# CDF

## User's Guide

Version 3.5, September 25, 2012

Space Physics Data Facility
NASA / Goddard Space Flight Center

Copyright © 2012 NASA/GSFC
Space Physics Data Facility
NASA/Goddard Space Flight Center
Greenbelt, Maryland 20771 (U.S.A.)

This software may be copied or redistributed as long as it is not sold for profit, but it can be incorporated into any other substantive product with or without modifications for profit or non-profit. If the software is modified, it must include the following notices:

- The software is not the original (for protection of the original author's reputations from any problems introduced by others)
- Change history (e.g. date, functionality, etc.)

This copyright notice must be reproduced on each copy made. This software is provided as is without any express or implied warranties whatsoever.

Internet: gsfc-cdf-support@lists.nasa.gov

Permission is granted to make and distribute verbatim copies of this document provided this copyright and permission notice are preserved on all copies.

# Contents


**1 Primer ..... 5**
1.1 Introduction ..... 5
1.2 Why use CDF? ..... 5
1.3 Conceptual Organization ..... 5
1.4 Features of the CDF Library ..... 6
1.4.1 File Format Options ..... 7
1.4.2 Data Encoding Options ..... 12
1.4.3 Compression ..... 12
1.4.4 Sparseness ..... 12
1.4.5 Variable Data Access Options ..... 12
1.5 Organizing Your Data in a CDF ..... 13
1.5.1 Variables ..... 13
1.6 Attributes ..... 16
1.7 CDF Toolkit ..... 17
1.8 Library Interface Routines ..... 18
1.8.1 Standard Interface ..... 18
1.8.2 Internal Interface ..... 19
1.9 CDF Java Interface ..... 19
1.10 How to create a CDF ..... 19
1.10.1 Sample C, Fortran, Java, Perl and C# Programs ..... 19
1.10.2 Creating a CDF with CDFEdit ..... 19
1.10.3 Creating a CDF with SkeletonTable ..... 19
**2 Concepts ..... 23**
2.1 CDF Library ..... 23
2.1.1 Interfaces ..... 23
2.1.2 CDF Modes ..... 24
2.1.3 Limits ..... 26
2.1.4 Scratch Files ..... 26
2.1.5 Caching Scheme ..... 26
2.2 CDFs ..... 28
2.2.1 Accessing ..... 28
2.2.2 Creating ..... 28
2.2.3 Opening ..... 28
2.2.4 Closing ..... 28
2.2.5 Deleting ..... 29
2.2.6 Naming ..... 29
2.2.7 Format ..... 29
2.2.8 Encoding ..... 31
2.2.9 Decoding ..... 33
2.2.10 Compression ..... 35
2.2.11 Limits ..... 35
2.3 Variables ..... 35
2.3.1 Types ..... 36
2.3.2 Accessing ..... 36
2.3.3 Opening ..... 36
2.3.4 Closing. ..... 36
2.3.5 Naming ..... 37
2.3.6 Numbering ..... 37
2.3.7 Deleting ..... 37
2.3.8 Dimensionality ..... 37
2.3.9 Data Specification ..... 38

2.3.10 Record Variance ... 38
2.3.11 Dimension Variance ... 39
2.3.12 Records. ... 40
2.3.13 Sparse Arrays ... 45
2.3.14 Compression ... 45
2.3.15 Majority ... 46
2.3.16 Single Value Access ... 47
2.3.17 Hyper Access ... 48
2.3.18 Sequential Access ... 50
2.3.19 Multiple Variable Access ... 51
2.3.20 Variable Pad Values. ... 52
2.4 Attributes ... 53
2.4.1 Naming ... 54
2.4.2 Numbering ... 54
2.4.3 Attribute Scopes ... 54
2.4.4 Deleting ... 55
2.4.5 Attribute Entries ... 55
2.5 Data Types ... 56
2.5.1 Integer Data Types ... 56
2.5.2 Floating Point Data Types ... 56
2.5.3 Character Data Types ... 56
2.5.4 EPOCH Data Types ... 57
2.5.5 TT2000 Data Types ... 57
2.5.6 Equivalent Data Types ... 58
2.6 Compression Algorithms ... 58
2.6.1 Run-Length Encoding ... 58
2.6.2 Huffman ... 58
2.6.3 Adaptive Huffman ... 59
2.6.4 GZIP ... 59
2.7 TT2000 and Leap Seconds ... 59
**3 Toolkit Reference ... 61**
3.1 Introduction ... 61
3.1.1 VMS and UNIX (including Mac OS X) ... 61
3.1.2 How to Invoke the GUI Toolkit for Macintosh OS X ... 62
3.1.3 How to Invoke the GUI Toolkit for Windows NT/95/98/2000/XP ... 63
3.1.4 How to Invoke the GUI Toolkit for Unix ... 63
3.1.5 Special Attributes ... 64
3.1.6 Special Qualifier ... 64
3.2 CDFedit ... 64
3.2.1 Introduction ... 64
3.2.2 Special Attribute Usage ... 65
3.2.3 Executing the CDFedit Program ... 65
3.2.4 Interaction with CDFedit ... 67
3.3 CDFexport ... 68
3.3.1 Introduction ... 68
3.3.2 Special Attribute Usage ... 68
3.3.3 Executing the CDFexport Program ... 68
3.3.4 Interaction with CDFexport ... 74
3.4 CDFconvert ... 74
3.4.1 Introduction ... 74
3.4.2 Executing the CDFconvert Program ... 75
3.4.3 Output from the CDFconvert Program ... 80
3.5 CDFcompare ... 80
3.5.1 Introduction ... 80
3.5.2 Executing the CDFcompare Program ... 80
3.5.3 Output from the CDFcompare Program ... 84

3.6 CDFstats ... 84
3.6.1 Introduction ... 84
3.6.2 Special Attribute Usage ... 84
3.6.3 Executing the CDFstats Program ... 85
3.6.4 Output from the CDFstats Program ... 88
3.7 SkeletonTable ... 89
3.7.1 Introduction ... 89
3.7.2 Special Attribute Usage ... 90
3.7.3 Executing the SkeletonTable Program ... 90
3.7.4 Output from the SkeletonTable Program ... 93
3.8 SkeletonCDF ... 93
3.8.1 Introduction ... 93
3.8.2 Executing the SkeletonCDF Program ... 94
3.8.3 Creating the Skeleton Table ... 96
3.9 CDFinquire ... 96
3.9.1 Introduction ... 96
3.9.2 Executing the CDFinquire Program ... 96
3.9.3 Output from the CDFinquire Program ... 97
3.10 CDFdir ... 97
3.10.1 Introduction ... 97
3.10.2 Executing the CDFdir Program ... 97
3.10.3 Output from the CDFdir Program ... 98
3.11 CDFmerge ... 98
3.11.1 Introduction ... 98
3.11.2 Executing the CDFmerge Program ... 98
3.12 CDFdump ... 101
3.12.1 Introduction ... 101
3.12.2 Executing the CDFdump Program ... 101
3.13 CDFirsdump ... 102
3.13.1 Introduction ... 102
3.13.2 Executing the CDFirsdump Program ... 103
3.14 CDFvalidate ... 104
3.14.1 Introduction ... 104
3.14.2 Executing the CDFvalidate Program ... 104
3.15 CDFleapsecondsinfo ... 105
3.15.1 Introduction ... 105
3.15.2 Executing the CDFleapsecondsinfo Program ... 105

# List of Figures

Figure 1.1 Conceptual View of a CDF, 0-Dimensional rVariable ..... 8
Figure 1.2 Conceptual View of a CDF, 2-Dimensional rVariables ..... 9
Figure 1.3 Conceptual View of a CDF, zVariables ..... 10
Figure 1.4 Multi-File Format ..... 11
Figure 1.5 Single-File Formats ..... 11
Figure 2.1 Physical vs. Virtual Dimensions ..... 39
Figure 2.2 Physical vs. Virtual Records, Standard Variable ..... 41
Figure 3.1 Window Sections, CDFedit ..... 67

# List of Tables

Table 1.1 Example Data Set - "Flat" Representation (0-Dimensional) ..... 14
Table 1.2 Example CDF - 2-Dimensional Representation (Conceptual) ..... 14
Table 1.3 Example CDF - Specification for 2-Dimensional Representation ..... 15
Table 1.4 Example CDF - 2-Dimensional Representation (Physical) ..... 15
Table 1.5 vAttribute rEntries for the Temperature rVariable ..... 17
Table 2.3 Cache Size Operations, Internal Interface ..... 28
Table 2.4 Equivalent Byte Orderings ..... 32
Table 2.5 Equivalent Single-Precision Floating-Point Encodings ..... 33
Table 2.6 Equivalent Double-Precision Floating-Point Encodings ..... 33
Table 2.7 Previous-missing Sparse Records Example, Conceptual View vs. Physical Storage ..... 43
Table 2.8 Default Pad Values. ..... 53
Table 2.9 Equivalent Data Types ..... 58
Table 3.1 Example rVariables, CDFstats Monotonicity Checking ..... 84

# 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 scalars, 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 metadata (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 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]).

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

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 a suite of utilities for creating, browsing, and modifying CDF files as well as exporting or importing CDF data to/from a regular text file or an eXtensible Markup Language (XML) file. These are very useful for architect ring 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 CDF library comes with C, Java and Fortran Application programming Interfaces (APIs), and the APIs provide the essential framework on which graphical and data analysis packages can be created. Perl APIs are also available as an optional package for those who wish to develop CDF applications in Perl. 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 on any of the CDF-supported platforms. These currently include VAX (OpenVMS and POSIX shell), Sun (SunOS & Solaris), DECstation (ULTRIX), DEC Alpha (OSF/1 or Tru64 & OpenVMS), Silicon Graphics Iris and Power Series (IRIX), IBM RS6000 series (AIX), HP 9000 series (HP-UX), NeXT (Mach), PC (DOS, Windows 3.x, Windows NT/95/98/2000/XP, Linux, Cygwin & MinGW), and Macintosh (Mac OS X or Linux). If you need to run the CDF library on an operating system that's not mentioned above, please contact the CDF support office at gsfc-cdf-support@lists.nasa.gov.

CDF is supported by commercial and open source data analysis/visualization software such as IDL, MATLAB, and IBM's Data Explorer (XP). For those who are familiar with a language like IDL or MATLAB can easily create sophisticated plots from CDF files instead of writing a lengthy program in C, Fortran, or Java.

