# CDF
## User's Guide

Version 2.5, January 10, 1995

National Space Science Data Center

National Space Science Data Center
NASA/Goddard Space Flight Center
Greenbelt, Maryland 20771 (U.S.A.)

NSI/DECnet — NSSDCA::CDFSUPPORT
Internet — cdfsupport@nssdca.gsfc.nasa.gov

# Contents

# List of Figures

# List of Tables

# Preface

### About This Document

This document is intended to serve as both a user's guide and reference manual for the Common Data Format (CDF). As such, it provides a primer for introducing the novice reader to the concepts of CDF as well as a reference manual for the advanced user.[1] However, it does not serve as a cookbook for the proper methods of designing a CDF.

The very first questions usually asked by a reader are: What is CDF?, How is CDF used?, and How is CDF useful for me? Although the reader will find the answers to these questions in this document, we provide here a brief description of the conceptual basis of CDF in order to provide a proper perspective when reading the remainder of this document.

### What is CDF?

CDF, in its most basic terms, is a conceptual data abstraction for storing, manipulating, and accessing multidimensional data sets. We refer to CDF as a data abstraction because we never discuss the actual physical format in which data sets are stored. Instead, we describe the form of the data sets and the means (interface) by which they may be manipulated. This important difference from traditional physical file formats is reflected in the orientation of the document toward defining form and function as opposed to a specification of the bits and bytes in an actual physical format. It is important to state here that the use of a data abstraction in no way inhibits access to physical data or necessarily makes such access inefficient. It merely provides a way of generalizing the data model and makes possible the specification of a uniform interface for manipulation of a data set. The data abstraction allows future extensibility and provides for conceptual simplicity while isolating machine and device dependence.

The contents of a CDF fall into two categories. The first is a series of records comprising a collection of variables consisting of scalers, vectors, and n-dimensional arrays. The second is a set of attribute entries (metadata) describing the CDF in global terms or specifically for a single variable. This dual function of CDF is what provides its "data set independence." Both the data dictionary (attributes) and the data objects (variables) are combined into an integrated data set. An important element of the CDF conceptual data abstraction is the "virtual" dimensional layer that allows data objects that share a subset of the overall CDF dimensionality to be projected into the full dimensional space. This capability is made available through the use of logical dimensional variances that indicate the subset of CDF dimensions that are applicable.

### How is CDF Used?

The origins of CDF date back to the development of the NASA Climate Data System at the National Space

[1] Programming reference manuals for C and Fortran users are provided as separate documents.

Science Data Center (NSSDC). As such, it has had three main requirements driving its development.

1. Facilitate ingestion of data sets and data products into CDF.
2. Utilize standard common terminology (metadata) to describe the data sets.
3. Development of higher level applications (e.g., NSSDC Graphics System [NGS]).

The above requirements imply two classes of users for CDF. One user class performs primarily data acquisition and is mainly involved in designing CDFs and the associated science metadata. The other user class builds high-level applications interacting with CDF at the programming level. CDF has two levels of access: one is through the programming interface layer and the other is through a high-level toolkit written using the programming interface layer.

The toolkit provides utilities for creating new CDFs and for browsing existing CDFs. These are very useful for architecturing a CDF and describing the metadata without using the programming level interfaces. The browsing tools allow a quick look at CDF data sets and aid in CDF validation.

The programming layer (CDF library) provides the essential framework on which graphical and data analysis packages can be created. The CDF library allows developers of CDF-based systems to easily create applications that permit users to slice data across multidimensional subspaces, access entire structures of data, perform subsampling of data, and access one data element independently regardless of its relationship to any other data element. CDF data sets are portable across any platform supported by CDF. These currently consist of VAX (VMS), Sun (SunOS & SOLARIS), DECstation (ULTRIX), DEC Alpha (OSF/1 & OpenVMS), Silicon Graphics Iris and Power Series (IRIX), IBM RS6000 series (AIX), HP 9000 series (HP-UX), NeXT (Mach), IBM PC (MS-DOS), and Macintosh (MacOS 7.0).

**How is CDF Useful to Me?**

Hopefully, the answers to the first two questions have provided a basis for answering this question. It is important to understand that CDF has been designed to solve a number of data management and storage problems and has shown itself to be quite flexible in storing a wide variety of data sets.

# Chapter 1

# Primer

## 1.1 Introduction

The CDF Primer is designed for scientists, researchers, programmers, and managers who want to learn about CDF without reading through this entire document or the programming reference guides. The primer will address what CDF is and how it can be used for storing and managing different types of data. A brief description of the tools and utilities available with CDF, in addition to program and toolkit examples, will be given. More detailed descriptions of the concepts presented herein are provided in the accompanying chapters of this document and the programming reference guides.

## 1.2 Why Use CDF?

When people first hear the term CDF they intuitively think of data formats in the traditional sense of the word (i.e., messy/convoluted storage of data on disk or tape). CDF is more than just a format. CDF is a "self-describing" format for managing data. In addition to the actual data being stored, CDF also stores user-supplied descriptions of the data, known as metadata. This self-describing property allows CDF to be a generic, data-independent format that can store data from a wide variety of disciplines.

In addition to being a self-describing data format, CDF is also a software library. The library routines are callable from C and Fortran and allow the user to randomly access and manage data and metadata without regard to their physical storage. This completely relieves the user of low-level I/O operations allowing more time for data analysis. The actual format used to store the data and metadata is completely transparent to the user.

The term "CDF" is also used to refer to the physical files that the CDF library generates. A data set stored using the CDF library is called a "CDF".

## 1.3 Conceptual Organization

An important feature of CDF is that it can handle data sets that are inherently multidimensional in addition to data sets that are scalar. To do this, CDF groups data by "variables" whose values are conceptually organized into arrays. The dimensionality of these variable arrays depends upon the data and is specified by the user when the CDF or a variable is created. For scalar data, as an example, the array of values would be 0-dimensional (i.e., a single value); whereas for image data the array would be 2-dimensional. Similarly, the array for volume data would be 3-dimensional. CDF allows users to specify arrays of up to ten dimensions. The array for a particular variable is called a "variable record." A collection of arrays, one for each variable, is referred to as a "CDF record." A CDF can, and usually does, contain multiple CDF records. This is useful for data with repeated observations at different times.

Two types of variables may exist in a CDF: rVariables[1] and zVariables.[2] Every rVariable in a CDF must have the same number of dimensions and dimension sizes. In the scalar data example the CDF's rVariables would be 0-dimensional, whereas for the image data example the CDF's rVariables would be 2-dimensional. Figures 1.1 and 1.2 illustrate 0-dimensional and 2-dimensional rVariables, respectively. zVariables may have a different number of dimensions and/or dimension sizes than that of the rVariables in a CDF. Figure 1.3 illustrates several zVariables. Note that a CDF may contain both rVariables and zVariables.[3] The term "variable" is used when describing a property that applies to both rVariables and zVariables.

It is important to note that there is no single "correct" way to store data in a CDF. The user has complete control over how the data values are stored in the CDF (within the confines of the variable array structure) depending on how the user views the data. This is the advantage of CDF. Data values are organized in whatever way makes sense to the user.

## 1.4 Features of the CDF Library

The CDF library is a flexible and extensible software package that gives the user many options for creating and accessing a CDF.

### 1.4.1 File Format Options

The CDF library gives the user the option to choose from one of two file formats in which to store the data and metadata. The first option is the traditional CDF multi-file format, which is also the default. This file format is illustrated in Figure 1.4 (assuming a CDF containing four variables). The `example.cdf` file contains all of the control information and metadata for the CDF. In addition to the `.cdf` file, a file exists for each variable in the CDF and contains only the data associated with that variable. This is illustrated by the files `example.v0` through `example.v3`. The second option is the single-file format, which, as illustrated in Figure 1.5, consists of only the `example.cdf` file. This file contains the control information, metadata, and the data values for each of the variables in the CDF. Both formats allow direct access. The advantage of the single-file format is that it minimizes the number of files one has to manage and makes it easier to transport CDFs across a network. The organization of the data within the single file may, however, become somewhat

[1]The "r" stands for "regular." rVariables are the type of variables that CDF has always supported. Perhaps "traditional" would have been a better term. We also considered "related" and "correlated" but those just didn't seem right.

[2]The "z" doesn't stand for anything special. We just like the letter "z."

[3]This is generally not recommended. In those situations where zVariables are necessary it is best to use all zVariables rather than a mixture of rVariables and zVariables.

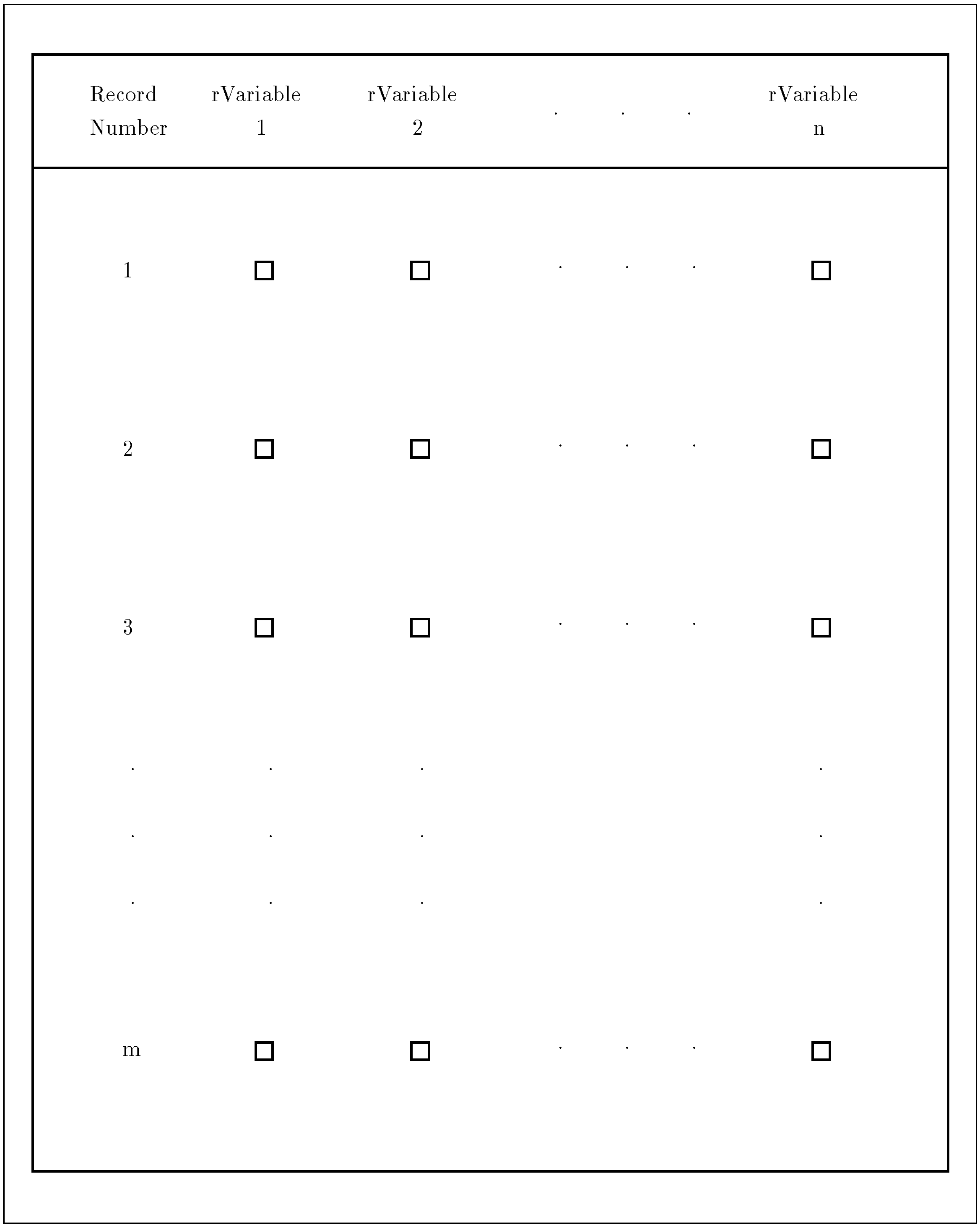


Figure 1.1: Conceptual View of a CDF, 0-Dimensional rVariables

convoluted, slightly increasing the data access time. The multi-file format, on the other hand, clearly delimits the data from the metadata and is organized in a consistent fashion within the files. Updating, appending,

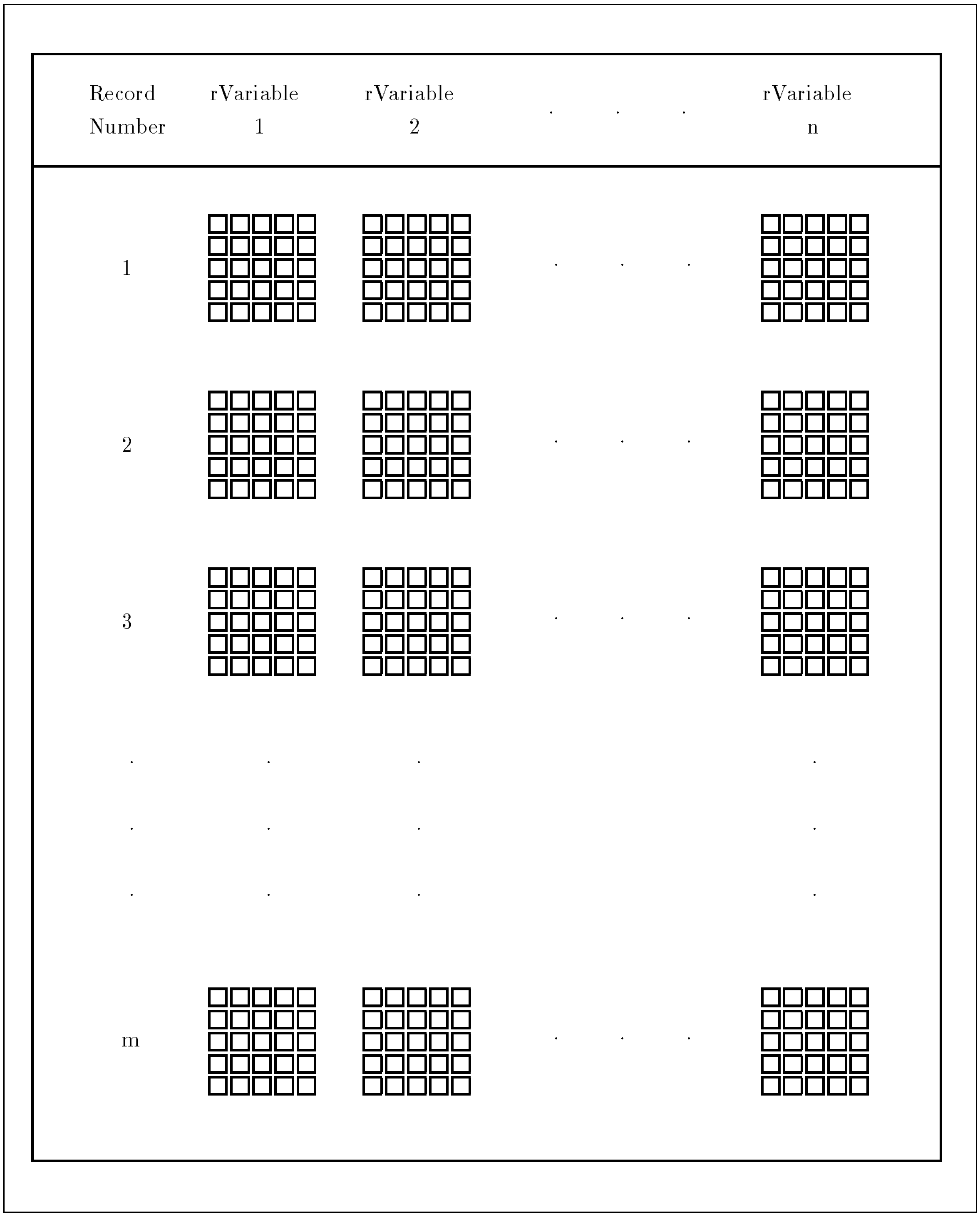


Figure 1.2: Conceptual View of a CDF, 2-Dimensional rVariables

and accessing data are also done with optimum efficiency.

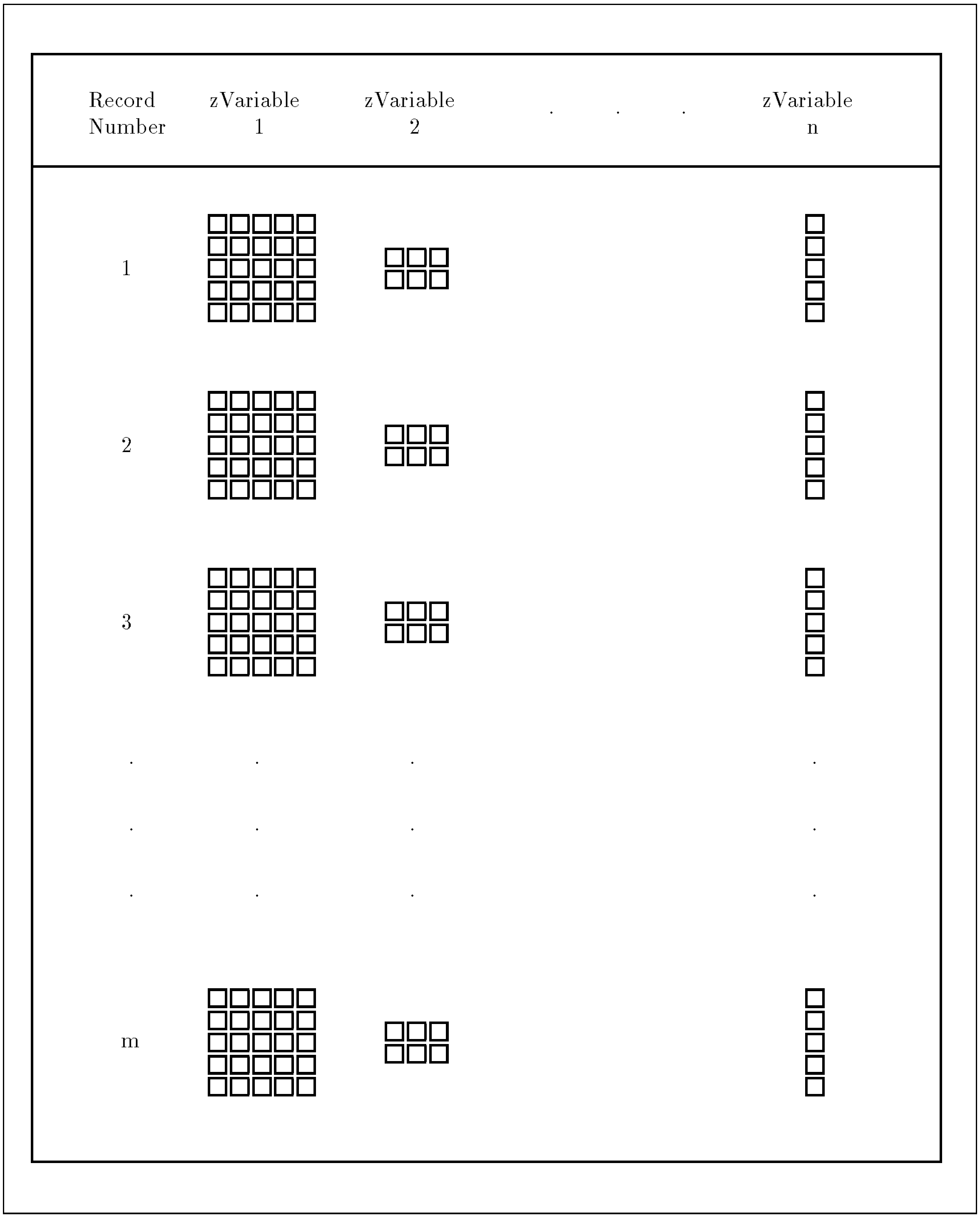


Figure 1.3: Conceptual View of a CDF, zVariables

### 1.4.2 Data Encoding Options

When creating a CDF, a user has the option of using any of the supported encodings:[4] VAX, Sun, SGi Personal Iris and Power Series, DECstation, DEC Alpha/OSF1, DEC Alpha/OpenVMS (`D_FLOAT` and `G_FLOAT`),

[4] This is a new feature of CDF V2.5.

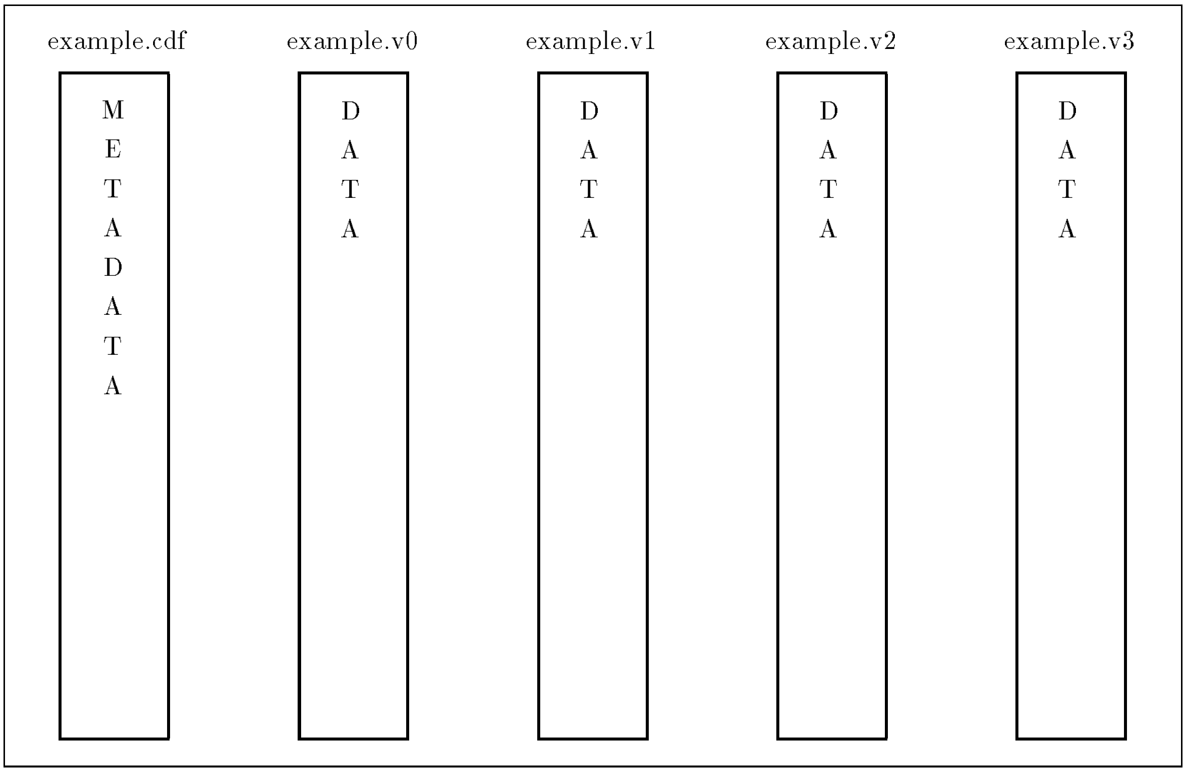


Figure 1.4: Multi-File Format

IBM RS6000 series, HP 9000 series, NeXT, IBM PC, Macintosh, or network (XDR - eXternal Data Representation). The created CDF may then be copied to any of the supported computers and read by the CDF library. When a value is read from the CDF, the CDF library may be requested to decode the value into the encoding of the computer being used or any of the other encodings (which may be desirable for various reasons). A CDF with any of the supported encodings may be read from and written to on any supported computer.

### 1.4.3 Variable Data Access Options

A program can access variable data one value at a time or it can access an entire multidimensional array structure or substructure spanning contiguous or non-contiguous record boundaries. The latter feature allows the user to perform aggregate access or uniform subsampling of the data at greatly increased rates over traditional value by value access.

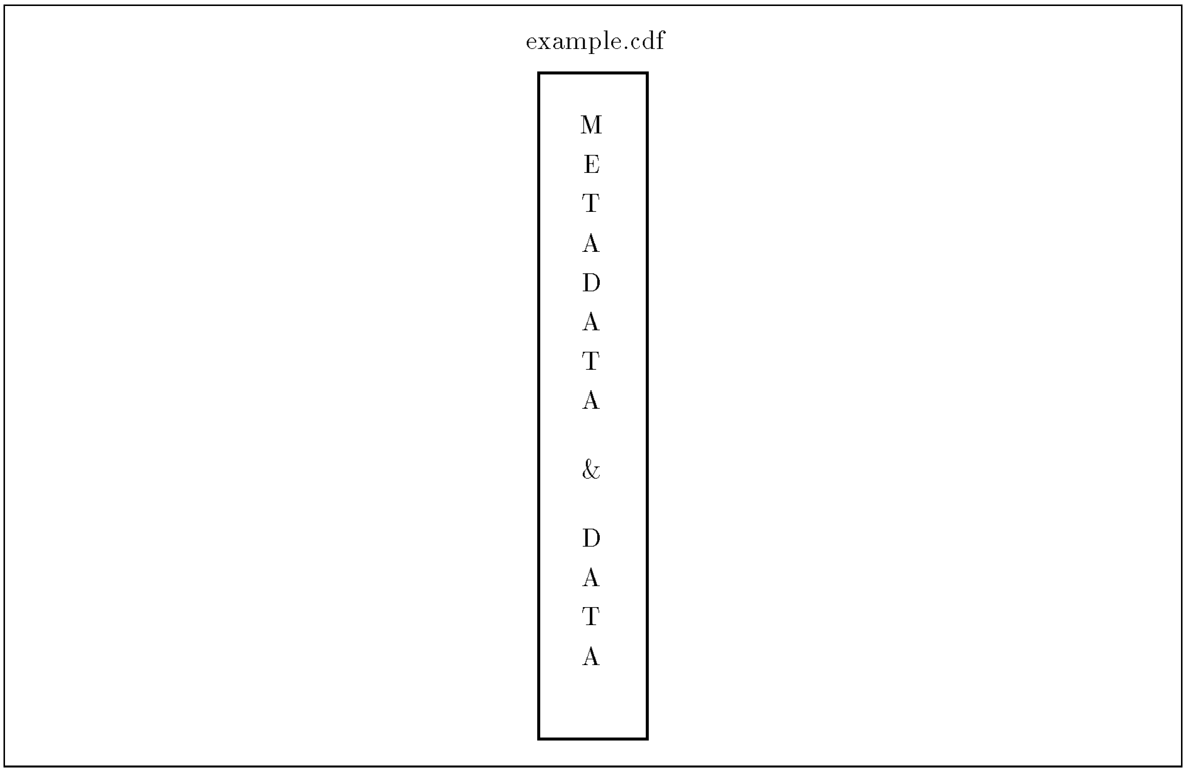


Figure 1.5: Single-File Format

## 1.5 Organizing Your Data in a CDF

### 1.5.1 Variables

The first component of a CDF is the actual data, organized into arrays for the individual variables. CDF can accommodate any type of data that can be organized into arrays. Two types of variables are supported: rVariables and zVariables.

#### rVariables

rVariables all have the same dimensionality (number of dimensions and dimension sizes). An example of the type of data set that may be stored in a CDF's rVariables is shown in Table 1.1. Each record holds one value for each of the four variables: `Time`, `Longitude`, `Latitude`, and `Temperature`. CDF can store scalar data in a "flat" (0-dimensional) representation such as this, but storage in this manner may hide fundamental relationships among the data values. Consistent repetitions found in the data for this example suggest another way to organize the data set. Note that every fourth record is an observation at the same point on Earth at different times. That fact is not immediately clear from this representation of the data. Looking more closely, we note that only two differing values are recorded for `Longitude` and, similarly, only two differing values are recorded for `Latitude`. This repetition suggests a 2-dimensional array structure whose dimensions are defined by `Longitude` and `Latitude`. For each of the two `Longitude` values there are two `Latitude` values. `Time` repeats for each `Longitude/Latitude` pair — the observations were taken simultaneously at the longitude/latitude locations. Because of `Time`'s repetition for `Longitude/Latitude` pairs, the number of `Time` values specifies the number of records needed in the CDF. Each record conceptually contains a 2-dimensional array per rVariable (Table 1.2). The array structure defines the dimensionality of the rVariables in the CDF. Although there are four rVariables, the array dimensions and the sizes of those dimensions are determined only by `Longitude` and `Latitude`. `Temperature` varies across the entire array while `Time` tells us how many records to expect. Therefore, the example, when reduced as described, defines a CDF with 2-dimensional rVariables. The number of discrete values for each rVariable that defines a dimension generates the size of that dimension. For example, `Longitude` has two unique values so the dimension defined by `Longitude` has a size of two.

| Record | rVariables | | | |
|---|---|---|---|---|
| Number | `Time` | `Longitude` | `Latitude` | `Temperature` |
| 1 | 0000 | -165. | +40. | 20.0 |
| 2 | 0000 | -165. | +30. | 21.7 |
| 3 | 0000 | -150. | +40. | 19.2 |
| 4 | 0000 | -150. | +30. | 20.7 |
| 5 | 0100 | -165. | +40. | 18.2 |
| 6 | 0100 | -165. | +30. | 19.3 |
| 7 | 0100 | -150. | +40. | 22.0 |
| 8 | 0100 | -150. | +30. | 19.2 |
| 9 | 0200 | -165. | +40. | 19.9 |
| 10 | 0200 | -165. | +30. | 19.3 |
| 11 | 0200 | -150. | +40. | 19.6 |
| 12 | 0200 | -150. | +30. | 19.0 |
| . | . | . | . | . |
| . | . | . | . | . |
| . | . | . | . | . |
| 93 | 2300 | -165. | +40. | 21.0 |
| 94 | 2300 | -165. | +30. | 19.5 |
| 95 | 2300 | -150. | +40. | 18.4 |
| 96 | 2300 | -150. | +30. | 22.0 |

Table 1.1: Example Data Set — "Flat" Representation (0-Dimensional)

Adding another independent rVariable, for instance `Pressure`, poses no difficulty for the example. `Temperature` would then be dependent on a specific `Longitude`, `Latitude`, and `Pressure` — a 3-dimensional array structure. In this 3-dimensional example `Longitude`, `Latitude`, and `Pressure` define the number of dimensions for the rVariables in the CDF, where the size of each dimension is determined by the number of discrete values contained in each of those rVariables. Additional dependent rVariables would be stored in the same way as `Temperature`.

Although conceptually there is a 2-dimensional array structure for each rVariable in each record of the CDF, this would not be an efficient way to store the data. For instance, the time for each record need only be stored once as opposed to being stored four times as shown in each 2-dimensional array (Table 1.2). This problem is circumvented by specifying "variances." For each rVariable there are variances associated with the array dimensions as well as the records. "Record variance" indicates whether or not an rVariable has unique values from record to record in the CDF. `Time` changes for each record so the record variance for `Time` is `[TRUE]`. One could also say that `Time` is record-variant. `Latitude` and `Longitude` repeat their values from record to record so the record variance for each is `[false]`. `Latitude` and `Longitude` are non-record-variant (NRV). The `Temperature` values change from record to record so they are record-variant. The record variances for

| Record Number | rVariables Time | Longitude | Latitude | Temperature |
|---|---|---|---|---|
| | **0000** — 0000 | **-165** — **-150** | **+40** — +40 | **20.0** — **19.2** |
| 1 | \| \| | \| \| | \| \| | \| \| |
| | 0000 — 0000 | -165 — -150 | **+30** — +30 | **21.7** — **20.7** |
| | **0100** — 0100 | -165 — -150 | +40 — +40 | **18.2** — **22.0** |
| 2 | \| \| | \| \| | \| \| | \| \| |
| | 0100 — 0100 | -165 — -150 | +30 — +30 | **19.3** — **19.2** |
| | **0200** — 0200 | -165 — -150 | +40 — +40 | **19.9** — **19.6** |
| 3 | \| \| | \| \| | \| \| | \| \| |
| | 0200 — 0200 | -165 — -150 | +30 — +30 | **19.3** — **19.0** |
| . | . | . | . | . |
| . | . | . | . | . |
| . | . | . | . | . |
| | **2300** — 2300 | -165 — -150 | +40 — +40 | **21.0** — **18.4** |
| 24 | \| \| | \| \| | \| \| | \| \| |
| | 2300 — 2300 | -165 — -150 | +30 — +30 | **19.5** — **22.0** |

Table 1.2: Example CDF — 2-Dimensional Representation (Conceptual)

this example are shown in Table 1.3.

Similarly, the term "dimension variance" indicates whether or not an rVariable changes with respect to the CDF dimensions. In the example above with 2-dimensional rVariables, the `Longitude` rVariable defines the first dimension of the CDF with its values repeating along the second dimension so its dimension variances would be `[TRUE,false]`. The `Latitude` rVariable defines the second dimension of the CDF with its values repeating along the first dimension so its dimension variances would be `[false,TRUE]`. Because the `Temperature` values change for each latitude/longitude location, its dimension variances are `[TRUE,TRUE]`. `Time` does not change from one latitude/longitude location to another, so its values are the same along both dimensions. The dimension variances for `Time` would be `[false,false]`. The dimension variances for the above example are shown in Table 1.3.

| | rVariables Time | Longitude | Latitude | Temperature |
|---|---|---|---|---|
| Record Variance | TRUE | false | false | TRUE |
| First Dimension Variance | false | TRUE | false | TRUE |
| Second Dimension Variance | false | false | TRUE | TRUE |

Table 1.3: Example CDF — Specification for 2-Dimensional Representation

When the record and dimension variances have been defined correctly, the amount of physical storage needed for the CDF is drastically reduced. In the above example, 2-dimensional arrays are not physically stored for each rVariable in a CDF record. Instead, the physical storage for each rVariable consists of just one value for `Time` in each CDF record, a single 1-dimensional array of values for the `Longitude` and `Latitude` rVariables (in only the first CDF record), and a full 2-dimensional array of values for `Temperature` in each CDF record. The actual physical storage (physical view) is shown in Table 1.4. The conceptual view of the CDF, however, is still that of one 2-dimensional array per rVariable in each CDF record as shown in

Table 1.2 (the physically stored values are shown in boldface type).

<table>
<tr><th>Record<br>Number</th><th colspan="4">rVariables</th></tr>
<tr><th></th><th>Time</th><th>Longitude</th><th>Latitude</th><th>Temperature</th></tr>
<tr><td>1</td><td>0000</td><td>-165 — -150</td><td>+40<br>|<br>+30</td><td>20.0 — 19.2<br>| |<br>21.7 — 20.7</td></tr>
<tr><td>2</td><td>0100</td><td></td><td></td><td>18.2 — 22.0<br>| |<br>19.3 — 19.2</td></tr>
<tr><td>3</td><td>0200</td><td></td><td></td><td>19.9 — 19.6<br>| |<br>19.3 — 19.0</td></tr>
<tr><td>.<br>.<br>.</td><td>.<br>.<br>.</td><td></td><td></td><td>.<br>.<br>.</td></tr>
<tr><td>24</td><td>2300</td><td></td><td></td><td>21.0 — 18.4<br>| |<br>19.5 — 22.0</td></tr>
</table>

Table 1.4: Example CDF — 2-Dimensional Representation (Physical)

### zVariables

zVariables are identical to rVariables in all but one respect. The dimensionality of a zVariable may be different from that of the rVariables (and other zVariables). This allows variables that do not "fit" into the dimensionality of the rVariables to be included in the same CDF. Using only rVariables to store certain data sets could result in wasted disk space or a CDF with rVariables whose dimensionality is not consistent with some of the data in the data set. zVariables are intended to solve these problems. Note that a CDF may also contain just zVariables.

Consider a data set that consists of some number of images, each containing 1024 by 1024 pixels. The data set also contains a palette that is used to map pixel values to the actual color/shade to be displayed. Palettes are also referred to as lookup tables or color lookup tables. For this example assume that each image pixel is stored in an 8-bit byte and the palette is a 1-dimensional array of 256 colors/shades. Indexing into the palette array with a pixel value gives the appropriate color/shade to use.

Attempting to store the images and the palette using only rVariables would result in one of two undesirable situations. If the CDF's rVariables are 2-dimensional with sizes `[1024,1024]` (to store the images), the palette would have to be stored in a 1024 by 1024 array that does not make sense logically and would waste disk space regardless of how the dimension variances are set. If the CDF's rVariables are 3-dimensional with sizes `[1024,1024,256]`, the images could be stored in an rVariable having dimension variances `[TRUE,TRUE,false]` and the palette could be stored in an rVariable having dimension variances `[false,false,TRUE]`. This would not waste any disk space but is not the intuitive way to store the data — nothing in the data set is 3-dimensional.

Using zVariables to store the images and palette would solve both problems. The images would be stored in a 2-dimensional zVariable with sizes `[1024,1024]` (and dimension variances `[TRUE,TRUE]`) and the palette would be stored in a zVariable defined to be 1-dimensional with a size of `[256]` (and dimension variance of `[TRUE]`). This would waste no disk space and logically makes sense.

The use of zVariables is recommended because of this added flexibility. Note that zVariables can always be used instead of rVariables. In the previous example involving the storage of temperature values, the zVariables defined in Table 1.5 could have been used instead. To some users this approach may be more logical. When using zVariables in this way, it becomes more important to use attributes to store the metadata needed to understand the data. In this example, attributes would be used to describe how the latitude/longitude zVariables relate to the dimensions of the temperature zVariable.

| zVariables | Dimensionality[5] | Variances[6] |
|---|---|---|
| `Time` | 0:[] | T/ |
| `Latitude` | 1:[2] | F/T |
| `Longitude` | 1:[2] | F/T |
| `Temperature` | 2:[2,2] | T/TT |

Table 1.5: zVariable Alternative

## 1.6 Attributes

The second component of a CDF is the metadata. Metadata values consist of user-supplied descriptive information about the CDF and the variables in the CDF by way of attributes and attribute entries. Attributes can be divided into two categories: attributes of global scope (gAttributes) and attributes of variable scope (vAttributes). gAttributes describe the CDF as a whole while vAttributes describe some property of each variable (rVariables and zVariables) in the CDF. Any number of attributes may be stored in a single CDF. Section 2.5 describes standard CDF attributes used by the NSSDC. The term "attribute" is used when describing a property that applies to both gAttributes and vAttributes.

gAttributes can include any information regarding the CDF and all of its variables collectively. Such descriptions could include a title for the CDF, data set documentation, or a CDF modification history. gAttributes may contain multiple entries (called gEntries). An example of this would be a modification history kept in the optional gAttribute, `MODS`. This attribute could be specified at CDF creation time and a gEntry made regarding creation date. Any subsequent changes made to the CDF, including additional variables, changes in min/max values, or modifications to variable values could be documented by writing additional gEntries to `MODS`.

vAttributes further describe the individual variables and their values. Examples of vAttributes would include such things as a field name for the variable, the valid minimum and maximum, the units in which the variable data values are stored, the format in which the data values are to be displayed, a fill value for errant or missing data, and a description of the expected order of data values: increasing or decreasing (monotonicity). The entries of a vAttribute correspond to the variables in the CDF. Each rEntry corresponds to an rVariable and each zEntry corresponds to a zVariable. Sample vAttribute rEntries for the `Temperature` rVariable from the example above are shown in Table 1.6.

