a field name in the field declaration. You cannot specify a facility-specific name for unnamed fields.
- Once you have assigned a facility-specific name to a field, the facility no longer recognizes the field's original name.
- Be careful when you use the NAME clause because it enables you to assign completely different names to the same field. If you use it improperly, the NAME clause can become a source of confusion.
- Avoid assigning dissimilar names to the same field. The NAME clause is designed only to allow you to make field names seem native to applications languages.

Example

The following example provides a VAX COBOL name and a VAX RPG II name for the ORDER_NUMBER field. Because of the NAME clause, VAX COBOL recognizes the field only by the name ORDER-NUMBER, and VAX RPG II recognizes the field only by the name ORDER#.

```
ORDER_NUMBER        DATATYPE IS UNSIGNED NUMERIC
                    SIZE IS 10 DIGITS
                    NAME FOR COBOL IS "ORDER-NUMBER",
                    NAME FOR RPG IS "ORDER#",
```

2.19 OCCURS Field Attribute Clause

Declares fixed-length, one-dimensional arrays.

Format

```
OCCURS n1 [ TIME[S] ]
        [ INDEXED FOR COBOL BY quoted-string [,...] ]
```

Parameters

n1

The number of occurrences of the array.

quoted-string

A VAX COBOL index name.

Syntax Rules

- An unsigned integer (n1) declares the number of occurrences in one-dimensional, fixed-length arrays. This integer is the upper bound of the array.
- The number of occurrences must be greater than zero.

Usage Notes

- The unsigned integer (n1) is the array's upper bound; the lower bound of an array declared with OCCURS is always 1. If you need to specify an array with a lower bound other than 1, use the ARRAY clause.
- Only VAX COBOL supports the INDEXED FOR COBOL BY optional field attribute clause. Other processors ignore it.
- You cannot use the INDEXED FOR COBOL BY optional field attribute clause with Version 3.0 of VAX COBOL or any earlier version. VAX COBOL supports INDEXED FOR COBOL BY in Version 3.1 and later.
- If you use a name as a COBOL index name, you cannot use that name as a field name or COBOL-specific name elsewhere in the record description.

Example

In the following example, the OCCURS clause is used twice to declare 20 instances of SUPPLIER where each instance is four 30-character strings. Note that OCCURS clauses can be nested.

```
SUPPLIER STRUCTURE                OCCURS 20 TIMES.
    SUPPLIER                      OCCURS 4 TIMES
                                  DATATYPE IS TEXT
                                  SIZE IS 30 CHARACTERS.
END SUPPLIER STRUCTURE.
```

2.20 OCCURS . . . DEPENDING Field Attribute Clause

Declares a variable-length, one-dimensional array.

Format

```
OCCURS n1 TO n2 [ TIME[S] ] DEPENDING [ ON ] field-name
        [ INDEXED FOR COBOL BY quoted-string [,...] ]
```

Parameters

n1, n2

The range for the number of occurrences.

field-name

The tag variable field, whose value determines the actual number of occurrences.

quoted-string

A VAX COBOL index name.

Syntax Rules

- Two unsigned integers (n1 and n2) specify a range for the number of occurrences; n1 specifies the minimum number of occurrences and must be greater than or equal to zero; n2 specifies the maximum number of occurrences and must be greater than or equal to n1.
- The actual number of occurrences varies according to the value of the named field.
- The field named in the DEPENDING clause must be an elementary field fixed in the record and not part of an array. Its field description must precede the array field description, and its value must never be less than n1 nor greater than n2. You must name the field.

- You must fully qualify the field name if it is not unique within the record. A fully qualified field name consists of an elementary field name preceded by the field names of as many of its direct ancestors as you need to specify the elementary field uniquely. You cannot omit the names of any of the ancestor fields within the fully qualified name, but once the elementary field name is identified uniquely, you can omit any remaining ancestors' field names. You must separate each element of a fully qualified field name from the next with a period.

Usage Notes

- The unsigned integers n1 and n2 declare the range for the array's upper bound; the lower bound of an array declared with OCCURS . . . DEPENDING is always 1. If you need to specify an array with a lower bound other than 1, use the ARRAY clause.
- If the tag variable's name is not unique within the record, none of the ancestors in its fully qualified name can be unnamed fields.
- Only VAX COBOL supports the INDEXED FOR COBOL BY field attribute clause. Other processors ignore it.
- You cannot use the INDEXED FOR COBOL BY optional field attribute clause with Version 3.0 of VAX COBOL or any earlier version. VAX COBOL supports INDEXED FOR COBOL BY in Version 3.1 and later.
- If you use a name as a COBOL index name, you cannot use that name as a field name or COBOL-specific name elsewhere in the record description.

Example

In the following example, a variable length array defines individual transactions within the STRUCTURE field SALES.

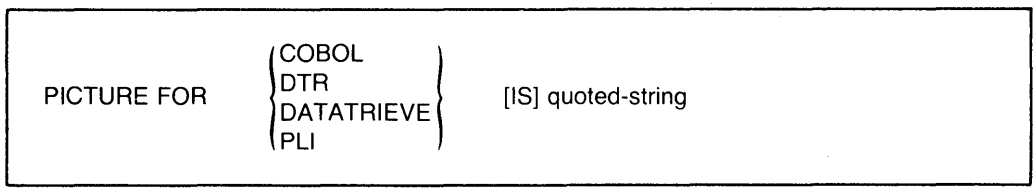
```
SALES STRUCTURE.
    TRANSACTION_COUNT                DATATYPE IS UNSIGNED WORD
                                     VALID FOR DTR IF
                                         "TRANSACTION_COUNT > 0".
    TRANSACTION STRUCTURE            OCCURS 1 TO 99 TIMES
                                         DEPENDING ON
                                         TRANSACTION_COUNT.
        TRANS_DATE                   DATATYPE IS DATE.
        ORDER_NUMBER                 DATATYPE IS UNSIGNED NUMERIC
                                     SIZE IS 10 DIGITS.
        AMOUNT                       DATATYPE IS UNSIGNED NUMERIC
                                     SIZE IS 8 DIGITS 2 FRACTIONS
                                     PICTURE FOR COBOL IS "9(6)V99".
    END TRANSACTION STRUCTURE.
END SALES STRUCTURE.
```

The fully qualified field name of TRANSACTION_COUNT is SALES.TRANSACTION_COUNT.

2.21 PICTURE Field Attribute Clause

Declares a field's picture string for a specified language or language processor.

Format



Parameter

quoted-string

The picture string for the specified language or language processor.

Syntax Rules

- The quoted string must be a valid picture string for the specified language or language processor. The CDDL compiler does not check the quoted string for validity or correct syntax.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.
- The data type implicit in the quoted string must be consistent with the data type you select with the DATATYPE clause.

Usage Notes

- Each language that requires a picture string will construct a default picture string if you do not provide one. The default picture string provides a concise, efficient description of the field. You should use the default picture string whenever possible and avoid facility-specific PICTURE clauses.
- The CDDL compiler does not check picture strings for conformity with the DATATYPE clause. In most cases, therefore, the default picture string is probably the best option.
- You can use the EDIT_STRING field attribute clause to provide an edited picture string for VAX DATATRIEVE.

Example

The following example contains a picture string for COBOL.

```
AMOUNT              DATATYPE UNSIGNED NUMERIC 8 DIGITS 2 FRACTIONS
                    PICTURE FOR COBOL IS "9(6)V99".
```

2.22 QUERY_HEADER Field Attribute Clause

Provides a label that VAX DATATRIEVE uses as a column heading for the field in printouts and reports.

Format

```
QUERY_HEADER FOR { DTR       } [ IS ] quoted-string [quoted-string] . . .
                 { DATATRIEVE }
```

Parameter

quoted-string

The VAX DATATRIEVE query header.

Syntax Rules

- The quoted string must be a valid VAX DATATRIEVE query header. The CDDL compiler does not check the quoted string for correct syntax.
- Quoted strings must be separated from each other by a space, tab, or comma.

Usage Notes

- Only VAX DATATRIEVE supports the QUERY_HEADER clause. Other language processors ignore it.
- If you specify more than one quoted string, VAX DATATRIEVE stacks them. Use multiple quoted strings to specify long query headers.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

Example

In the following example, the QUERY_HEADER clause specifies that VAX DATATRIEVE use TOTAL PRICE as the column header for the field TOTAL_PRICE.

```
TOTAL_PRICE                 DATATYPE IS VIRTUAL FIELD
                            COMPUTED BY DTR AS
                                "UNIT_PRICE * QUANTITY"
                            QUERY_HEADER FOR DTR IS "TOTAL PRICE".
```

2.23 QUERY_NAME Field Attribute Clause

Provides VAX DATATRIEVE with an alternate reference name for a field. VAX DATATRIEVE allows you to refer to a field either by its field name or by a specified query name.

Format

```
QUERY_NAME FOR {DTR       } [ IS ] quoted-string
               {DATATRIEVE}
```

Parameter

quoted-string

The VAX DATATRIEVE query name.

Syntax Rule

The quoted string must be a valid VAX DATATRIEVE query name. The CDDL compiler does not check the quoted string for correct syntax.

Usage Notes

- Only VAX DATATRIEVE supports the QUERY_NAME clause. Other processors ignore it.
- You can specify a query name for a field only if you have specified a field name for it in the field description statement. You cannot specify a query name for unnamed fields.
- QUERY_NAME is different from NAME because VAX DATATRIEVE recognizes both the query name and the original field name. With NAME, the facility-specific name is the only name recognized by the specified language or language processor.
- Choose a query name that is shorter and easier to remember than the actual field name. You can then choose a field name that is descriptive of the field's contents and purpose.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

Example

In the following example, the QUERY_NAME clause specifies TP as an alternate DATATRIEVE name for the field TOTAL_PRICE.

```
TOTAL_PRICE                DATATYPE IS VIRTUAL FIELD
                           COMPUTED BY DTR AS
                              "UNIT_PRICE * QUANTITY"
                           QUERY_NAME FOR DTR IS "TP".
```

2.24 STRUCTURE Field Description Statement

Defines a field that is divided into one or more subordinate fields.

Format

```
[ /* text */ ]

{ *          } STRUCTURE [ field-attribute ] . . . .
{ field-name }

    field-description-statement
    [field-description-statement] . . .

END [ field-name ] [ STRUCTURE ].
```

Parameters

text

Explanatory text describing the field.

field-name

The field's given name.

field-attribute

The characteristics of the STRUCTURE field.

field-description-statement

The characteristics of a subordinate field.

Syntax Rules

- The field name you assign can be up to 31 characters from the set A-Z, 0-9, _, and $. The first character must be a letter from A-Z, and the last character cannot be _ or $. If you are using a terminal of the VT200 family, you can use 8-bit alphabetic characters in field names. Remember that other terminals cannot reproduce 8-bit characters.
- If you use an asterisk (*) instead of a field-name, you create an unnamed field.

- If you do not specify a data type for the STRUCTURE, the CDDL assigns it the UNSPECIFIED data type.
- Subordinate field description statements describe contiguous portions of the field described by the STRUCTURE.
- There must be at least one subordinate field description statement. A subordinate field can be an elementary, a STRUCTURE, a COPY, or a VARIANTS field.
- CDDL accepts the keyword GROUP as a synonym for STRUCTURE, but the compiler issues a warning when you use GROUP.
- You must terminate the STRUCTURE and the END statements with periods.

Usage Notes

- Unnamed fields are similar to FILLER fields in VAX COBOL. You can use them to format print records or to reserve space in a record for future additions.
- In a STRUCTURE field, you can use any field attribute clauses allowed in an elementary field. However, if you use the DATATYPE field description statement, you cannot create subordinate fields that exceed the length of the structure field.
- A STRUCTURE field cannot contain the VIRTUAL FIELD datatype.
- Although the CDDL compiler accepts data type specifications for STRUCTURE fields, the feature may not be supported by the language or language processor you use with the CDD. Make sure the definitions you store in the dictionary are valid for the processor that will use them.
- Each field, except BIT fields, begins on the first byte following the preceding field. BIT fields begin on the bit immediately following the preceding field. You can modify this starting position with the ALIGNED clause. See Section 2.1.