## Compatibility with Previous CDF Releases

One of the CDF 3.0 requirements was an ability to create files bigger than 2G bytes. This requirement necessitated a change in the internal file structure since the 32-bit file offset had to be changed to a 64-bit file offset. As a result, CDF 2.7.2 or earlier won't be able to read CDF files that are created with CDF 3.0 or a later version. However, CDF 3.* can read files that are created with any of the previous CDF releases. If one is concerned about using CDF 3.0 or a later version due to the file compatibility problem with previous releases, one can create files in the CDF 2.7 format (optional) with CDF 3.*. The Backward File Compatibility with CDF 2.7 section of the CDF C Reference Manual or Fortran Reference Manual describes how to create files that can be read by CDF 2.7.2 or earlier, or IDL 6.2 or earlier. IDL 6.3 can read files created by CDF 3.0 or a later version. If a file is created in the CDF 2.7 format using the CDF 3.0 library or a later version, the maximum file size is 2G bytes.

**Note**: To minimize the scope of the coding changes as well as make the code functioning on both 32/64-bit machines when the 2G file size barrier is lift for V3.*, the maximum record number for each variable in a CDF stays the same as the previous versions. It is dictated by the 32-bit integer for the record counter, a 2G itself.

## Checksum

To ensure the data integrity in a CDF file, the checksum option has been added to the CDF Version 3.2. This is a form of redundancy check, a very simple measure for protecting the integrity of data by detecting error in data that is sent through space or time. It works by adding up the basic components of a message, typically the asserted bits, and storing the resulting value. Later, anyone can perform the same operation on the data, compare the result to the authentic checksum, and (assuming that the sums match) conclude that the message was probably not corrupted.

The checksum method used by the CDF is the popular MD5 algorithm. If the checksum bit is turned on for a CDF file, a 16-byte signature message (a.k.a. message digest) is computed from the entire file and appended to the end of the file

when the file is closed (after any create/write/update activities). Every time such file is open, other than the normal steps for opening a CDF file, this signature, serving as the authentic checksum, is used for file integrity check by comparing it to the re-computed checksum from the current file. If the checksums match, the file's data integrity is verified. Otherwise, an error message is issued. The checksum operation can be applied to CDF single-file format files that were created with V2.7 or later. Once the checksum is turned on for a particular file, the data integrity check of the file is performed every time it is open; and a new checksum is computed and stored when it is closed. This overhead (performance hit) may be noticeable for large files. Therefore, it is strongly encouraged to turn off the checksum once the file integrity is confirmed or verified. A couple of the utilities from the CDF toolkit that is distributed along with the core library provide the easy way to change the checksum option within a file. Use CDFedit to interactively modify the checksum mode. Or, use CDFconvert, a command line tool, to convert the source CDF file with the checksum to a non-checksum destination file.

CDF 3.1 or earlier doesn't recognize the checksum bit. Hence, the checksum bit will be ignored.

## How is CDF Useful to Me?

Hopefully, the answers to the first two questions have provided a basis for answering this question. If you still have questions or would like to learn more about CDF, please refer to the CDF Frequently Asked Questions (FAQ) page (`http://cdf.gsfc.nasa.gov/html/FAQ.html`) for more detailed information about CDF. 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

# 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, Fortran, and Java 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. If an application is written in Java, it can be executed without any modifications on any of the Java supported platforms.

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".

CDF files created on one operating system can be read without any modifications on any of the following CDF supported platforms: VAX (OpenVMS and POSIX shell), Sun (SunOS & Solaris), DECstation (ULTRIX), DEC Alpha (OSF/1 or Tru64 & OpenVMS), Silicon Graphics Iris and Power Series (IRIX), IBM RS6000 series (AIX), HP 9000 series (HP-UX), NeXT (Mach), PC (DOS, Windows 3.x, Windows NT/95/98/2000/XP, Linux, Cygwin & MinGW), and Macintosh (Mac OS X or Linux).

## 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. CDF's "variable" is a generic name or an object that represents data, and it does not have any scientific context associated it. For example, a variable can be data representing an independent variable, a dependent variable, time and date value, or whatever data might be (e.g. image, XML file, etc.). In other words, the variable doesn't contain any hidden meanings other than the data itself. One may describe a variable or one variable's relationship with other variable(s) through "attributes" (see the last paragraph of this section for more details). The dimensionality of a variable depends upon how the data is specified by the user. 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. As you can see, since all the rVariables must have the same dimensions and dimension sizes, there'll be a lot of disk space wasted if a few variables need big arrays and many variables need small arrays. Since zVariable is more efficient in terms of storage and offers more functionality than rVariable, use of zVariable is strongly recommended. 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.

So why would you want to use rVariables over zVariables? There's no reason to use rVariables at all (since zVariables are much more efficient) if you are creating a new CDF file. But if you are analyzing data files that were created with early CDF releases or contain rVariables for some reason, you'll need to use rVariables. One may wonder why there are rVariables and zVariables, not just zVariables. When CDF was first introduced in early 90's, only rVariables were available. The inefficiencies with rVariables were quickly realized and addressed with the introduction of zVariables in later CDF releases.

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 depending on how the user views the data. This is the advantage of CDF. Data values can be organized in whatever way makes sense to the user.

While CDF's variable is a mechanism for storing/representing data, CDF's "attribute" is a mechanism for describing the CDF file and the individual CDF variables in the file. There are two types of attributes in CDF: global attribute and variable attribute. Global attribute is used for describing the CDF file and variable attribute is used for describing individual variables. Examples of global attributes would include such things as file creation date, file author, source of data, and data set documentation. Examples of variable attributes 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, and a fill value for errant or missing data.

## 1.4 Features of the CDF Library

[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.
[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 z variables are necessary it is best to use all zVariables than a mixture of rVariables and zVariables.

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. 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,[4] 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, the default format when a CDF file is created. As illustrated in Figure 1.5, the whole CDF file consists of only a single 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. Use of single-file format (the default format) is recommended over the multi-file format albeit it slightly increases 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, and accessing data are also done with optimum efficiency. However, the multi-file format has the following restrictions[5]:

- Compression: Compression is not allowed for the CDF or any of its variables.

- Sparseness: Sparse records or arrays for variables are not allowed.

- Allocation: Pre-allocation of records or blocks of records is not allowed. For each variable, the maximum written record is the last allocated record.

- Deletion: Deletion of a single variable from a CDF is not allowed. Only deleting a whole CDF is possible.

[4] This file referred to as the dotCDF file.
[5] These features are covered in the following sections.

| Record Number | rVariable 1 | rVariable 2 | . | . | . | rVariable n |
|---|---|---|---|---|---|---|
| 1 | ❑ | ❑ | . | . | . | ❑ |
| 2 | ❑ | ❑ | | | | ❑ |
| 3 | ❑ | ❑ | | | | ❑ |
| . | . | . | | | | . |
| . | . | . | | | | . |
| . | . | . | | | | . |
| m | ❑ | ❑ | | | | ❑ |

**Figure 1.1 Conceptual View of a CDF, 0-Dimensional rVariable**

| Record Number | rVariable 1 | rVariable 2 | . . . | rVariable n |
|---|---|---|---|---|
| 1 | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | . . . | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ |
| 2 | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | . . . | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ |
| 3 | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | . . . | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ |
| . | . | . | | . |
| . | . | . | | . |
| . | . | . | | . |
| m | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | . . . | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ |

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

| Record Number | zVariable 1 | zVariable 2 | . . . | zVariable n |
|---|---|---|---|---|
| 1 | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | ❑❑❑❑❑<br>❑❑❑❑❑ | . . . | ❑<br>❑<br>❑<br>❑<br>❑ |
| 2 | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | ❑❑❑❑❑<br>❑❑❑❑❑ | . . . | ❑<br>❑<br>❑<br>❑<br>❑ |
| 3 | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | ❑❑❑❑❑<br>❑❑❑❑❑ | . . . | ❑<br>❑<br>❑<br>❑<br>❑ |
| . | . | . | | . |
| . | . | . | | . |
| . | . | . | | . |
| m | ❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑<br>❑❑❑❑❑ | ❑❑❑❑❑<br>❑❑❑❑❑ | . . . | ❑<br>❑<br>❑<br>❑<br>❑ |