[5] The notation used is `<num-dims>:[<dim-sizes>]` where `<num-dims>` is the number of dimensions and `<dim-sizes>` is zero or more dimension sizes separated by commas.

[6] The notation used is `<rec-vary>/<dim-varys>` where `<rec-vary>` is the record variance (`T` or `F`) and `<dim-varys>` is zero or more dimension variances.

The term "entry" is used when describing a property that applies to gEntries, rEntries, and zEntries.

| vAttribute | rEntry value |
|---|---|
| FIELDNAM | "Recorded temperature" |
| VALIDMIN | -40.0 |
| VALIDMAX | 50.0 |
| SCALEMIN | 17.0 |
| SCALEMAX | 24.0 |
| UNITS | "deg C" |
| FORMAT | "F4.1" |
| MONOTON | "FALSE" |
| FILLVAL | -999.9 |

Table 1.6: vAttribute rEntries for the `Temperature` rVariable

## 1.7 CDF Toolkit

A set of utility programs are provided with the CDF distribution which allow a user to perform a variety of operations on CDFs without having to write an application program. Each toolkit program is described in detail in Chapter 3.

The available toolkit programs are as follows:

`CDFedit`[7] Allows the display, creation, and modification of attribute and variable data in a CDF.

`CDFlist` Allows a sequential "flat" view of filtered variable data. `CDFlist` also allows the subsetting of a CDF (creating a new, smaller CDF).

`CDFwalk` Allows direct and sequential access to related variable values.

`CDFconvert` Allows the format, encoding, and majority of a CDF to be changed. In all cases a new CDF is created. The original CDF is not modified.

`SkeletonCDF`[8] Reads a specially formatted text file (called a skeleton table) and creates a skeleton CDF. A skeleton CDF is complete except for record-variant data.

`SkeletonTable` Reads a CDF and produces a specially formatted text file called a skeleton table. The skeleton table may be modified and then input to `SkeletonCDF` to create a skeleton CDF.

`CDFinquire` Displays the version of your CDF distribution and the default toolkit qualifiers.

`CDFstats` Produces a report containing various statistics about the variables in a CDF.

[7] `CDFedit` has replaced `CDFbrowse`. The alias/symbol `CDFbrowse` still exists in the "definitions" file on UNIX/VMS systems but now executes `CDFedit` in a browse-only mode.

[8] `SkeletonCDF` was previously named `CDFskeleton`.

| | |
|---|---|
| CDFcompare | Reports the differences between two CDFs. |
| CDFdir | Produces a directory listing of a CDF's files. For a multi-file CDF the variable files are listed in ascending numerical order. |

## 1.8 Library Interface Routines

The CDF library supports two programming interfaces, the Standard Interface and the Internal Interface. The Standard Interface is similar to the interface provided with Version 1 of CDF with several additions for new features. The Internal Interface is provided to allow additional functionality to be added to the CDF library without the need to modify the Standard Interface. Those features not available using the Standard Interface are made available using the Internal Interface (e.g., access to zVariables). The Internal Interface makes CDF extendable. The Standard and Internal interfaces are callable from both C and Fortran.

### 1.8.1 Standard Interface

The Standard Interface consists of three categories of software functions that are utilized to manipulate the components that make up a CDF: general CDF functions, rVariable functions, and attribute functions.

The general CDF functions are as follows:

| Callable from C | Callable from Fortran | Purpose |
|---|---|---|
| CDFcreate() | CDF_create() | Creates a new CDF. |
| CDFopen() | CDF_open() | Opens an existing CDF. |
| CDFdoc() | CDF_doc() | Inquires version/release and copyright notice. |
| CDFinquire() | CDF_inquire() | Inquires rVariable dimensionality, etc. |
| CDFclose() | CDF_close() | Closes a CDF. |
| CDFdelete() | CDF_delete() | Deletes a CDF. |
| CDFerror() | CDF_error() | Inquires error (status) code meaning. |

The rVariable functions are as follows:

| Callable from C | Callable from Fortran | Purpose |
|---|---|---|
| CDFvarCreate() | CDF_var_create() | Creates an rVariable. |
| CDFvarNum() | CDF_var_num() | Determines an rVariable number. |
| CDFvarRename() | CDF_var_rename() | Renames an rVariable. |
| CDFvarInquire() | CDF_var_inquire() | Inquires about an rVariable. |
| CDFvarPut() | CDF_var_put() | Writes an rVariable value. |
| CDFvarGet() | CDF_var_get() | Reads an rVariable value. |
| CDFvarHyperPut() | CDF_var_hyper_put() | Writes one or more rVariable values. |
| CDFvarHyperGet() | CDF_var_hyper_get() | Reads one or more rVariable values. |
| CDFvarClose() | CDF_var_close() | Closes an rVariable. |

The attribute functions are as follows:

| Callable from C | Callable from Fortran | Purpose |
|---|---|---|
| `CDFattrCreate()` | `CDF_attr_create()` | Creates an attribute. |
| `CDFattrNum()` | `CDF_attr_num()` | Determines an attribute number. |
| `CDFattrRename()` | `CDF_attr_rename()` | Renames an attribute. |
| `CDFattrInquire()` | `CDF_attr_inquire()` | Inquires about an attribute. |
| `CDFattrEntryInquire()` | `CDF_attr_entry_inquire()` | Inquires about an attribute rEntry. |
| `CDFattrPut()` | `CDF_attr_put()` | Writes an attribute rEntry. |
| `CDFattrGet()` | `CDF_attr_get()` | Reads an attribute rEntry. |

The Standard Interface may be used to access only rVariables and the vAttribute rEntries for rVariables. The Internal Interface must be used to access zVariables and the vAttribute zEntries for zVariables.

### 1.8.2 Internal Interface

The Internal Interface consists of one routine: `CDFlib` when called from C and `CDF_lib` when called from Fortran. The Internal Interface is used to perform all CDF operations. (In reality the Standard Interface is implemented via the Internal Interface.) The Internal Interface is used to add new CDF features (e.g., zVariables) without having to change the Standard Interface.

## 1.9 Examples

In this section several examples of how to use the CDF library and toolkit will be presented. The same CDF will be created two different ways: by using just the CDF library and by using the CDF library with the `SkeletonTable` toolkit program.

### 1.9.1 Creating a CDF, the Hard Way (But Not That Hard)

The first example program, written in C, creates a CDF with 2-dimensional rVariables using only CDF library function calls. The CDF created will contain the data and metadata values used in the example presented earlier in this chapter (minus some of the vAttributes/rEntries). An input file, `example.dat`, whose format is similar to that of Table 1.1 will be read and its data values written into the CDF.

```
/******************************************************************************
*
*  NSSDC/CDF           Create an example CDF (without using a skeleton table).
*
*  Version 1.0, 5-Jan-94, CDF, Inc.
*
*  Modification history:
*
*   V1.0   5-Jan-94, Joe Programmer     Original version.
*
******************************************************************************/

/******************************************************************************
```

```
*
* Note(s):
*
*   This program would have to be modified to run on a DEC Alpha because the
* C language 'long' data type is 8 bytes rather than 4 (the CDF data type of
* CDF_INT4 is always 4 bytes).
*
******************************************************************************/

/******************************************************************************
* Necessary include files.
******************************************************************************/

#include <stdio.h>
#include <stdlib.h>

#include "cdf.h"

/******************************************************************************
* Status handler.
******************************************************************************/

void StatusHandler (status)
CDFstatus status;
{
  char message[CDF_ERRTEXT_LEN+1];

  if (status < CDF_WARN) {
    printf ("An error has occurred, halting...\n");
    CDFerror (status, message);
    printf ("%s\n", message);
    exit (status);
  }
  else
    if (status < CDF_OK) {
      printf ("Warning, function may not have completed as expected...\n");
      CDFerror (status, message);
      printf ("%s\n", message);
    }
    else
      if (status > CDF_OK) {
        printf ("Function completed successfully, but be advised that...\n");
        CDFerror (status, message);
        printf ("%s\n", message);
      }
  return;
}

/******************************************************************************
* MAIN.
******************************************************************************/
```

```
main () {
  CDFid id;            /* CDF identifier. */
  CDFstatus status;    /* CDF completion status. */

  FILE *fp;            /* File pointer - used to read input data file. */

  long numDims = 2;                   /* Number of dimensions, rVariables. */
  static long dimSizes[2] = {2,2};   /* Dimension sizes, rVariables. */

  long dimVarys[2];    /* Dimension variances. */
  long indices[2];     /* Dimension indices. */
  long recNum;         /* Record number. */
  long attrNum;        /* Attribute number. */

  long TimeVarNum;     /* 'Time' rVariable number. */
  long LonVarNum;      /* 'Longitude' rVariable number. */
  long LatVarNum;      /* 'Latitude' rVariable number. */
  long TmpVarNum;      /* 'Temperature' rVariable number. */

  long Time;           /* 'Time' rVariable value. */
  float Lat;           /* 'Latitude' rVariable value. */
  float Lon;           /* 'Longitude' rVariable value. */
  float Tmp;           /* 'Temperature' rVariable value. */

  long TimeValidMin = 0;          /* 'Time' valid minimum (0000). */
  long TimeValidMax = 2359;       /* 'Time' valid maximum (2359). */

  float LonValidMin = -180.0;     /* 'Longitude' valid minimum. */
  float LonValidMax = 180.0;      /* 'Longitude' valid maximum. */

  float LatValidMin = -90.0;      /* 'Latitude' valid minimum. */
  float LatValidMax = 90.0;       /* 'Latitude' valid maximum. */

  float TmpValidMin = -40.0;      /* 'Temperature' valid minimum. */
  float TmpValidMax = 50.0;       /* 'Temperature' valid maximum. */

  /******************************************************************************
  * Create the CDF.
  ******************************************************************************/

  status = CDFcreate ("example1", numDims, dimSizes, NETWORK_ENCODING,
                      ROW_MAJOR, &id);
  if (status != CDF_OK) StatusHandler (status);

  /******************************************************************************
  * Create rVariables.
  ******************************************************************************/

  dimVarys[0] = NOVARY;
  dimVarys[1] = NOVARY;
```

```
status = CDFvarCreate (id, "Time", CDF_INT4, 1L, VARY, dimVarys,
                       &TimeVarNum);
if (status != CDF_OK) StatusHandler (status);

dimVarys[0] = VARY;
dimVarys[1] = NOVARY;
status = CDFvarCreate (id, "Longitude", CDF_REAL4, 1L, NOVARY, dimVarys,
                       &LonVarNum);
if (status != CDF_OK) StatusHandler (status);

dimVarys[0] = NOVARY;
dimVarys[1] = VARY;
status = CDFvarCreate (id, "Latitude", CDF_REAL4, 1L, NOVARY, dimVarys,
                       &LatVarNum);
if (status != CDF_OK) StatusHandler (status);

dimVarys[0] = VARY;
dimVarys[1] = VARY;
status = CDFvarCreate (id, "Temperature", CDF_REAL4, 1L, VARY, dimVarys,
                       &TmpVarNum);
if (status != CDF_OK) StatusHandler (status);

/******************************************************************************
* Create attributes.
******************************************************************************/

status = CDFattrCreate (id, "TITLE", GLOBAL_SCOPE, &attrNum);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrCreate (id, "VALIDMIN", VARIABLE_SCOPE, &attrNum);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrCreate (id, "VALIDMAX", VARIABLE_SCOPE, &attrNum);
if (status != CDF_OK) StatusHandler (status);

/******************************************************************************
* Write TITLE gAttribute gEntry.
******************************************************************************/

status = CDFattrPut (id, CDFattrNum(id,"TITLE"), 0L, CDF_CHAR, 50L,
                     "An example CDF (1).                                  ");
if (status != CDF_OK) StatusHandler (status);

/******************************************************************************
* Write vAttribute rEntries for 'Time' rVariable.
******************************************************************************/

status = CDFattrPut (id, CDFattrNum(id,"VALIDMIN"),
                     CDFvarNum(id,"Time"), CDF_INT4, 1L, &TimeValidMin);
if (status != CDF_OK) StatusHandler (status);
```

```
status = CDFattrPut (id, CDFattrNum(id,"VALIDMAX"),
                     CDFvarNum(id,"Time"), CDF_INT4, 1L, &TimeValidMax);
if (status != CDF_OK) StatusHandler (status);

/******************************************************************************
* Write vAttribute rEntries for 'Longitude' rVariable.
******************************************************************************/

status = CDFattrPut (id, CDFattrNum(id,"VALIDMIN"),
                     CDFvarNum(id,"Longitude"), CDF_REAL4, 1L, &LonValidMin);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"VALIDMAX"),
                     CDFvarNum(id,"Longitude"), CDF_REAL4, 1L, &LonValidMax);
if (status != CDF_OK) StatusHandler (status);

/******************************************************************************
* Write vAttribute rEntries for 'Latitude' rVariable.
******************************************************************************/

status = CDFattrPut (id, CDFattrNum(id,"VALIDMIN"),
                     CDFvarNum(id,"Latitude"), CDF_REAL4, 1L, &LatValidMin);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"VALIDMAX"),
                     CDFvarNum(id,"Latitude"), CDF_REAL4, 1L, &LatValidMax);
if (status != CDF_OK) StatusHandler (status);

/******************************************************************************
* Write vAttribute rEntries for 'Temperature' rVariable.
******************************************************************************/

status = CDFattrPut (id, CDFattrNum(id,"VALIDMIN"),
                     CDFvarNum(id,"Temperature"), CDF_REAL4, 1L,
                     &TmpValidMin);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"VALIDMAX"),
                     CDFvarNum(id,"Temperature"), CDF_REAL4, 1L,
                     &TmpValidMax);
if (status != CDF_OK) StatusHandler (status);

/******************************************************************************
* Read input values for rVariables and write them to the CDF.  Not
* every value must be written to the CDF - many of the values are redundant.
* The 'Time' value only has to be written once per CDF record (every 4 input
* records).  The 'Longitude' and 'Latitude' values are only written to the
* first CDF record (and only at the appropriate indices).  Each 'Temperature'
* value read is written to the CDF.
******************************************************************************/
```

```
  fp = fopen ("example.dat", "r");
  if (fp == NULL) {
    printf ("Error opening input file.\n");
    exit (-1);
  }

  for (recNum = 0; recNum < 24; recNum++) {
     for (indices[0] = 0; indices[0] < 2; indices[0]++) {
        for (indices[1] = 0; indices[1] < 2; indices[1]++) {
           fscanf (fp, "%d %f %f %f", &Time, &Lon, &Lat, &Tmp);

           if (indices[0] == 0 && indices[1] == 0) {
             status = CDFvarPut (id, TimeVarNum, recNum, indices, &Time);
             if (status != CDF_OK) StatusHandler (status);
           }

           if (recNum == 0 && indices[1] == 0) {
             status = CDFvarPut (id, LonVarNum, recNum, indices, &Lon);
             if (status != CDF_OK) StatusHandler (status);
           }

           if (recNum == 0 && indices[0] == 0) {
             status = CDFvarPut (id, LatVarNum, recNum, indices, &Lat);
             if (status != CDF_OK) StatusHandler (status);
           }

           status = CDFvarPut (id, TmpVarNum, recNum, indices, &Tmp);
           if (status != CDF_OK) StatusHandler (status);
        }
     }
  }

  fclose (fp);

  /****************************************************************************
  * Close CDF.
  ****************************************************************************/

  status = CDFclose (id);
  if (status != CDF_OK) StatusHandler (status);

  return;
}
```

### 1.9.2 Creating a CDF, an Easier Way

The CDF toolkit program `SkeletonCDF` is provided to make the task of creating a CDF easier for a programmer. `SkeletonCDF` reads a specially formatted text file called a skeleton table and generates a skeleton CDF. Everything about a CDF can be specified in a skeleton table except data values for variables that vary from record to record (record-variant). The toolkit program `SkeletonTable` is also provided. It reads an existing CDF and produces a skeleton table. The skeleton table for the CDF created using only the CDF library in Section 1.9.1 would be as follows.

```
! Skeleton table for the "example" CDF.
! Generated: Wed  5 Jan 1994 10:53:58

#header

                  CDF NAME: example1
             DATA ENCODING: NETWORK
                  MAJORITY: ROW
                    FORMAT: SINGLE

! Variables G.Attributes V.Attributes Records  Dims  Sizes
! --------- ------------ ------------ -------  ----  -----
     4/0          1             2       1/z      2    2 2


#GLOBALattributes

! Attribute         Entry       Data
! Name              Number      Type       Value
! ---------         ------      ----       -----

  "TITLE"               1:    CDF_CHAR     { "An example CDF (1).        " -
                                              "                        " } .


#VARIABLEattributes

  "VALIDMIN"
  "VALIDMAX"


#variables

! Variable            Data       Number     Record    Dimension
! Name                Type      Elements   Variance   Variances
! --------            ----      --------   --------   ---------

  "Time"            CDF_INT4       1          T         F F

  ! Attribute         Data
  ! Name              Type       Value
  ! --------          ----       -----

    "VALIDMIN"      CDF_INT4     { 0 }
    "VALIDMAX"      CDF_INT4     { 2359 } .
```

```
! Variable           Data      Number     Record     Dimension
! Name               Type      Elements   Variance   Variances
! --------           ----      --------   --------   ---------

  "Longitude"      CDF_REAL4      1          F          T F

  ! Attribute        Data
  ! Name             Type       Value
  ! --------         ----       -----

    "VALIDMIN"     CDF_REAL4    { -180.0 }
    "VALIDMAX"     CDF_REAL4    { 180.0 } .

  ! NRV values follow...

  [ 1, 1 ] = -165.0
  [ 2, 1 ] = -150.0


! Variable           Data      Number     Record     Dimension
! Name               Type      Elements   Variance   Variances
! --------           ----      --------   --------   ---------

  "Latitude"       CDF_REAL4      1          F          F T

  ! Attribute        Data
  ! Name             Type       Value
  ! --------         ----       -----

    "VALIDMIN"     CDF_REAL4    { -90.0 }
    "VALIDMAX"     CDF_REAL4    { 90.0 } .

  ! NRV values follow...

  [ 1, 1 ] = 40.0
  [ 1, 2 ] = 30.0


! Variable           Data      Number     Record     Dimension
! Name               Type      Elements   Variance   Variances
! --------           ----      --------   --------   ---------

  "Temperature"    CDF_REAL4      1          T          T T

  ! Attribute        Data
  ! Name             Type       Value
  ! --------         ----       -----

    "VALIDMIN"     CDF_REAL4    { -40.0 }
    "VALIDMAX"     CDF_REAL4    { 50.0 } .
```

```
#end
```

Assuming that `SkeletonCDF` was used to create a CDF containing the metadata and data in the above skeleton table, the following Fortran program would be used to complete the creation of the CDF.

```
        PROGRAM exampleSKT

C-----------------------------------------------------------------------------
C
C  NSSDC/CDF                    Create an example CDF (using skeleton table).
C
C  Version 1.0, 5-Jan-94, CDF, Inc.
C
C  Modification history:
C
C   V1.0   5-Jan-94, Joe Programmer      Original version.
C
C-----------------------------------------------------------------------------

        INCLUDE '../../include/cdf.inc'

        INTEGER*4 id            ! CDF identifier.
        INTEGER*4 status        ! CDF completion status.

        INTEGER*4 lun           ! Logical unit number for input data file.

        INTEGER*4 indices(2)    ! Dimension indices.
        INTEGER*4 rec_num       ! Record number.

        INTEGER*4 time_var_num  ! 'Time' rVariable number.
        INTEGER*4 tmp_var_num   ! 'Temperature' rVariable number.

        INTEGER*4 time          ! 'Time' rVariable value.
        REAL*4    lat           ! 'Latitude' rVariable value.
        REAL*4    lon           ! 'Longitude' rVariable value.
        REAL*4    tmp           ! 'Temperature' rVariable value.

        DATA lun/1/

C-----------------------------------------------------------------------------
C Open the CDF.
C-----------------------------------------------------------------------------

        CALL CDF_open ('example2', id, status)
        IF (status .NE. CDF_OK) CALL StatusHandler (status)

C-----------------------------------------------------------------------------
C Determine rVariable numbers.
C-----------------------------------------------------------------------------
```

```
        time_var_num = CDF_var_num (id, 'Time')
        IF (time_var_num .LT. CDF_OK) CALL StatusHandler (status)

        tmp_var_num = CDF_var_num (id, 'Temperature')
        IF (tmp_var_num .LT. CDF_OK) CALL StatusHandler (status)

C-----------------------------------------------------------------------------
C Read input values for rVariables and write them to the CDF.  Not
C every value must be written to the CDF - many of the values are redundant.
C The 'Time' value only has to be written once per CDF record (every 4 input
C records).  The 'Longitude' and 'Latitude' values are not written at all
C because they had been specified in the skeleton table.  Each 'Temperature'
C value read is written to the CDF.
C-----------------------------------------------------------------------------

        OPEN (lun, FILE='example.dat', ERR=99)

        DO rec_num = 1, 24
          DO x1 = 1, 2
            DO x2 = 1, 2
              indices(1) = x1
              indices(2) = x2

              READ (lun, *, ERR=99) time, lon, lat, tmp

              IF (indices(1) .EQ. 1 .AND. indices(2) .EQ. 1) THEN
                CALL CDF_var_put (id, time_var_num, rec_num, indices,
     .                            time, status)
                IF (status .NE. CDF_OK) CALL StatusHandler (status)
              END IF

              CALL CDF_var_put (id, tmp_var_num, rec_num, indices,
     .                          tmp, status)
              IF (status .NE. CDF_OK) CALL StatusHandler (status)
            END DO
          END DO
        END DO

        CLOSE (lun, ERR=99)

C-----------------------------------------------------------------------------
C Close CDF.
C-----------------------------------------------------------------------------

        CALL CDF_close (id, status)
        IF (status .NE. CDF_OK) CALL StatusHandler (status)

        STOP

C-----------------------------------------------------------------------------
C Input file error handler.
```

```
C-----------------------------------------------------------------------------

 99     WRITE (6,101)
 101    FORMAT (' ','Error reading input file')
        STOP

        END



C-----------------------------------------------------------------------------
C Status handler.
C-----------------------------------------------------------------------------

        SUBROUTINE StatusHandler (status)
        INTEGER*4 status

        INCLUDE '../../include/cdf.inc'

        CHARACTER message*(CDF_ERRTEXT_LEN)

        IF (status .LT. CDF_WARN) THEN
          WRITE (6,10)
10        FORMAT (' ','Error (halting)...')
          CALL CDF_error (status, message)
          WRITE (6,11) message
11        FORMAT (' ',A)
          STOP
        ELSE
          IF (status .LT. CDF_OK) THEN
            WRITE (6,12)
12          FORMAT (' ','Warning...')
            CALL CDF_error (status, message)
            WRITE (6,13) message
13          FORMAT (' ',A)
          ELSE
            IF (status .GT. CDF_OK) THEN
              WRITE (6,14)
14            FORMAT (' ','Be advised that...')
              CALL CDF_error (status, message)
              WRITE (6,15) message
15            FORMAT (' ',A)
            END IF
          END IF
        END IF

        RETURN
        END
```

The CDF was opened (since it already existed) and the values for only the `Time` and `Temperature` rVariables were written to the CDF. All of the other functions performed by the program in Section 1.9.1 were done by the `SkeletonCDF` program when it read the skeleton table.

# Chapter 2

# Concepts

## 2.1 CDF Library

The CDF library is the only way to access a CDF. Various properties of the CDF library are described in the following sections.

### 2.1.1 Interfaces

Several interfaces to the CDF library exist. They are described in the following sections.

#### Standard Interface

The Standard Interface[1] provides a standard set of routines with which to access a CDF. Not all CDF features are available with the Standard Interface. The Internal Interface must be used to perform operations not available with the Standard Interface routines (e.g., access to zVariables). The Standard Interface is callable from both C and Fortran applications. Table 2.1 lists the routines available when using the Standard Interface. Each routine is described in detail in the corresponding programmer's guide.

#### Internal Interface

The Internal Interface may be used to perform all supported CDF operations. The Internal Interface must be used to perform those operations not available with the Standard Interface. Table 2.2 lists the routines available when using the Internal Interface. Each is described in detail in the corresponding programmer's guide.

[1] The Standard Interface was previously known as the C/Fortran Interface(s). That was an unfortunate choice of terms since the Internal Interface is also callable from C and Fortran.

| Callable from C | Callable from Fortran | Purpose |
|---|---|---|
| `CDFcreate()` | `CDF_create()` | Creates a new CDF. |
| `CDFopen()` | `CDF_open()` | Opens an existing CDF. |
| `CDFdoc()` | `CDF_doc()` | Inquires version/release and copyright notice. |
| `CDFinquire()` | `CDF_inquire()` | Inquires rVariable dimensionality, encoding, etc. |
| `CDFclose()` | `CDF_close()` | Closes a CDF. |
| `CDFdelete()` | `CDF_delete()` | Deletes a CDF. |
| `CDFerror()` | `CDF_error()` | Inquires error (status) code meaning. |
| | | |
| `CDFvarCreate()` | `CDF_var_create()` | Creates an rVariable. |
| `CDFvarNum()` | `CDF_var_num()` | Determines an rVariable number. |
| `CDFvarRename()` | `CDF_var_rename()` | Renames an rVariable. |
| `CDFvarInquire()` | `CDF_var_inquire()` | Inquires about an rVariable. |
| `CDFvarPut()` | `CDF_var_put()` | Writes an rVariable value. |
| `CDFvarGet()` | `CDF_var_get()` | Reads an rVariable value. |
| `CDFvarHyperPut()` | `CDF_var_hyper_put()` | Writes one or more rVariable values. |
| `CDFvarHyperGet()` | `CDF_var_hyper_get()` | Reads one or more rVariable values. |
| `CDFvarClose()` | `CDF_var_close()` | Closes an rVariable. |
| | | |
| `CDFattrCreate()` | `CDF_attr_create()` | Creates an attribute. |
| `CDFattrNum()` | `CDF_attr_num()` | Determines an attribute number. |
| `CDFattrRename()` | `CDF_attr_rename()` | Renames an attribute. |
| `CDFattrInquire()` | `CDF_attr_inquire()` | Inquires about an attribute. |
| `CDFattrEntryInquire()` | `CDF_attr_entry_inquire()` | Inquires about an attribute entry. |
| `CDFattrPut()` | `CDF_attr_put()` | Writes an attribute entry. |
| `CDFattrGet()` | `CDF_attr_get()` | Reads an attribute entry. |

Table 2.1: Standard Interface Routines

| Callable from C | Callable from Fortran | Purpose |
|---|---|---|
| `CDFlib ()` | `CDF_lib ()` | Performs all available operations. |

Table 2.2: Internal Interface Routines

**CDF's IDL Interface**

The CDF distribution contains an interface that allows full access to the CDF library (and hence CDFs) from within IDL. CDF's IDL interface consists of a set of functions that mirror the functions in the Standard and Internal interfaces for C and Fortran applications. CDF's IDL interface is described in Appendix B.

IDL also provides its own interface to the CDF library (as well as other data formats) that differs from CDF's IDL interface. The differences are mainly syntactic with the functionality of the two interfaces being essentially the same. IDL's documentation describes their built-in CDF interface. Another difference between the two interfaces is that CDF's IDL interface is only available on those computers that support dynamic linking. Appendix B lists the computers on which this is the case.

### 2.1.2 CDF Modes

Once a CDF has been opened (or created and not yet closed), the CDF library may be configured to act on that CDF in one or more modes. These modes are specified independently for each open CDF.

#### Read-Only Mode

A CDF may be placed in read-only mode via the Internal Interface using the <<SELECT_,CDF_READONLY_MODE_>> operation. Only read access will be allowed on the CDF — all attempts to modify the CDF will fail. A CDF may be toggled in and out of read-only mode any number of times. (Note that attempts to modify a CDF may also fail if insufficient access privileges exist for the CDF — the file system enforces this access.)

#### zMode

A CDF may be placed into zMode[2] via the Internal Interface using the <<SELECT_,CDF_zMODE_>> operation. When in zMode a CDF's rVariables essentially disappear and are replaced by corresponding zVariables.[3] Likewise, the rEntries for a vAttribute become zEntries (because they are now associated with zVariables). While in zMode most operations involving rVariables/rEntries will fail. (Some inquiry operations will be allowed. For example, inquiring the number of rVariables is allowed [but will always be zero].) When zMode is used, the number of variables remains the same — rVariables simply change into zVariables. Note that the existing contents of the CDF are not changed — the CDF simply appears different.

Each new zVariable has the same exact properties as the corresponding (hidden) rVariable except for dimensionality and variances. The data specification (data type and number of elements), pad value, etc. stay the same. The dimensionality/variances of each zVariable are dependent on which zMode is currently being used: zMode/1 or zMode/2. In zMode/1 the dimensionality/variances stay exactly the same. In zMode/2, however, those dimensions with a false variance (NOVARY) are eliminated. Consider a CDF with an rVariable dimensionality of 2:[180,360][4] containing the following rVariables.

| rVariable Name | Variances |
|---|---|
| EPOCH | T/FF[5] |
| LATITUDE | T/TF |
| LONGITUDE | T/FT |
| HUMIDITY | T/TT |

If this CDF were to be placed into zMode/1, the following zVariables would replace the existing rVariables.

[2] There are actually two types of zMode — read on.

[3] In a future release of CDF, support for rVariables will be eliminated. zMode is provided to ease the transition from rVariables to the more flexible zVariables. rVariables are essentially a subset of zVariables.

[4] This notation is used throughout this document. In this case there are two dimensions whose sizes are 180 and 360. A dimensionality of zero is represented as 0:[].

[5] This notation is also used throughout this document. The record variance is before the slash and the dimension variances are after the slash. In this case the record variance is true (VARY) and the dimension variances are both false (NOVARY). The variances for a 0-dimensional variable would be represented as T/ (if the record variance were true).

| zVariable Name | Dimensionality | Variances |
|---|---|---|
| EPOCH | 2:[180,360] | T/FF |
| LATITUDE | 2:[180,360] | T/TF |
| LONGITUDE | 2:[180,360] | T/FT |
| HUMIDITY | 2:[180,360] | T/TT |

Note that the dimensionality of each zVariable is the same as it was for the rVariables in the CDF. However, if zMode/2 were used, the following zVariables would replace the existing rVariables.

| zVariable Name | Dimensionality | Variances |
|---|---|---|
| EPOCH | 0:[] | T/ |
| LATITUDE | 1:[180] | T/T |
| LONGITUDE | 1:[360] | T/T |
| HUMIDITY | 2:[180,360] | T/TT |

In this case the false dimensional variances were removed (which decreased the dimensionality in several of the variables).

A CDF can be placed into or taken out of zMode any number of times while it is open. Each time the zMode is changed for a CDF, it would be best to think of the CDF as being closed and reopened in that zMode. The numbering of variable/entries may or may not be as you would expect (and the scheme used could change in a future release of CDF). Most applications will simply select a zMode immediately after opening a CDF. (zMode being off is the default if a zMode is not selected.)

**NOTE:** Using zMode does not change the contents of a CDF. A CDF containing rVariables will appear to contain only zVariables when in zMode. If the same CDF is then opened without using zMode, the rVariables will still exist.

### -0.0 to 0.0 Mode

The floating-point value -0.0 is legal on those computers which use the IEEE 754 floating-point representation (e.g., UNIX-based computers, the Macintosh, and the IBM PC) but is illegal on VAXes and DEC Alphas running OpenVMS. Attempting to use -0.0 results in a reserved operand fault on a VAX and a high performance arithmetic fault on a DEC Alpha running OpenVMS. Because of this the CDF library can be told to convert -0.0 to 0.0 when read from or written to a CDF. When reading from a CDF the values physically stored in the CDF are not modified — only the values returned to an application are converted. When writing to a CDF the values physically stored are modified — -0.0 is converted to 0.0 before being written to the CDF. This mode is available on all supported computers but is only really necessary on VAXes and DEC Alphas running OpenVMS. The CDF library is told to convert -0.0 to 0.0 for a CDF via the Internal Interface using the <SELECT_,CDF_NEGtoPOSfp0_MODE_> operation. When this mode is disabled, a warning (NEGATIVE_FP_ZERO) is returned when -0.0 is read from a CDF (and the decoding is that of a VAX or DEC Alpha running OpenVMS) or written to a CDF (and the encoding is that of a VAX or DEC Alpha running OpenVMS).

### 2.1.3 Limits

#### Open CDFs

The only limit on the number of CDFs that may be open simultaneously is the operating system's limit on the number of open files that an application may have. Each open CDF will always have at least one associated open file (the `.cdf` file). The CDF library will open and close the variable files of a multi-file CDF as needed (see Sections 2.3.2 and 2.3.3).

### 2.1.4 Caching Scheme

The CDF library reads and writes to open files in 512-byte blocks. A cache of 512-byte memory buffers is maintained by the CDF library for each open file. The CDF library attempts to keep in the cache the set of file blocks currently being accessed. This results in fewer actual I/O operations to the file if repeated accesses to these blocks would occur. When the cache is completely full and a new block of the file is accessed, one of the cache buffers is written back to the file (if it was modified) and the new block is read into that cache buffer (unless the file is being extended in which case the cache buffer is simply cleared). This process is known as paging. By optimizing the number of cache buffers for a file, improved performance can be achieved. There is tradeoff between having too few cache buffers and having too many. Having too few cache buffers will cause excessive paging while having too many cache buffers *may* slow performance because of the overhead involved in maintaining the cache.

The CDF library attempts to choose optimal default cache sizes based on a CDF's format and number of variables.[6] This is difficult since the CDF library does not know how an application will access a CDF. For that reason an application may specify, via the Internal Interface, the number of cache buffers to be used for a file. The number of cache buffers may be changed as many times as necessary while a file is open (the first time will override the default used by the CDF library). Default cache sizes may be configured for your CDF distribution when it is built and installed. Consult your system manager for the values of these defaults.

The situations in which it will be necessary to specify a cache size will depend on how a CDF is accessed. For example, consider a variable in a multi-file, row-major CDF having a dimensionality of `2:[10,64]`, a data specification of `CDF_REAL8/1`, and variances of `T/TT`. This variable definition results in each record of the variable being written across 10 file blocks with the second dimension varying the fastest (since the CDF's variable majority is row-major). If single value reads were used to access this variable (see Section 2.3.13), only one cache buffer would be necessary if the second dimension were incremented the fastest (i.e., `[1,1]`, `[1,2]`,... `[10,63]`, `[10,64]`). This is because the values of a record would be accessed sequentially from the first block to the last block. If, however, the first dimension were incremented the fastest (i.e., `[1,1]`, `[2,1]`,... `[9,64]`, `[10,64]`), 10 cache buffers would improve performance. The values of a record are not being accessed sequentially but rather each read would be from a different block. Since the reads would be spread access 10 blocks, having 10 cache buffers would be optimal.

A similar situation arises when accessing variables in a single-file CDF. If values are accessed for each variable at a record number, then performance will be improved by setting the number of cache buffers to be greater than the number of variables. This is because the variable values will most likely be located in that many different file blocks for a particular record number.

Cache sizes are specified using the `<SELECT_,CDF_CACHESIZE_>`, `<SELECT_,rVAR_CACHESIZE_>`, `<SELECT_,`

[6] As variables are created in a single-file CDF, the CDF library will attempt to adjust the cache size accordingly. This may be overridden by an application if desired.

rVARs_CACHESIZE_>, <SELECT_,zVAR_CACHESIZE_>, and <SELECT_,zVARs_CACHESIZE_>, operations of the Internal Interface.

**NOTE:** The default cache sizes used by the CDF library are fairly conservative in order to minimize the problems that can arise due to memory limitations (especially on computers having limited memory such as the IBM PC and Macintosh). If the performance of your application is critical, it is very important to experiment with using larger cache sizes. Significant gains in performance can be achieved with the proper cache size. It is also important to allocate records if you know how many are to be written. This will reduce the fragmentation that can occur in a single-file CDF (which degrades performance because of the increased indexing that occurs for a variable). Allocating records is described in Section 2.3.8.

## 2.2 CDFs

The following sections describe various aspects of a CDF.

### 2.2.1 Accessing

Only Version 2 CDFs may be accessed with the current CDF distribution. Version 1 CDFs must be converted to Version 2 CDFs using the CDFconvert program in a CDF distribution prior to CDF V2.5 before they will be readable.

All supported CDF operations are available using the Internal Interface. A subset of these operations are available using the Standard Interface. The Obsolete Interface is no longer supported. (Applications written for CDF Version 1 must be ported to the Standard or Internal Interface of CDF Version 2.)

### 2.2.2 Creating

A CDF must be created by the CDF library. In a C application CDFs are created using either the CDFcreate function (Standard Interface) or the <CREATE_,CDF_> operation of the CDFlib function (Internal Interface). In a Fortran application CDFs are created using either the CDF_create subroutine (Standard Interface) or the <CREATE_,CDF_> operation of the CDF_lib subroutine (Internal Interface).

### 2.2.3 Opening

An application must open an existing CDF before access to that CDF is allowed by the CDF library. In a C application CDFs are opened using either the CDFopen function (Standard Interface) or the <OPEN_, CDF_> operation of the CDFlib function (Internal Interface). In a Fortran application CDFs are opened using either the CDF_open subroutine (Standard Interface) or the <OPEN_,CDF_> operation of the CDF_lib subroutine (Internal Interface).

### 2.2.4 Closing

It is absolutely essential that a CDF that has been created or modified by an application be closed before the program exits. If the CDF is not closed it will in most cases be corrupted and unreadable. This is because the cache buffers maintained by the CDF library will not have been written to the CDF file(s). An existing CDF that has been opened and only read from should also be closed. In a C application CDFs are closed using either the `CDFclose` function (Standard Interface) or the `<CLOSE_,CDF_>` operation of the `CDFlib` function (Internal Interface). In a Fortran application CDFs are closed using either the `CDF_close` subroutine (Standard Interface) or the `<CLOSE_,CDF_>` operation of the `CDF_lib` subroutine (Internal Interface).