Example

As this example shows, you can nest STRUCTURE field description statements. The STRUCTURE field ADDRESS, for example, has a subordinate STRUCTURE field, ZIP_CODE.

```
ADDRESS STRUCTURE.
    STREET                      DATATYPE IS TEXT
                                SIZE IS 30 CHARACTERS.
    CITY                        DATATYPE IS TEXT
                                SIZE IS 30 CHARACTERS.
    STATE                       DATATYPE IS TEXT
                                SIZE IS 2 CHARACTERS.
    ZIP_CODE STRUCTURE.
        NEW                     DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 4 DIGITS
                                BLANK WHEN ZERO.
        OLD                     DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 5 DIGITS.
    END ZIP_CODE STRUCTURE.
END ADDRESS STRUCTURE.
```

2.25 VALID FOR DATATRIEVE IF Field Attribute Clause

Causes VAX DATATRIEVE to validate value assignments to a field. VAX DATATRIEVE refuses to assign a value to a field if that value is not accepted by this validation expression.

Format

```
VALID FOR { DTR       } IF quoted-string [quoted-string] . . .
          { DATATRIEVE }
```

Parameter

quoted-string

VAX DATATRIEVE source text forming a validation expression.

Syntax Rule

The quoted strings must form a valid VAX DATATRIEVE validation expression. The CDDL compiler does not check the validation expression for correct syntax.

Usage Notes

- Only VAX DATATRIEVE can interpret these validation expressions. Other language processors ignore their presence.
- If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

Example

In the following example, the VALID FOR DATATRIEVE IF clause causes VAX DATATRIEVE to accept only positive integers for the field TRANSACTION_COUNT.

```
TRANSACTION_COUNT       DATATYPE IS UNSIGNED WORD
                        VALID FOR DTR IF
                            "TRANSACTION_COUNT > 0";
```

2.26 VARIANTS Field Description Statement

Defines a set of two or more fields mapping the same portion of a record definition. VARIANT fields are similar to fields defined with the REDEFINES clause in VAX COBOL and VAX DATATRIEVE.

There are two formats for the VARIANTS field description statement:

- VARIANTS is almost identical to the REDEFINES clause. Within an application program, you can refer to any of the VARIANT fields.
- VARIANTS OF uses the value of a tag variable at run time to determine which of the VARIANT fields is the current VARIANT.

2.26.1 VARIANTS Field Description Statement

Defines two or more logical views of the same portion of a record definition. The VARIANTS statement functions like the REDEFINES clause in VAX COBOL and VAX DATATRIEVE.

Format

```
VARIANTS .
        VARIANT.
                field-description-statement
                [field-description-statement] . . .
        END [ VARIANT ].

        VARIANT.
                field-description-statement
                [field-description-statement] . . .
        END [ VARIANT ].
                                .
                                .
                                .
END [ VARIANTS ].
```

Parameters

field-description-statement

A definition of the field characteristics for each subordinate field of each VARIANT.

Syntax Rules

- The vertical ellipsis in the format indicates that the VARIANT field description block can be repeated.
- The VARIANTS, VARIANT, and END statements all must end with periods.

Usage Notes

- Each VARIANT begins on the same byte in the record, subject to individual alignment options (see Section 2.1). The length of the longest VARIANT in the collection determines the overall length of the VARIANTS field description.

- Be sure that the VARIANTS collection you define conforms to the requirements of the language or language processor that will access the record definition.
- VAX DATATRIEVE requires each VARIANT to contain a STRUCTURE field description statement at the top of the VARIANT.

Example

The following example contains three VARIANT logical views of the same record. In an application program, you can refer to IN_STOCK, BACK_ORDER, or OUT_OF_STOCK depending on how you want to interpret the STOCK field.

```
STOCK STRUCTURE.
    VARIANTS.
        VARIANT.
            IN_STOCK STRUCTURE.
                PRODUCT_NO      DATATYPE IS TEXT
                                SIZE IS 8 CHARACTERS.
                DATE_ORDERED    DATATYPE IS DATE.
                STATUS_CODE     DATATYPE IS BYTE.
                QUANTITY        DATATYPE IS LONGWORD
                                ALIGNED ON LONGWORD.
                LOCATION        ARRAY 1:4
                                DATATYPE IS TEXT
                                SIZE IS 30 CHARACTERS.
                UNIT_PRICE      DATATYPE IS LONGWORD SCALE -2.
            END IN_STOCK STRUCTURE.
        END VARIANT.
        VARIANT.
            BACK_ORDER STRUCTURE.
                PRODUCT_NO      DATATYPE IS TEXT
                                SIZE IS 8 CHARACTERS.
                DATE_ORDERED    DATATYPE IS DATE.
                STATUS_CODE     DATATYPE IS BYTE.
                QUANTITY        DATATYPE IS LONGWORD
                                ALIGNED ON LONGWORD.
                SUPPLIER        ARRAY 1:4
                                DATATYPE IS TEXT
                                SIZE IS 30 CHARACTERS.
                UNIT_PRICE      DATATYPE IS LONGWORD
                                SCALE -2.
            END BACK_ORDER STRUCTURE.
        END VARIANT.
        VARIANT.
            OUT_OF_STOCK STRUCTURE.
                PRODUCT_NO          DATATYPE IS TEXT
                                    SIZE IS 8 CHARACTERS.
                DATE_LAST_SOLD      DATATYPE IS DATE.
            END OUT_OF_STOCK STRUCTURE.
        END VARIANT.
    END VARIANTS.
END STOCK STRUCTURE.
```

2.26.2 VARIANTS OF Field Description Statement

Names a tag variable whose value at run time determines which of the VARIANT fields is the current VARIANT.

Format

```
VARIANTS OF field-name .

    VARIANT  {VALUE  [IS]  }
             {VALUES [ARE] }  n1 [THRU n2] [n3 [THRU n4] ] ... .
        field-description-statement
        [field-description-statement] ...
    END [VARIANT] .

    VARIANT  {VALUE  [IS]  }
             {VALUES [ARE] }  n5 [THRU n6] [n7 [THRU n8] ] ... .
        field-description-statement
        [field-description-statement] ...
    END [VARIANT] .
                    .
                    .
                    .
END [VARIANTS] .
```

Parameters

field-name

The tag variable field, whose value determines the selection of the current VARIANT at run time.

n1, n2, n3, n4, n5, n6, n7, n8

Values to be compared to the value of the tag variable at run time to determine the current VARIANT.

field-description-statement

A definition of the field characteristics for the subordinate fields of each VARIANT.

Syntax Rules

- The vertical ellipsis in the format indicates that the VARIANT field description block can be repeated.
- The tag variable must be an elementary field fixed in the record and not part of an array, and it must precede the VARIANTS field description statement. It cannot be an unnamed field.
- You must fully qualify the tag variable's field name if it is not unique within the record. A fully qualified field name consists of an elementary field name preceded by the field names of as many of its direct ancestors as you need to specify the elementary field uniquely. You cannot omit the names of any of the ancestor fields within the fully qualified name, but once the elementary field name is identified uniquely, you can omit any remaining ancestors' field names. You must separate each element of a fully qualified field name from the next with a period. Furthermore, if the tag variable's name is not unique within the record, none of the ancestors in its fully qualified name can be unnamed fields.
- You must include a tag value clause for VARIANT.
- At run time, the values (n1, n2, n3, n4, . . .) you specify are compared to the value of the tag variable to determine the current VARIANT.
- Tag value specifications must conform to the following conditions:
  - The tag values n1, n2, n3, n4, . . . must be legal values as defined by the tag variable's data type. For example, if the tag variable is a TEXT field, the tag values must be quoted literals.
  - The values of n2, n4, n6, and n8 must be greater than or equal to the values of n1, n3, n5, and n7 in the collating sequence of the data type.
  - The range of values in one VARIANT must not overlap the range of values in any other VARIANT.
- The compiler ignores commas, but you can use them to make tag value specifications easier to read.
- The CDDL compiler does not check that the tag values you specify are legal. If you specify invalid values, you will receive error messages when you refer to the VARIANT field in an application.

Usage Notes

- The tag value clause specifies a distinct value or set of values for the tag variable in each VARIANT of a VARIANTS field collection. The tag variable can then be used at run time to find the current VARIANT.
- Languages that do not support tag variables ignore the tag value clause. For more information on language support of CDD record definitions, refer to the documentation for the language you are using.
- Each variant begins on the same byte in the record, subject to individual alignment options (see Section 2.1). The length of the longest VARIANT in the collection determines the overall length of the VARIANTS field description.
- Be sure that the VARIANT collection you define conforms to the requirements of the language or language processor that will access the record definition.
- VAX DATATRIEVE requires each VARIANT to contain a STRUCTURE field description statement at the top of the VARIANT.

Example

In the following example, RECORD_IDENTIFIER is the tag variable. The value of RECORD_IDENTIFIER at run time determines which VARIANT is current according to the translation table in the descriptive text.

```
STOCK STRUCTURE.
    /* RECORD_IDENTIFIER determines field type:
         S --> In-stock record.
         B --> Back-order record.
         O --> Out-of-stock record. */
    RECORD_IDENTIFIER           DATATYPE IS TEXT
                                SIZE IS 1 CHARACTER.
    VARIANTS OF RECORD_IDENTIFIER.
        VARIANT VALUE IS "S".
            IN_STOCK STRUCTURE.
                PRODUCT_NO      DATATYPE IS TEXT
                                SIZE IS 8 CHARACTERS.
                DATE_ORDERED    DATATYPE IS DATE.
                STATUS_CODE     DATATYPE IS BYTE.
                QUANTITY        DATATYPE IS LONGWORD
                                ALIGNED ON LONGWORD.
                LOCATION        ARRAY 1:4
                                DATATYPE IS TEXT
                                SIZE IS 30 CHARACTERS.
                UNIT_PRICE      DATATYPE IS LONGWORD SCALE -2.
            END IN_STOCK STRUCTURE.
        END VARIANT.
        VARIANT VALUE IS "B".
            BACK_ORDER STRUCTURE.
                PRODUCT_NO      DATATYPE IS TEXT
                                SIZE IS 8 CHARACTERS.
                DATE_ORDERED    DATATYPE IS DATE.
                STATUS_CODE     DATATYPE IS BYTE.
                QUANTITY        DATATYPE IS LONGWORD
                                ALIGNED ON LONGWORD.
                SUPPLIER        ARRAY 1:4
                                DATATYPE IS TEXT
                                SIZE IS 30 CHARACTERS.
                UNIT_PRICE      DATATYPE IS LONGWORD
                                SCALE -2.
            END BACK_ORDER STRUCTURE.
        END VARIANT.

        VARIANT VALUE IS "O".
            OUT_OF_STOCK STRUCTURE.
                PRODUCT_NO          DATATYPE IS TEXT
                                    SIZE IS 8 CHARACTERS.
                DATE_LAST_SOLD      DATATYPE IS DATE.
            END OUT_OF_STOCK STRUCTURE.
        END VARIANT.
    END VARIANTS.
END STOCK STRUCTURE.
```

The CDDL Compiler Command Descriptions 3

After you create a source file containing record definitions and data descriptions, use the CDDL compiler to place those definitions into the CDD. There are two command formats:

- The CDDL command, which compiles source files
- The CDDL/RECOMPILE command, which locates existing CDD record definitions and recompiles them

Both command formats allow you to create history list entries and listings of the source text.

Note

Before you begin storing definitions in the CDD, your system manager or data administrator should carefully plan, create, and protect your CDD directory hierarchy. See the *VAX CDD/Plus User's Guide, Appendix A*, for information about planning, creating, protecting, and maintaining the CDD directory hierarchy.

The following sections contain complete descriptions of CDDL commands, parameters, and qualifiers. These descriptions include:

- The syntax **format** you should use for each command
- The **parameters** of each command
- Command **qualifiers** that modify the functions of each command
- **Restrictions** on the ways you can use commands

- **Usage notes** to show you how to use each command
- **Required privileges** for each command
- **Examples** from the sample dictionary (Figure 1-1) to illustrate the use of each command

3.1 CDDL Command

Compiles source files and places record definitions in the CDD.