**Figure 1.3 Conceptual View of a CDF, zVariables**

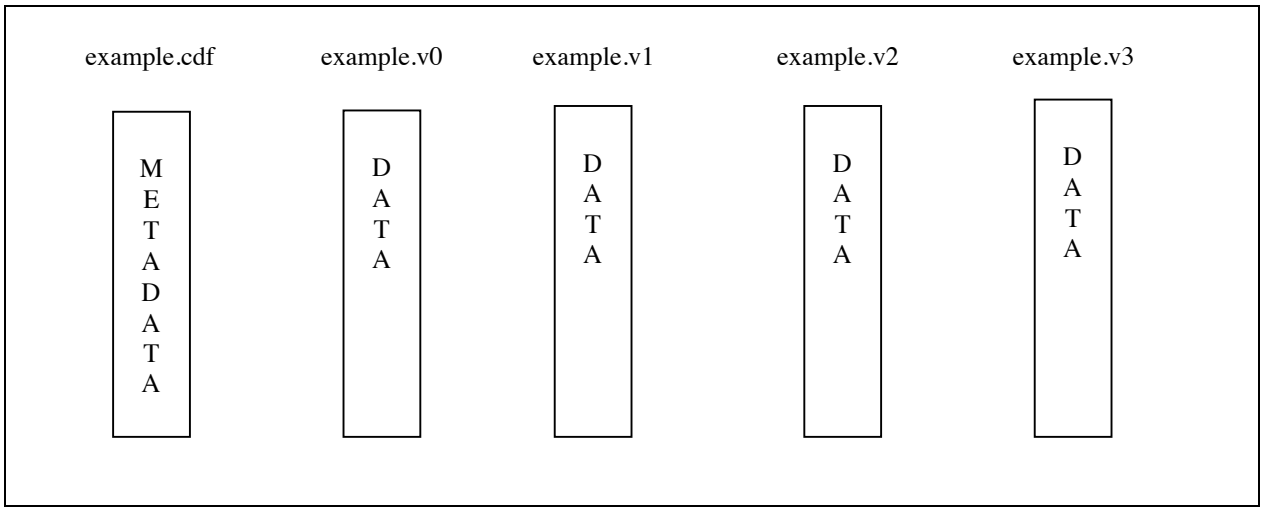


**Figure 1.4 Multi-File Format**

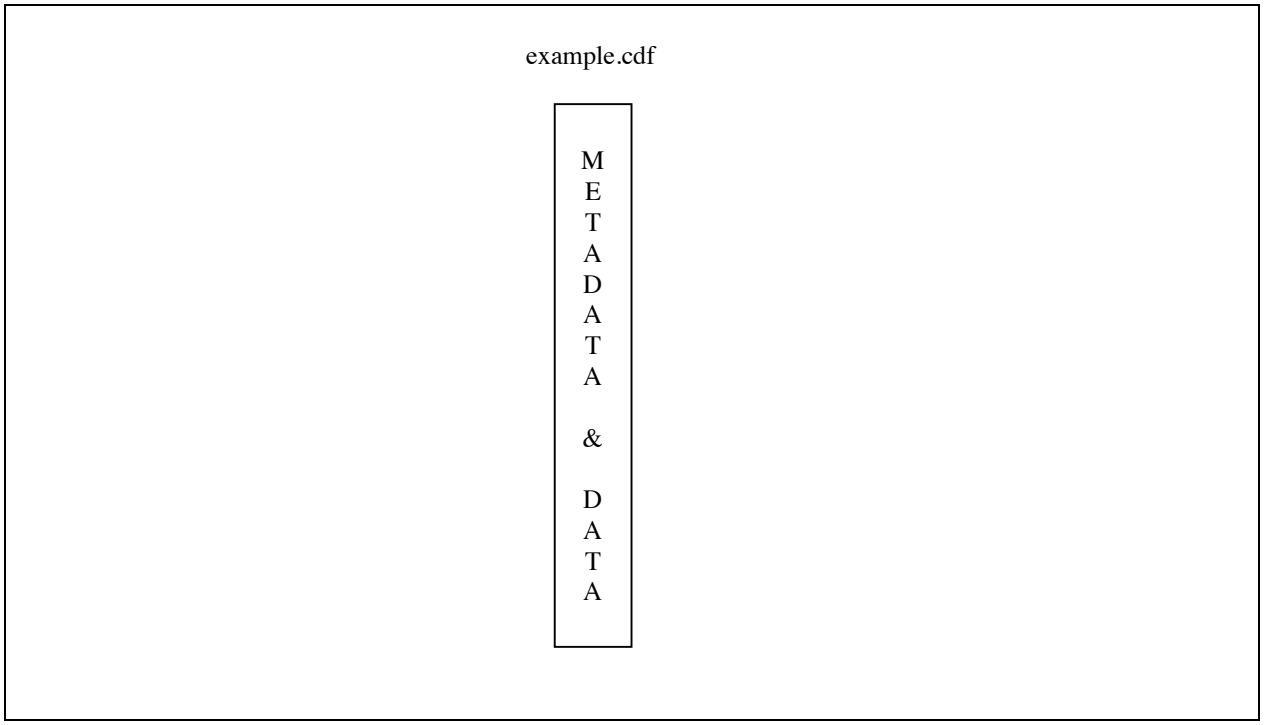


**Figure 1.5 Single-File Formats**

### 1.4.2 Data Encoding Options

When creating a CDF, a user has the option of using any of the supported encodings: VAX, Sun, SGi Personal Iris and Power Series, DECstation, DEC Alpha/OSF1, DEC Alpha/OpenVMS (D FLOAT, G FLOAT or IEEE FLOAT double-precision floating-point), IBM RS6000 series, HP 9000 series, NeXT, 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 Compression

Compression may be specified for a single-file CDF and the CDF library can be instructed to compress a CDF as it is written to disk. This compression occurs transparently to the user. When a compressed CDF is opened, the CDF library automatically decompresses it. An application does not have to even know that a CDF is compressed. Any type of access is allowed on a compressed CDF. When a compressed CDF is closed by an application, it is automatically recompressed as it is written back to disk.

The individual variables of a CDF can also be compressed. . The CDF library handles the compression and decompression of the variable values transparently. The application does not have to know that the variable is compressed as it accesses the variable's values. [6]

The CDF library supports several different compression algorithms. When compression is specified for a CDF or one of its variables, the compression algorithm to be used must be selected. There will be trade-offs between the different compression algorithms regarding execution performance and disk space savings.

The nature of the data in a CDF (or variable) will affect the selection of the best compression algorithm to be used.

### 1.4.4 Sparseness

Two types of sparseness are allowed for CDF variables: sparse records and sparse arrays. Sparse records are available now - sparse arrays won't be available until a future CDF release. When a variable is specified as having sparse records, only those records actually written to that variable will be stored in the CDF. This differs from variables without sparse records in that for those variables every record preceding the maximum record written is stored in the CDF. For example, if only the 1000th record were written to a variable without sparse records, the 999 preceding records would also be written using a pad value. If sparse records had been specified for the variable, only the 1000th record would be stored in the CDF (saving a considerable amount of disk space). Sparse records are ideal for variables containing gaps of missing data.

### 1.4.5 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.

---

[6]. If compressed data turns out to be larger than uncompressed data, it makes no sense to use compression, thus variable data will be stored in a CDF uncompressed, even if a compression is requested

## 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 (rVariable and zVariable) and they can coexist in the same CDF file. Use of zVariable over rVariable is strongly recommended since it is much more efficient and offers more functionality than the rVariable.

So why would you want to use rVariables over zVariables? There's no reason to use rVariables at all if you are creating a new CDF file. But if you are analyzing data files that were created with early CDF releases or contain rVariables for some reason, you'll need to use rVariables. One may wonder why there are rVariables and zVariables, not just zVariables. When CDF was first introduced in early 90's, only rVariables were available. The inefficiencies with rVariables were quickly realized and addressed with the introduction of zVariables in later CDF releases.

**rVariables[7]**

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 "at" (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 |

[7] Although rVariables are described here first, the trend among CDF users is toward CDFs containing only zVariables (since zVariables can do everything rVariables can do and more). zVariables are described in the next section.

| | | | | |
|---|---|---|---|---|
| . | | | | |
| . | | | | |
| . | | | | |
| 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). Specifying "variances" circumvents this problem. 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 this example are shown in Table 1.3.

| Record Number | rVariables<br>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** – 0000 | -165 – -150 | +40 – +40 | **18.2 – 22.0** |
| 2 | \| \| | \| \| | \| \| | \| \| |
| | 0000 – 0000 | -165 – -150 | +30 – +30 | **19.3 – 19.2** |
| | **0200** – 0000 | -165 – -150 | +40 – +40 | **19.9 – 19.6** |
| 3 | \| \| | \| \| | \| \| | \| \| |
| | 0000 – 0000 | -165 – -150 | +30 – +30 | **19.3 – 19.0** |
| . | | | | |
| . | | | | |
| . | | | | |
| | **2300** – 0000 | -165 – -150 | +40 – +40 | **21.0 – 18.4** |
| 24 | \| \| | \| \| | \| \| | \| \| |
| | 0000 – 0000 | -165 – -150 | +30 – +30 | **19.5 – 22.0** |

**Table 1.2 Example CDF - 2-Dimensional Representation (Conceptual)**

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).

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

**Table 1.4 Example CDF - 2-Dimensional Representation (Physical)**

**zVariables**

zVariables are similar to rVariables in all respects except that each zVariable can have a different dimensionality. This allows any set of variables to be stored in the same CDF without wasting space or creating confusion in how the variables are logically viewed.

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, it assumes 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 had a dimensionality of 2:[1024,1024][8] (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 had a dimensionality of 3:[1024,1024,256], the images could be stored in an rVariable having dimension variances T/TTF[9] and the palette could be stored in an rVariable having dimension variances F/FFT. 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 zVariable with dimensionality 2:[1024,1024] (and variances of T/TT) and the palette would be stored in a zVariable with a dimensionality of 1:[256] (and variances of F/T). 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 rVariable example where temperature values were being stored, zVariables could also have been used. Each zVariable would have the same dimensionality and their dimension variances would be used in the same way as they were used for the rVariables.

An even better example of how zVariables are preferred over rVariables in certain situations involves the storage of 1-dimensional arrays (vectors). Assume that five 1-dimensional arrays are being stored with dimension sizes of 2, 3, 5, 7, and 25. Using rVariables with a dimensionality of 1:[25] would waste considerable space while using rVariables with a dimensionality of 5:[2,3,5,7,25] and dimension variances of T/TFFFF, T/FTFFF, T/FFTFF, T/FFFTF, and T/FFFFT would be quite confusing to deal with zVariables with dimensionalities of 1:[2], 1:[3], 1:[5], 1:[7], and 1:[25] would be straight forward and efficient.

## 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. The term "attribute" is used when describing a property that applies to both gAttributes and vAttributes.

---

[8] The notation for dimensionality used here 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.

[9] The notation for variances used here is <rec-vary>/<dim-varys> where <rec-vary> is the record variance, T (TRUE) or F (false), and <dim-varys> is zero or more dimension variances.

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. A gAttribute 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 1.5.

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” |
| FORMATS | “F4.1” |
| MONOTON | “Increasing” |
| FILLVAL | -999.9 |

**Table 1.5 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[10] | Allows the display, creation, and modification of attribute and variable data in a CDF. |
| CDFexport[11] | Allows the contents of a CDF to be exported to the terminal screen, a text file, or another CDF. The CDF may be filtered in order to export a subset of its contents. |
| CDFconvert | Allows the format, encoding, majority, compression, and sparseness of a CDF to be changed. It also can reorganize a fragmented CDF file to make the file access more efficiently. In all cases a new CDF is created. The original CDF is not modified. |

[10] 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.
[11] CDFexport has replaced CDFlist and CDFwalk.

| | |
|---|---|
| SkeletonCDF[12] | 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, many of the configurable parameters, and the default CDF toolkit qualifiers. |
| CDFstats | Produces a report containing various statistics about the variables in a CDF. |
| 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. |
| CDFDump | Dumps the metadata/data in a CDF. |
| CDFIrsDump | Dumps the internal records (IRs) in a CDF. This is a debugging tool to show the internal data structure of a CDF. |
| CDFMerge | Merges several CDFs into a single one. Most suitable for combining time-sequenced CDFs. |
| CDFValidate | Validates CDFs to see if CDFs will pass data checking for certain data fields. |

## 1.8 Library Interface Routines

The core CDF library supports two programming interfaces, the Standard Interface and the Internal Interface. Standard Interface is easier-to-use and requires a much shorter learning curve than the Internal Interface, but it's not as efficient as Internal Interface. Standard Interface is recommended if you are not familiar with Internal Interface. The Standard and Internal interfaces are callable from both C and Fortran.