### 2.2.5 Deleting

An open CDF may be deleted at any time. The `.cdf` file is deleted along with any variable files if a multi-file CDF. Note that if the CDF is corrupted and cannot be opened by the CDF library you will have to delete the CDF file(s) manually using the capabilities of the operating system being used. In a C application CDFs are deleted using either the `CDFdelete` function (Standard Interface) or the `<DELETE_,CDF_>` operation of the `CDFlib` function (Internal Interface). In a Fortran application CDFs are deleted using either the `CDF_delete` subroutine (Standard Interface) or the `<DELETE_,CDF_>` operation of the `CDF_lib` subroutine (Internal Interface).

### 2.2.6 Naming

The pathname specified when opening or creating a CDF can be any legal pathname for the operating system being used. This includes logical symbols on VMS systems and environment variables on UNIX systems. Trailing blanks are also allowed but will be ignored. This is so Fortran applications do not have to be concerned with the trailing blanks of a Fortran `CHARACTER` variable. (C character string variables use terminating `NUL` characters.)

In almost all cases when a CDF pathname is specified, the `.cdf` extension should not be appended. (It will be appended automatically by the CDF library.) The exception to this is when a user has renamed an existing CDF with a different extension or with no extension (for whatever reason). When a CDF is opened, the CDF library first appends the `.cdf` extension to the pathname specified and then checks to see if that file exists.[7] If not, the CDF library will also check to see if a file exists whose pathname is exactly as specified (without `.cdf` appended). If this is the case, the CDF must be single-file. If the CDF is multi-file, an error occurs since the CDF library would have no idea as to how the variable files had been renamed. Note also that the CDF library always appends `.cdf` to the pathname specified when creating a CDF.

**NOTE:** The CDF toolkit programs will in some cases not recognize a CDF if it does not have an extension of `.cdf`.[8]

[7] Actually, the CDF library will check several possible extensions: `.cdf`, `.cdf;1`, `.CDF`, and `.CDF;1`. These extensions are checked because some CD-ROM drivers (primarily on UNIX machines) do peculiar things when making the files (CDFs) on a CD-ROM visible.

[8] Or `.cdf;1` or `.CDF` or `.CDF;1`.

### 2.2.7 Format

There are two CDF formats: multi-file and single-file. The choice of which format to use will depend on how the CDF is to be accessed. Note that the `CDFconvert` toolkit program can be used to change the format of an existing CDF (creating a new CDF with the desired format).

The default format for a created CDF was determined when your CDF distribution was built and installed. Consult your system manager for this default. In a user application, the Internal Interface must be used to change the format of a CDF. The format of an existing CDF can be changed only if no variables have been created in the CDF. If the `SkeletonCDF` toolkit program is used to create a CDF, the format is specified in the skeleton table (see Section 3.8).

A CDF's format is changed by using the `<PUT_,CDF_FORMAT_>` operation of the Internal Interface.

#### Multi-File CDFs

A multi-file CDF (`MULTI_FILE`) consists of one file (with extension `.cdf`) containing control information and attribute entry data and a separate file for each variable defined in the CDF (with extensions `.v0`,`.v1`,... for rVariables and `.z0`,`.z1`,... for zVariables). Each variable file contains the data values for the corresponding variable. (The control information for each variable is stored in the `.cdf` file.)

#### Single-File CDFs

A single-file CDF (`SINGLE_FILE`) consists of only one file (with extension `.cdf`). This file contains the control information for the entire CDF, the attribute entry data, and all of the variable data. An indexing scheme is used to provide efficient access to variable records.

**Indexing Scheme.** In single-file CDFs an indexing scheme is used to keep track of where a variable's records are located within the `.cdf` file. The order that variable (and attribute entry) values are written to a single-file CDF by an application may result in a variable's records being noncontiguous. There will be groups of contiguous records, but these groups will not be contiguous. This section will give a brief overview of how a variable's records are indexed.

For each variable in a single-file CDF one or more index records will exist. Each of these index records will contain one or more index entries.[9] An index entry consists of the following fields:

| | |
|---|---|
| `starting record` | The number of the first record in a group of contiguous records. |
| `ending record` | The number of the last record in a group of contiguous records. |
| `byte offset` | The byte offset within the `.cdf` file of the group of contiguous records. |

To find a particular variable record the CDF library must search through the index entries for that variable. Improved performance will result if there are fewer index entries to search. This can be achieved by having a

[9] Actually, all but the last index record will be completely full of index entries. The last index record will contain one or more index entries.

large number of records in each group of contiguous records (resulting in fewer groups/index entries). Even better would be only one group of contiguous records per variable. Techniques used to achieve fewer index entries are described in the "Allocated Records" and "Extend Records" subsections of Section 2.3.8.

It is possible to inquire the indexing statistics for a variable. Using the Internal Interface, an application may inquire the number of index records and total number of index entries for rVariables and zVariables using the `<GET_,rVAR_nINDEXRECORDS>`, `<GET_,rVAR_nINDEXENTRIES>`, `<GET_,zVAR_nINDEXRECORDS>`, and `<GET_, zVAR_nINDEXENTRIES>` operations.

#### Performance

The most efficient access to CDF variables will usually occur when the CDF has the multi-file format. The extra overhead involved with the indexing scheme used in single-file CDFs is small, so the difference may not be significant (especially if hyper reads/writes are used). The drawback to using the multi-file format is that more than one file is associated with a CDF (which may or may not be a problem for your system management).

There is a case in which the single-file format may be more efficient. If a CDF has a large number of variables (larger than the number of files that may be open at once by an application) and the variables values are accessed variable-by-variable (rather than accessing an entire variable before going to the next variable), the multi-file format may be much slower than the single-file format. This is because the CDF library will have to close one variable file and then open another as each variable value is accessed by the application (since the operating system's open file limit will be reached). If the application was to access every value for a variable before going on to the next variable, this would not occur (but it might create complications for your application).

Note that the format of a CDF can also be converted using the `CDFconvert` toolkit program (which creates a new CDF with the specified format). Section 3.4 describes `CDFconvert`.

### 2.2.8 Encoding

The encoding of a CDF determines how attribute entry data and variable data values are stored on disk in the CDF file(s). An application program never has to concern itself with the encoding of the CDF being accessed. The CDF library performs all of the encoding and decoding of data values for the application.

A CDF's encoding is specified when the CDF is created when using the Standard Interface but is set to the default encoding for your CDF distribution when created using the Internal Interface. The encoding of an existing CDF may be changed with the Internal Interface if no variable values or attribute entries have been written (variables and attributes may exist, however). If the `SkeletonCDF` toolkit program is used to create a CDF the encoding is specified in the skeleton table (see Section 3.8).

The encoding specified when creating/modifying a CDF may be any of the native encodings for the computers supported by CDF in addition to network (XDR) encoding.[10] A CDF with any supported encoding is also readable on any computer supported by CDF.

[10] This is a change from previous releases of CDF.

### Host Encodings

Host encoding (`HOST_ENCODING`) specifies that variable and attribute entry data values be written to the CDF in the native encoding of the computer being used. In addition, the following explicit host encodings are supported:

| | |
|---|---|
| `VAX_ENCODING` | VAX and microVAX computers running VMS. Double-precision floating-point values are encoded in Digital's `D_FLOAT` representation. |
| `ALPHAVMSd_ENCODING` | DEC Alpha computers running OpenVMS. Double-precision floating-point values are encoded in Digital's `D_FLOAT` representation. |
| `ALPHAVMSg_ENCODING` | DEC Alpha computers running OpenVMS. Double-precision floating-point values are encoded in Digital's `G_FLOAT` representation. |
| `SUN_ENCODING` | Sun computers. |
| `SGi_ENCODING` | Silicon Graphics Iris and Power Series computers. |
| `DECSTATION_ENCODING` | DECstation computers. |
| `ALPHAOSF1_ENCODING` | DEC Alpha computers running OSF/1. |
| `IBMRS_ENCODING` | IBM RS6000 series computers. |
| `HP_ENCODING` | HP 9000 series computers. |
| `IBMPC_ENCODING` | IBM PC personal computers. |
| `NeXT_ENCODING` | NeXT computers. |
| `MAC_ENCODING` | Macintosh computers. |

When `HOST_ENCODING` is specified, it is translated to the actual host encoding from the above list.

### Network Encoding

Network encoding (`NETWORK_ENCODING`) specifies that variable and attribute entry data values be written to the CDF in the XDR (External Data Representation) format. As values are written to the CDF, the CDF library encodes them into network encoding. Network encoded CDFs are readable on any machine supported by CDF.

### Equivalent Encodings

While an encoding parameter exists for each supported computer, not every encoding is different. The following sections describe which computers use the same encoding for the various data types.

**Character/1-Byte Integer Data Types.** Since each supported computer uses the ASCII character set and orders the bits in a byte the same way, the character and 1-byte integer data types (`CDF_CHAR`, `CDF_UCHAR`, `CDF_BYTE`, `CDF_INT1`, and `CDF_UINT1`) are encoded in the same way on each.

**Multiple-Byte Integer Data Types.** The multiple-byte integer data types (`CDF_INT2`, `CDF_UINT2`, `CDF_INT4`, and `CDF_UINT4`) are encoded in one of two ways: big-endian or little-endian. Big-endian has the least significant byte (LSB) in the highest memory location while little-endian has the LSB in the lowest memory location. The supported computers use big-endian or little-endian as shown in Table 2.3. Network (XDR) encoding uses big-endian encoding for multiple-byte integer data types.

| Big-Endian | Little-Endian |
|---|---|
| Sun | VAX |
| SGi Iris | DECstation |
| IBM RS6000 | IBM PC |
| HP 9000 | DEC Alpha (OSF/1) |
| NeXT | DEC Alpha (OpenVMS) |
| Macintosh | |

Table 2.3: Equivalent Byte Orderings

**Single-Precision Floating-Point Data Types.** The single-precision floating-point encodings on the supported computers are either IEEE 754 floating-point or Digital's `F_FLOAT` floating-point. There are also two different byte orderings for the computers that use IEEE 754 (big-endian and little-endian). The single-precision floating-point encodings for each supported computer are shown in Table 2.4. Network (XDR) encoding uses IEEE 754 (big-endian) encoding for single-precision floating-point data types.

| IEEE 754 (Big Endian) | IEEE 754 (Little Endian) | Digital's `F_FLOAT` |
|---|---|---|
| Sun | DECstation | VAX |
| SGi Iris | IBM PC | DEC Alpha/OpenVMS |
| IBM RS6000 | DEC Alpha/OSF1 | |
| HP 9000 | | |
| NeXT | | |
| Macintosh | | |

Table 2.4: Equivalent Single-Precision Floating-Point

**Double-Precision Floating-Point Data Types.** The double-precision floating-point encodings on the supported computers are either IEEE 754 floating-point, Digital's `D_FLOAT` floating-point, or Digital's `G_FLOAT` floating-point. There are also two different byte orderings for the computers that use IEEE 754 (big-endian and little-endian). The double-precision floating-point encodings for each supported computer are shown in Table 2.5. Network (XDR) encoding uses IEEE 754 (big-endian) encoding for double-precision floating-point data types.

| IEEE 754 (Big Endian) | IEEE 754 (Little Endian) | Digital's `D_FLOAT` | Digital's `G_FLOAT` |
|---|---|---|---|
| Sun | DECstation | VAX | DEC Alpha/OpenVMS/`G` |
| SGi Iris | IBM PC | DEC Alpha/OpenVMS/`D` | |
| IBM RS6000 | DEC Alpha/OSF1 | | |
| HP 9000 | | | |
| NeXT | | | |
| Macintosh | | | |

Table 2.5: Equivalent Double-Precision Floating-Point

**Performance**

The best performance when accessing (reading or writing) a CDF will occur when that CDF is in the host encoding of the computer being used (and host decoding is in effect — see Section 2.2.9). This is because no encoding or decoding has to be performed by the CDF library. A CDF that must be portable between two or more different types of computers should normally be network encoded. There may be cases, however, where it would be desirable to create a CDF with host encoding (e.g., on a slow machine) and then transfer it to a faster machine for processing or conversion to another encoding. Obviously, there are trade-offs as to which encoding should be used in any one particular case. Keep in mind that a CDF can always be converted to the host encoding of the machine being used (with `CDFconvert`) before being accessed.

### 2.2.9 Decoding

The decoding of a CDF determines how attribute entry and variable data values are passed to a calling application program from the CDF library. The default decoding when a CDF is initially opened is host decoding (the native encoding of the computer being used). When host decoding is in effect, all data values read by an application are immediately ready for manipulation and display. Almost all of your applications will simply use the default of host decoding and not be concerned with selecting a decoding. There are some situations, however, where selecting a different decoding will be advantageous. Some possibilities are as follows:

1. A client/server model where a number of CDFs are maintained on a server computer (in any of the supported encodings). Clients on different type computers could request data from a CDF on the server computer. The server computer would then select a decoding for the CDF based on the client's computer type and then read the data value(s). The value(s) could then be sent directly to the client computer by the server computer without a conversion being necessary by either the client or the server. The CDF library would perform the necessary conversions.
2. If data values were being read from a CDF and written in binary form to a file for use on a different type computer. The proper decoding could be selected for the CDF before any of the data values are read. No conversions would be necessary by the application program.

A CDF's decoding may be selected and reselected at any time after the CDF has been opened and as many times as necessary. A CDF's decoding is selected via the Internal Interface with the `<SELECT_, CDF_DECODING_>` operation. Also, a CDF's decoding does not affect the values that already exist in a CDF or any values subsequently written. A CDF's encoding determines how the values are written to the CDF file(s). Section 2.2.8 describes a CDF's encoding.

The supported decodings correspond to the supported encodings. They are as follows:

| | |
|---|---|
| `HOST_DECODING` | The data representation of the host computer. |
| `NETWORK_DECODING` | The External Data Representation (XDR). |
| `VAX_DECODING` | VAX and microVAX data representation. Double-precision floating-point values will be in Digital's `D_FLOAT` representation. |
| `ALPHAVMSd_DECODING` | DEC Alpha running OpenVMS data representation. Double-precision floating-point values will be in Digital's `D_FLOAT` representation. |

| | |
|---|---|
| `ALPHAVMSg_DECODING` | DEC Alpha running OpenVMS data representation. Double-precision floating-point values will be in Digital's `G_FLOAT` representation. |
| `SUN_DECODING` | Sun data representation. |
| `SGi_DECODING` | Silicon Graphics Iris and Power Series data representation. |
| `DECSTATION_DECODING` | DECstation data representation. |
| `ALPHAOSF1_DECODING` | DEC Alpha running OSF/1 data representation. |
| `IBMRS_DECODING` | IBM RS6000 series data representation. |
| `HP_DECODING` | HP 9000 series data representation. |
| `IBMPC_DECODING` | IBM PC data representation. |
| `NeXT_DECODING` | NeXT data representation. |
| `MAC_DECODING` | Macintosh data representation. |

#### Performance

The best performance when reading a CDF will occur when the CDF's decoding is the same as the CDF's encoding since no conversion will have to be performed by the CDF library. Since host decoding is the only directly usable decoding by an application, CDFs with the host's encoding will provide the best performance. Care should be taken when selecting the encoding for a CDF.

### 2.2.10 Data Types

CDF supports a variety of data types consistent with the types available with C and Fortran compilers on most computers. All data types are based on an 8-bit byte. The size of an element of a data type is the same regardless of the computer/operating system being used. The `<GET_,DATATYPE_SIZE_>` operation of the Internal Interface may be used to inquire the size in bytes of a particular data type.

#### Integer Data Types

| | |
|---|---|
| `CDF_BYTE` | 1-byte, signed integer. |
| `CDF_INT1` | 1-byte, signed integer. |
| `CDF_UINT1` | 1-byte, unsigned integer. |
| `CDF_INT2` | 2-byte, signed integer. |
| `CDF_UINT2` | 2-byte, unsigned integer. |
| `CDF_INT4` | 4-byte, signed integer. |
| `CDF_UINT4` | 4-byte, unsigned integer. |

**NOTE:** When using C on a DEC Alpha running OSF/1, keep in mind that a `long` is 8 bytes and that an `int` is 4 bytes. Use an `int` with the data types `CDF_INT4` and `CDF_UINT4` rather than a `long`.

**NOTE:** When using C on an IBM PC (MS-DOS), keep in mind that an `int` is 2 bytes and that a `long` is 4 bytes. Use a `long` with the data types `CDF_INT4` and `CDF_UINT4` rather than an `int`.

#### Floating Point Data Types

`CDF_REAL4 & CDF_FLOAT`
: 4-byte, single-precision floating-point.

`CDF_REAL8 & CDF_DOUBLE`
: 8-byte, double-precision floating-point.

A special case exists with respect to the value `-0.0` (negative floating-point zero). This value is legal on those computers that use the IEEE 754 floating-point representation (e.g., most UNIX-based computers and the IBM PC) but is illegal on VAXes and DEC Alphas running OpenVMS. Attempting to use `-0.0` will result in a reserved operand fault on a VAX and a high performance arithmetic fault on a DEC Alpha running OpenVMS. A warning is returned whenever `-0.0` is read by an application on a VAX or DEC Alpha running OpenVMS. The CDF library can be put into a mode where `-0.0` will be converted to `0.0` when detected (see Section 2.1.2). If `-0.0` is not being converted to `0.0`, the CDF toolkit programs are designed to display `-0.0` in all cases. This includes those computers that normally suppress the negative sign.

#### Character Data Types

`CDF_CHAR` 1-byte, character.

`CDF_UCHAR` 1-byte, unsigned character.

Character data types are unique for variables in that they are the only data types for which more than one element per value is allowed. Each variable value consists of a character string with the number of elements being the number of characters. More than one element is allowed for any of the data types when dealing with attribute entries.

#### EPOCH Data Type

`CDF_EPOCH` 8-byte, double precision floating point.

The EPOCH data type is used to store time values referenced from a particular epoch. For NSSDC applications that epoch is `01-Jan-0000 00:00:00.000`.[11] `CDF_EPOCH` values are the number of milliseconds since the epoch. CDF toolkit programs will display and expect `CDF_EPOCH` values in the format

[11] I know what you're thinking. The year 0 AD never existed. If it makes you feel better, think of the epoch year as 1 BC rather than 0 AD.

```
dd-mmm-yyyy hh:mm:ss.ccc
```

where

| | |
|---|---|
| dd | is the day of the month, 1-31, |
| mmm | is the month, Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec, |
| yyyy | is the year, AD, |
| hh | is the hour, 0-23, |
| mm | is the minute, 0-59, |
| ss | is the second, 0-59, and |
| ccc | is the millisecond, 0-999. |

Functions exist that parse, encode, compute, and break down CDF_EPOCH values. These functions are described in the CDF C Reference Manual for C applications and in the CDF Fortran Reference Manual for Fortran applications.

#### Equivalent Data Types

Certain data types are considered equivalent with respect to their representation in memory and in a CDF. The following groups contain equivalent data types.

| | | | | |
|---|---|---|---|---|
| CDF_CHAR | CDF_INT2 | CDF_INT4 | CDF_REAL4 | CDF_REAL8 |
| CDF_UCHAR | CDF_UINT2 | CDF_UINT4 | CDF_FLOAT | CDF_DOUBLE |
| CDF_INT1 | | | | CDF_EPOCH |
| CDF_UINT1 | | | | |
| CDF_BYTE | | | | |

Note that while the signed and unsigned forms of a data type are considered equivalent by the CDF library, they must be correctly interpreted by an application to produce the desired results.

### 2.2.11 Limits

Limits within a CDF are defined in the appropriate include files: cdf.h for C applications and cdf.inc for Fortran applications. The following limits exist:[12]

| | |
|---|---|
| CDF_MAX_DIMS | The maximum number of dimensions that rVariables/zVariables may have. |
| CDF_VAR_NAME_LEN | The maximum number of characters in a variable name. |
| CDF_ATTR_NAME_LEN | The maximum number of characters in an attribute name. |
| CDF_PATHNAME_LEN | The maximum number of characters in a pathname used to specify a CDF. |

Most of these limits can be raised. Contact CDF User Support if that becomes necessary.

[12] Previous releases of CDF limited the number of variables a CDF could contain. That limit has been eliminated except for multi-file CDFs on an IBM PC because of the 8.3 naming convention.

## 2.3 Variables

CDF variables are the mechanism for storing data. (Attributes are used to store metadata.) A new variable may be created in a CDF at any time. Two types of variables are supported: rVariables and zVariables.[13] The main difference is that all rVariables in a CDF have the same dimensionality whereas zVariables can have differing dimensionalities. In the following sections differences between the two types will be noted where appropriate.

### 2.3.1 Accessing

The Standard Interface deals exclusively with rVariables. No access to zVariables is provided. The Internal Interface may be used to access either rVariables or zVariables.

### 2.3.2 Opening

The CDF library automatically opens the variable files in a multi-file CDF as the variables are accessed. An application never has to concern itself with opening variables. The opening of variables does not apply to single-file CDFs since individual files do not exist for each variable.

### 2.3.3 Closing

The CDF library automatically closes the variable files in a multi-file CDF when the CDF itself is closed by an application.[14] Variable files are also closed automatically by the CDF library as other variables are accessed if insufficient file pointers exist to keep all of the variables open at once. This would be due to an open file quota enforced by the operating system being used.

A case also exists where it may be beneficial for an application to close a variable in a multi-file CDF. Since each open variable file uses some number of cache buffers, a large amount of system memory could be in use (see Section 2.1.4). This may not be a problem on VAX or UNIX machines but could result in a program crashing on an MS-DOS machine. If memory is limited, an application may want to close variables after they have been accessed in order to minimize the total number of cache buffers being used. In a C application rVariables are closed using either the `CDFvarClose` function (Standard Interface) or the `<CLOSE_, rVAR_>` operation of the `CDFlib` function (Internal Interface). zVariables are closed using the `<CLOSE_,zVAR_>` operation of the `CDFlib` function (Internal Interface). In a Fortran application rVariables are closed using either the `CDF_var_close` subroutine (Standard Interface) or the `<CLOSE_,rVAR_>` operation of the `CDF_lib` subroutine (Internal Interface). zVariables are closed using the `<CLOSE_,zVAR_>` operation of the `CDF_lib` subroutine (Internal Interface).

The closing of variables does not apply to single-file CDFs since individual files do not exist for each variable.

[13] The letters "r" and "z" don't stand for anything in particular. "r" sort of stands for "regular" since rVariables have always been supported by CDF.

[14] It is required that an application close a CDF before exiting.

### 2.3.4 Naming

Each variable in a CDF has a unique name. This applies to rVariables and zVariables together (i.e., an rVariable cannot have the same name as a zVariable). Variable names are case sensitive regardless of the operating system being used and may consist of up to `CDF_VAR_NAME_LEN` printable characters (including blanks). Trailing blanks, however, are ignored when the CDF library compares variable names. `"LAT"` and `"LAT     "` are considered to be the same name, so they cannot both exist in the same CDF. This was done because Version 1 of CDF padded variable names on the right with blanks out to eight characters. When a Version 1 CDF was converted to a Version 2 CDF these trailing blanks remained in the variable names. To allow CDF Version 2 applications to read such a CDF without having to be concerned with the trailing blanks, the trailing blanks are ignored by the CDF library when comparing variable names. The trailing blanks are returned as part of the name, however, when a variable is inquired by an application program.

### 2.3.5 Numbering

The rVariables in a CDF are numbered consecutively starting at one (1) for Fortran applications and starting at zero (0) for C applications. Likewise, the zVariables in a CDF are numbered consecutively starting at one (1) for Fortran applications and starting at zero (0) for C applications. The CDF library assigns variable numbers as the variables are created.

### 2.3.6 Deleting

A variable may be deleted from a single-file CDF.[15] Deleting a variable also causes the deletion of the corresponding attribute entries for the variable. The disk space used by the variable definition, the variable's data records, and the corresponding attribute entries becomes available for use as needed by the CDF library. Also, the variables which numerically follow the variable being deleted are renumbered immediately. (Each is decremented by one.) Variables are deleted using the `<DELETE_,rVAR_>` and `<DELETE_,zVAR_>` operations of the Internal Interface.

### 2.3.7 Dimensionality

Variable values are stored in arrays. A variable's dimensionality refers to the number of dimensions and the dimension sizes of these arrays.

Each rVariable in a CDF has the same dimensionality. An array of values exists for each rVariable at each record in a CDF. The values may not be physically stored but may be virtual (see Sections 2.3.8, 2.3.10, and 2.3.11).

A zVariable may have a dimensionality which is different from that of the rVariables and the other zVariables. An array of values exists for each zVariable at each record in a CDF. As with rVariables the values may not be physically stored but may be virtual. zVariables are intended for use in those situations where using an rVariable would waste disk space or not logically make sense.

A variable array having two or more dimensions also contains subarrays. For instance, in a 3-dimensional array with dimension sizes `[10,20,30]`, each array consists of ten 2-dimensional subarrays of size `[20,30]`,

[15] Variables may not currently be deleted from a multi-file CDF.

and each of those 2-dimensional subarrays consists of twenty 1-dimensional subarrays of size `[30]`. Subarrays will be referred to when discussing other properties of CDF variables.

### 2.3.8 Records

A CDF record is a set of variable arrays, one per rVariable and one per zVariable in the CDF. The variable arrays in a particular record are generally related to each other in some way (often time). This does not have to be the case and is not enforced by the CDF library in any way. A variable record is simply the corresponding variable array within a CDF record.

Physical variable records are actually stored in the CDF file(s). Virtual variable records are not actually stored but do exist in the conceptual view of the variable provided by CDF. Virtual records can occur in a CDF because of the following reasons:

1. If a variable's values do not vary from record to record (record variance of `NOVARY`), all of that variable's records beyond the first one are virtual and have the same values as the first record (only the first record is physically stored). If a record has not yet been written to that variable, then all of its records are virtual and contain the pad value for that variable.

2. If a variable's values do vary from record to record (record variance of `VARY`), then the records beyond the last record actually written are virtual and contain the pad value for that variable.

Record variance is described in Section 2.3.10.

The maximum record written is maintained by the CDF library for each variable in the CDF. The "maximum CDF record" is simply the maximum rVariable record written (of all the rVariables). This quantity is available through the Standard Interface when inquiring about a CDF. Because the Standard Interface does not allow access to zVariables, zVariables are not considered when determining the "maximum CDF record." The "maximum CDF record" would be used by applications dealing only with rVariables. The maximum record written for each rVariable and zVariable is available via the Internal Interface.

Figure 2.1 illustrates the relationships between physical and virtual variable records. Variable 1 has five records that were physically written. Only two records were physically written to variable 2 so the following records are virtual (containing the pad value for that variable). Only one record can be physically written to variable 3 because its record variance is `NOVARY`. The other records are virtual and contain the same values as the first record. Because a record has not been physically written to variable 4, all of its records are virtual containing the pad value for that variable. Likewise, since no records have been written to variable 5, all of its records are also virtual and contain the pad value for that variable. Note that if these are rVariables, the "maximum CDF record" would be five (when displayed by a toolkit program or in a Fortran application — it would be four in a C application).

Note that a variable's records do not have to be written sequentially starting at the first record. The records may be written in any order. For a variable with `VARY` record variance, if a new record more than one record beyond the current maximum record for a variable is written, the intervening records will be physically written and contain the pad value for that variable. Also, when one or more values are written to a new physical record, the entire record is physically written with the pad value for the variable being used for the unspecified values. Variable pad values are described in Section 2.3.17.

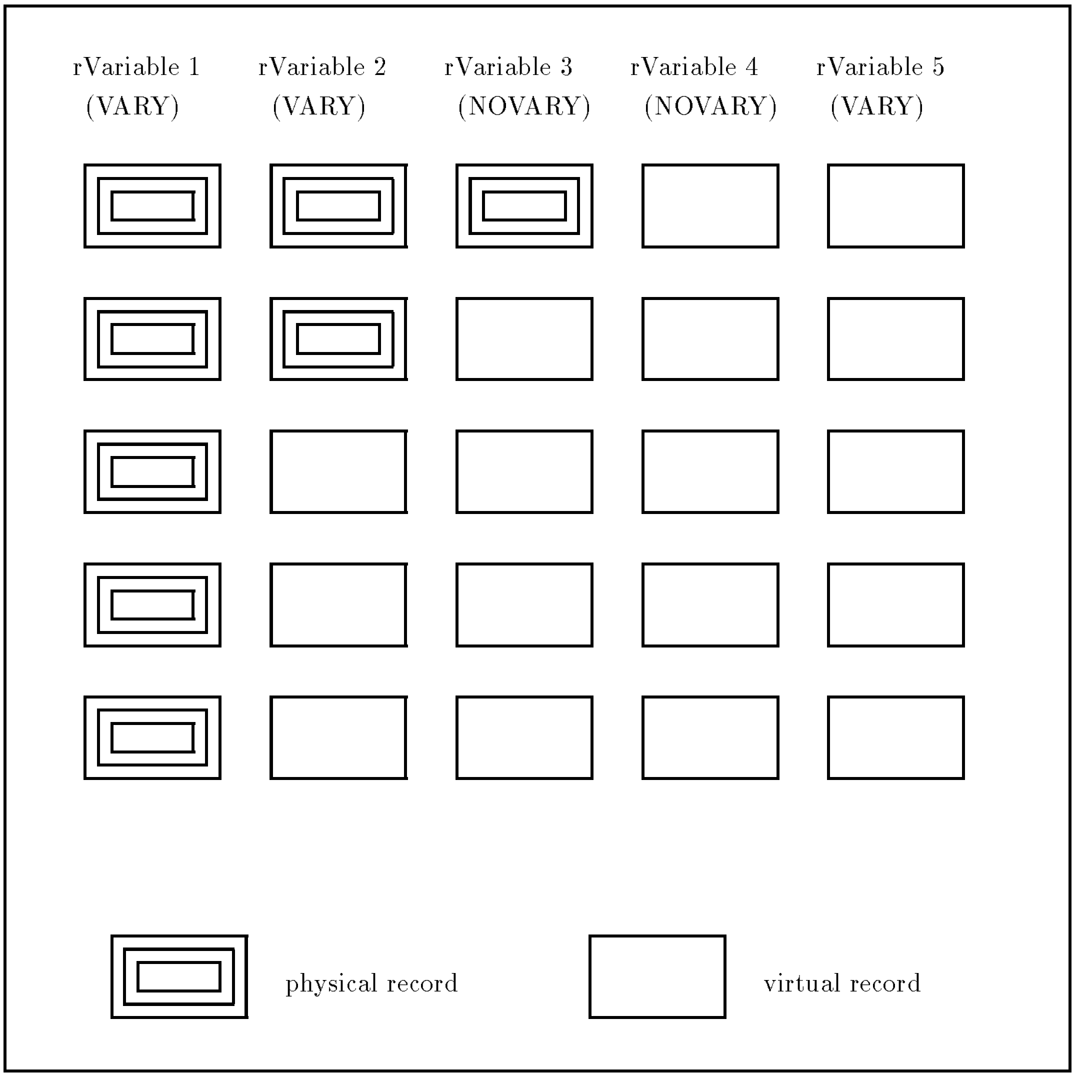


Figure 2.1: Physical vs. Virtual Records

### Numbering

The record numbers in a CDF are numbered starting at one (1) for Fortran applications and starting at zero (0) for C applications.

### Allocated Records

The Internal Interface may be used to allocate records for a variable in a single-file CDF.[16] Normally the number of records allocated would be the number that are to be written (assuming this can be determined). This can greatly improve performance when writing (and reading) values for the variable because of reduced overhead when searching the index entries (as described in Section 2.2.7). The application is expected to

[16] There is no reason to allocate records for a variable in a multi-file CDF.

write to all of the allocated records. When the CDF is closed, any allocated records for a variable which have not yet been written are written using the pad value for the variable. Records may only be allocated for a variable once but any number of records can eventually be written to the variable. For NRV variables, only one record may be allocated (because only one record will ever physically exist).

Performance is improved when using this method because the allocated records will be contiguous and will require only one index entry. This will greatly improve the time needed to locate a particular record when the variable is accessed. In addition, the CDF will be slightly smaller because of the reduced number of index records needed per variable.

Records are allocated using the `<PUT_,rVAR_ALLOCATERECS_>` and `<PUT_,zVAR_ALLOCATERECS_>>` operations of the Internal Interface.

#### Initial Records

The Internal Interface may be used to specify an initial number of records for a variable. The pad value for the variable is written at each record as if the application had done so itself. The Internal Interface allows this to be done more conveniently with only one function call. Note that the default pad value for the variable's data type will be used unless a pad value is specified for the variable (also using the Internal Interface). If a specific pad value is desired for a variable, then it must be specified before the number of initial records is specified.

Specifying a number of initial records for a variable would usually be done only for a CDF with the single-file format. Because the records would be allocated contiguously within the CDF file, the indexing scheme (see Section 2.2.7) would require fewer entries making the access to that variable more efficient. Note that this method is not as efficient as allocating records in those cases where all of the records are going to be written by the application. (The records would be written twice. Once with the pad value and then again by the application.)

The number of initial records specified would in most cases be the number of records planned for a variable. Note that additional records may be added to a variable at any time. After the initial records are written with the pad value, the actual data values would be written by the application. Note that for NRV variables the number of initial records must always be specified as one (1). This is because at most only one physical record will ever actually be written.

Initial records are written to rVariables and zVariables using the `<PUT_,rVAR_INITIALRECS_>` and `<PUT_, zVAR_INITIALRECS_>` operations of the Internal Interface.

#### Extend Records

As new records are written to a variable, additional records are allocated at the end of that variable. For CDFs with the multi-file format, only the records actually needed are appended to the corresponding variable file. The records always remain contiguous (allowing the location of a record to be determined very efficiently).

For single-file CDFs, however, the new records are allocated at the end of the `.cdf` file. The new records may not be located contiguously with the other records for the variable. This results in a new entry being created in the indexing scheme (see Section 2.2.7) used to locate variable records. The indexing scheme is more efficient if fewer entries are needed (resulting from larger blocks of contiguous records being allocated).

The CDF library attempts to optimize the indexing scheme by allocating more records than are actually needed with the assumption that they will eventually be used. A minimum number of records to allocate is maintained for each variable.

Because an application knows how records will be added to a variable more so than the CDF library, the Internal Interface may be used to explicitly specify the number of records to extend a variable when necessary. This only has an effect on CDFs with the single-file format and for variables that are not NRV (since at most only one physical record will ever be written for an NRV variable).

Note the distinction between records allocated and records actually written. The CDF library may allocate more records than are actually written by an application for the reasons stated above. When an application inquires the number of records for a variable, the number written, not the number allocated, will be returned by the CDF library.

Extend records are specified for rVariables/zVariables using the `<PUT_,rVAR_EXTENDRECS_>` and `<PUT_, zVAR_EXTENDRECS_>` operations of the Internal Interface.

### 2.3.9 Data Specification

Each variable in a CDF has a defined data specification. A variable's data specification consists of a data type and a number of elements of that data type. A variable's data specification is specified when the variable is created. The data specification of an existing variable may also be changed if either of the following conditions is true.

1. Values have not yet been written to the variable (including an explicitly written pad value — see Section 2.3.17).

2. The old data type and new data type are considered equivalent, and the number of elements for the variable are the same. Equivalent data types are described in Section 2.2.10.

#### Data Type

The supported data types are described in Section 2.2.10. Variables having any combination of data types may exist in the same CDF.

#### Number of Elements

In addition to a data type, each variable also has a number of elements. This refers to the number of elements of the data type at each variable value. For character string data types this is the number of characters in each string. (A variable value consists of the entire character string.) The character string can be thought of as an array of characters. For non-character string data types, this must always be one (1). An array of elements per variable value is not allowed for non-character string data types.

### 2.3.10 Record Variance

A variable's record variance specifies whether or not the variable's values change from record to record. The effect of a variable's record variance is defined as follows.

| | |
|---|---|
| `VARY` | The values do change from record to record. Each variable record is physically written with no gaps between records (i.e., if a record more than one beyond the maximum record is written, the intervening records are also physically written and contain pad values). If a record is read beyond the maximum record written to a variable, the pad value for the variable is returned. Variables of this type are referred to as record-variant (RV). |
| `NOVARY` | The values do not change from record to record. Only one record is physically written to the variable. Each record contains the same values (including virtual records beyond the first record). Variables of this type are referred to as non-record-variant (NRV). |

Section 2.3.8 describes variable records in more detail.

A variable's record variance is specified when the variable is created. The record variance of an existing variable may be changed only if values have not yet been written to that variable. (An explicit pad value may have been specified however.)

### 2.3.11 Dimension Variance

A variable's dimension variances specify whether or not the values change along the corresponding dimension. The effects of a dimension variance are defined as follows:

| | |
|---|---|
| `VARY` | The values do change along the dimension. All of the values for the dimension (or all of the subarrays) are physically stored. |
| `NOVARY` | The values do not change along the dimension. Only one value (or subarray) is physically written for that dimension. Each value (or subarray) along that dimension is the same (including virtual values/subarrays beyond the first value/subarray). |

Figure 2.2 illustrates the effect of dimension variances on a variable with 2-dimensional arrays (for a particular record). For variable 1 each value in the array is physically stored and therefore unique. Because variable 2 does not vary along the second dimension, each value along that dimension is the same so only one value for that dimension is physically stored (the other values are virtual). The same is true for variable 3 which does not vary along the first dimension. Variable 4 does not vary along either dimension. Only one value is physically stored for the array — all of the other values are the same (they are virtual).

A variable's dimension variances are specified when the variable is created. The dimension variances of an existing variable may be changed only if values have not yet been written to that variable. (An explicit pad value may have been specified, however.)