Format

CDDL file-specification

Qualifiers	*Defaults*
/[NO] ACL	/ACL
/AUDIT [= (quoted-string [, quoted-string] ...)]	
/AUDIT = file-specification	
/NOAUDIT	/NOAUDIT
/[NO]COPY_LIST	/NOCOPY_LIST
/[NO]DIAGNOSTICS	/NODIAGNOSTIC
/LISTING[=file-specification]	/LISTING
/NOLISTING	
/PATH = path-name	/PATH = CDD$DEFAULT
/[NO]REPLACE	/NOREPLACE
/[NO]VERSION	/NOVERSION
/V2	

Parameter

file-specification

The CDDL source file you want to compile. Each source file contains one or more CDD record definitions.

The file specification is a standard VMS file specification. The default file type is .DDL.

Qualifiers

/ACL

Creates an access control list for each record definition in the source file. The contents of the access control list created vary, depending on the qualifiers used and whether the target directory already contains one or more versions of the record definition.

- If the directory in which CDDL stores the record definition does not already contain a record definition with the same name, CDDL creates the default access control list, which grants you, the creator, the following privileges: CONTROL, LOCAL_DELETE, DTR_EXTEND/EXECUTE, HISTORY, DTR_MODIFY, DTR_READ, SEE, UPDATE, and DTR_WRITE.
- If the directory contains one or more versions of a record definition with the same name as the definition CDDL is compiling and you use the /REPLACE qualifier, CDDL copies the contents of the access control list of the replaced version into the access control list of the new version.
- If the directory contains one or more versions of a record definition with the same name as the definition CDDL is compiling and you use the /VERSION qualifier, CDDL copies the contents of the access control list of the highest existing version into the access control list of the new version.

/NOACL

Prevents the creation of an access control list.

/AUDIT [= (quoted-string [, quoted-string]...)]
/AUDIT = file-specification

Creates a history list entry auditing the creation of each record definition.

You can include explanatory text in the history list entries in two ways:

- By including quoted strings. Enclose each quoted string in double quotation marks, and enclose the series of strings in parentheses. The parentheses are optional if you specify only one quoted string.
- By specifying a file whose contents are to be included in the history list entry. The file specification is a standard VMS file specification, and the default file type is .DAT.

  You can include no more than 64 input strings in a history list entry. The compiler ignores any excess.

If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

/NOAUDIT

Prevents the creation of a history list entry.

/COPY_LIST

Expands in the listing file all template records included in a record description. The CDDL compiler creates CDDL source text from all template records and inserts that source text into the listing file. In the listing, the CDDL compiler inserts a "T" as the first character of each line that is part of an expanded template record.

/NOCOPY_LIST

Prevents the expansion of template records in the listing file.

/DIAGNOSTICS

Creates a diagnostics file that lists errors occurring during compilation. This qualifier is designed for use with the VAX Language-Sensitive Editor (LSE), and the diagnostics file is reserved for use by DIGITAL. LSE uses the file to display diagnostic messages and to position the cursor where a source error exists. The diagnostics file has the name of your source definition file and the default extension .DIA.

/NODIAGNOSTICS

Prevents creation of the diagnostics file. The default is /NODIAGNOSTICS.

/LISTING [=file-specification]

Writes an output file containing the command line, the source text, and CDDL messages. The file specification is a standard VMS file specification, and the default file type is .LIS.

/NOLISTING

Prevents creation of the listing file.

/PATH = path-name

Names a default directory from which to trace the path names in the CDDL source files. The CDDL uses your CDD$DEFAULT directory if you do not specify /PATH.

/REPLACE

Deletes an existing CDD record definition and inserts a new definition in its place. The new record definition copies the access control list and the history list of the record definition it replaces.

/NOREPLACE

Prevents the replacement of an existing record definition.

/VERSION

Creates an additional version of an existing CDD record. If you specify an absolute version number in the path name, the new record has that version number. If you do not, it has a version number one higher than the highest existing version.

If the logical name CDD$VERSION_LIMIT has been defined for your system, group, or process, the dictionary will store only the number of versions allowed by the quota CDD$VERSION_LIMIT specifies.

The newly created record definition copies the access control list of the highest existing version of the record definition. If no version exists, the newly created record definition contains the default access control list. You can prevent the creation of any access control list by using the /NOACL qualifier.

The newly created record definition copies the history list of the highest existing version.

/NOVERSION

Prevents the creation of a new version of an existing record definition.

/V2

The /V2 qualifier causes the compiler to use the CDD V2.0 defaults for the signs of the fixed point numbers. With this qualifier, BYTE, WORD, and LONGWORD data types are unsigned by default, but QUADWORD and OCTAWORD data types are signed.

If you do not specify the /V2 qualifier, all the fixed point data types are unsigned by default.

Restrictions

- You cannot specify both /NOLISTING and /COPY_LIST.
- You cannot specify both /REPLACE and /VERSION.
- You cannot specify both /REPLACE and /NOACL. When you use the /REPLACE qualifier, the new definition always copies the access control list of the definition it replaces.

Usage Notes

- You can include passwords in the DEFINE statement path name, but this is not recommended because passwords included in the source text can be

displayed with the DMU LIST/ITEM=SOURCE command. Instead, use the /PATH qualifier with the compile command if passwords are required in the path name.

- If you use any qualifiers, you must type the source file specification on the same command line as the CDDL command. If you press RETURN before giving the file specification, CDDL does not prompt you for it.
- If you specify the version number of a record definition in the source file and a record definition with the same name and version number already exists, the compilation will fail if you use the /VERSION qualifier. To correct this problem, you can change or eliminate the version number in the source file or use /REPLACE instead of /VERSION.

Required Privileges

- You need SEE and PASS_THRU access to the lowest existing ancestor of the dictionary object you are creating. You also need EXTEND unless you are using /REPLACE.
- If you use /REPLACE with the CDDL command, you need SEE, PASS_THRU, UPDATE, and either LOCAL_DELETE or GLOBAL_DELETE access to the dictionary object you are replacing.
- If you use /VERSION with the CDDL command, you need SEE, PASS_THRU, and UPDATE access to the highest version of the dictionary object you are compiling. Furthermore, if you use /NOACL with /VERSION, you must have CONTROL access to the highest existing version of the record definition.

Examples

The following command inserts the record definition contained in a file named EMPLOYEE.DDL into the CDD. The new record definition has the default access control list and a history list entry containing the explanation, "Initial compile". A source file can contain any number of record definitions, but the CDDL creates only one listing file for each source file.

```
$ CDDL/AUDIT="Initial compile" EMPLOYEE.DDL
```

When the source file is compiled, CDDL creates a listing file, EMPLOYEE.LIS, in your default VMS directory. EMPLOYEE.LIS contains the command line you entered, the CDDL source file, and certain CDDL messages.

```
Command Line:  CDDL/AUDIT="Initial compile" EMPLOYEE.DDL        Source File:
DB3:[CASADAY.CDD]EMPLOYEE.DDL;1

       0001  DEFINE RECORD CDD$TOP.CORPORATE.EMPLOYEE_LIST
       0002    DESCRIPTION IS
       0003      /* This record contains the master list of all
       0004         employees */.
       0005    EMPLOYEE STRUCTURE.
       0006      /* An employee's ID number is his
       0007         or her social security number */
       0008      ID                  DATATYPE IS UNSIGNED NUMERIC
       0009                          SIZE IS 9 DIGITS.
       0010      NAME STRUCTURE.
       0011        LAST_NAME         DATATYPE IS TEXT
       0012                          SIZE IS 15 CHARACTERS.
       0013        FIRST_NAME        DATATYPE IS TEXT
       0014                          SIZE IS 10 CHARACTERS.
       0015        MIDDLE_INITIAL    DATATYPE IS TEXT
       0016                          SIZE IS 1 CHARACTER.
       0017      END NAME STRUCTURE.
       0018
       0019      ADDRESS             COPY FROM
       0020                          CDD$TOP.CORPORATE.ADDRESS_RECORD.
       0021      DEPT_CODE           DATATYPE IS UNSIGNED NUMERIC
       0022                          SIZE IS 3 DIGITS.
       0023    END EMPLOYEE STRUCTURE.
       0024  END EMPLOYEE_LIST RECORD.
                                              1
%CDDL-S-RECORDCRE, record "CDD$TOP.CORPORATE.EMPLOYEE_LIST;1" created in the CDD
```

Example 3-1: Sample CDDL Listing File

The following example demonstrates how to create an additional version of a record definition.

The sample dictionary (Figure 1-1) contains two versions of the record definition CDD$TOP.PERSONNEL.STANDARDS.SALARY_RANGE. The file SALARY2.DDL contains the CDDL source file used to create CDD$TOP.PERSONNEL.STANDARDS.SALARY_RANGE;2:

```
$ TYPE SALARY2.DDL
DEFINE RECORD CDD$TOP.PERSONNEL.STANDARDS.SALARY_RANGE
    DESCRIPTION IS
        /* This record stores minimum salaries
        for the four incremental salary levels within
        each of 150 job classifications.  It reflects
        Personnel Policy effective 1/1/84. */.
    SALARY_RANGE     ARRAY 150,4
                     DATATYPE IS UNSIGNED NUMERIC
                     SIZE IS 8 DIGITS 2 FRACTIONS.
END SALARY_RANGE RECORD.
```

The following example compiles SALARY2.DDL and adds it to a directory already containing SALARY_RANGE;1.

```
$ CDDL/VERSION/AUDIT SALARY2.DDL
```

The /VERSION qualifier causes the CDDL compiler to create a new version of SALARY_RANGE while retaining SALARY_RANGE;1. Because the source file does not specify the version number of the record definition, CDDL gives the new definition a version number of 2, one higher than the highest existing version. SALARY_RANGE;2 copies the access control list of SALARY_RANGE;1 because /ACL is the default. SALARY_RANGE;2 also copies the history list of SALARY_RANGE;1 and adds an entry documenting its creation.

3.2 CDDL/RECOMPILE Command

Recompiles CDD record definitions from the source text stored in the dictionary. Use CDDL/RECOMPILE to update record definitions containing a COPY field description statement after you modify the template record that the field copies. CDDL/RECOMPILE recompiles those record definitions and updates the definitions referred to in the COPY field description statements.

Format

```
CDDL/RECOMPILE path-name [ , path-name ] ...
```

Qualifiers	*Defaults*
/[NO] ACL	/ACL
/AUDIT [= (quoted-string [, quoted-string] ...)]	
/AUDIT = file-specification	
/NOAUDIT	/NOAUDIT
/[NO] COPY_LIST	/NOCOPY_LIST
/LISTING [= file-specification]	/LISTING
/NOLISTING	
/PATH = path-name	/PATH = CDD$DEFAULT
/[NO] VERSION	/NOVERSION
/V2	

Parameter

path-name

Specifies a record definition to be recompiled. You can use the absolute or the relative path name.

Qualifiers

/ACL

Creates an access control list for the recompiled record definition. If you do not use the /VERSION qualifier, the recompiled definition copies the access control list of the version that you recompiled. If you use the /VERSION

qualifier, the recompiled definition copies the access control list of the highest existing version of the record definition.

/NOACL

Prevents the creation of an access control list when it is used with /VERSION. Unless you specify both /NOACL and /VERSION, you cannot prevent the creation of an access control list.

/AUDIT [= (quoted-string [, quoted-string]...)]
/AUDIT = file-specification

Creates a history list entry auditing the recompilation of each record definition.

You can include explanatory text in the history list entries in two ways:

- By including quoted strings. Enclose each quoted string in double quotation marks, and enclose the series of strings in parentheses. The parentheses are optional if you specify only one quoted string.
- By specifying a file whose contents are to be included in the history list entry. The file specification is a standard VMS file specification, and the default file type is .DAT. You can include no more than 64 input strings in a history list entry. The compiler ignores any excess.

If you are using a terminal of the VT200 family, you can use 8-bit characters in CDDL quoted strings.

/NOAUDIT

Prevents the creation of a history list entry documenting the recompilation.

/COPY_LIST

Expands in the listing file all template records included in a record description. The CDDL compiler creates CDDL source text from all template records and inserts that source text into the listing file. In the listing, the CDDL compiler inserts a "T" as the first character of each line that is part of an expanded template record.

/NOCOPY_LIST

Prevents the expansion of template records in the listing file.

/LISTING [=file-specification]

Writes an output file containing the source text of the record definition and CDDL messages. The file specification is a standard VMS file specification,

and the default file type is .LIS. /LISTING is the default; however, if you do not use /LISTING to explicitly name the listing file, CDDL creates a file called .LIS.