The C and the Fortran interfaces (APIs) are described in the CDF C Reference manual and the CDF Fortran reference manual, respectively. The CDF Java APIs are described in the CDF Java Reference Manual. The Perl interfaces are described in the Perl to CDF Interfaces document that is included in the CDF Perl distribution package. The C# interfaces are described in the C# to CDF Interfaces document that is included in the C#-CDF distribution package. The C, Fortran, and Java APIs are part of the standard CDF distribution package, but the Perl and C# APIs are available as optional packages. The Java APIs for the Unix[13] and Linux platforms are also available as an optional package. As of this writing, the Java APIs are not available for the VMS operating system.

### 1.8.1 Standard Interface

[12] SkeletonCDF was previously named CDFskeleton
[13] PC running CYGWIN/MinGW or Mac OS X can be considered a UNIX box while running the CDF tool programs.

There are two types of Standard Interfaces: Original and Extended. The Original Standard Interface was introduced in early 90's and they only provide a very limited functionality within the CDF library. For example, it can only handle rVariables, not zVariables, and has no access to zVariables's attribute entry (zEntries). Up until CDF 2.7.2, if you wanted to create or access zVariables and zEntries, you had to use the Internal Interface, which might be harder to use. The limitations of the Original Standard Interface were addressed with the introduction of the Extended Standard Interface in CDF 3.1. The new Extended Standard Interface allows many new operations that were only previously available through the Internal Interface.

### 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.

## 1.9 CDF Java Interface

The CDF Java Application Programming Interfaces (APIs) are based on the core CDF library's Internal Interface., and they support a near complete set of the Internal Interface functions. The Java APIs only support zVariables and treats rVariables as zVariables. This is not a problem since zVariable is a superset of rVariable. In another words, with zVariables, you can do everything with rVariables and more, but not vice versa.

## 1.10 How to create a CDF

A CDF file can be created either by using the programming interface (Standard Interface or Internal Interface) or the CDFEdit and SkeletonTable toolkit programs included in the standard CDF distribution package.

### 1.10.1 Sample C, Fortran, Java, Perl and C# Programs

Sample programs are included as part of the CDF standard distribution package. Below describes where the sample programs can be found.

| | Unix (including Mac OS X) | Windows |
|---|---|---|
| C | <cdf_dir>/samples | <cdf_dir>\samples |
| Fortran | <cdf_dir>/samples | <cdf_dir>\samples |
| Java | <cdf_dir>/cdfjava/examples | <cdf_dir>\samples |
| Perl | <perl_dir> | <perl_dir>\ |
| C# | <csharp_dir> | <csharp_dir> |

### 1.10.2 Creating a CDF with CDFEdit

The CDF toolkit program CDFEdit is one of the utility programs included in the CDF standard distribution package, and it allows users to create a CDF file without programming. CDFEdit allow interactively creating a new, empty CDF file, reading/modifying the data/metadata in an existing file.

### 1.10.3 Creating a CDF with SkeletonTable

The CDF toolkit program SkeletonCDF is another one of the utility programs included in the CDF standard distribution package, and it allows users to create a CDF file without programming. 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 in the CDF standard distribution package, and it reads an existing CDF file and produces a skeleton table. Below is a sample skeleton table file.

```
! 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
```

# Chapter 2

# 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

Two types of interfaces to the CDF core library exist for C and Fortran programs: the Standard Interface and the Internal Interface. The Standard Interface is easier-to-use and requires a much shorter learning curve than the Internal Interface, but it's not as efficient as the Internal Interface. The Standard Interface is recommended if you are not familiar with the Internal Interface. These interfaces are described in the following sections. The CDF Java Interface is described in the CDF Java Reference Manual.

#### Standard Interface

There are two types of the Standard Interfaces: Original and New. The Original Standard Interface was introduced in early 90's and they only provide a very limited functionality within the CDF library. For example, it can only handle rVariables, not zVariables, and has no access to zVariables's attribute entry (zEntries). Up until CDF 2.7.2, if you wanted to create or access zVariables and zEntries, you had to use the Internal Interface that is harder to use. The limitations of the Original Standard Interface were addressed with the introduction of the Extended Standard Interface in CDF 3.1. The Extended Standard Interface allows many new operations that were only previously available through the Internal Interface. Both the Original Standard Interface and Extended Standard Interface are callable from both C and Fortran applications and the functions/subroutines available in these interfaces are described in the CDF C Reference Manual and the CDF Fortran Reference Manual, respectively.

#### Internal Interface

The Internal Interface may be used to perform all supported CDF operations. The Internal Interface is much more efficient than the Standard Interface, but it's a bit more difficult to learn. It should be used to perform those operations not available with the Original Standard Interface and the Extended Standard Interface. The Extended Standard Interface offers almost everything average and sophisticated CDF users need. The Internal Interface is callable from both C and Fortran applications and its available operations are described in the CDF C Reference Manual and the CDF Fortran Reference Manual, respectively.

**CDF's IDL Interface**

IDL has a built-in support for CDF and CDF files can be created and manipulated in IDL. By default, CDF files created in IDL 6.3 or later can't be read by IDL 6.2 or earlier, or CDF 2.7.2 or earlier. However, IDL 6.3 or later can read files that were generated with IDL 6.2 or earlier, or CDF 2.7.2 or earlier. If you create files in IDL 6.3 or later and want to share files with colleagues who access CDF files using IDL 6.2 or earlier, or CDF 2.7.2 or earlier, you can do so by calling the CDF_SET_CDF27_BACKWARD_COMPATIBLE routine prior to creating CDF files (i.e. CDF_CREATE). Note that if this option is used, the maximum file size is 2G bytes.

### 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[1]. 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.) However, if the CDF is modified while not in read-only mode, subsequently setting read-only mode in the same session will not prevent further modifications to the CDF. When read-only mode is set, all metadata is read into memory and references to metadata are then directed there.

**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.

---

[1] This notation is used to specify a function to be performed on an item. The syntax is <function_,item_>.
[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 extensible 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:[].

| 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.

| rVariable 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 rValues.

| rVariable 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. zMode is only applicable to rVariables. It is strongly suggested that zVariables do not use any false dimensional variances..

**-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 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

[5] This notation is also used throughout this document. The record variance is before the slash and the dimension variances.

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 dotCDF file). The CDF library will open and close the variable files of a multi-file CDF as needed (see Sections 2.3.3 and 2.3.4).

### 2.1.4 Scratch Files

The CDF library will make use of scratch files when necessary. These scratch files are associated with an open CDF. Scratch files are used instead of core memory in an effort to prevent memory limitation problems (especially on the Macintosh and PC). The following types of scratch files are used.

| | |
|---|---|
| Staging | The staging scratch file is used when a CDF contains compressed variables. As each variable is accessed, a portion of the staging scratch file is allocated to hold a specific number of uncompressed records for that variable. The number of records allocated depends on the variable's blocking factor (see Section 2.3.12). The staging scratch file is also used (when necessary) with variables having sparse records. If the records being written are not first allocated, the staging scratch file will be used to minimize the indexing overhead (see Section 2.2.7) by trying to keep consecutive records contiguous in the dotCDF file. |
| Compression | The compression scratch file is used when writing to a compressed variable in a CDF. Because the CDF library does not know how well a block of variable records will compress, the compression algorithm first writes the compressed block to the compression scratch file. The compressed block is then copied to the dotCDF file. Note that when reading a compressed variable, a compressed block of records is decompressed directly to the staging scratch file because the CDF library knows the size of the uncompressed block of records. |
| Uncompressed dotCDF | When overall compression is specified for a CDF, the CDF library maintains an uncompressed version of the dotCDF file as a scratch file. |

By default, these scratch files are created in the current directory. On VMS systems the logical name CDF$TMP can be defined with an alternate directory in which to create scratch files. On UNIX and MS-DOS systems the environment variable CDF TMP would be used. An application can also select a directory to be used for scratch files with the <SELECT_,SCRATCHDIR_> operation of the Internal Interface (which will override a scratch directory specified with CDF$TMP/CDF TMP).

The caching scheme used by the CDF library (see Section 2.1.5) affects how these scratch files can impact performance. On machines with large amounts of core memory available, the cache size of a scratch files can be set high enough to result in no blocks actually being written (paged out) to that file. In that case, the scratch file is more like an allocated block of core memory.

### 2.1.5 Caching Scheme

The CDF library reads and writes to open files in 10240-byte blocks. The CDF library for each open file maintains a cache of 10240-byte memory buffers. 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 a 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 (although this is very rare). Having too many cache buffers may also cause problems on machines having limited memory such as the PC and Macintosh.

The CDF library attempts to choose optimal default cache sizes based on a CDF's format and the operating system being used. This is difficult because 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 (or use the CDFinquire toolkit program).

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 spread 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 2.3.16), only one cache buffer would be necessary for the variable file 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 (at least) 10 cache buffers would be optimal.

A similar situation arises when accessing standard variables in a single-file CDF. If values are accessed for each variable at a particular record number, then performance will be improved by setting the number of cache buffers for the dotCDF file to be equal to (or 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.

The Internal Interface is used to select and confirm the cache sizes being used for various files by the CDF library. Confirming a cache size (if it has not been explicitly selected) will determine the default being used. The operations used for each type of file are shown in Table 2.3.

**NOTE:** 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 sizes. It is also important to allocate records for uncompressed variables. This will reduce the fragmentation that can occur in the dotCDF file (which degrades performance because of the increased indexing that occurs). Allocating variable records is described in Section 2.3.12.

| File type | Selecting | Confirming |
|---|---|---|
| dotCDF file[6] | <SELECT_,CDF_CACHESIZE_> | <CONFIRM_,CDF_CACHESIZE_> |
| rVariable file | <SELECT_,rVAR_CACHESIZE_> | <CONFIRM_,rVAR_CACHESIZE_> |
| All rVariable files | <SELECT_,rVARs_CACHESIZE_> | <CONFIRM_,rVARs_CACHESIZE_> |
| zVariable file | <SELECT_,zVAR_CACHESIZE_> | <CONFIRM_,zVAR_CACHESIZE_> |
| All zVariable files | <SELECT_,zVARs_CACHESIZE_> | <CONFIRM_,zVARs_CACHESIZE_> |
| Staging scratch file | <SELECT_,STAGE_CACHESIZE_> | <CONFIRM_,STAGE_CACHESIZE_> |
| Compression scratch file | <SELECT_,COMPRESS_CACHESIZE_> | <CONFIRM_,COMPRESS_CACHESIZE_> |

**Table 2.3 Cache Size Operations, Internal Interface**

## 2.2 CDFs

The following sections describe various aspects of a CDF.