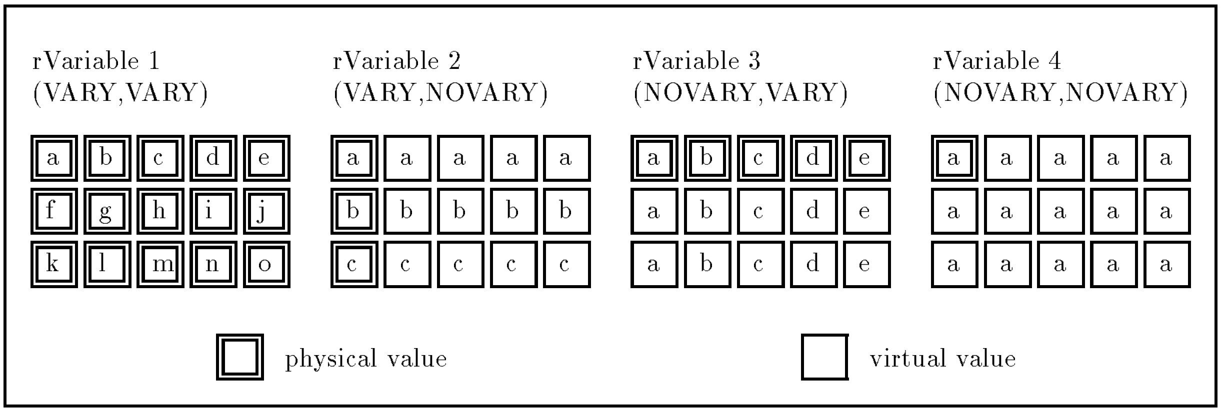


Figure 2.2: Physical vs. Virtual Dimensions

### 2.3.12 Majority

The variable majority of a CDF describes how variable values within each variable array (record) are stored. Each variable in a CDF has the same majority. The majority can be either row major or column major.

| | |
|---|---|
| `ROW_MAJOR` | Row majority. The first dimension changes the slowest. |
| `COL_MAJOR` | Column majority. The first dimension changes the fastest. |

For example, an array for an rVariable with `[VARY,VARY]` dimension variances in a 2-dimensional CDF with dimension sizes `[2,4]` and row majority would be stored as follows:

```
v(1,1), v(1,2), v(1,3), v(1,4), v(2,1), v(2,2), v(2,3), v(2,4)
```

where `v(i,j)` is the value at indices `(i,j)`. If the CDF had column majority, the array would be stored as follows:

```
v(1,1), v(2,1), v(1,2), v(2,2), v(1,3), v(2,3), v(1,4), v(2,4)
```

In each case `v(1,1)` is stored at the low address.

An application needs to be concerned with the majority of a CDF in the following cases:

1. When performing a variable hyper read, the values placed in the buffer by the CDF library will be in the variable majority of the CDF. The application must process the values according to that majority.

   When performing a variable hyper write, the CDF library expects the values in the buffer to be in the variable majority of the CDF. The application must place the values into the buffer in that majority.

2. When sequential access is used, the values are read/written in the order imposed by the variable majority of the CDF.

3. When single value reads/writes are performed, the majority could have an effect. The CDF library uses a caching scheme to optimize[17] the random access to variable values. If all of the values of a record are to be read/written, there may be an increase in performance if the values are accessed with (rather than against) the majority. For example, if the majority is row-major, increment the last index the fastest.

4. When performing a multiple variable read/write, the full-physical records in the buffer will/must be in the variable majority of the CDF.

A CDF's variable majority is specified when the CDF is created when using the Standard Interface but is set to the default variable majority for your CDF distribution when created using the Internal Interface. The majority of an existing CDF may be changed using the Internal Interface only if variable values have not yet been written. (Variables may exist and explicit pad values may have been specified, however.)

### 2.3.13 Single Value Access

Single value access allows only one value to be read from or written to a variable with a single call to the CDF library. Two parameters are specified when performing a single value read/write:

| | |
|---|---|
| `record number` | The record number at which to perform the access. |
| `dimension indices` | The indices within the record at which to perform the access. |

For 0-dimensional CDFs, the dimension indices are not applicable.

Single value access is sensitive to the record and dimension variances of a variable. For instance, if a variable has a record variance of `NOVARY` (with one record written) and a value is read from the fourth record, the CDF library will actually read the value from the first record (the record that is physically stored). If a value were written to the fourth record, the CDF library would actually write the value to the first record (the only record that actually physically exists). If the record variance is `VARY`, the values are written to the actual records. (The physical records are the same as the virtual records.) The same applies to any dimension variances that are `NOVARY`. When a set of indices is specified for a single value read/write, the index for a dimension whose variance is `NOVARY` is forced to the first index regardless of the actual index specified for that dimension (see Section 2.3.11).

In a C application single value access for rVariables is performed using either the `CDFvarGet` and `CDFvarPut` functions (Standard Interface) or the `<GET_,rVAR_DATA_>` and `<PUT_,rVAR_DATA_>` operations of the `CDFlib` function (Internal Interface). Single value access for zVariables is performed using the `<GET_,zVAR_DATA_>` and `<PUT_,zVAR_DATA_>` operations of the `CDFlib` function (Internal Interface). In a Fortran application single value access for rVariables is performed using either the `CDF_var_get` and `CDF_var_put` subroutines (Standard Interface) or the `<GET_,rVAR_DATA_>` and `<PUT_,rVAR_DATA_>` operations of the `CDF_lib` subroutine (Internal Interface). Single value access for zVariables is performed using the `<GET_,zVAR_DATA_>` and `<PUT_, zVAR_DATA_>` operations of the `CDF_lib` subroutine (Internal Interface).

[17] Since an application knows how it will be accessing a variable, it knows best how to optimize the caching scheme used. See Section 2.1.4 for details on how an application can control the CDF library caching scheme.

### 2.3.14 Hyper Access

Hyper access allows more than one value to be read from or written to a variable with a single call to the CDF library. In fact, the entire variable may be accessed at once (if a large enough memory buffer is available to your application). Hyper reads cause the CDF library to read from the variable record(s) in the CDF and place the values into a memory buffer provided by the application. Hyper writes cause the CDF library to take values from a memory buffer provided by the application and write them to the variable records in the CDF. Six parameters are specified when performing a hyper read/write:

| | |
|---|---|
| `record number` | The record number at which to start the access. |
| `record count` | The number of records to access. |
| `record interval` | The interval between records being accessed. An interval of two (2) would indicate that every other record is to be accessed. |
| `dimension indices` | The indices within each record at which the access should begin. |
| `dimension counts` | The number of values along each dimension that should be accessed. |
| `dimension intervals` | For each dimension, the interval between values being accessed. An interval of three (3) would indicate that every third value is to be accessed. |

For 0-dimensional CDFs, the dimension indices, counts, and intervals are not applicable.

A hyper access may or may not read/write a contiguous set of values stored for a variable in the CDF. However, the values in the memory buffer received/provided by the application are contiguous.

Hyper access is sensitive to the record and dimension variances of a variable. For instance, if a variable has a record variance of `NOVARY` (with one record written) and a hyper read of the first five records for that variable is requested, the CDF library will read the single record that is physically stored and place it five times (contiguously) into the memory buffer provided by the application. The same applies to any dimension variances that are `NOVARY`. For example, if the count for a dimension is three and the dimension variance is `NOVARY`, the one value (or subarray) physically stored will be read by the CDF library and placed into the application's memory buffer three times (contiguously).

**Example (Fortran application)**

Assume a 2-dimensional variable array with sizes `[2,4]`, row majority, a record variance of `VARY`, dimension variances of `[VARY,VARY]`, and hyper read parameters as follows:

| | |
|---|---|
| record number | `5` |
| record count | `2` |
| record interval | `1` |
| dimension indices | `1,1` |
| dimension counts | `2,4` |
| dimension intervals | `1,1` |

The values placed in the application's buffer would be as follows (with the first value being in low memory):

```
5(1,1) 5(1,2) 5(1,3) 5(1,4) 5(2,1) 5(2,2) 5(2,3) 5(2,4)
```

```
6(1,1) 6(1,2) 6(1,3) 6(1,4) 6(2,1) 6(2,2) 6(2,3) 6(2,4)
```

where `r(i,j)` is a physically stored value with `r` being the record number, `i` being the first dimension index, and `j` being the second dimension index. (`r`, `i`, and `j` are physical record numbers and dimension indices.)

If the dimension variances had been `[VARY,NOVARY]`, the values placed in the buffer would have been

```
5(1,1) 5(1,1) 5(1,1) 5(1,1) 5(2,1) 5(2,1) 5(2,1) 5(2,1)
6(1,1) 6(1,1) 6(1,1) 6(1,1) 6(2,1) 6(2,1) 6(2,1) 6(2,1)
```

If the record count had been `3` and the record interval `2`, the values placed in the buffer would have been

```
5(1,1) 5(1,2) 5(1,3) 5(1,4) 5(2,1) 5(2,2) 5(2,3) 5(2,4)
7(1,1) 7(1,2) 7(1,3) 7(1,4) 7(2,1) 7(2,2) 7(2,3) 7(2,4)
9(1,1) 9(1,2) 9(1,3) 9(1,4) 9(2,1) 9(2,2) 9(2,3) 9(2,4)
```

If the dimension counts had been `[2,2]` and the dimension intervals `[1,2]`, the values placed in the buffer would have been

```
5(1,1) 5(1,3) 5(2,1) 5(2,3)
6(1,1) 6(1,3) 6(2,1) 6(2,3)
```

If the CDF majority had been column major, the values placed in the buffer would have been...

```
5(1,1) 5(2,1) 5(1,2) 5(2,2) 5(1,3) 5(2,3) 5(1,4) 5(2,4)
6(1,1) 6(2,1) 6(1,2) 6(2,2) 6(1,3) 6(2,3) 6(1,4) 6(2,4)
```

Had these examples been for hyper writes, the CDF library would have expected to find the values in the application's buffer exactly as they were placed there during the corresponding hyper read. In the case where the record interval was `2`, the records being skipped would be written using the variable's pad value if they did not already exist. If they did already exist, they would not be affected.

In a C application hyper reads and writes for rVariables are performed using either the `CDFvarHyperGet` and `CDFvarHyperPut` functions (Standard Interface) or the `<GET_,rVAR_HYPERDATA_>` and `<PUT_,rVAR_HYPERDATA_>` operations of the `CDFlib` function (Internal Interface). Hyper reads and writes for zVariables are performed using the `<GET_,zVAR_HYPERDATA_>` and `<PUT_,zVAR_HYPERDATA_>` operations of the `CDFlib` function (Internal Interface). In a Fortran application hyper reads and writes for rVariables are performed using either the `CDF_var_hyper_get` and `CDF_var_hyper_put` subroutines (Standard Interface) or the `<GET_,rVAR_HYPERDATA_>` and `<PUT_,rVAR_HYPERDATA_>` operations of the `CDF_lib` subroutine (Internal Interface). Hyper reads/writes for zVariables are performed using the `<GET_,zVAR_HYPERDATA_>` and `<PUT_,zVAR_HYPERDATA_>` operations of the `CDF_lib` subroutine (Internal Interface).

### 2.3.15 Sequential Access

Sequential access provides a way to sequentially read/write the values physically stored for a variable. To use sequential access, a starting value must first be selected by specifying a record number and dimension indices. This selects the "current sequential value." A sequential read will return the value at the current sequential value and then automatically increment the current sequential value to the next value. Likewise, a sequential write will store a value at the current sequential value and then increment the current sequential value to the next value. Sequential reads are allowed until the end of the physical records has been reached (not the end of the virtual records [they never end]). Sequential reading will increment to the beginning of the next physical record if necessary. Sequential writing can be used to extend the physical records for a variable (as well as to overwrite existing values).

**Example (Fortran application)**

Assume a 2-dimensional array with sizes `[2,3]`, column majority, a record variance of `VARY`, dimension variances of `[VARY,VARY]`, nine (9) physical records written, and that the current sequential value has been set to record number `7` and indices `[2,2]`. Consecutive sequential reads would cause the following values to be read and returned to the application:

```
                     7(2,2) 7(1,3) 7(2,3)
8(1,1) 8(2,1) 8(1,2) 8(2,2) 8(1,3) 8(2,3)
9(1,1) 9(2,1) 9(1,2) 9(2,2) 9(1,3) 9(2,3)
END_OF_VAR
```

...where `r(i,j)` is a physically stored value with `r` being the record number, `i` the first dimension index, and `j` the second dimension index. (`r`, `i`, and `j` are physical record numbers and dimension indices.) The next sequential read after the last physical value would cause a status code indicating the end of the variable to be returned (`END_OF_VAR`).

Had the dimension variances been `[NOVARY,VARY]`, the values read would have been

```
       7(1,2) 7(1,3)
8(1,1) 8(1,2) 8(1,3)
9(1,1) 9(1,2) 9(1,3)
END_OF_VAR
```

Note that specifying the virtual value `7(2,2)` as the current sequential value caused physical value `7(1,2)` to actually be selected (because the first dimension variance is `NOVARY`).

Sequential access for rVariables is performed using the `<GET_,rVAR_SEQDATA_>` and `<PUT_,rVAR_SEQDATA_>` operations of the Internal Interface. Sequential access for zVariables is performed using the `<GET_,zVAR_SEQDATA_>` and `<PUT_,zVAR_SEQDATA_>` operations.

### 2.3.16 Multiple Variable Access

Multiple variable access allows an application to read from or write to multiple variables in a single operation. Multiple variable access works on either the rVariables or the zVariables of a CDF — not a mixture of the two. Up to all of the rVariables/zVariables may be accessed with a single call to the CDF library. For each variable specified in a multiple variable access, a full-physical record for that variable will be read/written. A full-physical record consists of all of the values exactly as they are physically stored in each variable record (the physical values). Virtual values do not apply when performing a multiple variable access (see Section 2.3.11). Three parameters are specified when performing a multiple variable read/write.

`variable count` The number of rVariables/zVariables that are being accessed.

`variable list` The rVariables/zVariables being accessed (specified by number).

`record numbers` The record numbers at which the reads/writes will take place. For rVariables the record numbers must all be the same. For zVariables the record numbers can vary (but for most applications will all be the same).

Multiple variable access is sensitive to the record variances of the variables being accessed. (Dimension variances do not apply since full-physical records are being read/written.) If a variable has a record variance of `NOVARY`, then a read/write to that variable will always occur at the first record regardless of the actual record number specified (since at most only one physical record will ever exist). If the record variance were `VARY`, the reads/writes would take place at the actual record numbers specified.

For a multiple variable write operation an application must place into a memory buffer each of the full-physical records to be written. The order of the full-physical records must correspond to the order of the list of variables specified, and the memory buffer must be contiguous — there can be no gaps between the full-physical records. This memory buffer is then passed to the CDF library which scans through the buffer writing the full-physical records to the corresponding variables.

Likewise, for a multiple variable read operation the CDF library places into a memory buffer provided by the application the full-physical records read. The order of the full-physical records will correspond to the order of the list of variables specified and the full-physical records will be contiguous. The application must then process the buffer as needed.

Care must be used when generating and processing the memory buffer containing the full-physical records. If C `struct` objects or Fortran `STRUCTURE` variables are being used, it may be necessary to order the variables being read/written such that there are no gaps between elements of the structures (assuming you are defining structures containing one element per full-physical record where an element is a scalar variable or an array depending on the corresponding variable definition). On some computers the C and Fortran compilers will place gaps between the elements of these structures so that memory alignment errors are not generated when the elements are accessed. In general, defining the structures so that "larger" data types are before "smaller" data types should result in no gaps (e.g., the Fortran `REAL*8` data type is "larger" than a `INTEGER*2`, which is "larger" than a `BYTE`). The list of variables would be adjusted accordingly.

The variable majority must also be considered when performing a multiple variable read/write since full-physical records are being accessed. The majority of the values in the full-physical records retrieved from/placed into the memory buffer must be the same as the variable majority of the CDF.

For example, consider a column-major CDF containing the following three zVariables (as well as others):

| zVariable Name | Data Specification | Dimensionality | Variances |
|---|---|---|---|
| `zVar1` | `CDF_INT2/1`[18] | `0:[]` | `T/` |
| `zVar2` | `CDF_CHAR/7` | `1:[5]` | `T/T` |
| `zVar3` | `CDF_REAL8/1` | `2:[2,4]` | `T/TT` |

[18] This notation is used throughout this document. The data type is before the slash and the number of elements is after the slash. In this case the data type is (`CDF_INT2`) and the number of elements is one (1).

If a Fortran application were to perform a multiple variable read on these three zVariables, it could define a `STRUCTURE` to receive the physical records as follows:

```
STRUCTURE /inputStruct/
  REAL*8      zVar3values(2,4)
  INTEGER*2   zVar1value
  CHARACTER*7 zVar2values(5)
END STRUCTURE
```

Note that because a full-physical record for the zVariable `zVar2` is an odd number of bytes it would most likely cause a gap in the `STRUCTURE` if not placed at the end (on some computers). An approach that would work on all computers would be to use `EQUIVALENCE` statements as follows:

```
INTEGER*2   zVar1value
CHARACTER*7 zVar2values(5)
REAL*8      zVar3values(2,4)
BYTE        buffer(101)
EQUIVALENCE (zVar3values,buffer(1))
EQUIVALENCE (zVar1value,buffer(65))
EQUIVALENCE (zVar2values,buffer(67))
```

The `EQUIVALENCE` statements ensure that the full-physical records will be contiguous. In each of the above examples, the order of the zVariables would be `zVar3`, `zVar1`, `zVar2`.

C applications must also be concerned with the ordering of full-physical records in the memory buffer. Even if a `void` memory buffer is used with type casting to access individual values, the alignment of the values in the memory buffer is important (on some computers).

### 2.3.17 Variable Pad Values

Variable pad[19] values are used in several situations.

1. When the first value is written to a new record, the other values in that record will contain the pad value. This also applies to hyper writes if less than the entire record is written. (The unwritten values will contain the pad value.)
2. When a new record is written that is more than one record beyond the last record already written, the intervening records will contain pad values. This will only happen if the record variance is `VARY`. (The variable values vary from record to record.)
3. When reading a record beyond the last record for a variable, pad values will be returned.
4. Allocated records for a variable which have not yet been written when a CDF is closed will be written using the variable's pad value.

[19] These were previously known as fill values but were renamed to avoid confusion with the `FILLVAL` attribute (an NSSDC standard attribute).

The pad value for a variable may be specified with the Internal Interface. It should be specified before any values are read from or written to the variable; otherwise the default pad value will be used. The pad value may be changed at any time (and any number of times) and will be in effect for all subsequent operations.

The default pad value for each data type is as follows:[20]

```
CDF_BYTE          0
CDF_INT1          0
CDF_UINT1         0
CDF_INT2          0
CDF_UINT2         0
CDF_INT4          0
CDF_UINT4         0
CDF_REAL4         0.0
CDF_FLOAT         0.0
CDF_REAL8         0.0
CDF_DOUBLE        0.0
CDF_EPOCH         01-Jan-0000 00:00:00.000 (0.0)
CDF_CHAR          " " (space character, 40 octal)
CDF_UCHAR         " " (space character, 40 octal)
```

## 2.4 Attributes

CDF attributes are the mechanism for storing metadata. (Variables are used to store data.) A new attribute may be created in a CDF at any time.

### 2.4.1 Naming

Each attribute in a CDF has a unique name. Attribute names are case sensitive regardless of the operating system being used and may consist of up to `CDF_ATTR_NAME_LEN` printable characters (including blanks). Trailing blanks, however, are ignored when the CDF library compares attribute names. `"UNITS"` and `"UNITS   "` are considered to be the same name, so they cannot both exist in the same CDF. This was done because Version 1 of CDF padded attribute names on the right with blanks out to eight characters. When a Version 1 CDF was converted to a Version 2 CDF these trailing blanks remained in the attributes names. To allow CDF Version 2 applications to read such a CDF without having to be concerned with the trailing blanks, the trailing blanks are ignored by the CDF when comparing attributes names. The trailing blanks are returned as part of the name, however, when an attribute is inquired by an application program.

### 2.4.2 Numbering

The attributes in a CDF are numbered consecutively starting at one (1) for Fortran applications and starting at zero (0) for C applications. The CDF library assigns attribute numbers as the attributes are created. Note that there are not separate lists of global and variable scoped attributes. Only one list of attributes exists in a CDF (containing both global and variable scoped attributes).

[20] These default pad values can be changed by your system manager when the CDF distribution is built.

### 2.4.3 Attribute Scopes

Attribute scopes declare the intended purpose of an attribute. Global scope attributes (gAttributes) describe some aspect of the entire CDF. Variable scope attributes (vAttributes) describe some property of each variable.

An attribute's scope exists to assist in the interpretation of its entries by CDF toolkit programs and user applications (e.g., entries of a vAttribute should correspond to variables). The CDF library also places some restrictions on the operations that may be performed on an attribute of a particular scope.[21] These restrictions consist of the following:

1. A gEntry operation may not be performed on a vAttribute.
2. A zEntry or rEntry operation may not be performed on a gAttribute.
3. While in zMode, only zEntry operations may be performed on vAttributes (see Section 2.1.2).

All other operations involving attributes and their entries remain available.

#### Assumed Scopes

CDF Version 1 did not allow the scope of an attribute to be explicitly declared. This led to ambiguities in the interpretation of attribute entries in the toolkit programs and user applications. CDF Version 2 does allow the scope of an attribute to be declared when the attribute is created. To ease the transition from Version 1 to Version 2, CDF distributions prior to CDF V2.5 contained the notion of assumed attribute scopes. Assumed attribute scopes arose when the CDF library had to guess the scope of an attribute in a Version 1 CDF (e.g., when the `CDFconvert` program converted a Version 1 CDF to a Version 2 CDF). Beginning with CDF V2.5, all assumed attribute scopes are converted to the corresponding definite scope. When a CDF is read this conversion occurs only in the CDF library — the CDF is not physically altered. When an existing CDF is written to, each assumed attribute scope detected will be physically converted to the corresponding definite scope. Note that if this automatic conversion is incorrect, the scope of an attribute can be corrected using the Internal Interface in an application program or by editing the CDF with the `CDFedit` program.

### 2.4.4 Deleting

An attribute may be deleted from a CDF. Deleting an attribute also deletes the corresponding entries. The disk space used by the attribute definition and the corresponding entries becomes available for use as needed by the CDF library. Also, the attributes which numerically follow the attribute being deleted are renumbered immediately. (Each is decremented by one.) Attributes are deleted using the `<DELETE_,ATTR_>` operation of the Internal Interface.

### 2.4.5 Attribute Entries

Attribute entries are used to actually store metadata. Each attribute in a CDF may have zero or more associated entries. For vAttributes two types of entries are supported: rEntries and zEntries. rEntries describe some property of the corresponding rVariable, and zEntries describe some property of the corresponding

[21] This was not necessarily the case in previous releases of CDF. These new restrictions should not, however, cause any conflicts with existing applications.

zVariable. Note that an entry does not have to exist for each variable in the CDF. For gAttributes only one type of entry is supported and is referred to as a gEntry. The gEntries are independent of anything else in the CDF and have meaning only to the application. Note that gEntries are sometimes referred to simply as "entries."

#### Accessing

The Standard Interface deals exclusively with rEntries (for vAttributes) and gEntries (for gAttributes). No access to zEntries is provided. The Internal Interface may be used to access any type of attribute entry.

#### Numbering

The rEntries and zEntries for a vAttribute and the gEntries for a gAttribute are numbered starting at one (1) for Fortran applications and starting at zero (0) for C applications. For vAttributes the entry numbers are in fact the variable numbers of the variables being described. rEntries correspond to rVariables and zEntries correspond to zVariables. For gAttributes the gEntry numbers have meaning only to the application.

The entry numbers used need not be contiguous (as are variable and attribute numbers). An application may choose to write any combination of entries for a particular attribute (keeping in mind that the entry numbers used for a vAttribute correspond to the existing variables).

#### Data Specification

Each entry for an attribute has a data specification and an associated value. A data specification consists of a data type and a number of elements of that data type. The supported data types are described in Section 2.2.10. The entries for an attribute may have any combination of data specifications.

For character string data types the number of elements is the number of characters in the string. For example, if a gEntry value for a gAttribute named `TITLE` were `"Example CDF Title."` (not including the double quotes), the data type would be `CDF_CHAR`, and the number of elements would be 18 (a character string of size 18).

For non-character string data types the number of elements is the size of an array of the data type. For example, if a zEntry value of a vAttribute named `RANGE` were `[100.0,900.0]`, the data type would be `CDF_REAL4`, and the number of elements would be two (an array of two values).

#### Deleting

An entry may be deleted from an attribute. The disk space used by the entry becomes available for use as needed by the CDF library. There is no renumbering of entries (as with deleting a variable or attribute). Entries are deleted using the `<DELETE_,gENTRY_>`, `<DELETE_,rENTRY_>`, and `<DELETE_,zENTRY_>` operations of the Internal Interface.

## 2.5 NSSDC Standard

CDFs generated for use by NSSDC applications (e.g., the NSSDC Graphics System [NGS]) are expected to conform to a specified standard. These applications expect certain attributes and make use of several optional attributes and variables to provide additional features. The NSSDC standard does not deal with zVariables or zEntries (for vAttributes).

### 2.5.1 Standard Attributes

There are several attributes that are required to exist in a CDF if it is to conform to the NSSDC standard. These required attributes are shown in Table 2.6. Several optional attributes are also shown.

| | Name | Data Type | Number of Elements[a] |
|---|---|---|---|
| Required gAttributes: | | | |
| | `TITLE` | Character (CDF_CHAR) | 50 |
| Required vAttributes: | | | |
| | `FIELDNAM` | Character (CDF_CHAR) | 20 |
| | `VALIDMIN` | (same as variable) | (same as variable) |
| | `VALIDMAX` | (same as variable) | (same as variable) |
| | `UNITS` | Character (CDF_CHAR) | 20 |
| | `FORMAT` | Character (CDF_CHAR) | 8 |
| Optional gAttributes: | | | |
| | `TEXT` | Character (CDF_CHAR) | 80 |
| | `MODS` | Character (CDF_CHAR) | 80 |
| Optional vAttributes: | | | |
| | `SCALEMIN` | (same as variable) | (same as variable) |
| | `SCALEMAX` | (same as variable) | (same as variable) |
| | `MONOTON`[b] | Character (CDF_CHAR) | 8 |
| | `CATDESC` | Character (CDF_CHAR) | 40 |
| | `FILLVAL` | (same as variable) | (same as variable) |

[a] If the data type is `CDF_CHAR`, blank pad the value on the right to the required number of elements (characters).
[b] We strongly suggest using the `MONOTON` attribute.

Table 2.6: NSSDC Standard CDF Attributes

The purpose of each attribute is defined as follows...

`TITLE` A title for the CDF data set. Only one gEntry should be written (gEntry number zero [0] for C applications, gEntry number one [1] for Fortran applications).

`FIELDNAM` The field name for an rVariable. One rEntry per rVariable should be written.

`VALIDMIN` The minimum valid value for an rVariable. Values below this minimum are treated as invalid or fill data. One rEntry per rVariable should be written.

| | |
|---|---|
| `VALIDMAX` | The maximum valid value for an rVariable. Values above this maximum are treated as invalid or fill data. One rEntry per rVariable should be written. |
| `UNITS` | The units of measurement for an rVariable. One rEntry per rVariable should be written. |
| `FORMAT` | The Fortran format specification to use when displaying the values for an rVariable. One rEntry per rVariable should be written. |
| `TEXT` | Documentation text about the CDF. Any number of gEntries may be written. |
| `MODS` | A modification history for the CDF. Each time the CDF is modified, a new gEntry would be added with an explanation of the changes. |
| `SCALEMIN` | The scale minimum for an rVariable. This is used when scaling the rVariable's values for display. If omitted, the `VALIDMIN` vAttribute rEntry will be used. One rEntry per rVariable should be written. |
| `SCALEMAX` | The scale maximum for an rVariable. This is used when scaling the rVariable's values for display. If omitted, the `VALIDMAX` vAttribute rEntry will be used. One rEntry per rVariable should be written. |
| `MONOTON` | The monotonicity of an rVariable. This specifies if the values strictly increase (`"INCREASE"`), strictly decrease (`"DECREASE"`), or are not monotonic (`"FALSE    "`) along a dimension or from record to record. Specifying an rVariable's monotonicity can greatly improve the efficiency of certain NSSDC applications (namely, the NSSDC Graphics System). One rEntry per rVariable should be written. |
| `CATDESC` | A catalog description for an rVariable. This is used for cataloging purposes by some NSSDC applications. One rEntry per rVariable should be written. |
| `FILLVAL` | The fill value for an rVariable. This value should not be confused with the pad value that may be specified for an rVariable (with the Internal Interface) that the CDF library actually uses when writing records. The values should agree but the `FILLVAL` vAttribute is not used by the CDF library in any way. It is, however, used by CDF applications (including the CDF toolkit). One rEntry per rVariable should be written. |

### 2.5.2 Standard Variables

There are no required CDF rVariables. The first eight characters of rVariable names must, however, be unique for NSSDC applications. Names longer than eight characters will be truncated when displayed.

The optional rVariable `EPOCH` is used by many NSSDC applications to store date/time values. Section 2.2.10 defines an EPOCH value. The record variance should be `VARY`, each dimension variance should be `NOVARY`, the data type should be `CDF_EPOCH`, and the number of elements should be one (1). If a CDF contains an `EPOCH` variable, the following variable names should not be used: `TIME`, `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `MSEC`, `IYEAR`, `IMONTH`, `IDAY`, `IHOUR`, `IMINUTE`, `ISECOND`, and `IMSEC`.

# Chapter 3

# Toolkit Reference

## 3.1 Introduction

The CDF toolkit is a set of utility programs that allow the creation, analysis, and modification of CDFs. The following sections will describe the use of these programs.

### 3.1.1 VMS, UNIX, & MS-DOS

Each program is executed at the command line (or may be executed from within your applications using the methods provided by the operating system being used). The following rules apply to the command line syntax:

1. Parameters are required unless noted otherwise. Parameters are shown in angle brackets (`<>`'s) in the sections which describe each toolkit program.

2. Qualifiers are optional unless noted otherwise.

3. Qualifiers can be truncated as long as no ambiguities result.

4. Optional parts of a command are shown in brackets (`[]`'s) in the sections which describe each toolkit program.

5. A vertical line (`|`) is used to separate two or more options in those cases when only one of the options may be specified.

6. Wildcard characters are allowed in CDF names to allow more than one CDF to be specified (where appropriate). Wildcard characters may be used in the CDF name but not the directory path portion of a specification. The wildcard characters supported are similar to those available on the operating system being used.

   **UNIX:** If a CDF specification is to contain a wildcard character, the entire specification must be enclosed in single quote marks (e.g., `'/disk3/sst*'`).

7. On VMS/OpenVMS systems, qualifiers begin with a slash (`/`). On UNIX and MS-DOS systems, qualifiers begin with a hyphen (`-`).

   **NOTE:** You can override the default notation by specifying a slash or hyphen as the first parameter/qualifier immediately after the program name. When this is done, you may have to adjust the syntax used as follows:

   (a) When the slash notation is used on UNIX systems, character stuffing will be necessary in pathnames (e.g., specify "`//disk1//CDFs`" rather than "`/dist1/CDFs`"). Also, double quote marks are required around options enclosed in parenthesis.

   (b) When the slash notation is used on MS-DOS systems, double quote marks may be needed around entire qualifier/option combinations.

8. On MS-DOS systems the executable names may be different from the names shown in this chapter (file names are limited to 8.3 characters). Where the names differ, the actual name will be noted.

   If you add the directory containing the toolkit executables to your path, you will have to use the 8-character (or fewer) names. If you use a command aliasing program, you could specify the aliases to be the names shown in this chapter with each pointing to the corresponding executable file name.

9. On UNIX systems all parameters/qualifiers entered at the command line are case sensitive. On VMS, OpenVMS, and MS-DOS systems parameters/qualifiers are not case sensitive. Note that variable names are always case sensitive regardless of the operating system being used.

10. If an option contains blanks, it will generally be necessary to enclose the entire option in double quote marks.

11. On UNIX systems, it may be necessary to execute "`stty tab3`" before running `CDFedit`, `CDFlist`, or `CDFwalk`.

In the following sections the available qualifiers and options for each of the toolkit programs will be presented. The default settings for these qualifiers and options will not be shown since they can be configured for a particular CDF distribution. Use `CDFinquire` to determine these defaults.

On VMS/OpenVMS systems you should have executed the command procedure named `DEFINITIONS.COM` before running any of the CDF toolkit programs. This will define the necessary logical names and symbols. Your system administrator knows the location of `DEFINITIONS.COM`.

On UNIX system you should have `source`'d the script file named `definitions.<shell>` where `<shell>` is the name of the shell you are using. This will define the necessary environment variables and aliases. Your system administrator knows the location of `definitions.<shell>`.

### 3.1.2 Macintosh

Each toolkit program is started by double-clicking on the appropriate icon. A dialog box will be displayed in which the parameters and qualifiers needed to execute the program are specified. When the parameters/qualifiers have been selected, clicking on `Enter` causes the initial execution to begin.

For the programs that use a full-screen interface (e.g., `CDFedit`, `CDFlist`, and `CDFwalk`), a "pasteboard" window is opened in which the program displays menus, prompts, etc. When the "pasteboard" window is closed (by exiting the execution), the parameters/qualifiers dialog box is redisplayed. A new set of parameters/qualifiers may be selected and executed or the program may be terminated.

For the programs that simply output to the screen (e.g., `CDFstats`, `CDFcompare`, and `CDFinquire`), a "standard output" window is opened in which the output will be written. When the execution completes, the "apple" and `File` menus are available in the menu bar. Under the `File` menu the following commands are available:

| | |
|---|---|
| `Execute` | Causes the parameters/qualifiers dialog box to be redisplayed. A new set of parameters/qualifiers may be selected and executed. The output from each execution is appended to the existing output. |
| `Save` | Saves the current output to a file named `<program-name>.so` where `<program-name>` is the name of the program. |
| `Save as...` | Saves the current output to the file specified in the standard output file dialog box that will be displayed. |
| `Clear` | Clears the current output. |
| `Quit` | Terminates the program. |

The vertical scroll bar as well as the `page up` and `page down` keys may be used to scroll through the output. When a large amount of text has been written, a dialog box may be displayed indicating that an output overflow is about to occur. The output may be saved to a file before being cleared (to allow the execution to continue).

The parameters/qualifiers dialog box for each program uses the standard Macintosh controls. Edit fields are used to enter text values (e.g., the pathname of a CDF). Leaving an edit field blank is allowed in some cases (which will be noted). Check boxes are used to enable or disable a qualifier. An `X` in the check box indicates that the qualifier is enabled. Radio buttons are used in groups to allow one of several options to be chosen for a qualifier. Generally, only one of the radio buttons in a group may be selected.

Several types of files are specified to the toolkit programs. These consist of CDFs, skeleton tables, and output files. In a parameters/qualifiers dialog box edit field or a toolkit program's prompt field a file must be specified using a full or partial pathname. Full pathnames consist of a volume name (which is also the corresponding folder name), zero or more folder names, and finally the file name (with or without an extension). These are all separated by semi-colons (`:`'s). Partial pathnames do not start with a volume name and may start with or without a semi-colon. If a partial pathname starts with a semi-colon, one or more folder names will follow, each separated by a semi-colon, followed by the file name. The first folder must exist in the currently selected folder. If a partial pathname starts without a semi-colon, then only a file name should be present and the file is (or will be) located in the currently selected folder. To ease in the selection of files in parameters/qualifiers dialog boxes, the corresponding edit fields are followed by a `Select` button. When clicked on, a standard input/output file dialog will be displayed in which a file may be specified. When that has been done the pathname of the selected file will appear in the edit field.

A directory/wildcard[1] specification is allowed for some of the CDF specifications required by the toolkit programs. This allows more than one of the CDFs in a directory to be selected. If a CDF specification ends with a folder name, then all of the CDFs in that folder will have been specified. A trailing semi-colon is not required (but may be present). The supported wildcard characters are the asterick (`*`) which matches zero or more characters and the question mark (`?`) which matches exactly one character.

In the following sections the available qualifiers and options for each of the toolkit programs will be presented. The default settings for these qualifiers and options will not be shown since they can be configured for a par-

[1] Macintosh folders are equivalent to the directories discussed here.

ticular CDF distribution. When a program is started, the settings shown in the initial parameters/qualifiers dialog box are the default qualifiers for your CDF distribution.

**NOTE:** You may find it necessary to increase the partition size available to a toolkit program when dealing with very large CDFs. You can do this by editing the "current size" field of the window opened when using the `Get Info` item of the `File` menu (from the Desktop menu bar) on the toolkit executable.

## 3.2 CDFdir

### 3.2.1 Introduction

The `CDFdir` utility is used to display a directory listing of a CDF's files.[2] The `.cdf` file is displayed first followed by the rVariable files and then the zVariable files (if either exist in a multi-file CDF) in numerical order.

### 3.2.2 Executing the CDFdir Program

The command line syntax for `CDFdir` is as follows:

**Usage:**

VMS:

```
$ CDFDIR <cdf-path>
```

UNIX:

```
% cdfdir <cdf-path>
```

MS-DOS:

```
> cdfdir <cdf-path>
```

**Parameter(s):**

`<cdf-path>`

The pathname of the CDF for which to display a directory listing (do not specify an extension).

[2] `CDFdir` is not available on a Macintosh.

**Example(s):**

VMS:

```
$ CDFDIR NCDS$DATA:GISS_WETL_CLIMATOLOGY
$ CDFDIR [-.TEMP]FGGE3B
```

UNIX:

```
% cdfdir ../cac_sst_blended
% cdfdir ~/CDFs/giss_wetl_climatology
```

MS-DOS:

```
> cdfdir ..\cac_sst
> cdfdir c:\cdfs\gisswetl
```

Help is displayed when `CDFdir` is executed without any arguments.

### 3.2.3 Output from the CDFdir Program

The format of the output from `CDFdir` is that of a directory listing on the operating system being used.

## 3.3 CDFcompare

### 3.3.1 Introduction

The CDFcompare program displays the differences between two CDFs. More than one pair of CDFs can be compared. This program would be used to verify changes made to a CDF (comparing it with the saved original) or to verify the conversions performed by `CDFconvert` (see Section 3.4).

### 3.3.2 Executing the CDFcompare Program

**Usage:**

VMS:

```
$ CDFCOMPARE [/[NO]LOG] [/[NO]ATTR] [/[NO]VAR] [/[NO]NUMBER]
             [/[NO]ETC] [/[NO]NEG2POSFP0] [/ZMODES=(<mode1>,<mode2>)]
```

```
               [/[NO]LOCATION] [/REPORT=(<types>)] [/CACHE=<size>]
               [/[NO]PAGE] [/[NO]STATISTICS] [/[NO]PERCENT]
               <cdf-spec-1> <cdf-spec-2>
```

UNIX:

```
% cdfcompare [-[no]log] [-[no]attr] [-[no]var] [-[no]number]
             [-[no]etc] [-[no]neg2posfp0] [-zmodes "<mode1>,<mode2>"]
             [-[no]location] [-report "<types>"] [-cache <size>]
             [-[no]page] [-[no]statistics] [-[no]percent]
             <cdf-spec-1> <cdf-spec-2>
```

MS-DOS:[3]

```
> cdfcompare [-[no]log] [-[no]attr] [-[no]var] [-[no]number]
             [-[no]etc] [-[no]neg2posfp0] [-zmodes "<mode1>,<mode2>"]
             [-[no]location] [-report "<types>"] [-cache <size>]
             [-[no]page] [-[no]statistics] [-[no]percent]
             <cdf-spec-1> <cdf-spec-2>
```

Macintosh:

Double-click on the CDFcompare icon. When the desired parameters/qualifiers have been selected in the dialog box, click on the Enter button to compare the CDFs. Clicking on the Help button will display online help. Clicking on the Quit button terminates CDFcompare.