/NOLISTING

Prevents creation of the listing file.

/PATH = path-name

Names a default directory from which to trace the path names in the CDDL source files. The CDDL uses your CDD$DEFAULT directory if you do not specify /PATH.

/VERSION

Recompiles an existing record definition and creates an additional version of it. CDDL/RECOMPILE/VERSION recompiles the record you specify in the path name parameter, but does not replace it. Instead, it creates an additional version of the record definition with a version number one greater than the highest existing version.

/NOVERSION

Prevents the creation of an additional version of the recompiled definition.

/V2

Causes the compiler to use the CDD V2.0 defaults for the signs of the fixed point numbers. With this qualifier, BYTE, WORD, and LONGWORD data types are unsigned by default, but QUADWORD and OCTAWORD data types are signed.

If you do not specify the /V2 qualifier, all the fixed point data types are unsigned by default.

Restrictions

- You cannot specify both /NOLISTING and /COPY_LIST.
- You must specify /VERSION if you specify /NOACL. Unless you are creating an additional version, the recompiled definition always copies the access control list of the definition it replaces.

Usage Notes

- You can recompile CDD record definitions only if the CDDL compiler created the record definition. Source text created by other processors may be unavailable or incompatible with CDDL syntax.
- The CDDL compiler accepts password specifications for the given names of new record definitions, but passwords included in the source can be displayed with the DMU LIST/ITEM=SOURCE command.

Required Privileges

- You need PASS_THRU and EXTEND access to the parent of the object you are recompiling. You also need SEE and PASS_THRU access to the object. If you are not using /VERSION, you need UPDATE and either LOCAL_DELETE or GLOBAL_DELETE access to the object.
- If you use /VERSION with the CDDL/RECOMPILE command, you need SEE, PASS_THRU, and UPDATE access to the highest version of the dictionary object you are compiling. Furthermore, if you use /NOACL with /VERSION, you must have CONTROL access to the highest existing version of the record.

Examples

In the following example, the field ADDRESS, which is copied from the CORPORATE directory, is recompiled because of modifications made to the template record, CDD$TOP.CORPORATE.ADDRESS_RECORD.

```
CDD$TOP.SALES.CUSTOMER_RECORD
    CUSTOMER STRUCTURE.
        NAME        DATATYPE IS TEXT
                    SIZE IS 30 CHARACTERS.
        ADDRESS     COPY FROM
                    CDD$TOP.CORPORATE.ADDRESS_RECORD.
    END CUSTOMER STRUCTURE.

$CDDL/RECOMPILE/AUDIT CDD$TOP.SALES.CUSTOMER_RECORD
```

In the following example, CDD$TOP.SALES.CUSTOMER_RECORD is again recompiled. In this case, however, the CDDL compiler creates an additional version because the /VERSION qualifier is used. The new version contains the same access control list and history list as the highest existing version of CDD$TOP.SALES.CUSTOMER_RECORD.

```
$CDDL/RECOMPILE/VERSION/LISTING=CUSTOMER.LIS/COPY_LIST -
$CDD$TOP.SALES.CUSTOMER_RECORD
```

When the above record definition is recompiled, CDDL creates a listing file, CUSTOMER.LIS, in your default VMS directory. CUSTOMER.LIS contains the command line you entered, the source text of the recompiled definition, and certain CDDL messages. Because /COPY_LIST is used, the CDDL compiler expands the definition to include source text created from the template record, CDD$TOP.CORPORATE.ADDRESS_RECORD.

```
VAX CDD Data Definition Language Utility  Version 3.0  5-MAR-1984  11:16:33:25  Page  1
Command Line: CDDL /RECOMPILE/VERSION/COPY_LIST CDD$TOP.SALES. CUSTOMER_RECORD
Source File:
  0001     DEFINE RECORD CDD$TOP.SALES.CUSTOMER_RECORD
  0002     DESCRIPTION IS
  0003          /* This record is of primary use to the marketing
  0004          department. It contains the names, addresses, and
  0005          phone numbers of all current customers. */.
  0006        CUSTOMER STRUCTURE.
  0007          NAME                    DATATYPE IS TEXT
  0008                                  SIZE IS 30 CHARACTERS.
  0009          ACCOUNT_NUMBER          DATATYPE IS UNSIGNED NUMERIC
  0010                                  SIZE IS 7 DIGITS.
  0011          ADDRESS
      T               ADDRESS STRUCTURE.
      T                   STREET      DATATYPE IS TEXT
      T                               SIZE IS 30 CHARACTERS.
      T                   CITY        DATATYPE IS TEXT
      T                               SIZE IS 30 CHARACTERS.
      T                   STATE       DATATYPE IS TEXT
      T                               SIZE IS 2 CHARACTERS.
      T                   ZIP_CODE STRUCTURE.
      T                       NEW         DATATYPE IS UNSIGNED NUMERIC
      T                                   SIZE IS 4 DIGITS
      T                                   BLANK WHEN ZERO.
      T                       OLD         DATATYPE IS UNSIGNED NUMERIC
      T                                   SIZE IS 5 DIGITS.
      T                   END ZIP_CODE STRUCTURE
      T               END ADDRESS STRUCTURE.
  0012                                CDD$TOP.CORPORATE.ADDRESS_RECORD.
  0013          TELEPHONE STRUCTURE.
  0014              AREA_CODE          DATATYPE IS UNSIGNED NUMERIC
  0015                                 SIZE IS 3 DIGITS.
  0016              NUMBER             DATATYPE IS UNSIGNED NUMERIC
  0017                                 SIZE IS 7 DIGITS.
  0018          END TELEPHONE STRUCTURE.
  0019        END CUSTOMER STRUCTURE.
  0020     END CUSTOMER_RECORD.
                                        1
%CDDL-S-RECOMPVER, record "CDD$TOP.SALES.CUSTOMER_RECORD;1" recompiled
and "CDD$TOP.SALES.CUSTOMER_RECORD;2" created in the CDD
```

SOURCE.DDL: The Source File for Examples in This Manual

A

```
DEFINE RECORD CDD$TOP.CORPORATE.ADDRESS_RECORD
    DESCRIPTION IS
        /* This record contains the standard format
        for addresses.  It provides the source from which all
        address fields in other record descriptions are copied. */.
    ADDRESS STRUCTURE.
        STREET                  DATATYPE IS TEXT
                                SIZE IS 30 CHARACTERS.
        CITY                    DATATYPE IS TEXT
                                SIZE IS 30 CHARACTERS.
        STATE                   DATATYPE IS TEXT
                                SIZE IS 2 CHARACTERS.
        ZIP_CODE STRUCTURE.
            NEW                 DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 4 DIGITS
                                BLANK WHEN ZERO.
            OLD                 DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 5 DIGITS.
        END ZIP_CODE STRUCTURE.
    END ADDRESS STRUCTURE.
END ADDRESS_RECORD.

DEFINE RECORD CDD$TOP.CORPORATE.EMPLOYEE_LIST
    DESCRIPTION IS
        /* This record contains the master list of all
        employees */.
    EMPLOYEE STRUCTURE.
        /* An employee's ID number is his
        or her social security number */
        ID                      DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 9 DIGITS.
        NAME STRUCTURE.
            LAST_NAME           DATATYPE IS TEXT
                                SIZE IS 15 CHARACTERS.
            FIRST_NAME          DATATYPE IS TEXT
                                SIZE IS 10 CHARACTERS.
```

(continued on next page)

```
            MIDDLE_INITIAL  DATATYPE IS TEXT
                            SIZE IS 1 CHARACTER.
        END NAME STRUCTURE.
        ADDRESS             COPY FROM
                            CDD$TOP.CORPORATE.ADDRESS_RECORD.
        DEPT_CODE           DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 3 DIGITS.
    END EMPLOYEE STRUCTURE.
END EMPLOYEE_LIST RECORD.

DEFINE RECORD CDD$TOP.CORPORATE.PRODUCT_INVENTORY
    DESCRIPTION IS
        /* This record is the primary location of inventory
        status information. */.
    INVENTORY STRUCTURE.
        STOCK STRUCTURE.
            /* RECORD_IDENTIFIER determines field type:
                 S --> In-stock record.
                 B --> Back-order record.
                 O --> Out-of-stock record. */
            RECORD_IDENTIFIER       DATATYPE IS TEXT
                                    SIZE IS 1 CHARACTER
                                    CONDITION FOR COBOL IS ON_HAND
                                        COBOL NAME "ON-HAND"
                                        VALUE IS "S"
                                    CONDITION FOR COBOL BACKORDER
                                        COBOL NAME "BACKORDER"
                                        VALUE IS "B"
                                    CONDITION FOR COBOL OUT_OF_STOCK
                                        COBOL NAME "OUT-OF-STOCK"
                                        VALUE IS "O"
                                    CONDITION FOR COBOL IS INVALID
                                        VALUES ARE "A", "C" THRU "N",
                                        "P" THRU "R", "T" THRU "Z".
            VARIANTS OF RECORD_IDENTIFIER.
                VARIANT VALUE IS "S".
                    IN_STOCK STRUCTURE.
                        PRODUCT_NO      DATATYPE IS TEXT
                                        SIZE IS 8 CHARACTERS.
                        DATE_ORDERED    DATATYPE IS DATE.
                        STATUS_CODE     DATATYPE IS BYTE.
                        QUANTITY        DATATYPE IS LONGWORD
                                        ALIGNED ON LONGWORD.
                        LOCATION        ARRAY 1:4
                                        DATATYPE IS TEXT
                                        SIZE IS 30 CHARACTERS.
                        UNIT_PRICE      DATATYPE IS LONGWORD SCALE -2.
                    END IN_STOCK STRUCTURE.
                END VARIANT.
                VARIANT VALUE IS "B".
                    BACK_ORDER STRUCTURE.
                        PRODUCT_NO      DATATYPE IS TEXT
                                        SIZE IS 8 CHARACTERS.
                        DATE_ORDERED    DATATYPE IS DATE.
                        STATUS_CODE     DATATYPE IS BYTE.
                        QUANTITY        DATATYPE IS LONGWORD
                                        ALIGNED ON LONGWORD.
```

(continued on next page)

```
                        SUPPLIER        ARRAY 1:4
                                        DATATYPE IS TEXT
                                        SIZE IS 30 CHARACTERS.
                        UNIT_PRICE      DATATYPE IS LONGWORD
                                        SCALE -2.
                    END BACK_ORDER STRUCTURE.
                END VARIANT.
                VARIANT VALUE IS "0".
                    OUT_OF_STOCK STRUCTURE.
                        PRODUCT_NO      DATATYPE IS TEXT
                                        SIZE IS 8 CHARACTERS.
                        DATE_LAST_SOLD DATATYPE IS DATE.
                    END OUT_OF_STOCK STRUCTURE.
                END VARIANT.
            END VARIANTS.
        END STOCK STRUCTURE.
    END INVENTORY STRUCTURE.
END PRODUCT_INVENTORY RECORD.

DEFINE RECORD CDD$TOP.PERSONNEL.STANDARDS.SALARY_RANGE
    DESCRIPTION IS
        /* This record stores minimum salaries
        for the five incremental salary levels within
        each of 200 job classifications.  */.
    SALARY_RANGE    ARRAY 200,5
                    DATATYPE IS UNSIGNED NUMERIC
                    SIZE IS 8 DIGITS 2 FRACTIONS.
END SALARY_RANGE RECORD.

DEFINE RECORD CDD$TOP.PERSONNEL.SERVICE.SALARY_RECORD
    DESCRIPTION IS
        /* This is the record containing salary
        information for all employees.  It is sensitive, and access
        is carefully restricted. */.
    SALARY STRUCTURE.
        EMPLOYEE_ID         DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 9 DIGITS.
        PAY STRUCTURE.
            JOB_CLASS       DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 3 DIGITS.
            INCR_LEVEL      DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 1 DIGIT.
            WEEKLY_SALARY   DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 6 DIGITS 2 FRACTIONS.
        END PAY STRUCTURE.
    END SALARY STRUCTURE.
END SALARY_RECORD RECORD.

DEFINE RECORD CDD$TOP.SALES.CUSTOMER_RECORD
    DESCRIPTION IS
        /* This record is of primary use to the
        marketing department.  It contains the names, addresses, and
        phone numbers of all current customers. */.
    CUSTOMER STRUCTURE.
        NAME                    DATATYPE IS TEXT
                                SIZE IS 30 CHARACTERS.
        ACCOUNT_NUMBER          DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 7 DIGITS.
```