### 2.2.1 Accessing

All supported CDF operations are available using the Internal Interface. The Extended Standard Interface is capable is doing almost all the CDF operations that are available with the Internal Interface. The Original Standard Interface was developed in early 90's and provides a very limited functionality, and its use is not recommended for users who are creating new CDF files. If you have a CDF file that contains only rVariables (very old), you can either use the Original Standard Interface or Internal Interface.

### 2.2.2 Creating

A CDF must be created by the CDF library. In a C application CDFs are created using either the CDFcreateCDF function (in Extended 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_cdf subroutine (in Extended Standard Interface) or the <CREATE_, CDF_> operation of the CDF_lib function (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 CDFopenCDF function (Extended 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_cdf subroutine (Extended Standard Interface) or the <OPEN_, CDF_> operation of the CDF_lib function (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 CDFcloseCDF function (Extended Standard Interface) or the <CLOSE_,CDF_> operation of the CDFlib function (Internal Interface). In a

[6] This also applies to the uncompressed CDF that is maintained as a scratch file.

Fortran application CDFs are closed using either the CDF_close_cdf subroutine (Extended Standard Interface) or the <CLOSE_,CDF_> operation of the CDF_lib function (Internal Interface).

### 2.2.5 Deleting

An open CDF may be deleted at any time. The dotCDF 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 CDFdeleteCDF function (Extended 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_cdf subroutine (Extended Standard Interface) or the <DELETE_,CDF_> operation of the CDF lib function (Internal Interface).

### 2.2.6 Naming

The file name specified when opening or creating a CDF can be any legal file name 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 strings use terminating NUL characters.)

In almost all cases when a CDF file name is specified, the .cdf extension should not be appended.[7] (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 file name specified and then checks to see if that file exists.[8] If not, the CDF library will also check to see if a file exists whose file name 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 file name 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.[9]

### 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 is single-file, and it can be changed if needed. In a user application, the Internal Interface must be used to change the format of a CDF by using the <PUT_,CDF_FORMAT_> operation of the Internal Interface. 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 Appendix A).

**Single-File CDFs**

---

[7] 6The file of a CDF having an extension of .cdf is referred to as the dotCDF file.

[8] 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 (e.g., CDFs) on a CD-ROM visible.

[9] Or .cdf;1 or .CDF or .CDF;1.

A single-file CDF (SINGLE FILE) consists of only one file (with extension .cdf). This file is referred to as the dotCDF file. The dotCDF 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 dotCDF 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 blocks of contiguous records, but these blocks will not be contiguous in the dotCDF file.

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. Because the indexing scheme is now hierarchical,[10] each index entry will point to either another index record (at a lower level in the hierarchy) or to a block of contiguous variable records (at the lowest level of the hierarchy). An index entry consists of the following fields:

| | |
|---|---|
| FirstRecord | The number of the first record in a block of contiguous variable records or the first record indexed in a lower-level index record. |
| LastRecord | The number of the last record in a block of contiguous variable records or the last record indexed in a lower-level index record. |
| ByteOffset | The byte offset within the dotCDF file of the block of contiguous variable records or the byte offset of a lower-level index record. |

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 larger number of records in each block of contiguous variable records (resulting in fewer overall index entries). Techniques used to achieve fewer index entries are outlined in the Allocated Records and Blocking Factor descriptions in Section 2.3.12.

It is possible to inquire the indexing statistics for a variable. Using the Internal Interface, an application may inquire the number of indexing levels in the hierarchy, the number of index records, and total number of entries for a variable using the <GET_,r/zVAR_nINDEXLEVELS_>,[11] <GET_,r/zVAR_nINDEXRECORDS_>, and <GET_,r/zVAR_nINDEXENTRIES_> operations.

**Multi-File CDFs**

A multi-file CDF (MULTI FILE) consists of one file (with extension .cdf referred to as the dotCDF file) 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 dotCDF file.)

**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

[10] As of CDF 2.6.

[11] This notation is used when an operation exists for both rVariables and zVariables. In this case, the actual operations are <GET_,zVAR_nINDEXLEVELS_> and <GET_,rVAR_nINDEXLEVELS_>.

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 the 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 automatically 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 Original Standard Interface but is set to the default encoding for your CDF distribution when created using the Internal Interface or the Extended Standard Interface. The encoding of an existing CDF may be changed with the Internal Interface or the Extended Standard 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.

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.[12] A CDF with any supported encoding is also readable on any computer supported by CDF.

**Host Encodings**

Host encoding (HOST_ENCODING) is the default, and it 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. 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. |
| ALPHAVMSi_ENCODING | DEC Alpha computers running OpenVMS. Double-precision floating-point values are encoded in IEEE representation. |
| ALPHAOSF1_ENCODING | DEC Alpha computers running OSF/1. |
| SUN_ENCODING | Sun computers. |
| SGi_ENCODING | Silicon Graphics Iris and Power Series computers. |
| DECSTATION_ENCODING | DECstation computers. |
| IBMRS_ENCODING | IBM RS6000 series computers. |

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

| | |
|---|---|
| HP_ENCODING | HP 9000 series computers. |
| PC_ENCODING | 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. All host encodings are readable and writeable on any machine supported by CDF.

**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 the network encoding. Network encoded CDFs are readable and writeable on any machine supported by CDF (as are all of the other encodings).

**Equivalent Encodings**

While an encoding 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, CDF UINT4, CDF INT8, and CDF TIME_TT2000) 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 2.4. Network (XDR) encoding uses big-Endian encoding for multiple-byte integer data types.

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

**Table 2.4 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 2.5. 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 | DEC Alpha (OSF/1) | DEC Alpha / OpenVMS/D |
| IBM RS6000 | DEC Alpha (OpenVMS/I) | DEC Alpha / OpenVMS/G |
| HP 9000 | Mac OS X | |
| NeXT | | |
| Macintosh PPC | | |
| (Network - XDR) | | |

**Table 2.5 Equivalent Single-Precision Floating-Point Encodings**

**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 2.6. Network (XDR) encoding uses IEEE 754 (big-Endian) encoding for double-precision floating-point data types.

| IEEE 754 (Big Endian) | IEEE 754 (Little Endian) |
|---|---|
| Sun | DECstation |
| SGi Iris | PC |
| IBM RS6000 | DEC Alpha/OSF/1 |
| HP 9000 | DEC Alpha/OpenVMS/I |
| NeXT | Mac OS X |
| Macintosh PPC | |
| (Network - XDR) | |
| Digital's D FLOAT | Digital's G FLOAT |
| VAX | DEC Alpha/OpenVMS/G |
| DEC Alpha/OpenVMS/D | |

**Table 2.6 Equivalent Double-Precision Floating-Point Encodings**

**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). 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 the CDFconvert utility included in the CDF standard distribution package) 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. This is the default. |
| 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. |
| ALPHAVMSi_DECODING | DEC Alpha running OpenVMS data representation. Double-precision floating- point values will be in IEEE representation. |
| ALPHAOSF1_DECODING | DEC Alpha running OSF/1 data representation. |
| SUN_DECODING | Sun data representation. |
| SGi_DECODING | Silicon Graphics Iris and Power Series data representation. |
| DECSTATION_DECODING | DECstation data representation. |
| IBMRS_DECODING | IBM RS6000 series data representation. |
| HP_DECODING | HP 9000 series data representation. |
| PC_DECODING | 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 Compression

A compression may be specified for individual variables and/or a single-file CDF that is performed when the CDF is closed and written to disk.[13] When compression is specified for a CDF, the CDF library maintains an uncompressed version of the dotCDF file in a scratch file. When the CDF is closed, the uncompressed dotCDF file is compressed and written to the file with the name specified when the CDF was opened/created. If the application program closing the CDF were to abnormally terminate before the dotCDF file was successfully compressed and written, the uncompressed dotCDF scratch file would remain in the scratch directory. The scratch directory used by the CDF library is described in Section 2.1.4.

Overall compression for a CDF is specified with the <PUT_,CDF_COMPRESSION_> operation of the Internal Interface. It may be respecified as often as desired. A CDF's overall compression may be inquired using the <GET_,CDF_COMPRESSION_> operation for an open CDF and the <GET_,CDF_INFO_> operation for a CDF that has not been opened (which saves the overhead of actually decompressing the CDF). The available compression algorithms are described in Section 2.6.

### 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:[14]

| | |
|---|---|
| CDF_MAX_DIMS | The maximum number of dimensions that rVariables/zVariables may have. |
| CDF_VAR_NAME_LEN256 | The maximum number of characters in a variable name. This limit was extended in CDF 3.0 to allow to create a longer variable name. |
| CDF_ATTR_NAME_LEN256 | The maximum number of characters in an attribute name. This limit was extended in CDF 3.0 to allow to create a longer attribute name. |
| CDF_PATHNAME_LEN | The maximum number of characters in the name of a file used to specify a CDF. |

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

## 2.3 Variables

CDF's "variable" is a generic name for an object that represents data where data can be 0-dimensional (scalar data) or multi-dimensional (up to 10-dimension), and it does not have any scientific context associated with it. For example, a

[13] Compression is not allowed with multi-file CDFs.
[14] 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 PC because of the 8.3 naming convention.

variable can be data representing an independent variable, a dependent variable, time and date value, or whatever data might be (e.g. image, XML file, etc.). In other words, a variable doesn't contain any hidden meanings other than the data itself. One may describe one variable's relationship with other variable(s) through "attributes".

There are two types of variables (rVariable and zVariable) in CDF, and they can happily coexist in a CDF: Every rVariable in a CDF must have the same number of dimensions and dimension sizes while each zVariable can have its own dimensionality. Since all the rVariables in a CDF must have the same dimensions and dimension sizes, there'll be a lot of disk space wasted if a few variables need big arrays and many variables need small arrays. Since zVariable is more efficient in terms of storage and offers more functionality than rVariable, use of zVariable is strongly recommended. As a matter of fact, there's no reason to use rVariables at all if you are creating a CDF file from scratch. One may wonder why there are rVariables and zVariables, not just zVariables. When CDF was first introduced, only rVariables were available. The inefficiencies with rVariables were quickly realized and addressed with the introduction of zVariables in later CDF releases.