**Parameter(s):**

<cdf-spec-1> <cdf-spec-2> (VMS, UNIX, & MS-DOS)
CDF1 and CDF2 edit fields (Macintosh)

The specifications of the CDFs to be compared. (Do not enter extensions.) These can be either a pathname specifying a single CDF or a directory/wildcard path specifying more than one CDF. Wildcards are allowed in the CDF name but not in the directory path.

If two directory/wildcard paths are specified, all of the CDFs with matching names will be compared. If a CDF pathname and a directory/wildcard path are specified, the CDF specified will be compared with the CDF in the directory/wildcard path having the same name. If two CDF pathnames are specified, the CDFs are compared. (This is the only way to compare two CDFs having different names.)

**Macintosh:** At the end of each field a button labeled Select is present. When selected, a standard input file dialog is displayed from which a single CDF may be selected.

[3] On MS-DOS systems the executable is named CDFCMP.EXE.

**Qualifier(s):**

`/[NO]LOG` (VMS)
`-[no]log` (UNIX & MS-DOS)
`Log progress` check box (Macintosh)

Specifies whether or not messages about the progress of each comparison are displayed.

`/[NO]PERCENT` (VMS)
`-[no]percent` (UNIX & MS-DOS)
`Display percentages` check box (Macintosh)

Specifies whether or not the percentage of a variable's values compared is displayed during the comparison of that variable. Message logging must also be enabled.

`/[NO]ATTR` (VMS)
`-[no]attr` (UNIX & MS-DOS)
`Compare attributes` check box (Macintosh)

Specifies whether or not attributes (and their entries) are to be compared.

`/[NO]VAR` (VMS)
`-[no]var` (UNIX & MS-DOS)
`Compare variables` check box (Macintosh)

Specifies whether or not variables are to be compared. Note that an rVariable will never be compared with a zVariable.

`/[NO]NUMBER` (VMS)
`-[no]number` (UNIX & MS-DOS)
`Compare numbers` check box (Macintosh)

Specifies whether or not numbering differences between attributes with the same names and between variables with the same names are to be displayed.

`/[NO]ETC` (VMS)
`-[no]etc` (UNIX & MS-DOS)
`Compare etc.` check box (Macintosh)

Specifies whether or not differences transparent to an application will be displayed. These would consist of the version/release/increment of the creating CDF library, format, encoding, etc.

`/ZMODES=(<mode1>,<mode2>)` (VMS)
`-zmodes "<mode1>,<mode2>"` (UNIX & MS-DOS)
`zMode1` and `zMode2` radio buttons (Macintosh)

Specifies the zModes that should be used with the CDF(s) being compared. Note that different zModes may be used for the two CDF(s) specifications. The zModes may be one of the following:

`0` Indicates that zMode should be disabled.

`1` Indicates that zMode/1 should be used. (The dimension variances of rVariables will be preserved.)

`2` Indicates that zMode/2 should be used. (The dimensions of rVariables having a variance of `NOVARY` [false] are removed.)

/[NO]NEG2POSFP0 (VMS)
-[no]neg2posfp0 (UNIX & MS-DOS)
-0.0 to 0.0 check box (Macintosh)

Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

/[NO]PAGE (VMS)
-[no]page (UNIX & MS-DOS)
Page output check box (Macintosh)

Specifies whether or not the output is displayed a page at a time.

/[NO]LOCATION (VMS)
-[no]location (UNIX & MS-DOS)
Display locations check box (Macintosh)

Specifies whether or not the locations of variable value differences are displayed. The locations are displayed in the form
<record-number>:[<index1>,<index2>,...,<indexN>]

/REPORT=(<types>) (VMS)
-report "<types>" (UNIX & MS-DOS)
Report info's/warnings/errors check boxes (Macintosh)

Specifies the types of return status codes from the CDF library that should be reported/displayed.

**VMS, UNIX, & MS-DOS:** <types> is a comma-separated list of zero or more of the following symbols: errors, warnings, or informationals. Note that these symbols can be truncated (e.g., e, w, and i).

/CACHE=<size> (VMS)
-cache <size> (UNIX & MS-DOS)
Cache size edit field (Macintosh)

Specifies the cache size to be used for the .cdf file of both CDFs in a comparison (i.e., the only files if they are single-file CDFs). If omitted, default cache sizes as chosen by the CDF library are used. The cache size is the number of 512-byte buffers to be used.

/[NO]STATISTICS (VMS)
-[no]statistics (UNIX & MS-DOS)
Display statistics check box (Macintosh)

Specifies whether or not caching statistics are displayed at the end of each comparison.

**Example(s):**

VMS:

```
$ CDFCOMPARE GISS_WETL GISS_WETL1
$ CDFCOMPARE/LOG/NOATTR/NUMBER/REPORT=(ERRORS) GISS_WETL CDF$SMPL:GISS_WETL
$ CDFCOMPARE/NOVAR/NOETC/ZMODES=(1,2) NCDS$SMPL: NCDS$DATA:
```

UNIX:

```
% cdfcompare giss_wetl giss_wetl1
% cdfcompare -log -noattr -number -report "errors" giss_wetl ../giss_wetlx
% cdfcompare -novar -noetc -zmodes "1,2" /user5/CDFs /user6/CDFs
```

MS-DOS:

```
> cdfcompare gisswetl c:\gisswetl
> cdfcompare -log -noattr -number -report "errors" gisswetl ..\..\giswetlx
> cdfcompare -novar -noetc -zmodes "1,2" a:\cdfs c:\cdf\cdfs
```

**VMS, UNIX, & MS-DOS:** Command line help is displayed when `CDFcompare` is executed without any arguments.

### 3.3.3 Output from the CDFcompare Program

The output from `CDFcompare` consists of messages indicating the differences found. If message logging is enabled, the progress of each comparison is also displayed.

## 3.4 CDFconvert

### 3.4.1 Introduction

The `CDFconvert` program is used to convert various properties of a CDF. In all cases new CDFs are created. (Existing CDFs are not modified.) Any combination of the following properties may be changed when converting a CDF.

1. A CDF created with an earlier release of the CDF library (e.g., CDF V2.4) may be converted to the current library release.
2. The format of the CDF may be changed (see Section 2.2.7).
3. The data encoding of the CDF may be changed (see Section 2.2.8).
4. The variable majority of the CDF may be changed (see Section 2.3.12).

### 3.4.2 Executing the CDFconvert Program

**Usage:**

VMS:

```
$ CDFCONVERT [/SKELETON=<skt-cdf-path>] [/[NO]LOG]
             [/[NO]PERCENT] [/REPORT=(<types>)]
             [/CACHE=<size>] [/[NO]PAGE] [/[NO]STATISTICS]
             <src-cdf-spec>
                [/ZMODE=<mode>] [/[NO]NEG2POSFP0]
             <dst-cdf-spec>
                [/SINGLE | /MULTI] [/ROW | /COLUMN] [/[NO]DELETE]
                [/ENCODING=<encoding> | /HOST | /NETWORK]
```

UNIX:

```
% cdfconvert [-skeleton <skt-cdf-path>] [-[no]log]
             [-[no]percent] [-report "<types>"]
             [-cache <size>] [-[no]page] [-[no]statistics]
             <src-cdf-spec>
                [-zmode <mode>] [-[no]neg2posfp0]
             <dst-cdf-spec>
                [-single | -multi] [-row | -column] [-[no]delete]
                [-encoding <encoding> | -host | -network]
```

MS-DOS:[4]

```
> cdfconvert [-skeleton <skt-cdf-path>] [-[no]log]
             [-[no]percent] [-report "<types>"]
             [-cache <size>] [-[no]page] [-[no]statistics]
             <src-cdf-spec>
                [-zmode <mode>] [-[no]neg2posfp0]
             <dst-cdf-spec>
                [-single | -multi] [-row | -column] [-[no]delete]
                [-encoding <encoding> | -host | -network]
```

Macintosh:

Double-click on the `CDFconvert` icon. When the desired parameters/qualifiers have been selected in the dialog box, click on the `Enter` button to convert the CDFs. Clicking on the `Help` button will display online help. Clicking on the `Quit` button terminates `CDFconvert`.

**Parameter(s):**

`<src-cdf-spec>` (VMS, UNIX, & MS-DOS)
`Source` edit field (Macintosh)

The source CDF(s). This can be either a single CDF pathname or a directory/wildcard path in which case all CDFs that match the specification will be converted. Wildcards are allowed in the CDF name but not in the directory path. In either case do not specify an extension.

[4] On MS-DOS systems the executable is named `CDFCVT.EXE`.

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard input file dialog is displayed from which a single CDF may be selected.

`<dst-cdf-spec>` (VMS, UNIX, & MS-DOS)
`Destination` edit field (Macintosh)

The destination of the converted CDF(s). This may be a single CDF pathname only if a single source CDF was specified. If the directory paths are the same, then a different CDF name must be specified. If the source CDF specification is a directory/wildcard path, then this must be a directory path (other than the source directory path). This may also be a directory path if only a single CDF is being converted. In any case do not specify an extension.

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard output file dialog is displayed in which a single CDF may be specified.

**Qualifier(s):**

`/SKELETON=<skt-cdf-path>` (VMS)
`-skeleton <skt-cdf-path>` (UNIX & MS-DOS)
`Skeleton` edit field (Macintosh)

The pathname of a skeleton CDF to be used during the conversions. (Do not enter an extension.) The skeleton CDF is used in the following cases:

1. If a format for the destination CDF was not specified, then the format of the skeleton CDF will be used.
2. If a variable majority for the destination CDF was not specified, then the variable majority of the skeleton CDF will be used.
3. If a data encoding for the destination CDF was not specified, then the data encoding of the skeleton CDF will be used.

Specifying a skeleton CDF is optional.

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard input file dialog is displayed from which the skeleton CDF may be selected.

`/[NO]LOG` (VMS)
`-[no]log` (UNIX & MS-DOS)
`Log progress` check box (Macintosh)

Specifies whether or not messages about the progress of each conversion are displayed.

`/[NO]PAGE` (VMS)
`-[no]page` (UNIX & MS-DOS)
`Page output` check box (Macintosh)

Specifies whether or not the output is displayed a page at a time.

`/[NO]PERCENT` (VMS)
`-[no]percent` (UNIX & MS-DOS)
`Display percentages` check box (Macintosh)

Specifies whether or not the percentage of a variable's values converted is displayed during the conversion of that variable. Message logging must also be enabled.

`/[NO]DELETE` (VMS)
`-[no]delete` (UNIX & MS-DOS)
`Delete existing` check box (Macintosh)

Specifies whether or not a destination CDF is deleted if it already exists.

`/SINGLE | /MULTI` (VMS)
`-single | -multi` (UNIX & MS-DOS)
`Source/Single/Multi` radio buttons (Macintosh)

The format of the destination CDF(s).

**VMS, UNIX, & MS-DOS:** This overrides the format of the skeleton CDF (if one was specified). If neither this qualifier nor a skeleton CDF is specified, then the format of a destination CDF will be the same as that of the source CDF.

**Macintosh:** Selecting `Single` or `Multi` will override the format of the skeleton CDF (if one was specified). If `Source` is selected (and a skeleton CDF was not specified), then the format of a destination CDF will be the same as that of the source CDF. (Selecting `Source` will not override the format of a specified skeleton CDF.)

`/ROW | /COLUMN` (VMS)
`-row | -column` (UNIX & MS-DOS)
`Source/Row/Column` radio buttons (Macintosh)

The variable majority of the destination CDF(s).

**VMS, UNIX, & MS-DOS:** This overrides the variable majority of the skeleton CDF (if one was specified). If neither this qualifier nor a skeleton CDF is specified, then the variable majority of a destination CDF will be the same as that of the source CDF.

**Macintosh:** Selecting `Row` or `Column` will override the variable majority of the skeleton CDF (if one was specified). If `Source` is selected (and a skeleton CDF was not specified), then the variable majority of a destination CDF will be the same as that of the source CDF. (Selecting `Source` will not override the variable majority of a specified skeleton CDF.)

`/ENCODING=<encoding> | /HOST | /NETWORK` (VMS)
`-encoding <encoding> | -host | -network` (UNIX & MS-DOS)
`Source/Host/Network/Sun...Vax` radio buttons (Macintosh)

The data encoding of the destination CDF(s).

**VMS, UNIX, & MS-DOS:** This overrides the data encoding of the skeleton CDF (if one was specified). If neither this qualifier nor a skeleton CDF is specified, then the data encoding of a destination CDF will be the same as that of the source CDF. The possible values of `<encoding>` are `host`, `network`, `sun`, `vax`, `decstation`, `sgi`, `ibmpc`, `ibmrs`, `mac`, `hp`, `next`, `alphaosf1`, `alphavmsd`, and `alphavmsg` (and their uppercase equivalents). Note that the host and network qualifiers are no longer necessary (but are supported for compatibility with previous CDF distributions).

**Macintosh:** Selecting `Host`, `Network`, or a specific machine will override the data encoding of the skeleton CDF (if one was specified). If `Source` is selected (and a skeleton CDF was not specified), then the data encoding of a destination CDF will be the same as that of the source CDF. (Selecting `Source` will not override the data encoding of a specified skeleton CDF.)

`/ZMODE=<mode>` (VMS)
`-zmode <mode>` (UNIX & MS-DOS)

`zMode` radio buttons (Macintosh)

Specifies the zMode that should be used with the source CDF(s). The zMode may be one of the following:

- `0` Indicates that zMode should be disabled.
- `1` Indicates that zMode/1 should be used. (The dimension variances of rVariables will be preserved.)
- `2` Indicates that zMode/2 should be used. (The dimensions of rVariables having a variance of `NOVARY` [false] are removed.)

Note that using zMode/1 or zMode/2 on a source CDF that contains rVariables will produce a destination CDF containing only zVariables. The zMode "view" provided for the source CDF is written to the destination CDF during the conversion.

`/[NO]NEG2POSFP0` (VMS)
`-[no]neg2posfp0` (UNIX & MS-DOS)
`-0.0 to 0.0` check box (Macintosh)

Specifies whether or not `-0.0` is converted to `0.0` by the CDF library when read from or written to a CDF. `-0.0` is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`/REPORT=(<types>)` (VMS)
`-report "<types>"` (UNIX & MS-DOS)
`Report info's/warnings/errors` check boxes (Macintosh)

Specifies the types of return status codes from the CDF library that should be reported/displayed.

**VMS, UNIX, & MS-DOS:** `<types>` is a comma-separated list of zero or more of the following symbols: `errors`, `warnings`, or `informationals`. Note that these symbols can be truncated (e.g., `e`, `w`, and `i`).

`/CACHE=<size>` (VMS)
`-cache <size>` (UNIX & MS-DOS)
`Cache size` edit field (Macintosh)

Specifies the cache size to be used for the `.cdf` file of both the source and destination CDF (i.e., the only files if they are both single-file CDFs). If omitted, default cache sizes as chosen by the CDF library are used. The cache size is the number of 512-byte buffers to be used.

`/[NO]STATISTICS` (VMS)
`-[no]statistics` (UNIX & MS-DOS)
`Display statistics` check box (Macintosh)

Specifies whether or not caching statistics are displayed at the end of each conversion.

**Example(s):**

VMS:

```
$ CDFCONVERT CDF$SMPL:TEMPLATE0 TEMPLATE0X
$ CDFCONVERT/LOG/REPORT=(ERRORS) CDF$SMPL: USER_DISK:[USER.CDF]
$ CDFCONVERT CAC_SST_BLENDED CAC_SST_BLENDEDX/SINGLE/NETWORK
```

```
$ CDFCONVERT/SKELETON=CDF$SMPL:TEMPLATE3 CAC_SST_BLENDED* [USER.CDF]
```

UNIX:

```
% cdfconvert ../samples/template0 template0x
% cdfconvert -log -report "errors" ../samples /disk4/user/cdf
% cdfconvert cac_sst_blended cac_sst_1 -single -network
% cdfconvert -skeleton template3 '../cdf/cac_sst*' ~user/cdf
```

MS-DOS:

```
> cdfconvert ..\samples\tplate0 tplate0x
> cdfconvert -log -report "errors" ..\samples c:\dir4\user\cdf
> cdfconvert cac_sst cac_sst1 -single -network
> cdfconvert -skeleton tplate3 a:\cdf\cac_sst dir5\cdf
```

**VMS, UNIX, & MS-DOS:** Command line help is displayed when `CDFconvert` is executed without any arguments.

### 3.4.3 Output from the CDFconvert Program

As `CDFconvert` executes, the name of each CDF being converted is displayed. If message logging is enabled, the progress of each conversion is also displayed.

## 3.5 CDFstats

### 3.5.1 Introduction

The `CDFstats` program produces a statistical report on a CDF's variable data. Both rVariables and zVariables are analyzed. For each variable it determines the actual minimum and maximum values (in all of the variable records), the minimum and maximum values within the valid range (if the `VALIDMIN` and `VALIDMAX` vAttributes and corresponding entries are present in the CDF), and the monotonicity. An option exists to allow fill values (specified by the `FILLVAL` vAttribute) to be ignored when collecting statistics.

Monotonicity refers to whether or not variable data values increase or decrease from record to record or along a dimension. This property is checked only if the variable varies along just one "dimension" (considering records to be another "dimension"). For example, consider a CDF with the 2-dimensional rVariables shown in Table 3.1.

The `EPOCH`, `LATITUDE`, and `LONGITUDE` rVariables would be checked for monotonicity but the `ELEVATION` and `TEMPERATURE` rVariables would not be checked.

An additional option exists that allows the `VALIDMIN`, `VALIDMAX`, `SCALEMIN`, and `SCALEMAX` attribute entries to be updated based on the actual minimum and maximum values found for a variable. The `MONOTON` attribute entries may also be updated based on the monotonicity found for a variable.

| rVariable | Record Variance | Dimension Variances |
|---|---|---|
| EPOCH | VARY | NOVARY,NOVARY |
| LATITUDE | NOVARY | VARY,NOVARY |
| LONGITUDE | NOVARY | NOVARY,VARY |
| ELEVATION | NOVARY | VARY,VARY |
| TEMPERATURE | VARY | VARY,VARY |

Table 3.1: Example rVariables, CDFstats Monotonicity Checking

### 3.5.2 Executing the CDFstats Program

**Usage:**

VMS:

```
$ CDFSTATS [/[NO]RANGE] [/[NO]FILL] [/OUTPUT=<file-path>] [/[NO]FORMAT]
           [/[NO]PAGE] [/[NO]UPDATE_VALIDS] [/[NO]UPDATE_SCALES]
           [/[NO]UPDATE_MONOTONIC] [/ZMODE=<mode>] [/[NO]NEG2POSFP0]
           [/REPORT=(<types>)] [/CACHE=<size>] <cdf-path>
```

UNIX:

```
% cdfstats [-[no]range] [-[no]fill] [-output <file-name>] [-[no]format]
           [-[no]page] [-[no]update_valids] [-[no]update_scales]
           [-[no]update_monotonic] [-zmode <mode>] [-[no]neg2posfp0]
           [-report "<types>"] [-cache <size>] <cdf-path>
```

MS-DOS:

```
> cdfstats [-[no]range] [-[no]fill] [-output <file-name>] [-[no]format]
           [-[no]page] [-[no]update_valids] [-[no]update_scales]
           [-[no]update_monotonic] [-zmode <mode>] [-[no]neg2posfp0]
           [-report "<types>"] [-cache <size>] <cdf-path>
```

Macintosh:

Double-click on the CDFstats icon. When the desired parameters/qualifiers have been selected in the dialog box, click on the Enter button to analyze the CDF. Clicking on the Help button will display online help. Clicking on the Quit button terminates CDFstats.

**Parameter(s):**

<cdf-path> (VMS, UNIX, & MS-DOS)
CDF edit field (Macintosh)

The pathname of the CDF to analyze. (Do not specify an extension.)

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard input file dialog is displayed from which the CDF may be selected.

**Qualifier(s):**

`/[NO]RANGE` (VMS)
`-[no]range` (UNIX & MS-DOS)
`Range checking` check box (Macintosh)

Specifies whether or not range checking will be performed. To perform range checking, the CDF must contain `VALIDMIN` and `VALIDMAX` attributes. A variable must also have an entry for each of these attributes in order for range checking to be performed on that variable.

`/[NO]FILL` (VMS)
`-[no]fill` (UNIX & MS-DOS)
`Use FILLVAL` check box (Macintosh)

Specifies whether or not fill values are ignored when collecting statistics (using the `FILLVAL` attribute if it exists).

`/OUTPUT=<file-path>` (VMS)
`-output <file-path>` (UNIX & MS-DOS)
`Output` edit field (Macintosh)

If this qualifier is specified, the statistical output is written to the named file. If the named file does not have an extension, `.sts` (UNIX & Macintosh) or `.STS` (VMS & MS-DOS) is appended automatically. If this qualifier is not specified, the output is displayed on the screen.

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard output file dialog is displayed in which the output file may be specified.

`/[NO]FORMAT` (VMS)
`-[no]format` (UNIX & MS-DOS)
`Use FORMAT` check box (Macintosh)

Specifies whether or not the `FORMAT` attribute is used when displaying variable values (if the `FORMAT` attribute exists and an entry exists for the variable).

`/[NO]PAGE` (VMS)
`-[no]page` (UNIX & MS-DOS)
`Page output` check box (Macintosh)

Specifies whether or not the output is displayed a page (screen) at a time. If the output is directed to a file, paging is not allowed.

`/[NO]UPDATE_VALIDS` (VMS)
`-[no]update_valids` (UNIX & MS-DOS)
`Update VALIDMIN/MAX` check boxes (Macintosh)

Specifies whether or not the `VALIDMIN` and `VALIDMAX` attribute entry values are updated for each variable based on the actual minimum and maximum values found (with fill values being ignored if requested). If the `VALIDMIN` and `VALIDMAX` attributes do not exist, they are created.

`/[NO]UPDATE_SCALES` (VMS)

-[no]update_scales (UNIX & MS-DOS)
Update SCALEMIN/MAX check boxes (Macintosh)

Specifies whether or not the SCALEMIN and SCALEMAX attribute entry values are updated for each variable based on the actual minimum and maximum values found (with fill values being ignored if requested). If the SCALEMIN and SCALEMAX attributes do not exist, they are created.

/[NO]UPDATE_MONOTONIC (VMS)
-[no]update_monotonic (UNIX & MS-DOS)
Update MONOTON check box (Macintosh)

Specifies whether or not the MONOTONIC attribute entry values are updated for each variable based on the monotonicity found (with fill values being ignored if requested). If the MONOTONIC attribute does not exist, it is created.

/ZMODE=<mode> (VMS)
-zmode <mode> (UNIX & MS-DOS)
zMode radio buttons (Macintosh)

Specifies the zMode that should be used with the CDF. The zMode may be one of the following:

0 Indicates that zMode should be disabled.

1 Indicates that zMode/1 should be used. (The dimension variances of rVariables will be preserved.)

2 Indicates that zMode/2 should be used. (The dimensions of rVariables having a variance of NOVARY [false] are removed.)

/[NO]NEG2POSFP0 (VMS)
-[no]neg2posfp0 (UNIX & MS-DOS)
-0.0 to 0.0 check box (Macintosh)

Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

/REPORT=(<types>) (VMS)
-report "<types>" (UNIX & MS-DOS)
Report info's/warnings/errors check boxes (Macintosh)

Specifies the types of return status codes from the CDF library that should be reported/displayed.

**VMS, UNIX, & MS-DOS:** <types> is a comma-separated list of zero or more of the following symbols: errors, warnings, or informationals. Note that these symbols can be truncated (e.g., e, w, and i).

/CACHE=<size> (VMS)
-cache <size> (UNIX & MS-DOS)
Cache size edit field (Macintosh)

Specifies the cache size to be used for the CDF's .cdf file (i.e., the only file if a single-file CDF). If omitted, a default cache size as chosen by the CDF library is used. The cache size is the number of 512-byte buffers to be used.

**Example(s):**

VMS:

```
$ CDFSTATS TEST1
$ CDFSTATS/REPORT=(ERRORS) GISS_SOIL
$ CDFSTATS/NOFILL/OUTPUT=TEMPLATE3/NORANGE CDF$SMPL:TEMPLATE3
```

UNIX:

```
% cdfstats giss_soil
% cdfstats -range -fill -report "errors" $CDF_SMPL/giss_soil
% cdfstats -norange -output template3 ../../samples/template3
```

MS-DOS:

```
> cdfstats gisssoil
> cdfstats -range -nofill -report "errors" a:\cdfs\gisssoil
> cdfstats -norange -output tplate3 ..\..\samples\tplate3
```

**VMS, UNIX, & MS-DOS:** Command line help is displayed when `CDFstats` is executed without any arguments.

### 3.5.3 Output from the CDFstats Program

The format of the output from `CDFstats` is as follows:

For each variable (rVariables and zVariables),

```
<number>.  <name>  <n-dims>:[<dim-sizes>]  <rec-vary>/<dim-varys>  (<data-type>/<n-elems>)

              min: <min-value>
     min in range: <min-value-in-range>
        valid min: <valid-min>, <low-values> low value(s)

              max: <max-value>
     max in range: <max-value-in-range>
        valid max: <valid-max>, <high-values> high value(s)

       fill value: <fill-value>, <fill-values> fill value(s)

        monotonic: <monotonicity>
```

If range checking and/or fill value filtering is disabled, the corresponding fields will not be displayed. The fields are defined as follows:

| | |
|---|---|
| <number> | The variable number. |
| <name> | The variable name. |
| <rec-vary> | The record variance of the variable — either a T or F. |
| <dim-varys> | The dimension variances of the variable — for each dimension either a T or F. This field is not present if there are zero (0) dimensions. |
| <data-type> | The data type of the variable (e.g., CDF_REAL4). |
| <n-elems> | The number of elements of the variable's data type. |
| <n-dims> | The number of dimensions of a zVariable. This field is not present for an rVariable. |
| <dim-sizes> | The dimension sizes of a zVariable. This field is not present for an rVariable or if the zVariable has zero (0) dimensions. |
| <min-value> | The minimum value found (regardless of any range checking performed). |
| <min-value-in-range> | The minimum value found within the valid range. |
| <valid-min> | The minimum valid value (VALIDMIN attribute entry value). |
| <low-values> | The number of values found that are less than the valid minimum. |
| <max-value> | The maximum value found (regardless of any range checking performed). |
| <max-value-in-range> | The maximum value found with the valid range. |
| <valid-max> | The maximum valid value (VALIDMAX attribute entry value). |
| <high-values> | The number of values found that are greater than the valid maximum. |
| <fill-value> | The fill value (FILLVAL attribute entry value). |
| <fill-values> | The number of fill values found. |
| <monotonicity> | The monotonicity of the variable. |

The <monotonicity> field may take on one of the following values.

| | |
|---|---|
| Steady (one value) | The variable has only one value in the CDF. |
| Steady (all values the same) | All values of the variable are the same. |
| Increase | Values strictly increase (with increasing record number/dimension index). |
| Decrease | Values strictly decrease (with increasing record number/dimension index). |
| noDecrease (some values the same) | Consecutive values either increase or are the same (with increasing record number/dimension index). |

| | |
|---|---|
| `noIncrease (some values the same)` | Consecutive values either decrease or are the same (with increasing record number/dimension index). |
| `False` | Consecutive values both increase and decrease. |
| `n/a` | The variable was not checked for monotonicity because it varies along more than one "dimension" (if records are considered another "dimension"). |

## 3.6 CDFinquire

### 3.6.1 Introduction

The `CDFinquire` program displays the version of the CDF distribution being used and the default toolkit qualifiers.[5]

### 3.6.2 Executing the CDFinquire Program

**Usage:**

VMS:

```
$ CDFINQUIRE [/ID] [/[NO]PAGE]
```

UNIX:

```
% cdfinquire [-id] [-[no]page]
```

MS-DOS:[6]

```
> cdfinquire [-id] [-[no]page]
```

Macintosh:

Double-click on the `CDFinquire` icon. When the desired parameters/qualifiers have been selected in the dialog box, click on the `Enter` button to inquire the CDF distribution. Clicking on the `Help` button will display online help. Clicking on the `Quit` button terminates `CDFinquire`.

[5] In CDF V2.5, the previous functionality of `CDFinquire` has been moved to the `SkeletonTable` toolkit program.
[6] On MS-DOS systems the executable is named `CDFINQ.EXE`.

**Parameter(s):**

`(None)`

**Qualifier(s):**

`/ID` (VMS)
`-id` (UNIX & MS-DOS)
`Id` check box (Macintosh)

Causes the version of your CDF distribution and the default toolkit qualifiers to be displayed. This qualifier is required.

`/[NO]PAGE` (VMS)
`-[no]page` (UNIX & MS-DOS)
`Page output` check box (Macintosh)

Specifies whether or not the output is displayed a page (screen) at a time.

**Example(s):**

VMS:

```
$ CDFINQUIRE/ID/PAGE
```

UNIX:

```
% cdfinquire -id -page
```

MS-DOS:

```
> cdfinquire -id -page
```

**VMS, UNIX, & MS-DOS:** Command line help is displayed when `CDFinquire` is executed without any arguments.

### 3.6.3 Output from the CDFinquire Program

The version of your CDF distribution is displayed first followed by the default toolkit qualifiers in the style of the operating system being used.

## 3.7 SkeletonTable

### 3.7.1 Introduction

The `SkeletonTable` program is used to create an ASCII text file called a skeleton table containing information about a given CDF. (`SkeletonTable` can also be instructed to output the skeleton table to the terminal screen.) It reads a CDF and writes into the skeleton table the following information.

1. Format (single or multi file), data encoding, variable majority.
2. Number of dimensions and dimension sizes for the rVariables.
3. gAttribute definitions (and gEntry values).
4. rVariable and zVariable definitions and vAttribute definitions (with rEntry/zEntry values).
5. Data values for all or a subset of the CDF's variables. Traditionally, only NRV variable values are written to a skeleton table. RV variable values may now also be written. NRV and RV variables are described in Section 2.3.10.

The above information is written in a format that can be "understood" by the `SkeletonCDF` program (see Section 3.8). `SkeletonCDF` reads a skeleton table and creates a new CDF (called a skeleton CDF).

### 3.7.2 Executing the SkeletonTable Program

**Usage:**

VMS:

```
$ SKELETONTABLE [/SKELETON=<skeleton-path>] [/[NO]LOG] [/ZMODE <mode>]
                [/NONRV | /NRVTABLE | /VALUES=<values>] [/[NO]SCREEN]
                [-[NO]NEG2POSFP0] [/[NO]FORMAT] [/REPORT=(<types>)]
                [/CACHE=<size>] [/[NO]PAGE] <cdf-path>
```

UNIX:

```
% skeletontable [-skeleton <skeleton-path>] [-[no]log] [-zmode <mode>]
                [-nonrv | -nrvtable | -values <values>] [-[no]screen]
                [-[no]neg2posfp0] [-[no]format] [-report "<types>"]
                [-cache <size>] [-[no]page] <cdf-path>
```

MS-DOS:[7]

[7] On MS-DOS systems the executable is named `CDF2SKT.EXE`.

```
> skeletontable [-skeleton <skeleton-path>] [-[no]log] [-zmode <mode>]
                [-nonrv | -nrvtable | -values <values>] [-[no]screen]
                [-[no]neg2posfp0] [-[no]format] [-report "<types>"]
                [-cache <size>] [-[no]page] <cdf-path>
```

Macintosh:

Double-click on the `SkeletonTable` icon. When the desired parameters/qualifiers have been selected in the dialog box, click on the `Enter` button to create the skeleton table. Clicking on the `Help` button will display online help. Clicking on the `Quit` button terminates `SkeletonTable`.

**Parameter(s):**

`<cdf-path>` (VMS, UNIX, & MS-DOS)
`CDF` edit field (Macintosh)

The pathname of the CDF from which the skeleton table will be created. (Do not enter an extension.)

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard input file dialog is displayed from which the CDF may be selected.

**Qualifier(s):**

`/SKELETON=<skeleton-path>` (VMS)
`-skeleton <skeleton-path>` (UNIX & MS-DOS)
`Skeleton` edit field (Macintosh)

The pathname of the skeleton table to be created. (Do not enter an extension because `.skt` [UNIX & Macintosh] or `.SKT` [VMS & MS-DOS] is appended automatically.) If this qualifier is not specified, the skeleton table will be named `<cdf-name>.skt` in the default/current directory (where `<cdf-name>` is the name portion of the CDF from which the skeleton table was created).

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard output file dialog is displayed in which the skeleton table may be specified.

`/VALUES=<values> | /NRVTABLE | /NONRV` (VMS)
`-values <values> | -nrvtable | -nonrv` (UNIX & MS-DOS)
`No values/.../Selected values` radio buttons (Macintosh)

Only one of these qualifiers may be specified. The meaning of each is as follows:

`/VALUES=<values>` (VMS)
`-values <values>` (UNIX & MS-DOS)
`No values/.../Selected values` radio buttons (Macintosh)

**VMS, UNIX, & MS-DOS:** The `<values>` option specifies which variable values should be put in the skeleton table. Select one of the options from the list which follows. **Macintosh:** Selecting one of the radio buttons described in the following list specifies which variable values should be put in the skeleton table.

`none` (VMS, UNIX, & MS-DOS)
`No values` radio button (Macintosh)

No variable values should be put in the skeleton table.

`nrv` (VMS, UNIX, & MS-DOS)
`NRV values` radio button (Macintosh)

Only NRV variable values should be put in the skeleton table.

`rv` (VMS, UNIX, & MS-DOS)
`RV values` radio button (Macintosh)

Only RV variable values should be put in the skeleton table.

`all` (VMS, UNIX, & MS-DOS)
`All values` radio button (Macintosh)

All variable values should be put in the skeleton table.

`<named>` (VMS, UNIX, & MS-DOS)
`Selected values` radio button (Macintosh)

Values of the named variables should be put in the skeleton table.

**VMS, UNIX, & MS-DOS:** `<values>` is a comma-separated list of delimited variable names with the entire list enclosed in double quote marks. **NOTE:** Do not use double quote marks to delimit a variable name.

**Macintosh:** The named variables are specified in the `Variables` edit field as a comma-separated list of delimited variable names.

`/NONRV` (VMS)
`-nonrv` (UNIX & MS-DOS)
Not supported (Macintosh)

Ignore NRV data. (No values are not placed in the skeleton table.)

`/NRVTABLE` (VMS)
`-nrvtable` (UNIX & MS-DOS)
Not supported (Macintosh)

Put NRV variable data values in the skeleton table.

**VMS, UNIX, & MS-DOS:** Note that only the "values" qualifier is actually needed. The others are supported for compatibility with previous CDF distributions.

`/[NO]LOG` (VMS)
`-[no]log` (UNIX & MS-DOS)
`Log progress` check box (Macintosh)

Specifies whether or not messages are displayed as the program executes.

`/ZMODE=<mode>` (VMS)
`-zmode <mode>` (UNIX & MS-DOS)
`zMode` radio buttons (Macintosh)

Specifies the zMode that should be used with the CDF. The zMode may be one of the following:

`0` Indicates that zMode should be disabled.

`1` Indicates that zMode/1 should be used. (The dimension variances of rVariables will be preserved.)

`2` Indicates that zMode/2 should be used. (The dimensions of rVariables having a variance of `NOVARY` [false] are removed.)

`/[NO]FORMAT` (VMS)
`-[no]format` (UNIX & MS-DOS)
`Use FORMAT` check box (Macintosh)

Specifies whether or not the `FORMAT` attribute is used when writing variable values (if the `FORMAT` attribute exists and an entry exists for the variable).

`/[NO]NEG2POSFP0` (VMS)
`-[no]neg2posfp0` (UNIX & MS-DOS)
`-0.0 to 0.0` check box (Macintosh)

Specifies whether or not `-0.0` is converted to `0.0` by the CDF library when read from a CDF. `-0.0` is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`/REPORT=(<types>)` (VMS)
`-report "<types>"` (UNIX & MS-DOS)
`Report info's/warnings/errors` check boxes (Macintosh)

Specifies the types of return status codes from the CDF library that should be reported/displayed.

**VMS, UNIX, & MS-DOS:** `<types>` is a comma-separated list of zero or more of the following symbols: `errors`, `warnings`, or `informationals`. Note that these symbols can be truncated (e.g., `e`, `w`, and `i`).

`/CACHE=<size>` (VMS)
`-cache <size>` (UNIX & MS-DOS)
`Cache size` edit field (Macintosh)

Specifies the cache size to be used for the CDF's `.cdf` file (i.e., the only file if a single-file CDF). If omitted, a default cache size as chosen by the CDF library is used. The cache size is the number of 512-byte buffers to be used.

`/[NO]SCREEN` (VMS)
`-[no]screen` (UNIX & MS-DOS)
`Output to screen` check box (Macintosh)

Specifies whether or not the skeleton table is to be displayed on the terminal screen (written to the "standard output"). If not, the skeleton table is written to a text file.

`/[NO]PAGE` (VMS)
`-[no]page` (UNIX & MS-DOS)
`Page output` check box (Macintosh)

If the skeleton table is being displayed on the terminal screen, specifies whether or not the output is displayed one page (screen) at a time.

**Example(s):**

VMS:

```
$ SKELETONTABLE/NOLOG/REPORT=(ERRORS) FGGE3B
$ SKELETONTABLE/SKELETON=FGGE3B/NONRV FGGE3B
$ SKELETONTABLE/SCREEN/VALUES="'Var1','Var2'"
```

UNIX:

```
% skeletontable -nolog -report "errors" fgge3b
% skeletontable -skeleton fgge3b -nonrv ../cdfs/fgge3b
% skeletontable -values "'Var1','Var2'"
```

MS-DOS:

```
> skeletontable -nolog -report "errors" fgge3b
> skeletontable -skeleton fgge3b -nonrv c:\temp\fgge3b
> skeletontable -values "'Var1','Var2'"
```

**VMS, UNIX, & MS-DOS:** Command line help is displayed when `SkeletonTable` is executed without any arguments.

### 3.7.3 Output from the SkeletonTable Program

The format of the skeleton table is described in Appendix A.

## 3.8 SkeletonCDF

### 3.8.1 Introduction

The `SkeletonCDF` [8] program is used to make a fully structured CDF, called a skeleton CDF, by reading a text file called a skeleton table. The skeleton table contains the information necessary to create a CDF that is complete in all respects except for record-variant (RV) variable values. (RV variables vary from record to record.) RV values are then written to the CDF by the execution of an application program. The `SkeletonCDF` program allows a CDF to be created with the following.

1. The necessary header information — the number of dimensions and dimension sizes for the rVariables, format, data encoding, and variable majority.