(continued on next page)

```
        ADDRESS                 COPY FROM
                                CDD$TOP.CORPORATE.ADDRESS_RECORD.
        TELEPHONE STRUCTURE.
            AREA_CODE           DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 3 DIGITS.
            NUMBER              DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 7 DIGITS.
        END TELEPHONE STRUCTURE.
    END CUSTOMER STRUCTURE.
END CUSTOMER_RECORD.

DEFINE RECORD CDD$TOP.SALES.JONES.LEADS_RECORD
    DESCRIPTION IS
        /* This record is in the personal directory of
        Jones, a supervisor in the marketing department.  It contains
        information about prospective customers and the revenues that
        landing these customers might generate. */.
    LEADS_RECORD STRUCTURE.
        CONTACT_NAME        DATATYPE IS TEXT
                            SIZE IS 30 CHARACTERS.
        COMPANY             DATATYPE IS TEXT
                            SIZE IS 30 CHARACTERS.
        ADDRESS             COPY FROM
                            CDD$TOP.CORPORATE.ADDRESS_RECORD.
        TELEPHONE STRUCTURE.
            AREA_CODE       DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 3 DIGITS.
            NUMBER          DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 7 DIGITS.
        END TELEPHONE STRUCTURE.
        POTENTIAL_ANN_SALES     DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 8 DIGITS 2 FRACTIONS.
    END LEADS_RECORD STRUCTURE.
END LEADS_RECORD.

DEFINE RECORD CDD$TOP.SALES.SALES_RECORD.
    SALES STRUCTURE.
        CUSTOMER_NAME           DATATYPE IS TEXT
                                SIZE IS 30 CHARACTERS.
        ACCOUNT_NUMBER          DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 7 DIGITS.
        TRANSACTION_COUNT       DATATYPE IS UNSIGNED WORD
                                VALID FOR DATATRIEVE IF
                                    "TRANSACTION_COUNT > 0".
        TRANSACTION STRUCTURE   OCCURS 1 TO 99 TIMES
                                    DEPENDING ON TRANSACTION_COUNT.
            TRANS_DATE          DATATYPE IS DATE
                                EDIT_STRING FOR DATATRIEVE
                                IS "NN/DD/YY".
            ORDER_NUMBER        DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 10 DIGITS
                                NAME FOR COBOL IS "ORDER-NUMBER".
            AMOUNT              DATATYPE IS UNSIGNED NUMERIC
                                SIZE IS 8 DIGITS 2 FRACTIONS
                                INITIAL_VALUE IS 0
                                PICTURE FOR COBOL IS "9(6)V99".
                END TRANSACTION STRUCTURE.
    END SALES STRUCTURE.
END SALES_RECORD RECORD.
```

(continued on next page)

```
DEFINE RECORD _CDD$TOP.CDD$EXAMPLES.PERSONNEL.SERVICE.SALARY_RECORD
    DESCRIPTION IS
        /* This is the record containing salary
        information for all employees.  It is sensitive, and access
        is carefully restricted. Direct deposit information added
        5-JAN-1984.*/.
    SALARY STRUCTURE.
        EMPLOYEE_ID         DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 9 DIGITS.
        PAY STRUCTURE.
            JOB_CLASS       DATATYPE IS TEXT
                            SIZE IS 3 CHARACTERS.
            INCR_LEVEL      DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 1 DIGIT.
            WEEKLY_SALARY   DATATYPE IS UNSIGNED NUMERIC
                            SIZE IS 6 DIGITS 2 FRACTIONS.
            DIRECT_DEP      DATATYPE IS TEXT
                            SIZE IS 1 CHARACTER
                            VALID FOR DTR IF
                               DIRECT_DEP=Y OR DIRECT_DEP=N.
        END PAY STRUCTURE.
    END SALARY STRUCTURE.
END SALARY_RECORD RECORD.

DEFINE RECORD _CDD$TOP.CDD$EXAMPLES.PERSONNEL.STANDARDS.SALARY_RANGE
    DESCRIPTION IS
        /* This record stores minimum salaries
        for the four incremental salary levels within
        each of 150 Job classifications.  It reflects
        Personnel Policy effective 1/1/84. */.
    SALARY_RANGE     ARRAY 150,4
                     DATATYPE IS UNSIGNED NUMERIC
                     SIZE IS 8 DIGITS 2 FRACTIONS.
END SALARY_RANGE RECORD.
```

CDDL Syntax Skeleton B

B.1 DEFINE...END

```
DEFINE RECORD path-name

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

    field-description-statement

END  [path-name]   [RECORD].
     [given-name]
```

B.2 Field Description Statements

B.2.1 Elementary Field Description

```
[ /* text */ ]

{  *       }
{field-name}   field-attribute [field-attribute] ... .
```

B.2.2 STRUCTURE Field Description

```
[ /* text */ ]

{ * }
{field-name}    STRUCTURE [field-attribute] ... .

      field-description-statement
      [field-description-statement] ...

END [field-name] [STRUCTURE] .
```

B.2.3 COPY Field Description

```
[ /* text */ ]

field-name        COPY [FROM] path-name [ALIGNED clause] .
```

B.2.4 VARIANT Field Description

Format 1:

```
VARIANTS .
      VARIANT .
            field-description-statement
            [field-description-statement] ...
      END [VARIANT] .

      VARIANT .
            field-description-statement
            [field-description-statement] ...
      END [VARIANT] .
                        .
                        .
                        .
END [VARIANTS] .
```

Format 2:

```
VARIANTS  OF field-name .

     VARIANT  {VALUE  [IS]  }
              {VALUES [ARE] }  n1 [THRU n2] [n3 [THRU n4] ] ... .
          field-description-statement
          [field-description-statement] ...
     END [VARIANT] .

     VARIANT  {VALUE  [IS]  }
              {VALUES [ARE] }  n5 [THRU n6] [n7 [THRU n8] ] ... .
          field-description-statement
          [field-description-statement] ...
     END [VARIANT] .
                          .
                          .
                          .
END [VARIANTS] .
```

B.3 General Field Attributes

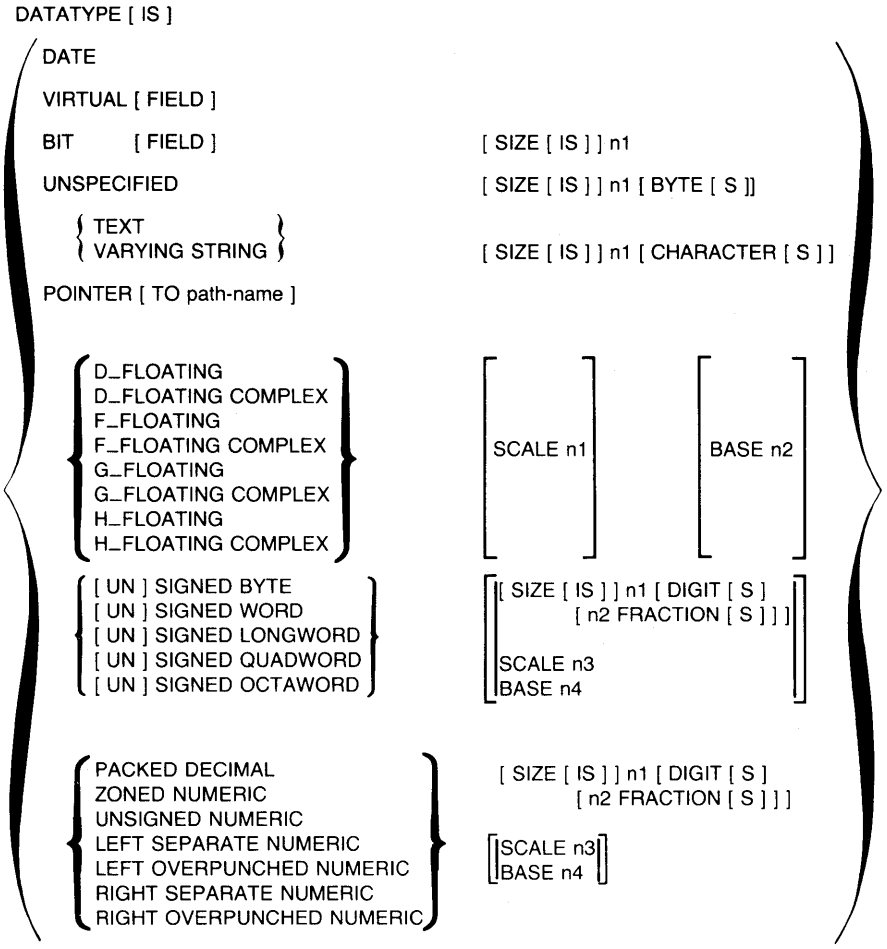
```
                  { BIT      }
                  { BYTE     }
ALIGNED [ON]      { WORD     }     [BOUNDARY]
                  { LONGWORD }
                  { QUADWORD }
                  { OCTAWORD }

[ROW_MAJOR   ]
[COLUMN_MAJOR]    ARRAY [n1 :] n2 [ [ n3 :] n4] ...

OCCURS n1 [ TIME [ S ] ]
    [ INDEXED FOR COBOL BY quoted-string [ ,... ] ]

OCCURS n1 TO n2 [ TIME [ S ] ] DEPENDING [ ON ] field-name
     [ INDEXED FOR COBOL BY quoted-string [ ,... ] ]

                  { complex-number      }
                  { fixed-point-number  }
INITIAL_VALUE [IS]{ floating-point-number }
                  { quoted-string       }
                  { hex-number          }
                  { octal-number        }
```



B.4 Facility-Specific Field Attributes

BLANK WHEN ZERO

COMPUTED BY { DTR / DATATRIEVE } AS quoted-string [quoted-string] ...

CONDITION FOR COBOL [IS] condition-name

[COBOL NAME [IS] quoted-string]

{ VALUE [IS] / VALUES [ARE] } n1 [THRU n2] [n3 [THRU n4]] ...

DEFAULT_VALUE FOR { DTR / DATATRIEVE } [IS] { fixed-point-number / quoted-string }

EDIT_CODE FOR RPG [IS] quoted-string

EDIT_STRING FOR { DTR / DATATRIEVE } [IS] quoted-string

EDIT_WORD FOR RPG [IS] quoted-string

JUSTIFIED RIGHT

MISSING_VALUE FOR { DTR / DATATRIEVE } [IS] fixed-point-number / quoted-string

NAME FOR { BASIC / COBOL / PL/I / RPG } [IS] quoted-string

PICTURE FOR { COBOL / DTR / DATATRIEVE / PL/I } [IS] quoted-string

QUERY_HEADER FOR { DTR / DATATRIEVE } [IS] quoted-string [quoted-string]...

QUERY_NAME FOR { DTR / DATATRIEVE } [IS] quoted-string

VALID FOR { DTR / DATATRIEVE } IF quoted-string [quoted-string]...

CDDL Error Messages C

This appendix lists the error messages generated by the CDDL compiler. After each message, there is an explanation of the message and of the action you should take to correct the problem.

For example, if you specify a negative number as the number of DIGITS in a DATATYPE clause, you receive the following message:

```
%CDDL-E-ILLNODIG, illegal number of digits
```

A CDDL error message contains the following elements:

- The facility name preceded by a percent sign (%) or a hyphen (-) and followed by a hyphen (-), such as %CDDL- or -CDDL-.
- The severity code followed by a hyphen (-). Table C-1 lists severity codes in order of increasing severity.

Table C-1: Explanation of Severity Codes

Code	Severity	Explanation
S	Success	Indicates that your command has executed successfully.
I	Information	Reports on actions taken by the software.
W	Warning	Indicates error conditions for which the compiler can take corrective action. You should check to make sure that this action has produced the results you want; the record definition placed in the dictionary may not be correct.
E	Error	Indicates conditions that are not fatal, but also are not correctable by the compiler. CDDL continues syntactic and semantic checking to detect as many errors as possible, but records with errors are not stored in the CDD. Therefore, you must correct any errors and compile any faulty record definitions again.
F	Fatal	Indicates that the compiler cannot continue syntactic checking and cannot store any definitions in the CDD. You must correct the error and compile the record definition again.