### 2.3.1 Types

With the introduction of compression and sparseness for variables, there now exist several different types of variables (in addition to the distinction between rVariables and zVariables). The various types of variables are as follows. . .

| | |
|---|---|
| "standard variable" | A variable in a single-file CDF that is not compressed nor has sparse records or arrays. |
| "compressed variable" | A variable in a single-file CDF that is compressed and may or may not have sparse records (but cannot have sparse arrays). |
| "variable with sparse records" | A variable in a single-file CDF that has sparse records and may be compressed, have sparse arrays, or have neither. |
| "variable with sparse arrays" | A variable in a single-file CDF that has sparse arrays and may or may not have sparse records (but cannot be compressed). |
| "multi-file variable" | A variable in a multi-file CDF. It cannot be compressed, have sparse records, or have sparse arrays. |

The term "variable" is used when a discussing a property that applies to all of the various variable types.

### 2.3.2 Accessing

The Original Standard Interface deals exclusively with rVariables while the Extended Standard Interface deals zVariables. The Internal Interface may be used to access either rVariables or zVariables.

### 2.3.3 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.

### 2.3.4 Closing.

The CDF library automatically closes the variable files in a multi-file CDF when the CDF itself is closed by an application.[15] 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 2.1.5). 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 function (Internal Interface). zVariables are closed using the <CLOSE_,zVAR_> operation of the CDF lib function (Internal Interface).

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

### 2.3.5 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 or CDF_VAR_NAME_LEN256 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.

### 2.3.6 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.

### 2.3.7 Deleting

A variable may be deleted from a single-file CDF.[16] 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 that numerically follow the variable being deleted are renumbered immediately. (Each is decremented by one.)

### 2.3.8 Dimensionality

[15] It is required that an application close a CDF before exiting.
[16] Variables may not currently be deleted from a multi-file CDF.

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.

A zVariable may have a dimensionality that 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], 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.

### 2.3.9 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 2.3.20).

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 2.5.5.

**Data Type**

The supported data types are described in Section 2.5. 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 data types (CDF CHAR and CDF UCHAR) 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 data types, this must always be one (1). An array of elements per variable value is not allowed for non-character 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 2.3.12 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 2.1 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 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 that 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.)

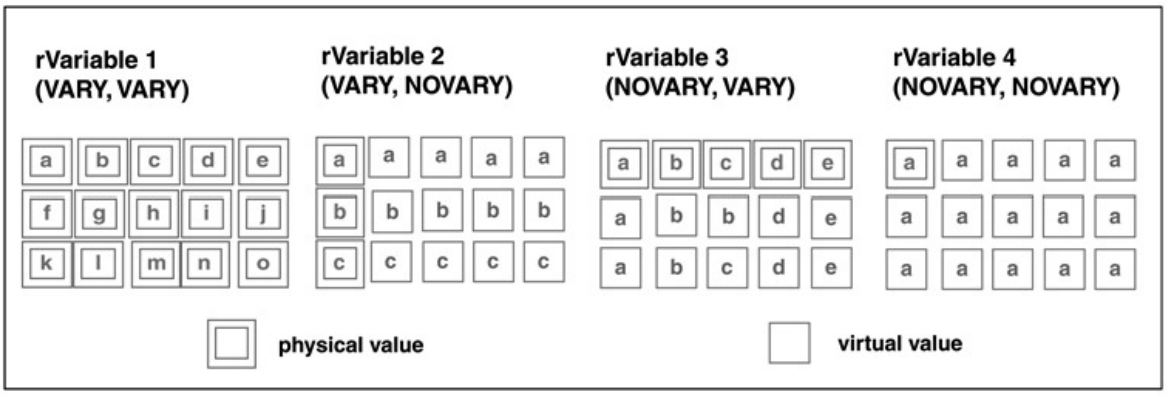


**Figure 2.1 Physical vs. Virtual Dimensions**

### 2.3.12 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.

3. If a variable has sparse records, then any unwritten records for that variable are virtual and contain either the pad value for that variable or the previous existing record's values (depending on the type of sparse records). Sparse records are described on page 48.

Record variance is described in Section 2.3.10. Variable pad values are described in Section 2.3.20.
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 2.2 illustrates the relationships between physical and virtual records for a standard variable. 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.

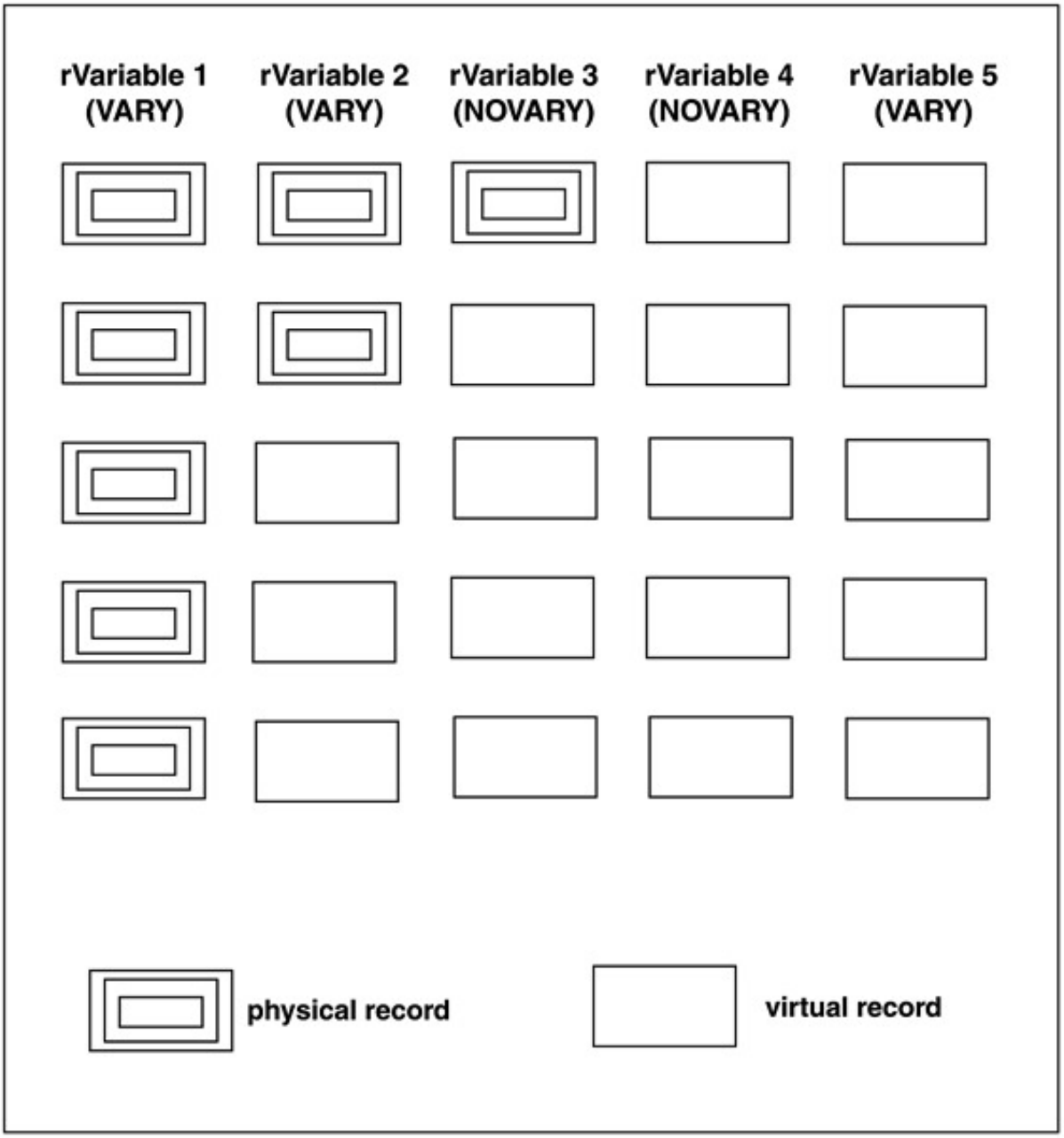


**Figure 2.2 Physical vs. Virtual Records, Standard Variable**

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 not having sparse records with a **VARY** record variance, if a new record more than one record beyond the current maximum record for the variable is written, the intervening records will be physically written and contain the pad value for that variable. For a variable having sparse records, only those records written by an application are physically stored. Unwritten records are virtual as described in **Sparse Records** on 48.

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 (if any). The remaining values in the record may or may not be subsequently written. Variable pad values are described in Section 2.3.20.

**Numbering**

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

**Sparse Records**

A variable in a single-file CDF can be specified as having sparse records.[17] If so, then only those records that are explicitly written to the variable will be physically stored. If a variable is not specified as having sparse records, then all of the records up to the maximum written will be physically stored. Sparse records are only allowed in single-file CDFs (where the indexing scheme used for variable records makes this possible). Considerable disk space can be saved in the dotCDF file for a variable that has gaps of missing data if that variable is specified as having sparse records.

For an uncompressed variable having sparse records, it is also beneficial if the blocks of records that are going to be written can first be allocated. This will allow the CDF library to optimize the indexing for the variable. Otherwise, the CDF library will use the staging scratch file to minimize the indexing needed. Note that records cannot be allocated for compressed variables (whether or not they have sparse records).

Two types of sparse records can be specified for a variable. They differ only in how unwritten records are presented in the conceptual view of the variable. These missing records are considered virtual records just like the records beyond the last record written. Pad-missing sparse records specifies that when a virtual record is read the variable's pad value should be returned. Previous-missing sparse records specifies that when a virtual record is read the previous existing record's values should be returned. If a previous record does not exist, the variable's pad value will be returned.

Note that previous-missing sparse records can also be used to save disk space for a variable if that variable's values do not change from record to record except occasionally. If the only records written were those that changed from the previous record, then the virtual records following each record actually written (physically stored) would all have the same value(s). This could save considerable disk space if the values do not change often. For example, consider a 0-dimensional variable having previous-missing sparse records that is being used to store temperature data. Each record corresponds to a temperature reading at a given time. Table 2.7 shows how the variable might appear conceptually along with which records are physically stored. Note that only three records are physically stored but that nine records appear in the conceptual view of the variable.