[8] This program was originally named `CDFskeleton`. It has been renamed to ease the confusion caused some users. Now, `SkeletonCDF` is used to create skeleton CDFs and `SkeletonTable` is used to create skeleton tables.

2. The gAttribute definitions and any number of gEntries for each.
3. The rVariable and zVariable definitions.
4. The vAttribute definitions and the entries corresponding to each variable.
5. The data values for those variables that are non-record-variant (NRV). NRV variables do not vary from record to record.[9]

The created CDF is referred to as a skeleton CDF.

### 3.8.2 Executing the SkeletonCDF Program

**Usage:**

VMS:

```
$ SKELETONCDF [/CDF=<cdf-path>] [/[NO]LOG] [/[NO]DELETE] [/[NO]FILLVAL]
              [/REPORT=(<types>)] [/[NO]NEG2POSFP0] <skeleton-path>
```

UNIX:

```
% skeletoncdf [-cdf <cdf-path>] [-[no]log] [-[no]delete] [-[no]fillval]
              [-report "<types>"] [-[no]neg2posfp0] <skeleton-path>
```

MS-DOS:[10]

```
> skeletoncdf [-cdf <cdf-path>] [-[no]log] [-[no]delete] [-[no]fillval]
              [-report "<types>"] [-[no]neg2posfp0] <skeleton-path>
```

Macintosh:

Double-click on the `SkeletonCDF` icon. When the desired parameters/qualifiers have been selected in the dialog box, click on the `Enter` button to create the skeleton CDF. Clicking on the `Help` button will display online help. Clicking on the `Quit` button terminates `SkeletonCDF`.

**Parameter(s):**

`<skeleton-path>` (VMS, UNIX, & MS-DOS)

[9] RV variable data may also be specified in a skeleton table but this is generally not done given the amount of RV data typically contained in a CDF.

[10] On MS-DOS systems the executable is named `SKT2CDF.EXE`.

`Skeleton` edit field (Macintosh)

The pathname of the skeleton table from which a skeleton CDF will be created. (Do not specify an extension.)

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard input file dialog is displayed from which a skeleton table may be selected.

**Qualifier(s):**

`/CDF=<cdf-path>` (VMS)
`-cdf <cdf-path>` (UNIX & MS-DOS)
`CDF` edit field (Macintosh)

The pathname of the CDF that will be created (overriding the pathname in the skeleton table). If this qualifier is not specified, the CDF pathname in the skeleton table is used. Do not specify an extension in the pathname.

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard output file dialog is displayed in which a CDF name may be specified.

`/[NO]LOG` (VMS)
`-[no]log` (UNIX & MS-DOS)
`Log progress` check box (Macintosh)

Specifies whether or not messages are displayed as the program executes.

`/[NO]NEG2POSFP0` (VMS)
`-[no]neg2posfp0` (UNIX & MS-DOS)
`-0.0 to 0.0` check box (Macintosh)

Specifies whether or not `-0.0` is converted to `0.0` by the CDF library when written to a CDF. `-0.0` is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`/[NO]DELETE` (VMS)
`-[no]delete` (UNIX & MS-DOS)
`Delete existing` check box (Macintosh)

Specifies whether or not the CDF will be deleted first if it already exists (essentially overwriting it).

`/[NO]FILLVAL` (VMS)
`-[no]fillval` (UNIX & MS-DOS)
`Use FILLVAL` check box (Macintosh)

Specifies whether or not entries of the `FILLVAL` vAttribute are used to set the pad values for the corresponding variables. If this qualifier is specified, the `FILLVAL` vAttribute must exist and only those variables with an entry for the `FILLVAL` vAttribute will be affected.

`/REPORT=(<types>)` (VMS)
`-report "<types>"` (UNIX & MS-DOS)
`Report info's/warnings/errors` check boxes (Macintosh)

Specifies the types of return status codes from the CDF library that should be reported/displayed.

**VMS, UNIX, & MS-DOS:** `<types>` is a comma-separated list of zero or more of the following symbols: `errors`, `warnings`, or `informationals`. Note that these symbols can be truncated (e.g., `e`, `w`, and `i`).

**Example(s):**

VMS:

```
$ SKELETONCDF FGGE3B
$ SKELETONCDF/NOLOG/CDF=[-.TEMP]FGGE3B_X/REPORT=(ERRORS) FGGE3B
```

UNIX:

```
% skeletoncdf fgge3b
% skeletoncdf -nolog -cdf ../fgge3b_x -report "errors" fgge3b
```

MS-DOS:

```
> skeletoncdf fgge3b
> skeletoncdf -nolog -cdf ..\fgge3b_x -report "errors" a:\fgge3b
```

**VMS, UNIX, & MS-DOS:** Command line help is displayed when `SkeletonCDF` is executed without any arguments.

### 3.8.3 Creating the Skeleton Table

A skeleton table is a text file having `.skt` (UNIX & Macintosh) or `.SKT` (VMS & MS-DOS) as a file extension. The normal method of creating and using a skeleton table would be to use `SkeletonTable` on an existing CDF that is similar to the CDF you want to create. Then edit the created skeleton table to meet your needs, and use `SkeletonCDF` to create the new CDF. The skeleton table could also be created from scratch with any text editor.

The format of the skeleton table is described in Appendix A.

## 3.9 CDFbrowse

`CDFbrowse` has been replaced by `CDFedit` (see Section 3.12).[11]

[11] The alias/symbol `CDFbrowse` still exists in the "definitions" file on UNIX/VMS systems but now executes `CDFedit` in a browse-only mode.

## 3.10 CDFlist

### 3.10.1 Introduction

The `CDFlist` program allows you to selectively list all (or a subset of) the variable data values from a CDF. The values can be displayed at the screen, output to a text file, or used to create a new CDF. (The new CDF will most likely be a subset of the original CDF.)

### 3.10.2 Executing the CDFlist Program

**Usage:**

VMS:

```
$ CDFLIST [/INITIAL=(<defaults>)] [/SPECIFICATION=<default-cdf-spec>]
          [/CACHE=<size>] [<initial-cdf-spec>]
```

UNIX:

```
% cdflist [-initial "<defaults>"] [-specification <default-cdf-spec>]
          [-cache <size>] [<initial-cdf-spec>]
```

MS-DOS:

```
> cdflist [-initial "<defaults>"] [-specification <default-cdf-spec>]
          [-cache <size>] [<initial-cdf-spec>]
```

Macintosh:

Double-click on the `CDFlist` icon. When the desired parameters/qualifiers have been selected in the dialog box, click on the `Enter` button to begin listing the CDF(s). Clicking on the `Help` button will display online help. Clicking on the `Quit` button terminates `CDFlist`.

**Parameter(s):**

`<initial-cdf-spec>` (VMS, UNIX, & MS-DOS)
`CDF` edit field (Macintosh)

The initial specification of the CDF(s) to list. (Do not specify an extension.) This may be either a single CDF pathname or a directory/wildcard path. Wildcards are allowed in the CDF name but not in the directory path. If this parameter is specified, then the "specification" qualifier may not be specified. If neither this parameter nor the "specification" qualifier are specified, then the initial CDF(s) specification will be the default/current directory.

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard input file dialog is displayed from which a single CDF may be selected.

**Qualifier(s):**

`/INITIAL=(<defaults>)` (VMS)
`-initial "<defaults>"` (UNIX & MS-DOS)
`Initial options` check boxes and radio buttons, see below (Macintosh)

The default settings that initially appear on the `Input Options`, `Output Options`, and `Listing Options` windows. Whether or not to automatically select zMode may also be specified. These settings are only the settings initially in effect. The user may change any of them at any time. More detailed descriptions of each option may be found in the appropriate sections that follow.

**VMS, UNIX, & MS-DOS:** `<defaults>` is a comma-separated list of settings consisting of one or more of the options in the list that follows. **Macintosh:** The initial settings are selected using the check boxes described in the list that follows.

`ZMODE` (VMS)
`zmode` (UNIX & MS-DOS)
`zMode` check box (Macintosh)

When specified, zMode is automatically selected when a CDF is opened.

`[NO]ERASE` (VMS)
`[no]erase` (UNIX & MS-DOS)
`Erase, new CDF` check box (Macintosh)

Specifies whether or not to erase the `Variable Selection` window when a new CDF is selected.

`[NO]FILTER` (VMS)
`[no]filter` (UNIX & MS-DOS)
`Use filters` check box (Macintosh)

Specifies whether or not to use the variable filters when generating output.

`[NO]FILL` (VMS)
`[no]fill` (UNIX & MS-DOS)
`Use FILLVAL` check box (Macintosh)

Specifies whether or not to use the `FILLVAL` attribute when filtering.

`[NO]FORMAT` (VMS)
`[no]format` (UNIX & MS-DOS)
`Use FORMAT` check box (Macintosh)

Specifies whether or not to use the `FORMAT` attribute when encoding values to be displayed.

`[NO]NEG2POSFP0` (VMS)
`[no]neg2posfp0` (UNIX & MS-DOS)
`-0.0 to 0.0` check box (Macintosh)

Specifies whether or not to convert `-0.0` to `0.0` when read from or written to a CDF. `-0.0` is an illegal floating-point value on VAXes and DEC Alphas running

OpenVMS.

`SINGLE | MULTI` (VMS)
`single | multi` (UNIX & MS-DOS)
`Single-file/Multi-file` radio buttons (Macintosh)

Specifies the format of created CDFs.

`HOST | NETWORK` (VMS)
`host | network` (UNIX & MS-DOS)
`Host/Network` radio buttons (Macintosh)

Specifies the encoding of created CDFs.

`HORIZONTAL | VERTICAL` (VMS)
`horizontal | vertical` (UNIX & MS-DOS)
`Horizontal/Vertical` radio buttons (Macintosh)

Specifies the type of text output desired while in zMode.

`EPOCH | EPOCH1` (VMS)
`epoch | epoch1` (UNIX & MS-DOS)
`Epoch/Epoch1` radio buttons (Macintosh)

Specifies the style in which `CDF_EPOCH` values should be displayed.

`/SPECIFICATION=<default-cdf-spec>` (VMS)
`-specification <default-cdf-spec>` (UNIX & MS-DOS)
Not supported (Macintosh)

**VMS, UNIX, & MS-DOS:** A default CDF(s) specification for the CDF name field. The user may choose this specification or enter another. This qualifier may not be specified if the initial CDF(s) specification parameter was specified.

`/CACHE=<size>` (VMS)
`-cache <size>` (UNIX & MS-DOS)
`Cache size` edit field (Macintosh)

Specifies the cache size to be used for a CDF's `.cdf` file (i.e., the only file if a single-file CDF). If omitted, a default cache size as chosen by the CDF library is used. The cache size is the number of 512-byte buffers to be used.

**Example(s):**

VMS:

```
$ CDFLIST
$ CDFLIST GISS_WETL
$ CDFLIST NCDS$DATA:FGGE3B
$ CDFLIST/INITIAL=(ZMODE,HORIZONTAL)/SPECIFICATION=NCDS$DATA:
```

UNIX:

```
% cdflist
```

```
% cdflist ../cac_sst_blended
% cdflist ~/CDFs
% cdflist -initial "zmode,horizontal" -specification ~/CDFs
```

MS-DOS:

```
> cdflist
> cdflist ..\cac_sst
> cdflist a:\cdfs
> cdflist -initial "zmode,horizontal" -specification a:\cdfs
```

### 3.10.3 Interaction with CDFlist

The user interface to `CDFlist` is contained on a single screen divided into several fixed windows and pop-up windows. The `Action Menu` controls the execution of the program. This menu allows you to move to a different window, start the data output, browse a help file, or exit the program. You must return to this menu each time you want to select a new variable, output option, listing type, or input CDF. This menu may be displayed at any time by entering the `Action` key. Use the `Up` and `Down` keys to highlight a choice in the `Action Menu` and then enter the `Select` key to execute that function. The `Action Menu` appears in Figure 3.1.

```
Generate Output
Select Input CDF name
Select Output Device
Select Variables
Select Monotonic Variables
Save Variables to file
Restore Variables from file
Select r/z/zMODE Variables
Select CDF Input Options
Select CDF Output Options
Select CDF Listing Options
Help
Exit
```

Figure 3.1: `Action Menu`, `CDFlist`

Two very important windows are the `Keypad Definition` and `Message` windows. They contain information on keyboard functionality (i.e., which keys to use to perform the operations described below) and program status/error messages.

When `CDFlist` is executed (and a CDF and/or directory is not entered on the command line), you must first specify the CDF you wish to list. If you enter a disk/directory/wildcard specification (without a CDF name), all of the CDFs meeting the specification will appear in a separate pop-up window. Use the `Up` and `Down` keys to choose a CDF and then enter the `Select` key to select that CDF.

The `Output` window will then be displayed with the valid choices. Use the `Up` and `Down` keys to move up and down within the window and then enter the `Select` key to select the desired output option. If output to an

ASCII text file or CDF is selected, you will be prompted for a file/CDF name. You may enter any file/CDF name valid for the operating system being used or accept the default file/CDF name that appears. Do not enter a file extension. (It is automatically appended.) The file extension for a generated ASCII text file is `.lis`.

The second window (that remains constant on the screen) is the `Variable Selection` window. This window allows the user to select the variables to be listed or put into a CDF along with any desired filters on the variables. You will automatically be placed in this window after you have entered an output option. You can also enter this window by requesting the `Action Menu` and then selecting the `Select Variables` option.

There are four fields on each line in the `Variable Selection` window: variable name, minimum filter value, maximum filter value, and output toggle. You can move forward to the next field by entering the `Next Field` key or backwards to the previous field with the `Prev Field` key. You may also use the `Up` and `Down` keys to move vertically through the fields. When the cursor is positioned in a variable name field, entering the `Pop-up` key will display the variables in the CDF in a pop-up window. Use the arrow keys to highlight a variable and then enter the `Select` key to select it. You can quit the pop-up window without selecting a variable by entering the `Quit` key. You may also use the `Page Up` and `Page Down` keys to traverse through the pop-up window. The first line in this pop-up menu contains `**SELECT ALL**`. Selecting this item causes all of the variables to be selected. The second line in this pop-up window contains `**RECORD NUM**`. This simply gives the user the opportunity to display/filter on the record numbers in the CDF.

After selecting a variable, the cursor is positioned in the minimum filter field. A filter is simply the range of data that you would like to see from the selected variable. If the CDF contains a `VALIDMIN` and `VALIDMAX` attribute entry for the selected variable, these values will be displayed as default minimum/maximum filter values. If just specifying a minimum filter value is sufficient, a maximum filter value need not be entered. You do not have to enter a filter for each selected variable. For a line of output to be written for a given record/indices, each filter must be passed. There are two ways to enter minimum and maximum filters. The first is by simply entering the desired values in the appropriate fields. The second method can only be used for variables that consist of a discrete set of values. This method involves entering the `Pop-up` key (when in the desired filter field), which displays a pop-up window that contains the discrete values for the variable. The desired filter value may then be selected. (The `Page Up` and `Page Down` keys may also be used to traverse through the window.)

The minimum/maximum fields (filters) will initially be highlighted. This indicates that the corresponding filter will be used for a variable (if filter values are present). A filter for a particular variable may also be turned off (without erasing the filter values) by entering the `Filter On/Off` key when on the variable name field. This will toggle off the highlighting of the filter values and indicates that no filtering is to be performed on that variable. Entering the `Filter On/Off` key again toggles filtering back on for that variable. Note that erasing the filter values is another way to turn off filtering for a variable.

The output toggle field simply gives the user the option not to view a variable they have selected. A user may only wish to filter on a variable and not view it. When positioned on this field, the user simply enters the `Out on` key to display the variable and `Out Off` key to not display the variable. Note that the user may also enter `Out On/Off` keys from the variable name field.

When all of the desired variables and filters have been selected, enter the `Action` key to display the `Action Menu`. Selecting `Generate Output` will cause the listing or new CDF to be generated. Note that the amount of time needed to generate the output will in most cases be significantly reduced if the `MONOTON` attribute and corresponding entries are present in the CDF (and the variables being filtered are monotonic). See the section explaining `Specifying Monotonic Variables` for more information on monotonic variables. When `CDFlist` completes the generation of the output, the `Action Menu` is redisplayed. It may then be used to select a new CDF to list, respecify the output option, select new variables/filters, get on-line help, select special options, or exit the program.

**MS-DOS:** A listing may be terminated early by entering the `Quit` key.

When on the `Action Menu` options are available to save/restore the current selections in the `Variable Selection` window. A user may wish to exit the program and upon returning restore the current selections quickly to the window. When the `Save Variables To File` option is selected, the user is prompted for a file name. (Do not enter an extension.) All of the selections in the `Variable Selection` window are stored in that file. A default file name of `cdflist` is available by just entering the `Select` key at the prompt. When the `Restore Variables From File` option is selected, the user is again prompted for a file name. (`cdflist` is again the default.) The selections are restored to the `Variable Selection` window from that file. `CDFlist` will check the current CDF for the existence of each variable (displaying only those variables that do exist).

Another option on the `Action Menu` is `Select r/z/zMODE Variables`. `CDFlist` groups variables in three separate manners: rVariables, zVariables, and zMode. All of the rVariables are contained in one group; zVariables are grouped together by dimensionality; and zMode consists of all of the variables in the CDF with the rVariables being made to appear as zVariables. Variables from only one of these groups at a time may be selected for filtering/output. See Section 2.3 for more information on r/zVariables and Section 2.1.2 for more information on zMode.

On the `Action Menu` three windows of special options are available. Each option brings up a window containing a set of small option selection lists (an example of which is shown in Figure 3.2). The user makes selections by moving the `*` (the current choice) with the `Up` and `Down` keys and moving to the next list by entering the `Select` key. When entering an option selection list, if the current choice is the one desired, just enter the `Select` key to move to the next list. To return back to the `Action Menu`, enter the `Quit` key at any time.

```
(*) Single
( ) Multi
```

Figure 3.2: Option Selection List, `CDFlist`

Selecting `Select CDF Input Options` causes a window to be displayed with the following option selection lists:

1. Whether or not to erase the variables in the `Variable Selection` window each time a new input CDF is selected. If users select a CDF that contains duplicate variables, they may wish to reuse the current selections. If users choose not to erase selected variables, `CDFlist` will check to make sure that the current variables actually exist when a new CDF is selected.

Selecting `Select CDF Output Options` causes a window to be displayed with the following option selection lists:

1. When selecting output to a CDF, whether the file format of the new CDF is single-file or multi-file.
2. When selecting output to a CDF, whether the encoding of the new CDF is network or host.

Selecting `Select CDF Listing Options` causes a window to be displayed with the following option selection lists:

1. The format used when displaying EPOCH values may be selected. (The `FORMAT` attribute is always ignored when displaying EPOCH values.)

2. Whether or not to use the `FORMAT` attribute when displaying values may be selected. (The `FORMAT` attribute and corresponding entries must exist in the CDF.)

3. Whether or not to use the `FILLVAL` attribute when filtering (assuming the `FILLVAL` attribute exists). If the `FILLVAL` attribute is to be used, fill values in the data are ignored when determining whether or not to filter out a set (line) of values (i.e., fill values outside the filter range will appear in the output).

4. Whether or not to ignore all of the filters when generating the output. This is a more convenient way to turn off filtering than to do so manually for each variable on the `Variable Selection` window.

5. Whether or not to convert `-0.0` to `0.0` when read from or written to a CDF. `-0.0` is an illegal floating-point value on VAXes and DEC Alphas running OpenVMS. (`CDFlist` may abort with a system error.)

6. Whether to display values vertically or horizontally while in zMode.

On the `Action Menu` the option `Specify Monotonic Variables` gives the user the opportunity to specify that a variable is monotonic. This is useful when a CDF does not contain the attribute `MONOTON` or has the entries incorrectly specified. When this option is executed, a pop-up window is displayed containing all of the variables in the CDF and their monotonicities. If a `MONOTON` attribute entry exists for a variable, it is set accordingly. An `I` is displayed if the values increase, `D` if the values decrease, and `F` if the variable is not monotonic (or if an entry does not exist). If a variable varies with more than one "dimension", `N/A` is displayed — these variables can never be specified monotonic. Once inside the pop-up window, the user traverses through the window as in any other pop-up window but with one exception — the cursor is positioned to the monotonic value of the variable. The user then may change that value by entering the `INCREASE` key for increasing values, the `DECREASE` key for decreasing values, or the `FALSE` key for not monotonic.

## 3.11 CDFwalk

### 3.11.1 Introduction

The `CDFwalk` program allows direct access to all variable data in a CDF. The data values of up to 27 variables may be displayed at once. (The CDF can have more variables but only 27 may be viewed at any one time.) The data value for each variable at a given record/indices location within the CDF conceptual view is displayed. The user can "walk" from one record/indices location to another sequentially (in any "direction") or a particular record/indices location can be "jumped" to directly. Variable values may also be used to directly "jump" to a record/indices location.

### 3.11.2 Executing the CDFwalk Program

**Usage:**

VMS:

```
$ CDFWALK [/INITIAL=(<defaults>)] [/SPECIFICATION=<default-cdf-spec>]
          [/CACHE=<size>] [<initial-cdf-spec>]
```

UNIX:

```
% cdfwalk [-initial "<defaults>"] [-specification <default-cdf-spec>]
          [-cache <size>] [<initial-cdf-spec>]
```

MS-DOS:

```
> cdfwalk [-initial "<defaults>"] [-specification <default-cdf-spec>]
          [-cache <size>] [<initial-cdf-spec>]
```

Macintosh:

Double-click on the `CDFwalk` icon. When the desired parameters/qualifiers have been selected in the dialog box, click on the `Enter` button to begin walking through the CDF(s). Clicking on the `Help` button will display online help. Clicking on the `Quit` button terminates `CDFwalk`.

**Parameter(s):**

`<initial-cdf-spec>` (VMS, UNIX, & MS-DOS)
`CDF` edit field (Macintosh)

The initial specification of the CDF(s) at which to begin "walking." (Do not specify an extension.) This may be either a single CDF pathname or a directory/wildcard path. Wildcards are allowed in the CDF name but not in the directory path. If this parameter is specified, then the "specification" qualifier may not be specified. If neither this parameter nor the "specification" qualifier are specified, then the initial CDF(s) specification will be the default/current directory.

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard input file dialog is displayed from which a single CDF may be selected.

**Qualifier(s):**

`/INITIAL=(<defaults>)` (VMS)
`-initial "<defaults>"` (UNIX & MS-DOS)
`Initial options` check boxes, see below (Macintosh)

The default settings that initially appear on the `Special Options` window. These setting are only the settings initially in effect. The user may change any of them at any time. More detailed descriptions of each option may be found in the appropriate sections that follow.

**VMS, UNIX, & MS-DOS:** `<defaults>` is a comma-separated list of settings consisting of one or more of the options in the list that follows. **Macintosh:** The initial settings are selected using the check boxes described in the list that follows.

`[NO]FORMAT` (VMS)

[no]format (UNIX & MS-DOS)
Use FORMAT check box (Macintosh)

Specifies whether or not to use the FORMAT when encoding values to be displayed.

[NO]NEG2POSFP0 (VMS)
[no]neg2posfp0 (UNIX & MS-DOS)
-0.0 to 0.0 check box (Macintosh)

Specifies whether or not to convert -0.0 to 0.0 when read from a CDF. -0.0 is an illegal floating-point value on VAXes and DEC Alphas running OpenVMS.

/SPECIFICATION=<default-cdf-spec> (VMS)
-specification <default-cdf-spec> (UNIX & MS-DOS)
Not supported (Macintosh)

A default CDF specification for the CDF name field. The user may choose this specification or enter another. This qualifier may not be specified if the initial CDF(s) specification parameter was specified.

/CACHE=<size> (VMS)
-cache <size> (UNIX & MS-DOS)
Cache size edit field (Macintosh)

Specifies the cache size to be used for a CDF's .cdf file (i.e., the only file if a single-file CDF). If omitted, a default cache size as chosen by the CDF library is used. The cache size is the number of 512-byte buffers to be used.

**Example(s):**

VMS:

```
$ CDFWALK
$ CDFWALK [-.TEMP]NOAA7
$ CDFWALK/INITIAL=(NOFORMAT,NEG2POSFP0)/SPECIFICATION=NCDS$DATA:
```

UNIX:

```
% cdfwalk
% cdfwalk ../cac_sst
% cdfwalk /disk3/CDFs/ssclimate
% cdfwalk -initial "noformat,neg2posfp0" -specification ~/CDFs
```

MS-DOS:

```
> cdfwalk
> cdfwalk ..\cac_sst
> cdfwalk c:\cdfs
> cdfwalk -initial "noformat,neg2posfp0" -specification c:\cdfs
```

### 3.11.3 Interaction with CDFwalk

The user interface to `CDFwalk` is contained on a single screen divided into several fixed windows and pop-up windows. The `Action Menu` controls the execution of the program. This menu allows you to "walk" through a CDF, select a new CDF through which to "walk", select r/zVariables, select `CDFwalk` special options, browse on-line help, or exit the program. This menu may be displayed at any time by entering the `Action` key. Use the `Up` and `Down` keys to highlight a choice in the `Action Menu`, and then enter the `Select` key to execute that function. The `Action Menu` appears in Figure 3.3.

```
Walk Thru CDF data
Select Input CDF name
Select Special Options
Select r/z Variables
Help
Exit
```

Figure 3.3: `Action Menu`, `CDFwalk`

Two very important windows are the `Keypad Definition` and `Message` windows. They contain information on keyboard functionality (i.e., which keys to use to perform the operations described below) and program status/error messages.

When `CDFwalk` is executed (and a CDF and/or directory/wildcard specification was not entered on the command line), you must first specify the CDF through which you wish to "walk." If you enter a directory/wildcard specification, all of the CDFs meeting that specification will appear in a separate pop-up window. Use the `Up` and `Down` keys to choose a CDF and then enter the `Select` key to "walk" through that CDF.

After the input CDF is selected, the `Record/Indices` window appears. The size of this window depends on the dimensionality of the variables being displayed (initially the rVariables if any exist). Vertically, the window will contain one row for the record number and one row for each dimension of the variables. Horizontally, each line of the window contains three fields. The first field contains a variable name; the second field contains a data value for the variable; and the third field contains a record number or dimension index.

The variable selected for the record number line will be one that only varies from record to record (not varying along any of the dimensions). If more than one variable of that type exists, a pop-up window will be displayed containing the list of variables. Select one of the variables, keeping in mind that you will have an opportunity to use it for positioning by value to a desired record. If a variable meeting the requirements mentioned above does not exist, only the record number field will be used.

The variable selected for each dimension line will be one that only varies along the corresponding dimension (and does not vary from record to record). Again, if more than one variable of that type exists in the CDF, a pop-up window will be displayed to allow you to select the variable to use for that dimension. Keep in mind that you will be able to use it to position by value to a desired index along that dimension. If for a particular dimension a variable meeting the requirements mentioned above does not exist, only the dimension index field will be used.

After the `Record/Indices` window has been specified (which will usually occur automatically), the `Variable Values` window will be displayed. It contains up to 27 variable names and the associated data values for the record/indices shown in the `Record/Indices` window.

Initially, the cursor will be positioned at the data value field of the record number line in the `Record/Indices` window (assuming that a variable was found meeting the above requirements; if not, the cursor will start at the record number field). You may now begin to "walk" through the CDF by using the right arrow to increment to and the left arrow to decrement to a new record. For each new record the data for each variable is updated in both the `Record/Indices` and `Variable Values` windows. If the CDF contains many records, you may want to advance quickly to a desired record. You may do this by entering the `Input` key. `CDFwalk` enters an input mode and prompts for the value to which to advance. If the exact value is not in the CDF for the variable, the record number is changed to correspond to the closest possible value. Another method of advancing to a desired value is by entering the `Pop-up` key. This will cause a pop-up window containing all of the values for the variable to be displayed. You may select the value desired as you would with any other pop-up window and `CDFwalk` will automatically position the record number to the corresponding value.

After positioning to a desired record (and if the CDF is multidimensional) you may now wish to "walk" through the dimension indices. This is done by entering the up and down arrow keys to position the cursor to the desired dimension line. "Walking" through dimensions is done in the same way as "walking" through the record numbers.

You may also wish to "walk" through the CDF by using the actual record number or dimension indices rather than data values. (This would also be necessary if variables meeting the above requirements were not found.) This is done by entering the `Next Field` key, which causes the cursor to advance to the next field (i.e., from the data value field to the record number/indices field). Entering the `Back Field` key causes the cursor to move back to the previous field. "Walking" through the CDF using the record number/indices fields is the same as using the data value fields but with a few exceptions. First, no `Pop-up` option is available with these fields. Also, in input mode (after entering the `Input` key) you may input either an absolute record number/index to which to move or a relative offset from the current record number/index. Prefixing the value with a `+/-` will increment/decrement the record number/index by that amount.

If more than 27 variables exist and a particular variable is not currently displayed in the `Variable Values` window, an option is provided to scroll new variables into that window. This is done by entering the `Scroll` key. A small pop-up window is displayed in which you may use the `Next/Prev Variable` keys to scroll in or scroll out variables. When the desired set of variables is displayed, enter the `Select` key to continue "walking" through the CDF.

When you are finished "walking" through the CDF, enter the `Action` key to display the `Action Menu`. You may input a new CDF, resume "walking" through the current CDF, select special options, read on-line help, or exit the program.

On the `Action Menu` selecting `Special Options` causes a window to be displayed with the following option selection lists:

1. Whether or not to use the `FORMAT` attribute when displaying values may be selected. (The `FORMAT` attribute and corresponding entries must exist in the CDF.)
2. Whether or not to convert `-0.0` to `0.0` when read from a CDF. `-0.0` is an illegal floating-point value on VAXes and DEC Alphas running OpenVMS. (`CDFwalk` may abort with a system error.)

The user makes a selection by moving the `*` (the current choice) with the `Up` and `Down` keys and moving to the next options selection list (if another exists) by entering the `Select` key. An example option selection list is shown in Figure 3.4. To return to the `Action Menu`, enter the `Quit` key at any time.

Another option on the `Action Menu` is `Select r/z Variables`. `CDFwalk` groups variables in two separate

```
(*) CDF FORMAT attribute
( ) CDFWALK defaults
```

Figure 3.4: Option Selection List, CDFwalk

manners: rVariables and zVariables. All of the rVariables are in one group and the zVariables are grouped together by dimensionality. Only one of these groups may be "walked" through at a time. For more information on the r/zVariables see Section 2.3.

## 3.12 CDFedit

### 3.12.1 Introduction

The CDFedit program allows the display and/or modification of practically all of the contents of a CDF by way of a full-screen interface. It is also possible to run CDFedit in a browse-only mode in order to prevent accidental modifications.[12]

### 3.12.2 Executing the CDFedit Program

**Usage:**

VMS:

```
$ CDFEDIT [/[NO]BROWSE] [/ZMODE=<mode>] [/[NO]FORMAT] [/[NO]PROMPT]
          [/[NO]NEG2POSFP0] [/REPORT=(<types>)] [/CACHE=<size>]
          [/[NO]STATISTICS] [/[NO]GWITHENTRIES] [/[NO]VWITHENTRIES]
          <cdf-spec>
```

UNIX:

```
% cdfedit [-[no]browse] [-zmode <mode>] [-[no]format] [-[no]prompt]
          [-[no]neg2posfp0] [-report "<types>"] [-cache <size>]
          [-[no]statistics] [-[no]gwithentries] [-[no]vwithentries]
          <cdf-spec>
```

MS-DOS:

```
> cdfedit [-[no]browse] [-zmode <mode>] [-[no]format] [-[no]prompt]
          [-[no]neg2posfp0] [-report "<types>"] [-cache <size>]
          [-[no]statistics] [-[no]gwithentries] [-[no]vwithentries]
          <cdf-spec>
```

[12] Running CDFedit in a browse-only mode provides the same functionality as CDFbrowse once did.

Macintosh:

Double-click on the `CDFedit` icon. When the desired parameters/qualifiers have been selected in the dialog box, click on the `Enter` button to begin editing the CDF(s). Clicking on the `Help` button will display online help. Clicking on the `Quit` button terminates `CDFedit`.

**Parameter(s):**

`<cdf-spec>` (VMS, UNIX, & MS-DOS)
`CDF` edit field (Macintosh)

The specification of the CDF(s) to edit. (Do not specify an extension.) This may be either a single CDF pathname or a directory/wildcard path. Wildcards are allowed in the CDF name but not in the directory path. If the "prompt" qualifier is used, this will appear as the initial specification at the prompt. If this parameter is omitted, the "prompt" qualifier must be specified (and the initial specification at the prompt will be the default/current directory).

**Macintosh:** At the end of this field a button labeled `Select` is present. When selected, a standard input file dialog is displayed from which a single CDF may be selected. Also present is a button labeled `New`. When selected, a standard output file dialog is displayed in which a single CDF may be specified.

**Qualifier(s):**

`/[NO]BROWSE` (VMS)
`-[no]browse` (UNIX & MS-DOS)
`Browse only` check box (Macintosh)

Specifies whether or not a browsing mode is desired. In browsing mode the creation, modification, or deletion of a CDF is not allowed.

`/ZMODE=<mode>` (VMS)
`-zmode <mode>` (UNIX & MS-DOS)
`zMode` radio buttons (Macintosh)

Specifies which zMode should be used. The zMode may be one of the following:

`0` Indicates that zMode should be disabled.

`1` Indicates that zMode/1 should be used. (The dimension variances of rVariables will be preserved.)

`2` Indicates that zMode/2 should be used. (The dimensions of rVariables having a variance of `NOVARY` [false] are removed.)

`/[NO]FORMAT` (VMS)
`-[no]format` (UNIX & MS-DOS)
`Use FORMAT` check box (Macintosh)

Specifies whether or not the `FORMAT` attribute is used when displaying variable values (if the `FORMAT` attribute exists and an entry exists for the variable).

`/[NO]PROMPT` (VMS)

`-[no]prompt` (UNIX & MS-DOS)
`Prompt for CDF` check box (Macintosh)

Specifies whether or not a prompt is issued for the CDF(s) specification. When enabled the prompt will be issued both at program startup and after editing the current CDF(s) specification (at which point a new CDF[s] specification may be specified).

**VMS, UNIX, & MS-DOS:** If a CDF(s) specification was entered on the command line, that CDF(s) specification will appear at the prompt. (Otherwise, the current/default directory will appear at the prompt.)

**Macintosh:** If a CDF(s) specification was entered in the `CDF` edit field, that CDF(s) specification will appear at the prompt. (Otherwise, the current directory will appear at the prompt.)

`/[NO]NEG2POSFP0` (VMS)
`-[no]neg2posfp0` (UNIX & MS-DOS)
`-0.0 to 0.0` check box (Macintosh)

Specifies whether or not `-0.0` is converted to `0.0` by the CDF library when read from or written to a CDF. (The CDF is not modified.) `-0.0` is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`/REPORT=(<types>)` (VMS)
`-report "<types>"` (UNIX & MS-DOS)
`Report info's/warnings/errors` check boxes (Macintosh)

Specifies the types of return status codes from the CDF library that should be reported/displayed.

**VMS, UNIX, & MS-DOS:** `<types>` is a comma-separated list of zero or more of the following symbols: `errors`, `warnings`, or `informationals`. Note that these symbols can be truncated (e.g., `e`, `w`, and `i`).

`/CACHE=<size>` (VMS)
`-cache <size>` (UNIX & MS-DOS)
`Cache size` edit field (Macintosh)

Specifies the cache size to be used for a CDF's `.cdf` file (i.e., the only file if a single-file CDF). If omitted, a default cache size as chosen by the CDF library is used. The cache size is the number of 512-byte buffers to be used.

`/[NO]STATISTICS` (VMS)
`-[no]statistics` (UNIX & MS-DOS)
`Display statistics` check box (Macintosh)

Specifies whether or not caching statistics are displayed at the end of each editing session.

`/[NO]GWITHENTRIES` (VMS)
`-[no]gwithentries` (UNIX & MS-DOS)
`gEntries with gAttributes` check box (Macintosh)

Specifies whether or not gEntries are displayed with the gAttributes or on separate menus (with one menu per gAttribute).

`/[NO]VWITHENTRIES` (VMS)
`-[no]vwithentries` (UNIX & MS-DOS)
`rEntries/zEntries with vAttributes` check box (Macintosh)

Specifies whether or not rEntries/zEntries are displayed with the vAttributes or on separate menus (with one menu per vAttribute).

**Example(s):**

VMS:

```
$ CDFEDIT [.SAMPLES]
$ CDFEDIT/ZMODE=2/NOFORMAT GISS_WETLX
$ CDFEDIT/BROWSE/PROMPT/REPORT=(ERRORS)
```

UNIX:

```
% cdfedit samples
% cdfedit -zmode 2 -noformat giss_wetl
% cdfedit -browse -prompt -report "errors"
```

MS-DOS:

```
> cdfedit samples
> cdfedit -zmode 2 -noformat giss_wetl
> cdfedit -browse -prompt -report "errors"
```

### 3.12.3 Interaction with CDFedit

Interaction with `CDFedit` is through a series of menus and windows. Extensive on-line help is provided and will not be repeated here.[13] The on-line help does refer to the sections of a window by name. Figure 3.5 illustrates the various sections of the possible types of windows.

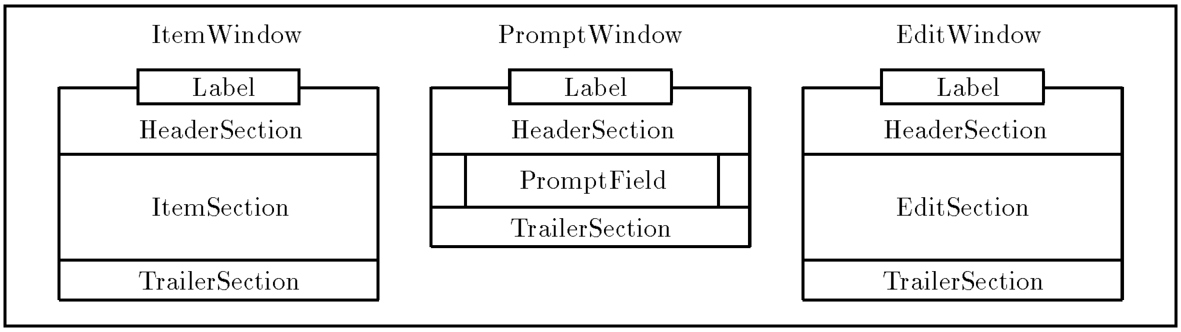


Figure 3.5: Window Sections, `CDFedit`

ItemWindows are used when a choice is to be made from a list of one or more items (e.g., functions to perform, CDFs to edit, variable names, etc.). In some cases the entire list of items may not fit on the screen at once. When this occurs, the ItemSection may be scrolled to display hidden items. Some ItemWindows have a percentage indicator at the bottom right portion of the ItemSection. The percentage indicator shows which part of the ItemSection is being displayed.