- The diagnostic error message name followed by a comma (,). This name identifies the message. In the following list of error messages, the messages are alphabetized by diagnostic error message name.
- The diagnostic error message. The message is a brief description of the problem. Error messages may contain string substitutions identifying the particular file names or path names in question. These string substitutions are indicated by angle brackets (< >) within a message. For example:

```
unexpected <token> deleted
```

If you received this message, CDDL would substitute the actual token for <token>.

Normally, you can correct CDDL errors by checking the source file and compiling again. IF YOU RECEIVE A FATAL ERROR INDICATING AN INTERNAL CDDL PROBLEM, OR IF YOU RECEIVE A CDD OR UTILITIES ERROR MESSAGE THAT IS NOT DOCUMENTED, you should immediately notify your system manager or the person responsible for maintaining the dictionary. If your facility has a service contract with DIGITAL, your system manager should submit a Software Performance Report (SPR) on the forms provided by DIGITAL. Be sure to include a VMS BACKUP copy of the dictionary file and a listing of the source file and commands that produced the error.

ABBREV, found <keyword> abbreviated as <abbreviation>

Explanation: CDDL issues this message when you abbreviate a keyword.

User Action: None.

ALLFAILED, none of the defined records was created

Explanation: CDDL could not compile any of the record definitions in the source file.

User Action: Refer to the other messages generated by the compiler for specific errors.

APPROXVAL, converted value is approximately <number>

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record description and CDD calculated an approximate value for a field's literal value within the record.

User Action: None.

ATTMLTDEF, field attribute conflicts with earlier declaration

Explanation: You have defined a field attribute more than once (for example, by assigning two query names to one field).

User Action: Correct the error and compile the record definition again.

ATTNOTDEF, DATATYPE clause not specified

Explanation: You have omitted a DATATYPE clause from an elementary field description statement.

User Action: Specify a data type and compile the record definition again.

AUDITFAIL, error occurred while creating audit entry

Explanation: The COPY field description statement automatically adds an entry to the history list of template records as they are copied. If the audit fails, the compiler issues this message. The record definition, however, is stored in the CDD.

User Action: None.

BADALIGN, illegal length of alignment filler field

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

BADCHAR, illegal character

Explanation: The parser has encountered an illegal character in the source file. If no other error messages appear, the parser has ignored the bad character and compiled the record definition successfully.

User Action: If the record compilation was unsuccessful, refer to the documentation for the accompanying error messages for explanation and suggested action.

BADCLAUSE, clause may prompt diagnostics when compiled with CDDL

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record description and the operation generated a clause that might cause the CDDL compiler to issue error messages.

User Action: None.

BADCPYLST, unable to expand copied template record source in listing

Explanation: CDDL tried unsuccessfully to expand a template record in the listing file.

User Action: Check to make sure that the template record has not been deleted from the CDD.

BADDATTIM, invalid date/time value

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record containing an invalid VMS absolute date and time value.

User Action: None.

BADDIMENS, dimension lower bound must be less than upper bound

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

BADEXTVAL, illegal CONDITION FOR COBOL VALUES — external values ignored.

Explanation: You used DMU/EXTRACT or CDDL/COPY_LIST to generate CDDL source for a record containing an illegal EXTERNAL value.

User Action: Correct the EXTERNAL value, recompile the record definition, and reenter DMU/EXTRACT or CDDL/COPY_LIST.

BADFIELD, syntax error in field name

Explanation: You have entered a field name incorrectly.

User Action: Correct the error and compile the record definition again.

BADFORMAT, does not conform to record description protocol version 4

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record that does not conform to version 4 of the CDD record description protocol.

User Action: Use the instructions at beginning of error message list in this appendix.

BADFRACTS, FRACTIONS clause conflicts with DIGITS clause

Explanation: You have specified a number of FRACTIONS greater than the number of DIGITS.

User Action: Correct the error and compile the record definition again.

BADIDENT, illegal identifier

Explanation: The parser encountered an apparent identifier such as a keyword or a path name, but the identifier was not legal. If no other error messages appear, the parser has ignored the illegal identifier and compiled the record definition successfully.

User Action: If the record compilation was unsuccessful, refer to the documentation for the accompanying error messages for explanation and suggested action.

BADLENGTH, length for structure is inconsistent with prior declaration

Explanation: You have given a STRUCTURE field an explicit data type and length, but that length is less than the total length of the STRUCTURE's subfields.

User Action: Correct the error and compile the record definition again.

BADOCCURS, maximum number of occurrences must be greater than 0

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

BADOVRLAY, VARIANTS field does not have an OVERLAY data type

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

BADPROTCL, <quoted string> is not a legal dictionary object

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

BADTAGVAR, illegal tag variable

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

BASEGTR1, base must be greater than 1

Explanation: You have illegally specified a number with a base of 0 or 1.

User Action: Correct the error and compile the record definition again.

BASENOT10, scale factor not applied to non-decimal values

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record description containing literal values for fields with a base other than ten. The CDD did not apply any scaling factor to these fields.

User Action: None.

CANTTRANS, unsupported language feature in record <given name>

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

CDDERROR, error encountered while creating record

Explanation: Another CDD message identifying the problem always follows this message.

User Action: Refer to the documentation for the accompanying error message for explanation and user action.

CDDOBJERR, CDD error at object <given name>

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

COBINDEX, index for COBOL illegal in this context — value ignored

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record description containing an invalid attribute value. The CDD substituted a default value for the incorrect attribute value.

User Action: None.

COMPBYDTR, COMPUTED BY DTR clause allowed only on VIRTUAL fields

Explanation: You have illegally included the COMPUTED BY clause in the description of a field with a data type other than VIRTUAL FIELD.

User Action: Correct the error and compile the record definition again.

COMPLEX, value conversion error on complex data type

Explanation: CDDL could not convert an INITIAL VALUE specification to a complex number.

User Action: Check your INITIAL VALUE specification and make sure you have specified it correctly.

CONVERR, value conversion error

Explanation: CDDL could not convert a value.

User Action: Check your value specification and make sure you have specified it correctly.

COPYERROR, error encountered while copying record <quoted string>

Explanation: CDDL was unable to copy a template record. An accompanying message explains the problem.

User Action: If the accompanying message is a CDDL message, read the explanation of that message elsewhere in this appendix. If it is a CDD message, see Appendix D of the *VAX Common Data Dictionary Utilities Reference Manual*. If it is a system message, see the VMS documentation set.

COPYNOLIS, the /COPY_LIST qualifier conflicts with /NOLISTING qualifier

Explanation: You specified /COPY_LIST and /NOLISTING on the same CDDL command line.

User Action: Choose either /COPY_LIST or /NOLISTING and compile the record definition again.

COPYNONAM, error encountered while copying record

Explanation: CDDL was unable to copy a template record. An accompanying message explains the problem.

User Action: If the accompanying message is a CDDL message, read the explanation of that message elsewhere in this appendix. If it is a CDD message, see Appendix D of the *VAX Common Data Dictionary Utilities Reference Manual*. If it is a system message, see the VMS documentation set.

DEFNOTSET, error encountered while setting default directory

Explanation: An error occurred when CDD tried to set CDD$DEFAULT.

User Action: Check your definition of CDD$DEFAULT; it is illegal or points to an inaccessible directory or subdictionary.

DEFVALUE, default value substituted for invalid CDD attribute value

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record description containing an invalid attribute value. The CDD substituted a default value for the incorrect attribute value.

User Action: None.

DEPENDING, depending item illegal in this context — value ignored

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

DELETETOK, unexpected <token> deleted

Explanation: The CDDL has deleted a superfluous keyword or token.

User Action: None.

DEPINARRY, field named in DEPENDING clause is within an array

Explanation: You have illegally defined the field that governs the number of occurrences in an OCCURS. . .DEPENDING clause as part of an array.

User Action: Edit the source file to define this field as an elementary field, not as part of an array. Then compile the record definition again.

DEPNOTBEF, depending item not found before field definition

Explanation: You must define the field governing the number of occurrences in an OCCURS. . .DEPENDING clause in the record before you declare the array.

User Action: Correct the error and compile the record definition again.

DEPNOTELM, depending item is not an elementary field

Explanation: You have illegally defined the field governing the number of occurrences in an OCCURS. . .DEPENDING clause as part of an array or a structure field.

User Action: Correct the error and compile the record definition again.

DEPNOTFND, field named in DEPENDING clause not found

Explanation: You have referred to a nonexistent field in an OCCURS. . .DEPENDING clause.

User Action: Edit the source file to define the field governing the number of occurrences as an elementary field, not as part of an array. Then compile the record definition again.

DEPNOTUNQ, depending item field name is not unique

Explanation: The specification for the field governing the number of occurrences in an OCCURS. . .DEPENDING clause matches more than one field.

User Action: Edit the source file to make the specification unique, perhaps by fully qualifying the field name. Then compile the record definition again.

DSCNOTTRM, DESCRIPTION clause not terminated before EOF

Explanation: You have omitted the */ terminator from a DESCRIPTION IS clause, and the parser has read the entire source file looking for it.

User Action: Edit the file and insert */ at the appropriate place. Then compile the source file again.

DTYPEKWRD, use keyword DATATYPE rather than TYPE

Explanation: You used TYPE instead of DATATYPE to specify a data type. CDDL compiled the definition successfully, but warns that DATATYPE is the preferred keyword.

User Action: None.

DUPNAME, field name is not unique

Explanation: Two or more fields within the same STRUCTURE have the same field name. This is a legal CDDL construction, but it may cause problems. Some language compilers, for example, do not allow name duplication within a STRUCTURE.

User Action: Consider giving each field within the STRUCTURE a unique name.

ELEMFIELD, tag variable must be an elementary field

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record description containing a tag variable that is not an elementary field.

User Action: Use the instructions at beginning of error message list in this appendix.

EMPTYREC, a record must contain a field description

Explanation: You have defined a record that does not contain a field description. Every record definition must contain at least one field description.

User Action: Add at least one field description to the record definition and compile it again.

EMPTYSTRC, a structure must contain at least one sub-field description

Explanation: You have used a STRUCTURE field description statement but have not included a subordinate field description. Legal subordinate fields include elementary, STRUCTURE, COPY, and VARIANTS fields.

User Action: Add a subordinate field description to your STRUCTURE field description and compile the record definition again.

EXTRANATT, extraneous CDD attribute encountered — value ignored

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

EXTTAGVAL, a tag value cannot be expressed as a COBOL EXTERNAL value

Explanation: You have illegally used a COBOL EXTERNAL quoted string as a tag value.

User Action: Re-enter a legal tag value.

FLTVALIMP, decimal representation of floating point value is imprecise

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record description containing a floating point value. When converting the floating point value to its decimal equivalent, the CDD generated only an approximate value.

User Action: None.

HEXCONERR, conversion error — value could not be converted to hex

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

IGNORETAG, unable to use tag variable — tag values ignored

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

ILLBOUND, lower bound cannot exceed upper bound

Explanation: You have specified an array with a lower bound greater than the upper bound.

User Action: Correct the error and compile the record definition again.

ILLCOBCND, illegal COBOL condition in template record

Explanation: You have tried to copy into a record a template record with illegal COBOL condition attributes.

User Action: Correct the attributes, recompile the definition, and copy the template record again.

ILLDTYPE, illegal or unsupported data type — code is <number>

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

ILLEXTVAL, duplicate INITIAL_VALUE clauses — all but one ignored

Explanation: You used DMU/EXTRACT or CDDL/COPY_LIST to generate CDDL source for a record containing duplicate INITIAL_VALUE clauses. The EXTERNAL clause was ignored.

User Action: None.

ILLFQNAME, illegal fully qualified name

Explanation: You have illegally included a password in a fully qualified name.

User Action: Correct the syntax and try again.

ILLHEXOCT, illegal hex or octal number

Explanation: The parser encountered an apparent hexadecimal or octal number because of the prefix (%X or %O), but a legal hexadecimal or octal string did not follow. If no other error message appears, the parser has ignored the illegal number and attempted to compile the record definition.

User Action: If the record compilation was unsuccessful, refer to the documentation for the accompanying error messages for explanation and suggested action.

ILLINDXNM, index name defined elsewhere in record definition

Explanation: You specified the same string value as a COBOL index name and as a field name or COBOL-specific field name in a record definition. You must use a unique string value for COBOL index names.