Sparse records are specified for a variable using the <PUT_,r/zVAR_SPARSERECORDS_> operation of the Internal Interface. One of the following types of sparse records must be specified:

| | |
|---|---|
| NO_SPARSERECORDS | The variable does not have sparse records. |
| PAD_SPARSERECORDS | The variable has pad-missing sparse records. The notation sRecords.PAD is used by the CDF toolkit for pad-missing sparse records. |
| PREV_SPARSERECORDS | The variable has previous-missing sparse records. The notation sRecords.PREV is used by the CDF toolkit for previous-missing sparse records. |

[17] Sparse records are not allowed for a variable in a multi-file CDF.

| Record | Temperature | |
|---|---|---|
| 1 | 101.4 | (Physical) |
| 2 | 101.4 | (Virtual) |
| 3 | 101.5 | (Physical) |
| 4 | 101.5 | (Virtual) |
| 5 | 101.5 | (Virtual) |
| 6 | 101.5 | (Virtual) |
| 7 | 101.5 | (Virtual) |
| 8 | 101.6 | (Physical) |
| 9 | 101.6 | (Virtual) |

**Table 2.7 Previous-missing Sparse Records Example, Conceptual View vs. Physical Storage**

The <GET_,r/zVAR_SPARSERECORDS_> operation can be used to inquire the type of sparse records.

**Allocated Records**.

The Internal Interface may be used to allocate records for an uncompressed variable in a single-file CDF[18] 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 normally expected to write to all of the allocated records. For NRV variables, only one record may be allocated (because only one record will ever physically exist). If the variable has sparse records, only those blocks of records that are going to be written would be allocated. Records cannot be allocated by an application for compressed variables because they are allocated automatically by the CDF library when their compressed size is known.

Performance is improved when using this method because the allocated records will be as contiguous as possible requiring the fewest number of index entries. 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.

Note that records do not have to be allocated by an application before they are written to a variable. The CDF library will automatically allocate any needed records based on the variable's blocking factor. Also, records may be allocated at any time (not only before records have been written as in previous CDF releases).

Records are allocated using the <PUT_,r/zVAR_ALLOCATERECS_> and <PUT_,r/zVAR_ALLOCATEBLOCK_> operations of the Internal Interface. The number of records allocated for a variable can be inquired using the <GET_,r/zVAR_NUMallocRECS_> operation. The maximum record allocated for a variable can be inquired using the <GET_,r/zVAR_MAXallocREC_> operation. The exact records allocated for a variable can be determined using a combination of the <GET_,r/zVAR_ALLOCATEDTO_> and <GET_,r/zVAR_ALLOCATEDFROM_> operations.

**Initial Records**

The Internal Interface may be used to specify an initial number of records to be written for a variable.[19] 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 explicitly specified for the variable. If a specific pad value is desired for a variable, then it must be specified before the number of initial records is specified. Also, any compression or sparseness for the

[18] There is no reason to allocate records for a variable in a multi-file CDF.
[19] The use of allocated records would in most cases be more efficient than specifying initial records.

variable must be specified before writing the initial records because those properties cannot be changed after records have been written.

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 as contiguously as possible 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. This is because 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. For NRV variables the number of initial records must always be specified as one (1). This is because only one physical record will ever actually be written. Initial records for a variable may be specified only once.

Initial records are written to variables using the <PUT_,r/zVAR_INITIALRECS_> operation of the Internal Interface. Explicit pad values are specified using the <PUT_,r/zVAR_PADVALUE_> operation.

**Blocking Factor**.

A variable's blocking factor[20], a number of records, affects how its records are allocated in the CDF file(s). For NRV variables the blocking factor is not applicable because only one physical record will ever exist. For variables in a multi-file CDF the blocking factor is not used because only those records written by an application will exist in the variable files. But for the other types of variables in a single-file CDF, the blocking factor can have a significant impact on I/O performance. The following sections will describe how a variable's blocking factor is used in each case.

**Standard Variables** Space in the dotCDF file for records written to a standard variable is either allocated explicitly by an application or automatically by the CDF library. If the records are allocated by the application the exact number needed can be specified. This can be used to optimize the indexing for the variable resulting in fewer (or even just one) index entries that must be searched when accessing the variable. If the records are not allocated by the application, however, they must be automatically allocated by the CDF library. Because the CDF library wants to optimize the indexing for a variable, it may allocate additional records beyond those needed at the time in an attempt to minimize the number of index entries. The variable's blocking factor specifies the minimum number of records to allocate when an application writes to an unallocated record. This is based on the assumption that the addition records allocated will eventually be written. If that is not the case, the allocated but unwritten records will simply waste space in the dotCDF file. The best way to prevent that situation is for an application to explicitly allocate the records that are going to be written. An application can specify a blocking factor for a variable or let the CDF library use a default blocking factor. Note that setting the blocking factor too low (and not allocating the records being written) may result in excessive indexing for a variable. Even using the default blocking factor for a variable may result in excessive indexing unless the records to be written are first allocated. The indexing scheme used by the CDF library is described in Section 2.2.7.

**Compressed Variables** The blocking factor for compressed variables specifies the number of records that will be compressed together. The CDF library stages the records of a compressed variable in a scratch file. The number of records in the staging area is also based on the variable's blocking factor. When necessary, the CDF library compresses the records in the staging area and writes the compressed block of records to the dotCDF file. Each block of compressed records has an associated index entry (see Section 2.2.7). Setting the blocking factor high will minimize the indexing for a variable but will increase the time needed to access an individual record because the entire block in which it is compressed will have to be decompressed. If the blocking factor is too low, the decompression of an individual record will not take as long but excessive indexing may result (which will increase the access overhead). Also, most compression algorithms work better as the number of records (bytes) being compressed is increased. Note that if the compressed variable also has sparse records, the blocking factor becomes the maximum number of records per compressed block. Depending on which records are written some of the compressed blocks may contain fewer

[20] A variable's blocking factor was previously called its "extend records."

records. The blocking factor for a compressed variable may be explicitly specified by an application or a default may be used as determined by the CDF library (the default blocking size for compressed variable, 5120 bytes, decided by the record size). Using the default for the compressed variables may be too small and inefficient if writing a large number of records. Once a record has been written to the variable, however, the blocking factor cannot be changed.

**Uncompressed Variables With Sparse Records** The CDF library uses a staging area scratch file for uncompressed variables with sparse records. This is done in an attempt to minimize the indexing for the variable (as described in Section 2.2.7) when the records being written are not first allocated by an application. The blocking factor specifies the number of records to be maintained in the staging area for the variable (which will be the maximum number of unallocated consecutive records that would be stored contiguously in a block when written by an application). An explicit blocking factor can be specified or a default determined by the CDF library may be used.

Blocking factors are explicitly specified for variables using the <PUT_,r/zVAR_BLOCKINGFACTOR_> operation of the Internal Interface. The blocking factor may be inquired using the <GET_,r/zVAR_BLOCKINGFACTOR_> operation. If an explicit blocking factor has not been specified, the default blocking factor for the variable will be returned.

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. Both the number of records written to a variable and the number of records allocated for that variable may be inquired using the Internal Interface.

**Deleting**

The records of a variable in a single-file CDF may be deleted.[21] If the variable has sparse records, the deleted records simply cease to exist. A gap of one or more missing records will be formed. But if the variable does not have sparse records, the records following the block of deleted records are immediately renumbered to fill in the gap created. The record numbers remain consecutive without a gap.

Variable records are deleted using the <DELETE_,r/zVAR_RECORDS_> operation of the Internal Interface.

### 2.3.13 Sparse Arrays

The idea being that only those values actually written to a variable array (record) will be physically stored. Currently, unwritten values in each variable array are physically stored using the variable's pad value. Note that specifying a compression for a variable will in many cases result in a disk space savings similar to that of sparse arrays. The exact differences in disk space savings and execution overhead between sparse arrays and variable compression will not be known until sparse arrays have been implemented. No effort is planned to add this feature to the CDF.

### 2.3.14 Compression

A compression may be specified for a variable in a single-file CDF that gets performed automatically as values are written.[22] The values are transparently decompressed as they are read from the variable. The values of a variable are compressed in blocks of one or more variable records. The blocking factor for a compressed variable (described beginning on page 50) specifies the number of records in each block (or the maximum number in the case of a compressed variable with sparse records). Properly setting the blocking factor involves a trade-off between the compression percentage achieved and execution speed when accessing values in individual variable records. The CDF

[21] Variable records may be deleted from a multi-file CDF.
[22] Note that variable compression is not allowed in a multi-file CDF.

library also uses a staging area scratch file to minimize access overhead for a compressed variable. Note that if a block of variable records actually increases in size when compressed, the block of records will be stored uncompressed in the CDF. This could happen if the blocking factor is set too low or simply because of the nature of the data and the compression algorithm being used.

The compression for a variable is specified with the <PUT_,r/zVAR_COMPRESSION_> operation of the internal interface. A variable's compression may be inquired with the <GET_,r/zVAR_COMPRESSION_> operation. Section 2.6 describes the available compression algorithms.

**Reserve Percentage.**

If a value in a compressed block of records is changed, the amount of compression achieved for that block may also change. If it increases, the block of compressed records may have to be moved in the dotCDF file. This will most likely result in the dotCDF file increasing in size if the block of compressed records is placed at the end (leaving a block of unused bytes where the compressed block of records previously existed). This is not a desirable situation considering that the variable compression is supposed to make the CDF smaller. To alleviate this potential problem a reserve percentage may be selected for a compressed variable. When a compressed block of variable records is initially written to the dotCDF file some additional space will be allocated. This will allow that block of compressed records to expand in size if necessary. The reserve percentage is interpreted as follows:

| | |
|---|---|
| 0 | No reserve space is allocated. This is the default. |
| 1..100 | Allocates that percentage of the uncompressed size of the block of variable records (as a minimum). For example, if a 1000-byte block of records compressed down to 600 bytes and the reserve percentage is 70%, then 700 bytes would actually be allocated for the block in the dotCDF file. If the reserve percentage is 50%, then 600 bytes would of course still have to be allocated. |
| 101… | Allocates that percentage of the size of the compressed block of variable records but not exceeding the uncompressed size. For example, if a 1000- byte block of records compressed down to 800 bytes and the reserve percentage is 110%, then 880 bytes would be allocated for the block. |

Even specifying a reserve percentage for a compressed variable does not guarantee that the problem with moving blocks of compressed records as the variable's values are changed will be avoided. If a CDF does become fragmented in this way remember that the CDFconvert utility can always be used to create a new CDF with each variable's compression being optimized (e.g., no fragmentation).