[13] It is our intention that the use of `CDFedit` be as intuitive as possible. You may not even need the on-line help. We're sure you'll let us know.

PromptWindows are used when a textual response is required (e.g., a CDF specification, a new attribute name, a variable value, etc.). If the text is too long to fit into the PromptField, the "more" indicators ("<" and ">") at the left and right ends of the PromptField will display where hidden characters exist.

EditWindows are used to display/edit a text file or group of lines. EditWindows are currently only used to display on-line help.

# Appendix A

# Skeleton Table Format

## A.1 Introduction

Skeleton tables are both created by and read by CDF utility programs. `SkeletonTable` creates a skeleton table by reading a CDF. `SkeletonCDF` creates a CDF by reading a skeleton table. In almost all cases the format of the skeleton tables read and written will be the same. Any differences are minor and will be described where appropriate.

The skeleton table has a free format (except where noted) — you need not be concerned with any column alignments, spaces between fields, or spaces between successive lines. However, certain syntax rules do apply to skeleton tables.

1. Lines are limited to 132 characters.
2. Keywords for the header section, gAttributes section, vAttributes section, rVariables section, and end section must always be specified (in that order). The zVariables section is optional — its keyword may be omitted.
3. An exclamation point (`!`) at any point signifies a comment until the end of the line. Any characters encountered after the exclamation point will be ignored. An exclamation point may begin a line (making the entire line a comment). Exclamation points inside delimited character strings are part of the string and do not cause the start of a comment.
4. Attribute and variable names must be delimited. Any character not in the name may be used as the delimiter with the following exceptions:
   (a) Do not use an exclamation point (`!`) to delimit an attribute or variable name.
   (b) Do not use a period (`.`) to delimit an attribute name in the variables section.
   (c) Do not use a left square bracket (`[`) or a numeral to delimit a variable name.
5. When specifying a character string attribute entry value, do not use a hyphen (`-`) to delimit the string or strings (if the string is split across one or more lines).
6. All items are referenced from one (1). These include gAttribute gEntry numbers and NRV variable index values.

In the descriptions that follow, optional fields are shown in brackets ([...]).

## A.2 Header Section

The header section contains general information about the CDF. The format of the header section is as follows:

```
#header

                    CDF NAME: <cdf-name>
               DATA ENCODING: <data-encoding>
                    MAJORITY: <variable-majority>
                      FORMAT: <cdf-format>

!     Variables   G.Attributes V.Attributes   Records     Dims       Sizes
!     ---------   ------------ ------------   -------     ----    -----------
   <rVars>/<zVars>   <gAttrs>    <vAttrs>   <n-recs>/z  <n-dims>  <dim-sizes>
```

The fields are defined as follows:

| | |
|---|---|
| `<cdf-name>` | The name of the CDF. When SkeletonTable creates a skeleton table, this will be the name of the corresponding CDF (not the full pathname specified). When SkeletonCDF reads a skeleton table, this will be the name of the CDF created unless a CDF pathname is specified on the command line. If the CDF name in the skeleton table is to be used, a full pathname must be specified (if desired) or else the CDF will be created in the default/current directory. |
| `<data-encoding>` | The data encoding of the CDF. When specifying a data encoding to the SkeletonCDF program, the following encodings are valid: HOST, NETWORK, VAX, ALPHAVMSd, ALPHAVMSg, SUN, SGi, DECSTATION, ALPHAOSF1, IBMRS, HP, IBMPC, MAC, and NeXT. When a skeleton table is created by SkeletonTable, all of the above encodings with the exception of HOST are possible. Data encoding is described in Section 2.2.8. |
| `<variable-majority>` | The variable majority of the CDF. This may be either ROW or COLUMN. Variable majority is described in Section 2.3.12. |
| `<cdf-format>` | The format of the CDF. This may be either SINGLE or MULTI. CDF formats are described in Section 2.2.7. Note that this line is optional. Skeleton tables created by SkeletonTable in CDF V2.0 did not have this line because the single-file option did not exist. To allow SkeletonCDF to read skeleton tables created with SkeletonTable in CDF V2.0, this line was made optional. If omitted, SkeletonCDF will create a CDF with the default format for your CDF distribution. Consult your system manager to determine this default. SkeletonTable (in CDF V2.1 and beyond) always generates this line regardless of the version of the CDF being read. |

`<rVars>` The number of rVariables in the CDF. `SkeletonTable` always places the correct number here. However, when `SkeletonCDF` reads a skeleton table, this value is ignored (but a place holder is necessary). The number of rVariables created is determined by the number of rVariable definitions in the rVariable definitions section.

`<zVars>` The number of zVariables in the CDF. `SkeletonTable` always places the correct number here. However, when `SkeletonCDF` reads a skeleton table, this value is ignored (but a place holder is necessary). The number of zVariables created is determined by the number of zVariable definitions in the zVariable definitions section.

`<gAttrs>` The number of gAttributes in the CDF. `SkeletonTable` always places the correct number here. However, when `SkeletonCDF` reads a skeleton table, this value is ignored (but a place holder is necessary). The number of gAttributes created is determined by the number of definitions in the gAttributes section.

`<vAttrs>` The number of vAttributes in the CDF. `SkeletonTable` always places the correct number here. However, when `SkeletonCDF` reads a skeleton table, this value is ignored (but a place holder is necessary). The number of vAttributes created is determined by the number of definitions in the vAttributes section.

`<n-recs>` The (maximum) number of rVariable records in the CDF. `SkeletonTable` always places the correct number here. However, when `SkeletonCDF` reads a skeleton table, this value is ignored (but a place holder is necessary). The number of records written to the CDF depends on whether or not any values are specified for variables. NRV variables are described in Section 2.3.10.

`<n-dims>` The number of dimensions for the rVariables in the CDF.

`<dim-sizes>` The dimension sizes for the rVariables in the CDF — one value per dimension. If the rVariables have zero (0) dimensions, this field would be left blank.

An example header section for a CDF with 2-dimensional rVariables follows:

```
#header

                  CDF NAME: sample2
             DATA ENCODING: NETWORK
                  MAJORITY: ROW
                    FORMAT: SINGLE

! Variables G.Attributes V.Attributes Records  Dims   Sizes
! --------- ------------ ------------ -------  ----  -------
    14/0         18            4        1/z      2   180 360
```

If the rVariables had zero dimensions, the header section would be as follows:

```
#header
```

```
               CDF NAME: sample0
          DATA ENCODING: NETWORK
               MAJORITY: ROW
                 FORMAT: SINGLE

! Variables G.Attributes V.Attributes Records  Dims  Sizes
! --------- ------------ ------------ -------  ----  -----
    14/0         18             4        1/z      0
```

## A.3 gAttributes Section

The gAttributes section contains the definition of each gAttribute as well as any gEntries for those gAttributes. The format of the gAttributes section is as follows:

```
#GLOBALattributes

[<global-scope-attribute-definition>
 <global-scope-attribute-definition>
 <global-scope-attribute-definition>
                 .
                 .
                 .
 <global-scope-attribute-definition>]
```

Where `<global-scope-attribute-definition>`, needless to say, is a gAttribute definition.

Zero or more gAttribute definitions are allowed. (There is no limit on the number of attributes that a CDF may have.) The format of each gAttribute definition is as follows:

```
! Attribute        Entry       Data
! Name             Number      Type            Value
! ---------        ------      ----            -----

  <attr-name>    [<entry-n>:    <data-type>    <value>
                  <entry-n>:   [<data-type>]   <value>
                  <entry-n>:   [<data-type>]   <value>
                      .              .             .
                      .              .             .
                      .              .             .
                  <entry-n>:   [<data-type>]   <value>] .         ! note the "."
```

The fields are defined as follows:

`<attr-name>` The name of the gAttribute. The name must be delimited with a character not appearing in the name itself (e.g., "TITLE" or 'History'). The delimiting characters are not part of the gAttribute name in the CDF.

`<entry-n>` The gEntry number. Zero or more gEntries may be specified for a gAttribute, and there are no restrictions on the gEntry numbers that may be used (except that they must be greater than zero).

`<data-type>` The data type for the gEntry. The data type must be one of the following: `CDF_BYTE`, `CDF_INT1`, `CDF_UINT1`, `CDF_INT2`, `CDF_UINT2`, `CDF_INT4`, `CDF_UINT4`, `CDF_REAL4`, `CDF_FLOAT`, `CDF_REAL8`, `CDF_DOUBLE`, `CDF_EPOCH`, `CDF_CHAR`, or `CDF_UCHAR`. The `<data-type>` field is optional for all but the first gEntry specified. If omitted, the data type of the previous gEntry is assumed.

`<value>` The value(s) for the gEntry. A period (`.`) follows the value(s) of the last gEntry for a gAttribute.

**Attribute Entry Values**

An attribute entry can have more than one element of the specified data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), each character is an element. The character string must be delimited with a character not appearing in the string itself, and the entire delimited string must be enclosed in braces (e.g., `{ "The CDF title." }`). If the string will not fit on one line, it may be continued on additional lines. The substrings are each delimited with a unique character, and a dash (`-`) is placed at the end (after the terminating delimiter) of each line except the last one. For example,

```
{ "This is a longer " -
  "CDF title that will" -
  " not fit on one line." }
```

For non-character string data types, the elements are enclosed in braces and separated by commas (e.g., `{ 1, 2, 3 }`). If the elements will not all fit on one line, they may be continued on additional lines. For example,

```
{ 1.0, 2.0, 3.0, 4.0, 5.0,
  6.0, 7.0, 8.0, 9.0, 10.0 }
```

Note that an individual element value may not be split across lines.

The format of a value for the `CDF_EPOCH` data type (which is also considered a non-character string data type) is defined in Section 2.2.10. A `CDF_EPOCH` value may not be split across two lines.

Several example gAttribute definitions follow:

```
#GLOBALattributes

! Attribute         Entry       Data
! Name              Number      Type        Value
! ---------         ------      ----        -----
```

```
  "TITLEa"              1:    CDF_CHAR     { "CDAW-9A; SABRE" }.

  ^TITLEb^              1:    CDF_CHAR     { "CDAW-9A; SABRE " -
                                             "Backscatter Radar, 20s." }.

  "History"             1:    CDF_CHAR     { "CDF created 02-Jan-1961" }
                        2:                 { "CDF modified 23-Oct-1964" }.

  "TIMES"               1:    CDF_EPOCH    { 04-Jul-1976 12:00:00.000,
                                             31-Oct-1976 00:00:00.000 }
                        2:                 { 25-Dec-1976 01:10:00.000,
                                             01-Jan-1977 01:10:30.000 }.

  &Factors&             1:    CDF_REAL4    { 12.5 }
                        2:                 { 17.4 }
                        3:                 {  8.5 }
                        4:    CDF_INT2     { 7 }
                        5:                 { 12 }.
```

## A.4 vAttributes Section

The vAttributes section contains the names of the vAttributes in the CDF. Any rEntries or zEntries for these vAttributes are defined in the rVariables/zVariables sections (following the definition of the corresponding variable). The format of the vAttributes section is as follows:

```
#VARIABLEattributes

[<attribute-name>
 <attribute-name>
 <attribute-name>
        .
        .
        .
 <attribute-name>]
```

Where `<attribute-name>` is a vAttribute name delimited with a character not appearing in the name itself (e.g., `"VALIDMIN"` or `'Units'`). The delimiting characters are not part of the vAttribute name in the CDF. There may be zero or more vAttribute names. (There is no limit on the number of attributes that a CDF may have.)

An example vAttributes section follows:

```
#VARIABLEattributes

  "FIELDNAM"
  "VALIDMIN"
  "Units"
```

## A.5 rVariables Section

The rVariables section contains the definition of each rVariable in the CDF, the values for any vAttribute rEntries associated with each rVariable, and (optionally) data values for those rVariables. The format of the rVariables section is as follows:

```
#variables

[<variable-definition>
 <variable-definition>
 <variable-definition>
           .
           .
           .
 <variable-definition>]
```

Where `<variable-definition>` is an rVariable definition. The format of each rVariable definition is as follows:

```
! Variable            Data         Number     Record     Dimension
! Name                Type        Elements   Variance    Variances
! --------            ----        --------   --------    ---------

  <var-name>     <var-data-type>  <n-elems> <rec-vary>  <dim-varys>

  !  Attribute           Data
  !  Name                Type           Value
  !  ---------           ----           -----

    [<attr-name>  <entry-data-type>    <entry-value>
     <attr-name>  <entry-data-type>    <entry-value>
     <attr-name>  <entry-data-type>    <entry-value>
          .              .                   .
          .              .                   .
          .              .                   .
     <attr-name>  <entry-data-type>    <entry-value>].        ! Note the "."

   [[<rec-num>:]<indices> = <value>
    [<rec-num>:]<indices> = <value>
    [<rec-num>:]<indices> = <value>
        .            .
        .            .
        .            .
    [<rec-num>:]<indices> = <value>]
```

Each field is defined as follows:

`<var-name>` The name of the rVariable. The name must be delimited with a character not appearing in the name itself (e.g., `"EPOCH"` or `'Temperature'`). The delimiting characters are not part of the rVariable name in the CDF.

`<var-data-type>` The data type for the rVariable. The data type must be one of the following: `CDF_BYTE`, `CDF_INT1`, `CDF_UINT1`, `CDF_INT2`, `CDF_UINT2`, `CDF_INT4`, `CDF_UINT4`, `CDF_REAL4`, `CDF_FLOAT`, `CDF_REAL8`, `CDF_DOUBLE`, `CDF_EPOCH`, `CDF_CHAR`, or `CDF_UCHAR`.

`<n-elems>` The number of elements of the data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in each string. For non-character string data types, this value must be one (1).

`<rec-vary>` The record variance of the rVariable. This must be either `T` (the values vary from record to record) or `F` (the values do not vary from record to record).

`<dim-varys>` The dimension variances of the rVariable. For each dimension there must be either a `T` (the values vary along that dimension) or `F` (the values do not vary along that dimension). Each dimension variance must be separated by at least one space. If the rVariables have zero dimensions, this field would be left blank.

`<attr-name>` The name of the vAttribute for which to specify an rEntry for this rVariable. The vAttribute must have been specified in the vAttributes section. The name must be delimited with a character not appearing in the name itself (e.g., `"SCALEMAX"` or `'range'`). The delimiting characters are not part of the vAttribute name in the CDF.

`<entry-data-type>` The data type for the vAttribute rEntry. The data type must be one of the following: `CDF_BYTE`, `CDF_INT1`, `CDF_UINT1`, `CDF_INT2`, `CDF_UINT2`, `CDF_INT4`, `CDF_UINT4`, `CDF_REAL4`, `CDF_FLOAT`, `CDF_REAL8`, `CDF_DOUBLE`, `CDF_EPOCH`, `CDF_CHAR`, or `CDF_UCHAR`.

`<entry-value>` The value(s) for the vAttribute rEntry. The format of attribute entry values is described in Section A.3.

**NOTE:** The last rEntry MUST be followed by a period (`.`). If no rEntries are specified for an rVariable, the period must still be present.

`<rec-num>` The record number of an rVariable value. This will be present only for record-variant (RV) rVariables.

`<indices>` The indices of an rVariable value. The indices are enclosed in brackets and separated by commas (e.g., `[23,1]` or `[1,80]`). If the rVariables have zero dimensions, `[]` would be specified (the brackets are still required).

`<value>` The value at the given record/indices. For character string data types (`CDF_CHAR` or `CDF_UCHAR`) the string must be delimited with a unique character and enclosed in braces (`{...}`) in the same manner as for an attribute entry for a character string data type. For non-character string data types the value is not enclosed in braces (the braces are not necessary because there can only be one element). The format for `CDF_EPOCH` values is described in Section 2.2.10.

The vAttribute rEntries are optional. If omitted, the terminating period is still required. The rVariable values are also optional.

Several sample rVariable definitions for a CDF with 2-dimensional rVariables follow:

```
! Variable            Data       Number     Record     Dimension
! Name                Type       Elements   Variance   Variances
! --------            ----       --------   --------   ---------

  "Latitude"        CDF_REAL4       1          F          F T

  ! Attribute         Data
  ! Name              Type       Value
  ! --------          ----       -----

    "VALIDMIN"      CDF_REAL4    { -90.0 }
    "VALIDMAX"      CDF_REAL4    { 90.0 }
    "scale"         CDF_REAL4    { -60.0, 60.0 }.

    [1,1] = -60.0
    [1,2] = -30.0
    [1,3] = 0.0
    [1,4] = 30.0
    [1,5] = 60.0


! Variable            Data       Number     Record     Dimension
! Name                Type       Elements   Variance   Variances
! --------            ----       --------   --------   ---------

  "EPOCH"           CDF_EPOCH       1          T          F F

  ! Attribute         Data
  ! Name              Type       Value
  ! --------          ----       -----

    "scale"         CDF_EPOCH    { 10-Oct-1991 00:00:00.000,
                                   20-Oct-1991 23:59:59.999 }.


! Variable            Data       Number     Record     Dimension
! Name                Type       Elements   Variance   Variances
! --------            ----       --------   --------   ---------

  'Tmp'             CDF_INT2        1          T          T T

  ! Attribute         Data
  ! Name              Type       Value
  ! --------          ----       -----

    'Fieldname'     CDF_CHAR     { "Temperature (C)" }.
```

```
! Variable           Data       Number     Record     Dimension
! Name               Type      Elements   Variance   Variances
! --------           ----      --------   --------   ---------

  "Pres_lvl"       CDF_REAL4       1          T          F F

  ! Attribute        Data
  ! Name             Type       Value
  ! --------         ----       -----

    .   ! no attribute entries

    1:[1,1] = 1013.1
    2:[1,1] = 1015.0
    3:[1,1] = 1012.3
```

A sample variable definition for a CDF with 0-dimensional rVariables follows:

```
! Variable           Data       Number     Record     Dimension
! Name               Type      Elements   Variance   Variances
! --------           ----      --------   --------   ---------

  "Latitude"       CDF_REAL4       1          F

  ! Attribute        Data
  ! Name             Type       Value
  ! --------         ----       -----

    "VALIDMIN"     CDF_REAL4    { -90.0 }
    "VALIDMAX"     CDF_REAL4    { 90.0 }.

    [] = -12.3
```

## A.6 zVariables Section

The optional zVariables section contains the definition of each zVariable in the CDF, the values for any vAttribute zEntries associated with each zVariable, and (optionally) data values for those zVariables. The format of the zVariables section is as follows:

```
#zVariables

[<variable-definition>
 <variable-definition>
 <variable-definition>
           .
```

```
          .
          .
 <variable-definition>]
```

Where `<variable-definition>` is a zVariable definition. The format of each zVariable definition is as follows:

```
! Variable        Data       Number                      Record    Dimension
! Name            Type       Elements   Dims   Sizes   Variance   Variances
! --------        ----       --------   ----   -----   --------   ---------

  <var-name> <var-data-type> <n-elems> <dims> <sizes> <rec-vary> <dim-varys>

  !  Attribute           Data
  !  Name                Type           Value
  !  ---------           ----           -----

     [<attr-name>  <entry-data-type>    <entry-value>
      <attr-name>  <entry-data-type>    <entry-value>
      <attr-name>  <entry-data-type>    <entry-value>
           .               .                  .
           .               .                  .
           .               .                  .
      <attr-name>  <entry-data-type>    <entry-value>].         ! Note the "."

   [[<rec-num>:]<indices> = <value>
    [<rec-num>:]<indices> = <value>
    [<rec-num>:]<indices> = <value>
        .            .
        .            .
        .            .
    [<rec-num>:]<indices> = <value>]
```

Each field is defined as follows:

| | |
|---|---|
| `<var-name>` | The name of the zVariable. The name must be delimited with a character not appearing in the name itself (e.g., `"EPOCH"` or `'Temperature'`). The delimiting characters are not part of the zVariable name in the CDF. |
| `<var-data-type>` | The data type for the zVariable. The data type must be one of the following: `CDF_BYTE`, `CDF_INT1`, `CDF_UINT1`, `CDF_INT2`, `CDF_UINT2`, `CDF_INT4`, `CDF_UINT4`, `CDF_REAL4`, `CDF_FLOAT`, `CDF_REAL8`, `CDF_DOUBLE`, `CDF_EPOCH`, `CDF_CHAR`, or `CDF_UCHAR`. |
| `<n-elems>` | The number of elements of the data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`) this is the number of characters in each string. For non-character string data types this value must be one (1). |
| `<dims>` | The number of dimensions for the zVariable. |

`<sizes>` — The dimension sizes — one value per dimension. If the zVariable has zero (0) dimensions, this field would be left blank.

`<rec-vary>` — The record variance of the zVariable. This must be either `T` (the values vary from record to record) or `F` (the values do not vary from record to record).

`<dim-varys>` — The dimension variances of the zVariable. For each dimension there must be either a `T` (the values vary along that dimension) or `F` (the values do not vary along that dimension). Each dimension variance must be separated by at least one space. If the zVariable has zero dimensions, this field would be left blank.

`<attr-name>` — The name of the vAttribute for which to specify a zEntry for this zVariable. The vAttribute must have been specified in the vAttributes section. The name must be delimited with a character not appearing in the name itself (e.g., `"SCALEMAX"` or `'range'`). The delimiting characters are not part of the vAttribute name in the CDF.

`<entry-data-type>` — The data type for the vAttribute zEntry. The data type must be one of the following: `CDF_BYTE`, `CDF_INT1`, `CDF_UINT1`, `CDF_INT2`, `CDF_UINT2`, `CDF_INT4`, `CDF_UINT4`, `CDF_REAL4`, `CDF_FLOAT`, `CDF_REAL8`, `CDF_DOUBLE`, `CDF_EPOCH`, `CDF_CHAR`, or `CDF_UCHAR`.

`<entry-value>` — The value(s) for the vAttribute zEntry. The format of attribute entry values is described in Section A.3.

**NOTE:** The last zEntry MUST be followed by a period (`.`). If no zEntries are specified for a zVariable, the period must still be present.

`<rec-num>` — The record number of an zVariable value. This will be present only for record-variant (RV) zVariables.

`<indices>` — The indices of an zVariable value. The indices are enclosed in brackets and separated by commas (e.g., `[23,1]` or `[1,80]`). If the zVariable has zero dimensions, `[]` would be specified (the brackets are still required).

`<value>` — The value at the given record/indices. For character string data types (`CDF_CHAR` or `CDF_UCHAR`) the string must be delimited with a unique character and enclosed in braces (`{...}`) in the same manner as for an attribute entry for a character string data type. For non-character string data types the value is not enclosed in braces (the braces are not necessary because there can only be one element). The format for `CDF_EPOCH` values is described in Section 2.2.10.

The vAttribute zEntries are optional. If omitted, the terminating period is still required. The zVariables values are also optional.

Several sample zVariable definitions follow:

```
! Variable          Data      Number                       Record   Dimension
! Name              Type     Elements  Dims  Sizes  Variance  Variances
! --------          ----     --------  ----  -----  --------  ---------
```

```
  "Instrument"    CDF_CHAR      10       0              F

  ! Attribute       Data
  ! Name            Type       Value
  ! --------        ----       -----

    "FIELDNAM"    CDF_CHAR     { "Measuring instrument" }.

  [] = { "Gonkulator" }


! Variable          Data      Number                    Record   Dimension
! Name              Type     Elements  Dims  Sizes   Variance  Variances
! --------          ----     --------  ----  -----   --------  ---------

  "Ticks"         CDF_BYTE       1       1     3        T         T

  ! Attribute       Data
  ! Name            Type       Value
  ! --------        ----       -----

  .    ! no attribute entries

  1:[1] = 1
  1:[2] = 2
  1:[3] = 3
  2:[1] = 3
  2:[2] = 2
  2:[3] = 1


! Variable           Data      Number                        Record    Dimension
! Name               Type     Elements  Dims    Sizes      Variance  Variances
! --------           ----     --------  ----  ----------   --------  ---------

  "WIND VELOCITY"  CDF_REAL4      1       3   360 180 10      T       T T T

  ! Attribute       Data
  ! Name            Type       Value
  ! --------        ----       -----

    "FIELDNAM"    CDF_CHAR     { "Wind velocity." }
    "VALIDMIN"    CDF_REAL4    { 0.0 }
    "VALIDMAX"    CDF_REAL4    { 300.0 }
    "UNITS   "    CDF_CHAR     { "Knots" }
    "FORMAT  "    CDF_CHAR     { "F9.1    " }.
```

## A.7 End Section

This section simply consists of the keyword `#end`. This section is required.

## A.8 Example Skeleton Table

An example skeleton table containing rVariables and zVariables follows:

```
! Skeleton table for the "example2" CDF.
! Generated: Thursday, 17-Nov-1994 14:07:58
! CDF created/modified by CDF V2.4.10
! Skeleton table created by CDF V2.5.0


#header

                 CDF NAME: example2
            DATA ENCODING: NETWORK
                 MAJORITY: ROW
                   FORMAT: SINGLE

! Variables G.Attributes V.Attributes Records  Dims  Sizes
! --------- ------------ ------------ -------  ----  -----
      4           1            7          1       2   11 7


#GLOBALattributes

! Attribute        Entry       Data
! Name             Number      Type        Value
! ---------        ------      ----        -----

  "TITLE"              1:    CDF_CHAR     { "Title for example2 CDF." } .


#VARIABLEattributes

  "FIELDNAM"
  "VALIDMIN"
  "VALIDMAX"
  "SCALEMIN"
  "SCALEMAX"
  "UNITS"
  "FORMAT"


#variables
```

```
! Variable           Data       Number     Record     Dimension
! Name               Type      Elements   Variance   Variances
! --------           ----      --------   --------   ---------

  "EPOCH"          CDF_EPOCH      1          T          F F

  ! Attribute        Data
  ! Name             Type       Value
  ! --------         ----       -----

    "FIELDNAM"     CDF_CHAR    { "Time since 0 A.D.   " }
    "VALIDMIN"     CDF_EPOCH   { 01-Jan-0000 00:00:00.000 }
    "VALIDMAX"     CDF_EPOCH   { 01-Jan-2089 00:00:00.000 }
    "SCALEMIN"     CDF_EPOCH   { 01-Apr-1986 07:00:00.000 }
    "SCALEMAX"     CDF_EPOCH   { 01-Apr-1986 23:00:00.000 }
    "UNITS"        CDF_CHAR    { "milliseconds (UT)   " }
    "FORMAT"       CDF_CHAR    { "E14.0    " } .

! Variable           Data       Number     Record     Dimension
! Name               Type      Elements   Variance   Variances
! --------           ----      --------   --------   ---------

  "LONGITUD"       CDF_REAL4      1          F          T F

  ! Attribute        Data
  ! Name             Type       Value
  ! --------         ----       -----

    "FIELDNAM"     CDF_CHAR    { "Longitude variable  " }
    "VALIDMIN"     CDF_REAL4   { 0.0 }
    "VALIDMAX"     CDF_REAL4   { 180.0 }
    "SCALEMIN"     CDF_REAL4   { -50.0 }
    "SCALEMAX"     CDF_REAL4   { 50.0 }
    "UNITS"        CDF_CHAR    { "Degrees             " }
    "FORMAT"       CDF_CHAR    { "F8.3    " } .

    [1,1] = -50.0
    [2,1] = -40.0
    [3,1] = -30.0
    [4,1] = -20.0
    [5,1] = -10.0
    [6,1] = 0.0
    [7,1] = 10.0
    [8,1] = 20.0
    [9,1] = 30.0
    [10,1] = 40.0
    [11,1] = 50.0

! Variable           Data       Number     Record     Dimension
! Name               Type      Elements   Variance   Variances
```

```
! --------           ----      --------   --------   ---------

  "LATITUDE"        CDF_REAL4       1          F         F T

  ! Attribute        Data
  ! Name             Type       Value
  ! --------         ----       -----

    "FIELDNAM"      CDF_CHAR     { "Latitude variable   " }
    "VALIDMIN"      CDF_REAL4    { 0.0 }
    "VALIDMAX"      CDF_REAL4    { 90.0 }
    "SCALEMIN"      CDF_REAL4    { -30.0 }
    "SCALEMAX"      CDF_REAL4    { 30.0 }
    "UNITS"         CDF_CHAR     { "Degrees             " }
    "FORMAT"        CDF_CHAR     { "F8.3    " } .

    [1,1] = -30.0
    [1,2] = -20.0
    [1,3] = -10.0
    [1,4] = 0.0
    [1,5] = 10.0
    [1,6] = 20.0
    [1,7] = 30.0

! Variable           Data       Number     Record    Dimension
! Name               Type      Elements   Variance   Variances
! --------           ----      --------   --------   ---------

  "TEMPERATURE"     CDF_INT4        1          T         T T

  ! Attribute        Data
  ! Name             Type       Value
  ! --------         ----       -----

    "FIELDNAM"      CDF_CHAR     { "Temperature         " }
    "VALIDMIN"      CDF_INT4     { 0 }
    "VALIDMAX"      CDF_INT4     { 50 }
    "SCALEMIN"      CDF_INT4     { 0 }
    "SCALEMAX"      CDF_INT4     { 10 }
    "UNITS"         CDF_CHAR     { "Deg C               " }
    "FORMAT"        CDF_CHAR     { "I2      " } .


#zVariables

! Variable          Data      Number                 Record   Dimension
! Name              Type     Elements  Dims  Sizes  Variance  Variances
! --------          ----     --------  ----  -----  --------  ---------

  "BIAS"          CDF_INT4       1       0              T
```

```
  ! Attribute       Data
  ! Name            Type       Value
  ! --------        ----       -----

    "FIELDNAM"    CDF_CHAR     { "Correction bias for temperature" }
    "VALIDMIN"    CDF_INT4     { -5 }
    "VALIDMAX"    CDF_INT4     { 5 }
    "UNITS   "    CDF_CHAR     { "deg C               " }
    "FORMAT  "    CDF_CHAR     { "I2      " } .

    1:[] = 34
    2:[] = 28
    3:[] = 17


! Variable          Data      Number                 Record   Dimension
! Name              Type     Elements  Dims  Sizes  Variance  Variances
! --------          ----     --------  ----  -----  --------  ---------

  "Coefficients"  CDF_REAL4      1       1     3        F         T

  ! Attribute       Data
  ! Name            Type       Value
  ! --------        ----       -----

    "FIELDNAM"    CDF_CHAR     { "Temperature model coefficients." }
    "FORMAT  "    CDF_CHAR     { "F9.1    " } .

  [1] = -0.0254
  [2] = 14.2338
  [3] = -9.9444


! Variable          Data      Number                 Record   Dimension
! Name              Type     Elements  Dims  Sizes  Variance  Variances
! --------          ----     --------  ---- ------- --------  ---------

  "TMP-model"     CDF_REAL4      1       2  360 180     T        T T

  ! Attribute       Data
  ! Name            Type       Value
  ! --------        ----       -----

    "FIELDNAM"    CDF_CHAR     { "Temperature model." }
    "VALIDMIN"    CDF_REAL4    { -20.0 }
    "VALIDMAX"    CDF_REAL4    { 50.0 }
    "SCALEMIN"    CDF_REAL4    { 0.0 }
    "SCALEMAX"    CDF_REAL4    { 30.0 }
    "UNITS   "    CDF_CHAR     { "deg C               " }
    "FORMAT  "    CDF_CHAR     { "F9.6    " } .
```

```
#end
```

# Appendix B

# IDL Support

## B.1 Introduction

The CDF distribution includes a set of IDL functions/procedures that allow access to the CDF library (hereafter referred to as the CDF/IDL interface). The functions/procedures are used in the same manner as the Standard and Internal Interfaces are used in programs written in C. CDFs can be both read and written/modified.

**NOTE:** There is, however, one difference when using the Internal Interface (`CDFlib`). The current CDF must be selected each time `CDFlib` is called. This is because of how IDL does the dynamic mapping to the CDF library at run-time. Failure to select the current CDF may result in IDL crashing.

Research Systems, Inc. (the developers of IDL) have also implemented an interface to CDF that is part of the IDL product. It differs from the interface provided with the CDF distribution in that it is intended more for the non-programmer (and is functionally similar to other interfaces they provide). Which interface you choose to use will depend on your personal preferences.

CDF's interface relies on IDL's support of dynamic linking. IDL in turn relies on the operating system being used to support dynamic linking in a reasonable way. Currently, only the VAX (VMS), DEC Alpha (OpenVMS & OSF/1), Sun (SunOS & SOLARIS), HP (HP-UX), and IBM RS6000 (AIX) support dynamic linking in a way that makes it possible to use CDF's interface to IDL. Because IDL's CDF interface is built into the IDL executable, it does not rely on dynamic linking and is therefore available on every machine on which IDL is supported.

## B.2 Using CDF's IDL Interface

This section assumes that you are using the "definitions" file provided with the CDF distribution (and that it has been properly configured and named by your system manager). The "definitions" file is a command file named `DEFINITIONS.COM` on VMS systems and a script file named `definitions.<shell>` on UNIX systems (where `<shell>` is the shell being used). The "definitions" file defines the logical name `IDL_STARTUP` on VMS systems and the environment variable `IDL_STARTUP` on UNIX systems to the full directory path of an

IDL startup file provided with the CDF distribution. When IDL is started, this startup file is automatically executed (because `IDL_STARTUP` is defined). The startup file performs the following functions...

1. Makes known to IDL the functions/procedures that are used to call the CDF library (and perform other utility operations).
2. Sets IDL's internal paths. These are used when searching for a batch file to be executed or when locating online help files.
3. Executes a batch file (`cdf.pro`) that creates a number of local variables necessary for calling CDF functions/procedures and then interpreting the results.

A list of messages should be displayed indicating the CDF functions/procedures that have been compiled.

If you are already using your own startup file (pointed to by your definition of `IDL_STARTUP`), you should remove the definition of `IDL_STARTUP` from the CDF "definitions" file being used and do one of the following...

1. Add the contents of the CDF/IDL startup file to your own startup file.
2. Have your startup file execute the CDF/IDL startup file.

The definition of `IDL_STARTUP` in the CDF "definitions" file being used points to the location of the CDF/IDL startup file.

## B.3 CDF "Include" Files

A number of parameters (macro constants) are necessary when calling functions in the CDF library from a C program. These parameters are defined in `cdf.h`. When calling the CDF functions/procedures in IDL, local variables are used to accomplish the same thing. An IDL batch[1] file, `cdf.pro`, is provided that creates/initializes the necessary local variables. `cdf.pro` is executed automatically by the startup file when you run IDL. All necessary CDF parameters will be available including the status codes commonly used (`CDF_OK` and `CDF_WARN`). Most users will not need the other status codes since the `CDFerror` function can be used to determine the explanation of a status code. If you do require the other status codes, execute the batch file `cdf1.pro` (the IDL internal path will know where to find it). If you are using the Internal Interface (`CDFlib`), you will also need the parameters (local variables) defined in `cdf2.pro`.

IDL limits the number of local variables that may be defined in any one function/procedure. This limit might be reached if you have to include one or more of the CDF "include" files. If that happens you will have to use one or more of the "mapping" functions provided with the CDF/IDL interface. The use of these "mapping" functions eliminates the need to include the corresponding 'include' file.

| Include File | Mapping Function | Purpose |
|---|---|---|
| cdf.pro | mcp.pro | Map CDF parameters |
| cdf1.pro | msc.pro | Map status codes |
| cdf2.pro | mii.pro | Map Internal Interface |

Each function returns the longword value that corresponds to the parameter passed in as a character string. For example, the status code value for `NO_SUCH_CDF` would be retrieved as follows...

[1] These are also known as include files.

```
IDL> status = CDFopen ('test1', id)
IDL> if (status eq MSC('NO_SUCH_CDF')) ...
```

In this case the batch file `cdf1.pro` would not have to be included.

## B.4 On-Line Help

IDL provides an on-line help facility to which help for user written procedures/functions can be added. A specially formatted file is provided with the CDF distribution that contains on-line help for the procedures/functions in the CDF/IDL interface. This file is named `cdf25cdfif.help` and is located in the "help" directory of the CDF distribution. If you are using the appropriate "definitions" file, IDL will be able to locate this file when online help is requested. Choose the topic called `CDF25CDFIF` rather than `CDF`. (`CDF` provides online help for IDL's CDF interface — not the interface provided with the CDF distribution.)

## B.5 Available Functions

The CDF functions/procedures available at the IDL command line exactly mirror the Standard and Internal Interface functions available for a program written in C.[2] The CDF C Reference Manual describes the functionality of each in detail. The syntax required when calling each function is as follows...

```
IDL> status = CDFlib (op1, ...)
IDL> status = CDFcreate (CDFpath, numDims, dimSizes, encoding, majority, $
IDL>                     id)
IDL> status = CDFopen (CDFpath, id)
IDL> status = CDFdoc (id, version, release, copyright)
IDL> status = CDFinquire (id, numDims, dimSizes, encoding, majority, $
IDL>                      maxRec, numVars, numAttrs)
IDL> status = CDFclose (id)
IDL> status = CDFdelete (id)
IDL> status = CDFerror (status, text)
IDL> status = CDFattrCreate (id, attrName, attrScope, attrNum)
IDL> status = CDFattrRename (id, attrNum, attrName)
IDL> status = CDFattrInquire (id, attrNum, attrName, attrScope, maxEntry)
IDL> status = CDFattrEntryInquire (id, attrNum, entryNum, dataType, $
IDL>                               numElems)
IDL> status = CDFattrPut (id, attrNum, entryNum, dataType, numElems, value)
IDL> status = CDFattrGet (id, attrNum, entryNum, value)
IDL> status = CDFvarCreate (id, varName, dataType, numElems, recVary, $
IDL>                        dimVarys, varNum)
IDL> status = CDFvarRename (id, varNum, varName)
```

[2] With the exception of the multiple variable access operations available via the Internal Interface (<GET_,rVARs_RECDATA_>, <PUT_,rVARs_RECDATA_>, <GET_,zVARs_RECDATA_>, and <PUT_,zVARs_RECDATA_>). Because these operations deal with variables having different data types, the use of structures to store the values would be the logical method to use. Unfortunately, IDL does not currently support the manipulation of structures from within a C application (which is how CDF's IDL interface is implemented). Support of these operations will hopefully be added in the future.

```
IDL> status = CDFvarInquire (id, varNum, varName, dataType, numElems, $
IDL>                         recVary, dimVarys)
IDL> status = CDFvarPut (id, varNum, recNum, indices, value)
IDL> status = CDFvarGet (id, varNum, recNum, indices, value)
IDL> status = CDFvarHyperPut (id, varNum, recStart, recCount, recInterval, $
IDL>                          indices, counts, intervals, buffer)
IDL> status = CDFvarHyperGet (id, varNum, recStart, recCount, recInterval, $
IDL>                          indices, counts, intervals, buffer)
IDL> status = CDFvarClose (id, varNum)
IDL> attrNum = CDFattrNum (id, attrName)
IDL> varNum = CDFvarNum (id, varName)
IDL> status = CDFlib (op1, ...)
IDL> epoch = computeEPOCH (year, month, day, hour, minute, second, $
IDL>                       millisecond)
IDL> EPOCHbreakdown, epoch, year, month, day, hour, minute, second, $
IDL>                 millisecond
IDL> epoch = parseEPOCH (epochString)
IDL> encodeEPOCH, epoch, epochString
IDL> encodeEPOCH1, epoch, epochString
IDL> encodeEPOCH2, epoch, epochString
IDL> encodeEPOCH3, epoch, epochString
```

The IDL data types to use for integer arguments are always longwords or arrays of longwords using IDL's data type of `TYP_LONG`.[3] Attribute entries and variable values should have the data types corresponding their CDF data types. CDF pathnames and variable/attribute names are IDL character strings. `CDFid` and `CDFstatus` are also longwords. The CDF C Reference Manual may be used as a guide (especially for `CDFlib`). Arguments passed to the CDF functions/procedures must obviously be created and initialized before the call. Arrays of longwords are necessary for dimension sizes, variances, etc.[4] Arguments returned by the CDF functions will be created to be of the proper data type and size (they do not have to be created before calling the CDF function).