User Action: Correct the error and compile the record definition again.

ILLINTVAL, if scale is not 0 and base is not 10, value must be expressed in hex or octal

Explanation: You have entered an illegal initial value. If you have used a scale other than 0 and a base other than 10, you must identify the initial value as either an octal or hexadecimal number.

User Action: Correct the error and compile the record definition again.

ILLLITERL, does not conform to record description protocol version 4

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

ILLMINMAX, minimum cannot exceed maximum number of occurrences

Explanation: In an OCCURS. . .DEPENDING clause, you have specified a minimum number of occurrences greater than the maximum number.

User Action: Correct the error and compile the record definition again.

ILLNAMCHR, a field name contains a character other than A-Z, 0-9, $, or _

Explanation: You have used an illegal character in a field name. A field name can contain only letters, integers, $, or _.

User Action: Correct the error and compile the record definition again.

ILLNAMSIZ, a field name's length is 0 or greater than 31

Explanation: The number of characters in a field name must be greater than 0 but less than 32.

User Action: Correct the error and compile the record definition again.

ILLNCHAR, illegal character found in numeric input

Explanation: During initial value, tag value, or condition value conversion, the parser encountered an illegal character, such as a period.

User Action: Correct the error and compile the record definition again.

ILLNODIG, illegal number of digits

Explanation: You have specified an illegal number of DIGITS. DIGITS specifications must be greater than 0 but less than 32.

User Action: Correct the error and compile the record definition again.

ILLNUMBER, illegal number

Explanation: You have entered an illegal number (for example, 2E with no scale specified). If no other error message appears, the parser has ignored the illegal number and compiled the record definition successfully.

User Action: If the record compilation was unsuccessful, refer to the documentation for the accompanying error messages for explanation and suggested action.

ILLOCCURS, number of occurrences must be greater than zero

Explanation: In an OCCURS clause, you have specified a number of occurrences less than or equal to 0.

User Action: Correct the error and compile the record definition again.

ILLOVPNCH, illegal overpunched character

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

ILLQUOSTR, illegal quoted string

Explanation: The parser has encountered an illegal string in the source file. Make sure, for example, that all open quotations are subsequently closed and that quoted strings do not exceed a single line of CDDL source text.

User Action: Correct the error and compile the record definition again.

ILLRELVER, CDDL does not support relative version numbers

Explanation: You specified a relative version in a path name used by CDDL, but CDDL recognizes only absolute version numbers. To specify a version other than the highest to CDDL, you must use an absolute version number.

User Action: Correct the error and compile the record definition again.

ILLSCALE, illegal scale value

Explanation: You have specified an illegal SCALE. SCALE specifications cannot be greater than 127 nor less than -128.

User Action: Correct the error and compile the record definition again.

ILLVERNUM, version number cannot be used on a directory name

Explanation: You specified a version number with a directory's given name. Because CDD does not support multiple versions of dictionary directories, you cannot use a version number with directories.

User Action: Correct the error and compile the record definition again.

ILLVFSTRC, STRUCTUREs cannot have the virtual field datatype

Explanation: You have tried to specify a virtual field datatype in a STRUCTURE field description statement.

User Action: Modify and recompile the record definition.

ILLVIRFLD, STRUCTUREs cannot be VIRTUAL FIELDS — UNSPECIFIED assumed

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

INSERTDOT, inserted "." at the end of the previous line

Explanation: CDDL has inserted a missing period at the end of a source line.

User Action: None.

INSERTTOK, inserted <token> before <token>

Explanation: CDDL has inserted a missing source token.

User Action: None.

INSMATCH, <keyword> inserted to match <keyword> inserted earlier

Explanation: You omitted a keyword. CDDL has inserted it.

User Action: None.

INTERROR, internal CDDL error

Explanation: This message indicates a problem with the compiler.

User Action: Use the instructions at beginning of error message list in this appendix.

INV1STCHR, the first character of a field name is not A-Z

Explanation: You have begun the name of a field with an illegal character. A field name must begin with a letter.

User Action: Correct your error and compile the record definition again.

INVALLIT, invalid literal — length is zero

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

INVALREC, record to be copied is invalid

Explanation: You have tried to copy an incomplete template record definition.

User Action: Examine the template, supply the missing attributes, and compile the record definition again. Then you can copy the complete template record definition with the COPY field description statement.

INVBITLEN, invalid length for bit field

Explanation: The legal range for the length of a field with the BIT data type is from 1 to 65,535 bits. You have specified a BIT field outside of this range.

User Action: Correct the error and compile the record definition again.

INVDATTIM, invalid date/time string — value set to 17-NOV-1858 00:00

Explanation: You have specified an invalid quoted string as an INITIAL, CONDITION, or TAG value whose data type is defined as DATE. When a quoted string is invalid, CDDL sets the date or time field to binary zero. The field then yields the Smithsonian base date and time, 17-NOV-1858 00:00.

User Action: Correct the error and compile the record definition again.

INVFLDLEN, field's length is invalid

Explanation: You have specified an invalid length for a field. Text, unspecified, and varying string fields must have a length greater than 0 but less than 65,536 characters.

User Action: Correct the error and compile the record definition again.

INVLENGTH, field length is not multiple of 8 bits — extra bits ignored

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description. The generated CDDL source may not accurately reflect the record description.

User Action: Examine the source text of the record description to ensure the correctness of the generated CDDL source.

INVLSTCHR, the last character of a field name is not A-Z or 0-9

Explanation: You have ended a field name with an illegal character. A field name must end with a letter or an integer.

User Action: Correct the error and compile the record definition again.

JUSTIFIED, JUSTIFIED RIGHT clause allowed only with TEXT or UNSPECIFIED fields

Explanation: You have attempted to use the JUSTIFIED RIGHT clause in the description of a field whose data type is something other than TEXT or UNSPECIFIED.

User Action: Correct the error and compile the record definition again.

KEYWDREPL, replaced <keyword> with <keyword>

Explanation: You have used an incorrect keyword, and CDDL has replaced it with the correct one.

User Action: None.

LANGNAMES, language specific names cannot be used with unnamed fields

Explanation: You have attempted to specify a language-specific name for an unnamed field. Only named fields can have language-specific names.

User Action: Correct the error and compile the record definition again.

LINETRUNC, output buffer overflow — line truncated

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record description, and a line of the generated source exceeds 255 characters.

User Action: Examine the source text of the record description to learn what should be in the truncated line.

MLTJUSTRT, JUSTIFIED RIGHT clause specified more than once

Explanation: You have specified JUSTIFIED RIGHT more than once for a single field.

User Action: Correct the error and compile the record definition again.

MULTARRAY, array clause specified more than once

Explanation: You have specified more than one OCCURS. . .DEPENDING, ARRAY, or OCCURS clause for a single field.

User Action: Correct the error and compile the record definition again.

MULTBLANK, BLANK WHEN ZERO clause specified more than once

Explanation: You have defined a single field as BLANK WHEN ZERO more than once.

User Action: Correct the error and compile the record definition again.

MULTDTYPE, DATATYPE clause specified more than once

Explanation: You have specified more than one data type clause for a single field.

User Action: Correct the error and compile the record definition again.

MULTSCALE, SCALE clause conflicts with earlier declaration

Explanation: Within a single DATATYPE clause you have specified SCALE more than once, or you have specified both SCALE and FRACTIONS and their values are inconsistent.

User Action: Correct the error and compile the record definition again.

MULTVALIS, VALUE IS clause specified more than once

Explanation: You have illegally specified more than one VALUE IS or VALUES ARE clause within one VARIANT field description.

User Action: Correct the error and compile the record definition again.

NEWVERCRE, new version <given name> created in the CDD

Explanation: You created an additional version of a dictionary object.

User Action: None.

NODEPITEM, depending item fieldname does not exist within the record

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

NODTYPATT, field's data type attribute is missing

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

NOLENGATT, field's length attribute is missing

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

NOMATCH, name mismatch in "END" statement

Explanation: The name you specified in the END statement does not match the name you specified in the DEFINE or STRUCTURE statement.

User Action: Correct the error and compile the record definition again.

NOROOTATT, does not conform to record description protocol version 4

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

NOTCDDL, record <quoted string> not created by CDDL, cannot be recompiled

Explanation: You have tried to recompile a definition created by a facility (such as VAX DATATRIEVE) other than CDDL. CDDL does not support this operation.

User Action: None.

NOTCDDREC, object is not a record

Explanation: With the COPY field description statement, you can copy dictionary objects only if the CDD type is CDD$RECORD. You tried to use COPY with another type.

User Action: None.

NOTCOMPLX, complex literal must be used with complex data type

Explanation: You have incorrectly attempted to specify a complex number as the initial, tag, or condition value of a field whose data type is not one of the FLOATING COMPLEX data types.

User Action: Correct the error and compile the record definition again.

NOTFIXED, fixed point literal not allowed in this context

Explanation: You have specified a fixed point number as the initial, tag, or condition value of a field whose data type cannot be expressed as a fixed point number.

User Action: Edit the source file to change the initial, tag, or condition value or to define a compatible data type. Then compile the record definition again.

NOTFLOAT, floating point literal not allowed in this context

Explanation: You have specified a floating point number as the initial, tag, or condition value of a field whose data type cannot be expressed as a floating point number.

User Action: Edit the source file to change the initial, tag, or condition value or to define a compatible data type. Then compile the record definition again.

NOTRECDEF, object <given name> is not a record definition

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a dictionary object that is not a CDD$RECORD.

User Action: Be sure the path name you specify identifies a CDD record description and enter the command again.

NOTRECORD, object <quoted string> is not a record

Explanation: The CDDL/RECOMPILE command allows you to recompile objects only if the type of the object is CDD$RECORD. You have tried to use the CDDL/RECOMPILE command with another type.

User Action: None.

NOTREPLCE, object to be replaced is not a record

Explanation: The /REPLACE qualifier to the compile command allows you to replace dictionary objects only if the type is CDD$RECORD. You tried to use /REPLACE with another type.

User Action: None.

NOTSTRING, string literal not allowed in this context

Explanation: You have specified an ASCII string as the initial, tag, or condition value of a field whose data type cannot be expressed as an ASCII string.

User Action: Edit the source file to change the initial, tag, or condition value or to specify a compatible data type. Then compile the record definition again.

NUMCONERR, conversion error — number could not be converted to decimal

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record description containing a value the CDD could not convert to a decimal equivalent.

User Action: None.

OBJEXTUVS, another version already exists — record not created

Explanation: You tried to create a record definition with the same path name as an already existing dictionary object, but you did not specify /VERSION.

User Action: Choose a different given name, or use the /VERSION qualifier.

OBJEXTVNS, object already exists — not superseded

Explanation: You tried to create a record definition with the same path name and version number as an already existing dictionary object.

User Action: Choose a different given name or version number for the object.

OVERFLOW, value conversion overflow

Explanation: You have specified a numeric value that is too large for the defined field.

User Action: Specify a smaller value, or redefine the length of the field and recompile the record.

PSSWDSYN, syntax error in password specification

Explanation: You supplied an invalid password specification. Passwords contain from 1 to 64 printable ASCII characters, including space and tab, but excluding open parenthesis [(], close parenthesis [)], and period [.].

User Action: Correct the syntax and compile the record definition again.

QUERYNAME, QUERYNAME clause cannot be used for unnamed fields

Explanation: You have attempted to specify a VAX DATATRIEVE QUERY_NAME for an unnamed field. You can specify query names only for named fields.

User Action: Correct the error and compile the record definition again.

RECNOTCRE, error in record definition — record not created

Explanation: The record defined in the source file record was not created because of an error or errors.

User Action: Use the specific error messages you received to edit the source file.

RECNOTFND, record to be replaced not found

Explanation: The /REPLACE qualifier to the COMPILE command allows you to replace existing record definitions. This warning indicates that you tried to replace a record definition that does not exist. The new definition, however, has been stored in the CDD.

User Action: None.

RECOMPACL, /VERSION qualifier must be specified to use /[NO]ACL qualifier

Explanation: In order to use either /ACL or /NOACL with CDDL/RECOMPILE, you must use the version qualifier. Otherwise, the newly created record definition copies the access control list of the definition it replaces.