**NOTE:** There is one difference between how the Internal Interface (`CDFlib`) is used in IDL and in a C/Fortran application. In IDL the current CDF must be selected each time `CDFlib` is called. Failure to select the current CDF may result in IDL crashing.

Also provided are two functions, `row_to_col` and `col_to_row`, that may be used to change the majority of an array (or arrays) of values. This function is necessary because CDF can store variable values in either row or column major order. The syntax of `row_to_col` is as follows...

```
IDL> status = row_to_col (inArray, outArray, numDims, dimSizes, numBytes, $
                          arrayCount
```

where...

[3] On DEC Alpha's running OSF/1, don't confuse the C data type of `long` with IDL's data type of `TYP_LONG`. The C `long` is eight bytes while IDL's `TYP_LONG` is four bytes.

[4] If an array in a call to a CDF function won't be used because a variable is 0-dimensional, a dummy array must still be passed to avoid an error from the CDF library. Note that embedding a call to `lonarr` [such as `lonarr(1)`] in the call to the CDF function will work but will result in a memory leak because the allocated array will not be freed.

| | |
|---|---|
| `inArray` | The array (or arrays) of values to convert — row major ordering. Any valid IDL data type is allowed. |
| `outArray` | The converted array (or arrays) of values — column major ordering. This array will be created by `row_to_col`. |
| `numDims` | The number of dimensions in the array (or arrays). This value must be a longword. |
| `dimSizes` | The size of each dimension. This must be an array of longwords — one longword per dimension. |
| `numBytes` | The number of bytes in each value. This value must be a longword. |
| `arrayCount` | The number of arrays to convert. The order of the arrays will not be affected. This value must be a longword. |
| `status` | Completion status code. This value will be a longword. The possible values are defined in `cdf1.pro`. |

Multiple arrays (records) of CDF variable values should always be stored with the last dimension of the IDL array corresponding to the record number. For example, assume a CDF variable with two dimensions whose sizes are `[180,360]`. If `50` records are to be stored, an IDL array with three dimensions whose sizes are `[180,360,50]` should be used. If the CDF variables have row major ordering, `row_to_col` would be called with `numDims` set to `2`, `dimSizes` set to `[180,360]`, and `arrayCount` set to `50`.

The syntax of `col_to_row` is as follows...

```
IDL> status = col_to_row (inArray, outArray, numDims, dimSizes, numBytes, $
                          arrayCount
```

where...

| | |
|---|---|
| `inArray` | The array (or arrays) of values to convert — column major ordering. Any valid IDL data type is allowed. |
| `outArray` | The converted array (or arrays) of values — row major ordering. This array will be created by `col_to_row`. |
| `numDims` | The number of dimensions in the array (or arrays). This value must be a longword. |
| `dimSizes` | The size of each dimension. This must be an array of longwords — one longword per dimension. |
| `numBytes` | The number of bytes in each value. This value must be a longword. |
| `arrayCount` | The number of arrays to convert. The order of the arrays will not be affected. This value must be a longword. |
| `status` | Completion status code. This value will be a longword. The possible values are defined in `cdf1.pro`. |

`col_to_row` would be used in those cases where an array of values created by IDL (in column-major ordering) are to be written to a row-major CDF.

## B.6 Example IDL Session

The following IDL session will open a CDF (unsuccessfully at first — typo), inquire about a variable named `LATITUDE`, and then close the CDF.

```
.
.
IDL> status=CDFopen('yrdy3',id)
IDL> status=CDFerror(status,text)
IDL> print,text
NO_SUCH_CDF: The specified CDF does not exist.
IDL> status=CDFopen('test2',id)
IDL> status=CDFerror(status,text)
IDL> print,text
CDF_OK: Function completed successfully.
IDL> status=CDFvarInquire(id,CDFvarNum(id,'LATITUDE'),varName,dataType,$
IDL> numElems,recVary,dimVarys)
IDL> status=CDFerror(status,text)
IDL> print,text
CDF_OK: Function completed successfully.
IDL> print,dataType,numElems
          21           1
IDL> status=CDFclose(id)
IDL> status=CDFerror(status,text)
IDL> print,text
CDF_OK: Function completed successfully.
.
.
```

As you can see, checking the return status from each call to the CDF library can be fairly tedious. A procedure such as the following could be used to ease status code checking...

```
pro checkstatus, status
@cdf
if (status lt CDF_OK) then begin
  statusT = CDFerror (status, text)
  print, text
endif
return
end
```

The sample IDL session could then be as follows...

```
.
.
IDL> status=CDFopen('yrdy3',id)
IDL> checkstatus,status
NO_SUCH_CDF: The specified CDF does not exist.
IDL> checkstatus,CDFopen('test2',id)
CDF_OK: Function completed successfully.
IDL> status=CDFvarInquire(id,CDFvarNum(id,'LATITUDE'),varName,dataType,$
IDL> numElems,recVary,dimVarys)
IDL> checkstatus,status
CDF_OK: Function completed successfully.
IDL> print,dataType,numElems
          21           1
IDL> checkstatus,CDFclose(id)
CDF_OK: Function completed successfully.
.
.
```

As you can see, a call to a CDF function can be embedded in a call to `checkstatus`.

## B.7 Example IDL Function

The following example function is also available in the "samples" subdirectory of the CDF distribution in a file named `idlload1.pro`.

```
;
;
; This function will load all data for a specified rVariable from a CDF into
; an IDL variable.  All of the IDL/CDF interface routines may be entered at
; the IDL command line but for demonstration purposes they will be used in
; an IDL function.
;
; Argument descriptions:                             data type
; CDF_name      input CDF name                       string
; var_name      variable name in CDF                 string
; data          array to be returned to IDL          (see creation below)
;
; Executing
; status = get_var('SAMPLE_CDF','VARNAME',data)
;
function get_var, CDF_name, var_name, data
;
; include all CDF definitions
;
@cdf
;
; Open the CDF, note that CDF_id will be returned as the proper data type
```

```
;
status = CDFopen(CDF_name,CDF_id)
;
; check the status of the open, if a error is detected print a error and
; return
;
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; Inquire about the structure of the CDF
;
status = CDFinquire(CDF_id,num_dims,dim_sizes,encoding,majority,$
                    max_record_num,num_vars,num_attrs)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; Get the number in the CDF of the variable
;
var_num = CDFvarNum(CDF_id, var_name)
if(var_num lt CDF_WARN)then begin
   status = CDFerror(var_num, error_mes)
   print,error_mes
   return,status
endif
;
; Inquire about the variable
;
status = CDFvarInquire(CDF_id, var_num, out_var_name, data_type, $
                       num_bytes, record_variance, dim_variances)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; We will retrieve some of the metadata from this variable.
; FIELDNAM is a CDF standard attribute and is a description of the
; variable.
;
attr_num = CDFattrNum(CDF_id, "FIELDNAM")
if(attr_num lt CDF_WARN)then begin
   status = CDFerror(attr_num, error_mes)
   print,error_mes
   return,status
endif
```

```
status = CDFattrInquire(CDF_id, attr_num, out_attr_name, $
                        attr_scope, max_entry)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; Need the data type and size of this attribute
;
status = CDFattrEntryInquire(CDF_id, attr_num, var_num, $
                             attr_data_type, num_elements)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
status = CDFattrGet(CDF_id, attr_num, var_num, attr_value)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
print,'Loading variable ',var_name,': ',attr_value
;
; We will now get ready to load all data for this variable,
; use the CDF structure to initialize all parameters to perform this
; operation
;
recStart = 0l
recCount = max_record_num+1
recInterval=1l
indices = lonarr(num_dims)
counts = lonarr(num_dims)
intervals = lonarr(num_dims)
for dim_num=0, num_dims-1 do begin
    indices(dim_num) = 0;
    counts(dim_num) = dim_sizes(dim_num)
    intervals(dim_num) = 1
endfor
status = CDFvarHyperGet(CDF_id,var_num, recStart,recCount,recInterval, $
                        indices,counts,intervals,inbuf)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; IDL needs to have the CDF in column majority in order to plot properly,
; we will supply a routine that will convert from ROW to COLUMN major.
; Converting character data is not supported in this release
```

```
;
if((majority eq ROW_MAJOR) and $
   (data_type ne CDF_CHAR and data_type ne CDF_UCHAR))then begin
   print,'Converting from ROW to COLUMN major'
;
; Need to know the number of bytes of this data type (e.g. 1,2,4,8)
;
   nbytes,data_type,num_bytes_dt
;
; Now convert
;
   status = row_to_col (inbuf, data, num_dims, dim_sizes, num_bytes_dt, $
                        recCount)
endif else begin
;
; No conversion necessary, load input buffer into return data buffer
;
   data = inbuf
endelse
status=CDFclose(CDF_id)
if (status lt CDF_WARN) then begin
  st = CDFerror (status, error_mes)
  print, error_mes
  return, status
endif
return,CDF_OK
end

pro nbytes,data_type,num_bytes
@cdf
num_bytes = 0l
Case 1 of
 (data_type eq CDF_INT1)   : num_bytes = 1l
 (data_type eq CDF_INT2)   : num_bytes = 2l
 (data_type eq CDF_INT4)   : num_bytes = 4l
 (data_type eq CDF_UINT1)  : num_bytes = 1l
 (data_type eq CDF_UINT2)  : num_bytes = 2l
 (data_type eq CDF_UINT4)  : num_bytes = 4l
 (data_type eq CDF_REAL4)  : num_bytes = 4l
 (data_type eq CDF_REAL8)  : num_bytes = 8l
 (data_type eq CDF_EPOCH)  : num_bytes = 8l
 (data_type eq CDF_BYTE)   : num_bytes = 1l
 (data_type eq CDF_FLOAT)  : num_bytes = 4l
 (data_type eq CDF_DOUBLE) : num_bytes = 8l
 (data_type eq CDF_CHAR)   : num_bytes = 1l
 (data_type eq CDF_UCHAR)  : num_bytes = 1l
endcase
return
end
```

# Appendix C

# Status Codes

## C.1 Introduction

A status code is returned from most CDF functions. The `cdf.h` (for C) and `CDF.INC` (for Fortran) include files contain the numerical values (constants) for each of the status codes (and for any other constants referred to in the explanations). The CDF library Standard Interface functions `CDFerror` (for C) and `CDF_error` (for Fortran) can be used within a program to inquire the explanation text for a given status code. The Internal Interface can also be used to inquire explanation text.

There are three classes of status codes: informational, warning, and error. The purpose of each is as follows:

| | |
|---|---|
| Informational | Indicates success but provides some additional information that may be of interest to an application. |
| Warning | Indicates that the function completed but possibly not as expected. |
| Error | Indicates that a fatal error occurred and the function aborted. |

Status codes fall into classes as follows:

Error codes < `CDF_WARN` < Warning codes < `CDF_OK` < Informational codes

`CDF_OK` indicates an unqualified success (it should be the most commonly returned status code). `CDF_WARN` is simply used to distinguish between warning and error status codes.

## C.2 Status Codes and Messages

The following list contains an explanation for each possible status code. Whether a particular status code is considered informational, a warning, or an error is also indicated.

`ATTR_EXISTS` Named attribute already exists — cannot create or rename. Each attribute in a CDF must have a unique name. Note that trailing blanks are ignored by the CDF library when comparing attribute names. [Error]

`ATTR_NAME_TRUNC` Attribute name truncated to `CDF_ATTR_NAME_LEN` characters. The attribute was created but with a truncated name. [Warning]

`BAD_ALLOCATE_RECS` An illegal number of records to allocate for a variable was specified. For RV variables the number must be one or greater. For NRV variables the number must be exactly one. [Error]

`BAD_ARGUMENT` An illegal/undefined argument was passed. Check that all arguments are properly declared and initialized. [Error]

`BAD_ATTR_NAME` Illegal attribute name specified. Attribute names must contain at least one character, and each character must be printable. [Error]

`BAD_ATTR_NUM` Illegal attribute number specified. Attribute numbers must be zero (0) or greater for C applications and one (1) or greater for Fortran applications. [Error]

`BAD_CACHESIZE` An illegal number of cache buffers was specified. The value must be at least one (1). [Error]

`BAD_CDF_EXTENSION` An illegal file extension was specified for a CDF. In general, do not specify an extension except possibly for a single-file CDF which has been renamed with a different file extension or no file extension. [Error]

`BAD_CDF_ID` CDF identifier is unknown or invalid. The CDF identifier specified is not for a currently open CDF. [Error]

`BAD_CDF_NAME` Illegal CDF name specified. CDF names must contain at least one character, and each character must be printable. Trailing blanks are allowed but will be ignored. [Error]

`BAD_CDFSTATUS` Unknown CDF status code received. The status code specified is not used by the CDF library. [Error]

`BAD_DATA_TYPE` Unknown data type specified. The CDF data types are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]

`BAD_DECODING` Unknown decoding specified. The CDF decodings are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]

`BAD_DIM_COUNT` Illegal dimension count specified. A dimension count must be at least one (1) and not greater than the size of the dimension. [Error]

`BAD_DIM_INDEX` One or more dimension index is out of range. A valid value must be specified regardless of the dimension variance. Note also that the combination of dimension index, count, and in-

| | |
|---|---|
| | terval must not specify an element beyond the end of the dimension. [Error] |
| BAD_DIM_INTERVAL | Illegal dimension interval specified. Dimension intervals must be at least one (1). [Error] |
| BAD_DIM_SIZE | Illegal dimension size specified. A dimension size must be at least one (1). [Error] |
| BAD_ENCODING | Unknown data encoding specified. The CDF encodings are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error] |
| BAD_ENTRY_NUM | Illegal attribute entry number specified. Entry numbers must be at least zero (0) for C applications and at least one (1) for Fortran applications. [Error] |
| BAD_EXTEND_RECS | An illegal number of extend records has been specified. The number of extend records must be at least zero (0). A value of zero indicates that the default should be used. Note that extend records are only significant for variables that are not NRV and are in single-file CDFs. [Error] |
| BAD_FNC_OR_ITEM | The specified function or item is illegal. Check that the proper number of arguments are specified for each operation being performed. Also make sure that `NULL_` is specified as the last operation. [Error] |
| BAD_FORMAT | Unknown format specified. The CDF formats are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error] |
| BAD_INITIAL_RECS | An illegal number of records to initially write has been specified. The number of initial records must be at least one (1). [Error] |
| BAD_MAJORITY | Unknown variable majority specified. The CDF variable majorities are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error] |
| BAD_MALLOC | Unable to allocate dynamic memory — system limit reached. Contact CDF User Support if this error occurs. [Error] |
| BAD_NEGtoPOSfp0_MODE | An illegal `-0.0` to `0.0` mode was specified. The `-0.0` to `0.0` modes are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error] |
| BAD_NUM_DIMS | The number of dimensions specified is out of the allowed range. Zero (0) through `CDF_MAX_DIMS` dimensions are allowed. If more are needed, contact CDF User Support. [Error] |
| BAD_NUM_ELEMS | The number of elements of the data type is illegal. The number of elements must be at least one (1). For non-character string variables, the number of elements must always be one (1). [Error] |

| | |
|---|---|
| `BAD_NUM_VARS` | Illegal number of variables in a record access operation. [Error] |
| `BAD_READONLY_MODE` | Illegal read-only mode specified. The CDF read-only modes are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error] |
| `BAD_REC_COUNT` | Illegal record count specified. A record count must be at least one (1). [Error] |
| `BAD_REC_INTERVAL` | Illegal record interval specified. A record interval must be at least one (1). [Error] |
| `BAD_REC_NUM` | Record number is out of range. Record numbers must be at least zero (0) for C applications and at least one (1) for Fortran applications. Note that a valid value must be specified regardless of the record variance. [Error] |
| `BAD_SCOPE` | Unknown attribute scope specified. The attribute scopes are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error] |
| `BAD_VAR_NAME` | Illegal variable name specified. Variable names must contain at least one character and each character must be printable. [Error] |
| `BAD_VAR_NUM` | Illegal variable number specified. Variable numbers must be zero (0) or greater for C applications and one (1) or greater for Fortran applications. [Error] |
| `BAD_zMODE` | Illegal zMode specified. The CDF zModes are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error] |

`CANNOT_CHANGE`

Because of dependencies on the value, it cannot be changed. Some possible causes of this error follow:

1. Changing a CDF's data encoding after a variable value (including a pad value) or an attribute entry has been written.
2. Changing a CDF's format after a variable has been created.
3. Changing a CDF's variable majority after a variable value (excluding a pad value) has been written.
4. Changing a variable's data specification after a value (including the pad value) has been written to that variable.
5. Changing a variable's record variance after a value (excluding the pad value) has been written to that variable.
6. Changing a variable's dimension variances after a value (excluding the pad value) has been written

to that variable.

7. Allocating or writing "initial" records to a variable after a value (excluding the pad value) has been written to that variable.
8. Changing an attribute entry's data specification where the new specification is not equivalent to the old specification.

[Error]

| | |
|---|---|
| `CDF_CLOSE_ERROR` | Error detected while trying to close CDF. Check that sufficient disk space exists for the `.cdf` file and that it has not been corrupted. [Error] |
| `CDF_CREATE_ERROR` | Cannot create CDF specified — error from file system. Make sure sure that sufficient privilege exists to create the `.cdf` file in the disk/directory location specified and that an open file quota has not already been reached. [Error] |
| `CDF_EXISTS` | The CDF named already exists — cannot create it. The CDF library will not overwrite an existing CDF. [Error] |
| `CDF_INTERNAL_ERROR` | An unexpected condition has occurred in the CDF library. Report this error to CDFsupport. [Error] |
| `CDF_NAME_TRUNC` | CDF pathname truncated to `CDF_PATHNAME_LEN` characters. The CDF was created but with a truncated name. [Warning] |
| `CDF_OK` | Function completed successfully. |
| `CDF_OPEN_ERROR` | Cannot open CDF specified — error from file system. Check that the `.cdf` file is not corrupted and that sufficient privilege exists to open it. Also check that an open file quota has not already been reached. [Error] |
| `CDF_READ_ERROR` | Failed to read CDF file — error from file system. Check that the `.cdf` file is not corrupted. [Error] |
| `CDF_WRITE_ERROR` | Failed to write CDF file — error from file system. Check that the `.cdf` file is not corrupted. [Error] |
| `CORRUPTED_V2_CDF` | This Version 2 CDF is corrupted. An error has been detected in the CDF's control information. If the CDF file(s) are known to be valid, please contact CDF User Support. [Error] |
| `END_OF_VAR` | The sequential access current value is at the end of the variable. Reading beyond the end of the last physical value for a variable is not allowed (when performing sequential access). [Error] |
| `FUTURE_CDF` | The CDF distribution being used is older than the CDF distribution which created/modified the CDF. In this case, there is no guarantee that the CDF will be successfully accessed. [Warning] |

| | |
|---|---|
| `IBM_PC_OVERFLOW` | An operation involving a buffer greater than 64k bytes in size has been specified. [Error] |
| `ILLEGAL_FOR_SCOPE` | The operation is illegal for the attribute's scope. For example, only gEntries may be written for gAttributes — not rEntries or zEntries. [Error] |
| `ILLEGAL_IN_zMODE` | The attempted operation is illegal while in zMode. Most operations involving rVariables or rEntries will be illegal. [Error] |
| `ILLEGAL_ON_V1_CDF` | The specified operation (i.e., opening) is not allowed on Version 1 CDFs. [Error] |
| `MULTI_FILE_FORMAT` | The specified operation is not applicable to CDFs with the multi-file format. For example, it does not make sense to inquire indexing statistics for a variable in a multi-file CDF (indexing is only used in single-file CDFs). [Informational] |
| `NEGATIVE_FP_ZERO` | One or more of the values read/written are `-0.0` (an illegal value on VAXes and DEC Alphas running OpenVMS). [Warning] |
| `NO_ATTR_SELECTED` | An attribute has not yet been selected. First select the attribute on which to perform the operation. [Error] |
| `NO_CDF_SELECTED` | A CDF has not yet been selected. First select the CDF on which to perform the operation. [Error] |
| `NO_DELETE_ACCESS` | Deleting is not allowed (read-only access). Make sure that delete access is allowed on the CDF file(s). [Error] |
| `NO_ENTRY_SELECTED` | An attribute entry has not yet been selected. First select the entry number on which to perform the operation. [Error] |
| `NO_MORE_ACCESS` | Further access to the CDF is not allowed because of a severe error. If the CDF was being modified, closing the CDF may save some of the modifications made since the CDF was opened. [Error] |
| `NO_PADVALUE_SPECIFIED` | A pad value has not yet been specified. The default pad value is currently being used for the variable. The default pad value was returned. [Informational] |
| `NO_STATUS_SELECTED` | A CDF status code has not yet been selected. First select the status code on which to perform the operation. [Error] |
| `NO_SUCH_ATTR` | The named attribute was not found. Note that attribute names are case-sensitive. [Error] |
| `NO_SUCH_CDF` | The specified CDF does not exist. Check that the pathname specified is correct. [Error] |
| `NO_SUCH_ENTRY` | No such entry for specified attribute. [Error] |
| `NO_SUCH_VAR` | The named variable was not found. Note that variable names are case-sensitive. [Error] |

| | |
|---|---|
| `NO_VAR_SELECTED` | A variable has not yet been selected. First select the variable on which to perform the operation. [Error] |
| `NO_VARS_IN_CDF` | This CDF contains no rVariables. The operation performed is not applicable to a CDF with no rVariables. [Informational] |
| `NO_WRITE_ACCESS` | Write access is not allowed on the CDF file(s). Make sure that the CDF file(s) have the proper file system privileges and ownership. [Error] |
| `NOT_A_CDF` | Named CDF is corrupted or not actually a CDF. Contact CDF User Support if you are sure that the specified file is a CDF. [Error] |
| `READ_ONLY_DISTRIBUTION` | Your CDF distribution has been built to allow only read access to CDFs. Check with your system manager if you require write access. [Error] |
| `READ_ONLY_MODE` | The CDF is in read-only mode — modifications are not allowed. [Error] |
| `SINGLE_FILE_FORMAT` | The specified operation is not applicable to CDFs with the single-file format. For example, it does not make sense to close a variable in a single-file CDF. [Informational] |
| `TOO_MANY_VARS` | A multi-file CDF on an IBM PC may contain only a limited number of variables because of the 8.3 file naming convention of MS-DOS. This consists of 100 rVariables and 100 zVariables. [Error] |
| `UNSUPPORTED_OPERATION` | The attempted operation is not supported at this time. [Error] |
| `VAR_ALREADY_CLOSED` | The specified variable is already closed. [Informational] |
| `VAR_CLOSE_ERROR` | Error detected while trying to close variable file. Check that sufficient disk space exists for the variable file and that it has not been corrupted. [Error] |
| `VAR_CREATE_ERROR` | An error occurred while creating a variable file in a multi-file CDF. Check that a file quota has not been reached. [Error] |
| `VAR_EXISTS` | Named variable already exists - cannot create or rename. Each variable in a CDF must have a unique name (rVariables and zVariables can not share names). Note that trailing blanks are ignored by the CDF library when comparing variable names. [Error] |
| `VAR_NAME_TRUNC` | Variable name truncated to `CDF_VAR_NAME_LEN` characters. The variable was created but with a truncated name. [Warning] |
| `VAR_OPEN_ERROR` | An error occurred while opening variable file. Check that sufficient privilege exists to open the variable file. Also make sure that the associated variable file exists. [Error] |
| `VAR_READ_ERROR` | Failed to read variable as requested — error from file system. Check that the associated file is not corrupted. [Error] |

| | |
|---|---|
| `VAR_WRITE_ERROR` | Failed to write variable as requested — error from file system. Check that the associated file is not corrupted. [Error] |
| `VIRTUAL_RECORD_DATA` | One or more of the records are virtual (never actually written to the CDF). Virtual records do not physically exist in the CDF file(s) but are part of the conceptual view of the data provided by the CDF library. Virtual records are described in the Concepts chapter in the CDF User's Guide. [Informational] |

# Appendix D

# Release Notes

## D.1 Introduction

This chapter should provide you with all of the information you need to know to begin using CDF V2.5 assuming that you are already familar with CDF V2.4. For additional information contact CDF User Support.

## D.2 Supported Systems

CDF V2.5 is currently supported on the following computers/operating systems.

1. VAX (VMS)
2. Sun (SunOS & SOLARIS)
3. DECstation (ULTRIX)
4. Silicon Graphics Iris & Power Series (IRIX)
5. IBM RS6000 series (AIX)
6. HP 9000 series (HP-UX)
7. IBM PC (MS-DOS)
8. NeXT (Mach)
9. DEC Alpha (OSF/1 & OpenVMS)
10. Macintosh (MacOS 7.0)

Additional ports are planned for the future. They include Cray (UNICOS), Macintosh (AUX), and a 32-bit distribution for the IBM PC (MS-DOS).

## D.3 Changes for CDF V2.5

Several changes have been made to the CDF distribution for V2.5. CDF V2.5 differs from CDF V2.4 in the following ways.

1. The distribution has been ported to the Macintosh (MacOS 7.0). A version of the CDF library is available for Symantec THINK C and Macintosh Programmer's Workshop (MPW) C and Fortran applications. The CDF toolkit programs have user interfaces typical of Macintosh applications.
2. A shareable version of the CDF library is available on DEC Alphas running OSF/1. Because of this, support for CDF's IDL interface has also been added for that platform.
3. Create/write access for a CDF with any supported encoding is now supported on all platforms. Previous CDF distributions only supported host and network encodings when creating/writing a CDF.
4. Support for Version 1 CDFs has been eliminated. If necessary, use CDF V2.4 to convert your Version 1 CDFs to Version 2 CDFs before accessing them with CDF V2.5.
5. A cache size qualifier has been added to most of the CDF toolkit programs. With this qualifier, the cache size to be used with open CDFs can be adjusted for increased performance.
6. Most of the functionality of `CDFinquire` has been moved to `SkeletonTable`. `CDFinquire` may still be used to inquire the version of the CDF distribution being used and the default qualifiers for the CDF toolkit programs. This was done in an effort to eliminate redundant capabilities between the two programs.
7. The ability to delete variables, attributes, and attribute entries from a CDF has been added.
8. An operation has been added to the Internal Interface which allows variable records to be allocated in a single-file CDF (but not yet written). The application program is then expected to write to the allocated variable records before closing the CDF. This method is more efficient than using "initial records".
9. The option of converting `-0.0` to `0.0` by the CDF library when reading from a CDF now also applies to writing to a CDF. When `-0.0` to `0.0` mode is enabled, `-0.0` would be converted to `0.0` by the CDF library before being written to a CDF.
10. An encoding qualifier has been added to the `CDFconvert` toolkit program to allow any supported encoding to be selected for the destination CDF of a conversion.
11. Support for the Obsolete Interface has been eliminated. Applications written for CDF Version 1 must now be modified for the Standard (or Internal) Interface of CDF Version 2.
12. The CDF V2.5 distribution is configured for a default format of single-file (previous CDF distributions defaulted to multi-file). This may be changed before building/installing the CDF distribution if desired.
13. A statistics qualifier has been added to some of the CDF toolkit programs. When selected, caching statistics are displayed when a CDF is closed.
14. `CDFskeleton` has been renamed to `SkeletonCDF`. This should ease the confusion experienced by some users. `SkeletonTable` creates a skeleton table and `SkeletonCDF` creates a skeleton CDF. (Note that the symbol `CDFSKELETON` on VMS systems and the environment variable `cdfskeleton` on UNIX systems is still defined in the corresponding "definitions" file for compatibility with existing user systems.)

15. CDFs created with CDF V2.5 will use less disk space than CDFs created with previous CDF distributions. The savings will consist of 1689 bytes per CDF regardless of its contents plus an additional savings of 128 bytes per variable. **NOTE:** A side effect of this is that V2.5 CDFs may only be read with the CDF V2.5 distribution. Previous CDF distributions will not be able to read CDFs created with CDF V2.5.

16. The CDF library has been modified to use less memory for each open CDF (unless a large cache size is specified). This should ease the memory limitation problems encountered on the IBM PC and Macintosh. Access to attribute entry data may be slower because of this change.

17. Assumed attribute scopes are no longer supported. The CDF library will now automatically convert an assumed scope to its definite equivalent scope. Because of this the `CDFscope` toolkit program has also been eliminated. If necessary, use CDF V2.4 to correct attribute scopes. Note that the Internal Interface of CDF V2.5 may also be used to correct an attribute scope.

18. Dynamic-link libraries (DLLs) are now supplied with the MS-DOS distribution for Microsoft and Borland Windows applications.

19. Qualifiers have been added to `CDFedit` which allow attribute entries to either be displayed with the attributes or on separate menus (with one for each attribute).

# Appendix E

# Glossary

| | |
|---|---|
| attribute | A CDF object with which entries of metadata are associated. |
| big-endian | The byte ordering in which the most significant byte (MSB) is stored in the lowest memory location. |
| caching | The method used by the CDF library to improve performance when accessing a file. An attempt is made to keep commonly accessed blocks of the file in memory rather than repeatedly reading them from disk. |
| CDF | This term is used in more than one way... 1. The actual files that contain your data/metadata. For example: The CDF library must be used to create a "CDF." 2. The software distribution containing the CDF library, include files, and toolkit. For example: We like using "CDF" to store our data. |
| CDF distribution | The directory of software consisting of the CDF library, include files, and toolkit. |
| CDF library | The software library that is used to access a CDF. |
| CDF toolkit | A set of utility programs which ease the creation, modification, and verification of CDFs. |
| `CDF_OK` | A completion status code indicating unqualified success. |
| `cdf.h` | An include file used in C applications. |
| `cdf.inc` | An include file used in Fortran applications. |
| `cdfmsf.inc` | An include file used in Microsoft Fortran applications. |
| column-major | The variable majority where the first index of a multidimensional array of values increments the fastest. |

| | |
|---|---|
| conceptual view | The way that values along a dimension having a variance of `NOVARY` are made to appear as if they do actually exist (only one value is actually physically stored). This also applies to records beyond the last record actually stored. The conceptual view of a variable consists of "virtual" records and values (in addition to the physical records and values actually stored). |
| current | When the Internal Interface is used, current objects/states are those items affected when an operation is performed. For example, a current CDF is selected and then any operation performed involving a CDF is performed on that CDF (until a different current CDF is selected). |
| data specification | For a variable or attribute entry the data type and number of elements of that data. |
| data type | For a variable or attribute entry, the type of data being stored (e.g., integer, floating-point, character). |
| decoding | The integer/floating-point representation of data values passed to an application by the CDF library as they are read from a CDF. This is independent of the way the data values are physically stored in the CDF. |
| dimensionality | The number of dimensions and the dimension sizes for the rVariables or a zVariable. |
| encoding | The integer/floating-point representation of the data values physically stored in a CDF. |
| entry | A CDF object in which metadata is stored. An entry is associated with an attribute. |
| `EPOCH` | A variable of the NSSDC Standard used to store "time" values. The `CDF_EPOCH` data type is used with this variable. |
| error code | A status code indicating that a fatal condition was encountered. The operation was aborted. |
| extend records | The number of records actually allocated for a variable in a single-file CDF when a new record is written. More records may be allocated than are actually needed in order to keep a variable's records as contiguous as possible (with the assumption that the records will eventually be written by an application). |
| format | In reference to a CDF, the way in which files are used to store the CDF's control/data/metadata. This may be single-file or multi-file. |
| full-physical record | A variable record consisting of values exactly as physically stored in the CDF. |
| gAttribute | A global scoped attribute. |
| gEntry | An entry for a gAttribute. |

| | |
|---|---|
| global scope | Global scope indicates that an attribute describes some property of the entire CDF. |
| host decoding | The decoding of the computer currently being used. |
| host encoding | The encoding of the computer currently being used. |
| hyper access | A variable access method in which multiple records/values are read/written for a variable. |
| IDL Interface | A set of functions callable from within IDL (Interactive Data Language) that allow access to CDFs. The CDF distribution contains an IDL interface in addition to the CDF interface built into IDL by Research Systems, Inc. (RSI — the distributors of IDL). |
| IEEE 754 | The floating-point representation of XDR. |
| include file | A file, included by a C or Fortran application, that contains constants recognized by the CDF library pertaining to various aspects of CDF objects/states. |
| indexing | The method used in a single-file CDF to keep track of where each variable's records are located. |
| informational code | A status code indicating success but providing some additional information that may be of interest. |
| Internal Interface | A set of routines in the CDF library callable from C and Fortran applications that provide all types of access to CDFs. |
| item | When the Internal Interface is used, an object or state on which an operation is performed. |
| `libcdf.a` | The CDF library on UNIX systems. |
| `libcdf.lib` | The CDF library on MS-DOS systems. |
| `LIBCDF.OLB` | The CDF library on VMS and OpenVMS systems. |
| little-endian | The byte ordering in which the least significant byte (LSB) is stored in the lowest memory location. |
| majority | The order in which the values of a multidimensional array are stored. This may be either row-major or column-major. |
| monotonicity | The property of a variable that specifies whether or not that variable's values increment or decrement (or neither) along a dimension or from record to record. |
| multi-file | A CDF format. Multi-file CDFs consist of one file for control/metadata and one file per variable of data. |
| multiple variable access | A variable access method in which one full-physical record is read/written for each of one or more variables. |
| network encoding | The encoding that uses the XDR representation. |

| | |
|---|---|
| `NOVARY` | A record/dimension variance indicating that the values do not change from record to record or along a dimension. |
| NRV variable | Non-record variant variable. A variable whose values do not change from record to record (a record variance of `NOVARY`). |
| NSSDC | National Space Science Data Center. |
| NSSDC Standard | A set of guidelines for creating a CDF containing attributes expected by NSSDC tools. |
| number of elements | For a variable the number of instances of the data type at each value. For an attribute entry the number of instances of the data type for that entry. |
| object | When the Internal Interface is used, an item that exists and may be accessed/manipulated (e.g., a CDF or variable). |
| operation | When the Internal Interface is used, an action performed on an item (e.g., creating or writing). |
| pad value | A value written to a variable by the CDF library in those cases where a physical record must be written but not all of its values have been specified by an application. For example, when a single value is written to a new record, all of the other values are written using the pad value. |
| physical record | A variable record actually stored in a CDF. |
| physical value | A variable value actually stored in a CDF. |
| read-only | A mode of the CDF library in which modifications to a CDF are not allowed. |
| rEntry | An entry for a vAttribute corresponding to an rVariable. |
| row-major | The variable majority where the last index of a multidimensional array of values increments the fastest. |
| RV variable | Record variant variable. A variable whose values change from record to record (a record variance of `VARY`). |
| rVariable | "R" variable. A CDF object in which data values are stored. All rVariables have the same dimensionality. |
| scope | The intended use for an attribute. This may be global scope or variable scope. |
| sequential access | A variable access method in which values are read/written in the physical order in which they are stored in the CDF. |
| single-file | A CDF format. Single-file CDFs are entirely contained within one file. |
| single value access | A variable access method in which exactly one value is read/written for a variable. |

| | |
|---|---|
| skeleton CDF | A CDF consisting of only control, metadata (attributes and attribute entries), and NRV variable values. |
| skeleton table | A text file containing the control, metadata, and NRV variable values of a skeleton CDF. A skeleton table is read by the `SkeletonCDF` toolkit program, which then creates the corresponding skeleton CDF. The `SkeletonTable` toolkit program can be used to create a skeleton table from a CDF. |
| Standard Interface | A set of routines in the CDF library callable from C and Fortran applications that provide access to a commonly used subset of the capabilities of the Internal Interface. This interface was defined with the release of CDF V2.0 and has not changed since. New features since that time are available only through the Internal Interface (e.g., zVariables and zMode). |
| state | When the Internal Interface is used, a property pertaining to an object (e.g., a CDF's format or variable's data specification). |
| status code | The result of a CDF function/subroutine call. `CDF_OK` indicates unqualified success. |
| status handler | A function/subroutine that acts upon a status code received from the CDF library. |
| variable scope | Variable scope indicates that an attribute describes some property of each variable. |
| variance, record | The property of a variable that specifies whether or not its values change from record to record. |
| variance, dimension | The property of a variable that specifies whether or not the values along a dimension change or stay the same. |
| `VARY` | A record/dimension variance indicating that the values change from record to record or along a dimension. |
| vAttribute | A variable scoped attribute. |
| virtual record | A variable record that is not actually stored in a CDF but does appear in the conceptual view of the CDF. Virtual records would be those records beyond the first record of an NRV variable and those records beyond the last record actually written to an RV variable. |
| virtual value | A variable value this is not actually stored in a CDF but does appear in the conceptual view of the CDF. Virtual values would be those values beyond the first value of a dimension whose variance is `NOVARY`. |
| warning code | A status code indicating that the operation did complete but probably not as expected. |
| XDR | External Data Representation. An integer/floating-point representation using big-endian byte ordering and the IEEE 754 floating-point representation. |

| | |
|---|---|
| zEntry | An entry for a vAttribute corresponding to a zVariable. |
| zMode | A mode of the CDF library in which rVariables are made to appear as zVariables (and rEntries appear as zEntries). |
| zVariable | "Z" variable. A CDF object in which data values are stored. zVariables can have dimensionalities that are different than those of the rVariables (and each other). |

# Index