User Action: If you want the newly created object to have no access control list, use the /VERSION qualifier.

RECOMPERR, error encountered while recompiling record <quoted string>

Explanation: An error occurred when CDDL was trying to recompile a record. This message is followed by another that explains the problem.

User Action: Refer to the documentation for the accompanying error message for explanation and suggested action.

RECOMPVER, record <given name> recompiled and <given name> created in the CDD

Explanation: CDDL recompiled an existing record description, creating an additional version of it.

User Action: None.

RECORDCRE, record <given name> created in the CDD

Explanation: CDDL created a record description and inserted it in the dictionary.

User Action: None.

RELVERNUM, relative version number on path name is being ignored

Explanation: You included a relative version number as part of a path name, but CDDL recognizes only absolute version numbers. CDDL created the record definition with version number one higher than the previous highest.

User Action: If you want the version number you specified, you can change it with the DMU RENAME command.

REPLACACL, the /[NO]ACL qualifier conflicts with the /REPLACE qualifier

Explanation: When you use CDDL/REPLACE, the new object always copies the access control list of the object it replaces. You cannot specify /ACL or /NOACL with CDDL/REPLACE.

User Action: Reenter the command without the /NOACL qualifier.

RMSERROR, RMS error in file <quoted string>

Explanation: This message is followed by an RMS message explaining why CDDL was unable to open, close, read, or write a file.

User Action: Refer to the *VAX/VMS System Message and Recovery Procedures Manual*.

SOMEFAIL, some of the defined records were not created

Explanation: Some of the records defined in your source file were not compiled because of an error or errors.

User Action: Remove the successful definitions from the file, correct errors in those remaining, and compile the file again.

SPELLCORR, identifier <keyword> replaced with <keyword> by the spelling corrector

Explanation: You have misspelled a keyword, but CDDL has corrected the spelling.

User Action: None.

STRCONERR, conversion error — string could not be converted to text

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record description containing an invalid VMS absolute date and time value. The CDD displays the value in hexadecimal, not as an ASCII date and time specification.

User Action: None.

STRMERGED, merged <string> and <string> to form <keyword>

Explanation: You have left a space in the middle of a keyword, and CDDL has joined the two portions.

User Action: None.

STRUCKWRD, use keyword STRUCTURE rather than GROUP

Explanation: You used GROUP instead of STRUCTURE to specify a structure field. The CDDL compiled the definition successfully, but warns that STRUCTURE is the preferred keyword.

User Action: None.

SYNTAX, syntax error near <quoted string>

Explanation: You have a syntax error in the CDDL command line. An accompanying message explains the problem.

User Action: Refer to the documentation for the accompanying error message for explanation and suggested action.

TAGINARRY, field named in tag variable clause is within an array

Explanation: You have incorrectly defined the tag variable as part of an array.

User Action: Edit the source file to make sure the tag variable is defined as an elementary field that is not part of an array. Then compile the record definition again.

TAGNOTALL, VALUE IS clause requires tag variable clause

Explanation: You have not specified a tag variable in the VARIANTS statement, but you have specified a tag value for one of the variants.

User Action: Edit the source file to make usage of the tag variable and the tag value consistent. Then, compile the record definition again.

TAGNOTBEF, tag variable not found before field definition

Explanation: You must define the tag variable field in the record before you include the VARIANTS OF statement.

User Action: Correct the error and compile the record definition again.

TAGNOTELM, tag variable is not an elementary field

Explanation: You have either defined the tag variable as part of an array or you have defined it as a field that is not elementary. The tag variable field must be elementary and cannot be part of an array.

User Action: Correct the error and compile the record definition again.

TAGNOTFND, field named in tag variable clause not found

Explanation: You have referred to a nonexistent field in a tag variable clause.

User Action: Edit the source file to make sure the tag variable is defined as an elementary field, not as part of an array. Then compile the record definition again.

TAGNOTUNQ, tag variable field name is not unique

Explanation: Your tag variable specification matches more than one field in the record.

User Action: Edit the source file to make the specification unique, perhaps by fully qualifying the field name. Then compile the record definition again.

TAGREQUIR, tag variable clause required

Explanation: If you specify a tag variable with a VARIANTS OF statement, each variant must declare a value or values with the VALUE IS or VALUES ARE clauses. You have omitted one or more of these tag value clauses.

User Action: Correct the error and compile the record definition again.

TOOFEWPAR, too few or missing parameter

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a dictionary object that is not a CDD$RECORD.

User Action: Be sure the path name you specify identifies a CDD record description and enter the command again.

TOOMANPAR, too many or conflicting parameters

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a dictionary object that is not a CDD$RECORD.

User Action: Be sure the path name you specify identifies a CDD record description and enter the command again.

TOOMANVAL, no more than two values permitted in a range

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

TRUNCATIO, value conversion truncation error

Explanation: The value you specified for an initial, tag, or condition value was too large to fit in the field as defined.

User Action: Specify a smaller value, or redefine the field and recompile the record.

UNDERFLOW, value conversion underflow

Explanation: The value you specified for an initial, tag, or condition value was too small to fit in the field as defined.

User Action: Specify a different value, or redefine the field and recompile the record.

UNEXPTOK, found <token> when expecting one of <token>

Explanation: CDDL is unable to determine what you were defining when it encountered a syntax error.

User Action: Correct the error and compile the record definition again.

UNPRINTCH, text string contains one or more unprintable characters

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for a record description containing an unprintable text string. The CDD displays the string's hexadecimal equivalent.

User Action: None.

UNSPROTCL, object <object name> does not conform to record description protocol version 4

Explanation: This object does not conform to the protocol used to describe CDD record definitions.

User Action: Use the instructions at beginning of error message list in this appendix.

VALCONERR, conversion error — value could not be converted

Explanation: You used DMU EXTRACT/RECORD or CDDL/COPY_LIST to generate CDDL source for an invalid record description.

User Action: Use the instructions at beginning of error message list in this appendix.

VALUEOVFL, value overflow, value set to zero

Explanation: An integer you specified in the source file is too large to fit into a 32-bit longword. CDDL converts all integers to longwords.

User Action: None.

VALTRUNC, overflow occurred during conversion — value truncated

Explanation: You have specified a value that is too large for the defined field. The hexadecimal or octal number has been truncated.

User Action: None.

VERREPLAC, the /VERSION qualifier conflicts with the /REPLACE qualifier

Explanation: You specified /VERSION and /REPLACE on the same CDDL command line.

User Action: Correct the error and enter the command again.

VIRTFIELD, VIRTUAL field cannot have INITIAL or CONDITION value clauses

Explanation: You have assigned an initial or condition value to a VIRTUAL FIELD. Virtual fields take up no space in the record and cannot store values. They are valid only when used with the COMPUTED BY DATATRIEVE clause.

User Action: Correct the error and compile the record definition again.

CDDL Reserved Words D

Certain CDDL keywords are reserved. You cannot use CDDL reserved words in field names or a path name within a CDDL source file. CDDL reserved words are:

END
FROM
GROUP
IS
ON
RECORD
SIZE
STRUCTURE
VARIANTS

Although you cannot use these reserved words as path names in CDDL source files, you can use them as given names within path names. For example, the following statements are legal within CDDL source files because they use CDDL reserved words only as *parts* of longer path names:

```
DEFINE RECORD CDD$TOP.TEST.STRUCTURE.

END PAYROLL.SIZE.RECORD.

COPY FROM ON.ON.ON.
```

The following statements are illegal within CDDL source files because they use CDDL reserved words as the *entire* path name:

```
DEFINE RECORD FROM.

END SIZE.

COPY FROM ON.
```

Additional CDDL Notes E

E.1 Support of the VAX Language-Sensitive Editors (LSE)

Version 3.3 and later of CDD supports the VAX Language-Sensitive Editor (LSE). If the VAX Language-Sensitive Editor is installed on your system, you can use this feature to help write, compile, and debug CDDL definitions. The CDDL Language-Sensitive Editor provides templates and menus to walk you through CDDL options and syntax. It is especially useful for users unfamiliar with CDDL.

To invoke the Language-Sensitive Editor, type LSE at the DCL prompt. The following command, for example, creates a file ADDRESS.CDDL and displays a CDDL record definition template to guide you through the process of describing a CDDL record:

```
$ LSE ADDRESS.CDDL
```

When LSE compiles your source definition, it expects a file type of .CDDL. The CDDL compiler now recognizes both file types .CDDL and .DDL.

E.2 /DIAGNOSTICS Qualifier for CDDL Command

The /DIAGNOSTICS qualifier with the CDDL command creates a diagnostics file that lists errors occurring during compilation. /DIAGNOSTICS is designed for use from the LSE environment. /DIAGNOSTICS lists errors in a file that has the default name of your definition file and the extension .DIA. The diagnostic file is reserved for use by DIGITAL. LSE uses the diagnostic file to display diagnostic messages and to position the cursor on the line and column where a source error exists.

You cannot use /DIAGNOSTICS with CDDL/RECOMPILE.

For complete information on using LSE, see the *VAX Language-Sensitive Editor User's Guide*.

E.3 The CDDL ALIGNED Clause

Be careful when you use the CDDL ALIGNED clause.

- You should not use the ALIGNED clause in template records. When CDDL stores the template record, the position of an aligned field is fixed within the record and is not changed when the record is copied into another record definition. Therefore, the newly created field may not align properly in the new record definition.
- Records created with the ALIGNED clause using previous versions of CDDL may not have aligned fields properly. CDD Version 3.1 corrected this alignment problem. However, if you recompile the records using the ALIGNED clause, data already stored will no longer match the recompiled data definition.

Index

In this index, a page number followed by a "t" indicates a table reference. A page number followed by an "e" indicates an example reference. A page number followed by an "f" indicates a figure reference.

A

B

C

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local DIGITAL subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local DIGITAL subsidiary or approved distributor
Internal[1]	————	SDC Order Processing - WMO/E15 *or* Software Distribution Center Digital Equipment Corporation Westminster, Massachusetts 01473

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

VAX Common Data Dictionary
Data Definition Language
Reference Manual
AA–K085D–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

What I like best about this manual is _____

_____

What I like least about this manual is _____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear - Fold Here

Cut Along Dotted Line

Reader's Comments

VAX Common Data Dictionary
Data Definition Language
Reference Manual
AA–K085D–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

What I like best about this manual is _____

_____

What I like least about this manual is _____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear - Fold Here

Cut Along Dotted Line

Reader's Survey

VAX Common Data Dictionary
Data Definition Language
Reference Manual
AA-K085D-TE

1. How useful are the following methods for finding information in this manual?

	Most	Very	Moderately	Not Very	Not at All
Table of contents	☐	☐	☐	☐	☐
Divider pages (if applicable)	☐	☐	☐	☐	☐
Index (circle: book or master)	☐	☐	☐	☐	☐
Other (specify) _____	☐	☐	☐	☐	☐

2. What feature do you most want to see improved in this manual? Why?

_____

3. How helpful are these sources when you use the software this manual describes?

	Most	Very	Moderately	Not Very	Not at All
Handbook or user's guide	☐	☐	☐	☐	☐
Introduction or overview	☐	☐	☐	☐	☐
Reference manual	☐	☐	☐	☐	☐
Quick reference guide	☐	☐	☐	☐	☐
Online help	☐	☐	☐	☐	☐
Online tutorial (if available)	☐	☐	☐	☐	☐
Other: colleague, telephone support services (specify) _____	☐	☐	☐	☐	☐

4. What business tasks are you using the software described by this manual to solve (for example: billing, funds transfer, report writing)?

_____

5. Please estimate, if you can, how long the following VAX Information Architecture products have been used at your site:

VAX ACMS _____	VAX CDD/Plus _____	VAX DATATRIEVE _____
VAX Data Distributor _____	VAX DBMS _____	VAX RALLY _____
VAX Rdb/VMS _____	VAX SQL _____	VAX TEAMDATA _____
VAX TDMS _____	VIDA with IDMS/R _____	

6. This release of VAX Information Architecture documentation uses a 7x9 format for quick reference guides. Do you prefer such books in a 7x9 or a 4x8 pocket guide format? _____

Thank you for your assistance.

May we contact you at work for further information? ☐ Yes ☐ No

Name _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

Do Not Tear - Fold Here and Tape

digital™

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK02–2/N53
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear - Fold Here