The reserve percentage for a compressed variable is selected with the <SELECT_,r/zVAR_RESERVEPERCENT_> operation. A variable's reserve percentage may be confirmed with the <CONFIRM_,r/zVAR_RESERVEPERCENT_> operation.

### 2.3.15 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. The default variable majority is row-major.

| | |
|---|---|
| ROW_MAJOR | Row majority. The first dimension changes the slowest. |
| COLUMN_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[23] 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.)

### 2.3.16 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:

| | |
|---|---|
| RecordNumber | The record number at which to perform the access. |
| DimensionIndices | The indices within the record at which to perform the access. |

For 0-dimensional variables, 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

[23] Since an application knows how it will be accessing a variable, it knows best how to optimize the caching scheme used. See Section 2.1.5 for details on how an application can control the CDF library caching scheme.

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 must be performed using the <GET_,zVAR_DATA_> and <PUT_,zVAR_DATA_> operations of CDFlib. 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 function (Internal Interface). Single value access for zVariables must be performed using the <GET_,zVAR_DATA_> and <PUT_,zVAR_DATA_> operations of CDF lib.

### 2.3.17 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:

| | |
|---|---|
| RecordNumber | The record number at which to start the access. |
| RecordCount | The number of records to access. |
| RecordInterval | The interval between records being accessed. An interval of two (2) would indicate that every other record is to be accessed. |
| DimensionIndices | The indices within each record at which the access should begin. |
| DimensionCounts | The number of values along each dimension that should be accessed. |
| DimensionIntervals | 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 variables, 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 writes for rVariables are performed using the CDFvarHyperPut function (Standard Interface) or the <PUT_,rVAR_HYPERDATA_> operation of the CDFlib function (Internal Interface). Hyper writes for zVariables must be performed using the <PUT_,zVAR_HYPERDATA_> operation of CDFlib. Hyper reads for rVariables are performed using the CDFvarHyperGet function (Standard Interface) or the <GET_,rVAR_HYPERDATA_> operation of CDFlib. Hyper reads for zVariables must be performed using the <GET_,zVAR_HYPERDATA_> operation of CDFlib.

In a Fortran application, hyper writes for rVariables are performed using the CDF_var_hyper_put subroutine (Standard Interface) or the <PUT_,rVAR_HYPERDATA_> operation of the CDF lib function (Internal Interface). Hyper writes for zVariables must be performed using the <PUT_,zVAR_HYPERDATA_> operation of CDF lib. Hyper reads for

rVariables are performed using the CDF_var_hyper_get subroutine (Standard Interface) or the <GET_,rVAR_HYPERDATA_> operation of CDF lib. Hyper reads for zVariables must be performed using the <GET_,zVAR_HYPERDATA_> operation of CDF lib.

### 2.3.18 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).

If the variable has sparse records, the virtual records in a gap of missing records are not skipped. The type of sparse records (see Section 2.3.12) will determine the values returned. When a virtual record in a gap of missing records is read, the informational status code VIRTUAL RECORD DATA is returned (rather than END OF VARIABLE). Sequential writes will create any necessary record in a gap of missing records (i.e., sequential writes do not skip virtual records in a gap of missing records).

**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 variables is performed using the <GET_,r/zVAR_SEQDATA_> and <PUT_,r/zVAR_SEQDATA_> operations of the Internal Interface.

### 2.3.19 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.

| | |
|---|---|
| VariableCount | The number of rVariables/zVariables that are being accessed. |
| VariableList | The rVariables/zVariables being accessed (specified by number). |
| RecordNumbers | 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 that 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 INT2/14[24] | 0:[] | T/ |
| zVar2 | CDF_CHAR/7 | 1:[5] | T/T |
| ZVar3 | CDF REAL8/1 | 2:[2,4] | T/TT |

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, and 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).

Multiple variable writes are performed using the <PUT_,r/zVARs_RECDATA_> operation of the Internal Interface. Multiple variable reads are performed using the <GET_,r/zVARs_RECDATA_> operation. The selection of record numbers is performed using the <SELECT_,r/zVARs_RECNUMBER_> operation.

### 2.3.20 Variable Pad Values.

Variable pad[25] values are used in several situations. .

1. When the first value is written to a new record (for records containing multiple values), 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. For a variable not having sparse records, when a new record is written that is more than one record beyond the last record already written, the intervening records will also be written and will contain pad values. This does not apply to NRV variables because only one physical record actually exists.

[24] 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).
[25] These were previously known as fill values but were renamed to avoid confusion with the FILLVAL attribute.

3. For a variable having the pad-missing style of sparse records, if a record is read from a gap of missing records, pad values will be returned. The previous-missing style of sparse records would cause the previous existing record's values to be returned (unless there is no previous record in which case pad values would be returned).

4. When reading a record beyond the last record written for a variable, pad values will be returned except if the variable has the previous-missing style of sparse records. In that case, the last written record's values are returned (unless there are no written records in which case pad values are returned).

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 are shown in Table 2.8.[26]

| Data Type | Default Pad Value |
|---|---|
| CDF_BYTE | 0 |
| CDF_INT1 | 0 |
| CDF_UINT1 | 0 |
| CDF_INT2 | 0 |
| CDF_UINT2 | 0 |
| CDF_INT4 | 0 |
| CDF_UINT4 | 0 |
| CDF_INT8 | 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 |
| CDF_EPOCH16 | 01-Jan-0000 00:00:00.000.000.000.000 |
| CDF_TIME_TT2000 | 0000-01-01T00:00:00.000000000[27] |
| CDF_CHAR | " " (space character) |
| CDF_UCHAR | " " (space character) |

**Table 2.8 Default Pad Values.**

Variable pad values are specified using the <PUT_,r/zVAR_PADVALUE_> operation of the Internal Interface. The pad value being used for a variable can be inquired with the <GET_,r/zVAR_PADVALUE_> operation. If a pad value has not been explicitly specified for a variable, the default pad value (based on the variable's data type) will be returned along with the NO_PADVALUE_SPECIFIED informational status code. The existence of an explicitly specified pad value can be confirmed for a variable (without actually inquiring the value) using the <CONFIRM_,r/zVAR_PADVALUE_> operation.

## 2.4 Attributes

CDF attributes are the mechanism for storing metadata. A new attribute may be created in a CDF at any time.

[26] These default pad values can be changed by your system manager when the CDF distribution is built.
[27] It is a value of –9223372036854775807, one more than the minimum value for an 8-byte integer. We present it a such date/time, an invalid date for CDF_TIME_TT2000 data type, which is similar to CDF_EPOCH/EPOCH16 data type.

### 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 or CDF_ATTR_NAME_LEN256 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).

### 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.[28] 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.

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

### 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 that 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 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 2.5. The entries for an attribute may have any combination of data specifications.

For character 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 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 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.

### 2.5.1 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. |
| CDF_INT8 | 8-byte, signed integer. |

**NOTE:** When using C on a 64-bit operating system (e.g. DEC Alpha running OSF/1, and Linux 64-bit on Intel), 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*.

### 2.5.2 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 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.

### 2.5.3 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. Currently, the character set supported by the CDF is limited to **ASCII** set of characters. Non-conforming characters are either rejected or will not be properly handled/displayed by the CDF library.

### 2.5.4 EPOCH Data Types

| | |
|---|---|
| CDF_EPOCH | 8-byte, double precision floating point. |
| CDF_EPOCH16 | two 8-byte, double precision floating point. |

The CDF_EPOCH and CDF_EPOCH16 data types are used to store date and time values referenced from a particular epoch. For CDF that epoch is 01-Jan-0000 00:00:00.000 and 01-Jan-0000 00:00:00.000.000.000.000, respectively..[29]

CDF_EPOCH values are the number of milliseconds since the epoch. The standard format used to display a CDF_EPOCH value is

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

where dd is the day of the month (01-31), mmm is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec), yyyy is the year (0000-9999) hh is the hour (00-23), mm is the minute (00-59), ss is the second (00-59), and ccc is the millisecond (000-999).

CDF_EPOCH16 values are the number of picoseconds since the epoch. The standard format used to display a CDF_EPOCH16 value is

dd-mmm-yyyy hh:mm:ss.ccc.mmm.nnn.ppp

where dd is the day of the month (01-31), mmm is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec), yyyy is the year (0000-9999) hh is the hour (00-23), mm is the minute (00-59), ss is the second (00-59), ccc is the millisecond (000-999)., mmm is the microsecond (000-999)., nnn is the nanosecond (000-999)., and ppp is the picosecond (000-999).

Functions exist that parse, encode, compute, and decompose CDF_EPOCH and CDF_EPOCH16 values. These functions are described in the CDF C Reference Manual for C applications and in the CDF Fortran Reference Manual for Fortran applications.

### 2.5.5 TT2000 Data Types

| | |
|---|---|
| CDF_TIME_TT2000 | 8-byte, signed integer. |

The CDF_TIME_TT2000 data type, an alternative to CDF_EPOCH and CDF_EPOCH16, is used to store date and time values referenced from J2000 (2000-01-01T12:00:00.000000000). The values also include leap seconds. Please read our requirements analysis at http://cdf.gsfc.nasa.gov/html/leapseconds_requirements.html and development approach at http://cdf.gsfc.nasa.gov/html/leapseconds.html for more details.

[29] 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 (or simply year 0) rather than 0 AD. Also, year 0 is considered to be a leap year.

CDF_TIME_TT2000 values are the number of nanoseconds since J2000. The standard format used to display a CDF_TIME_TT2000 value is in ISO 8601 format

yyyy-mm-ddThh:mm:ss.cccuuunnn

where yyyy is the year (1707-2292), mm is the month (01-12),dd is the day of the month (01-31), hh is the hour (00-23), mm is the minute (00-59), ss is the second (00-59 or 00-60 if leap second), ccc is the millisecond (000-999), uuu is microseconds (000-999) and nnn is nanoseconds (000-999).

Functions exist that parse, encode, compute, and decompose CDF_TIME_TT2000 values. These functions, similar to those for CDF_EPOCH and CDF_EPOCH16 data types, are described in the CDF C Reference Manual for C applications and in the CDF Fortran Reference Manual for Fortran applications.

### 2.5.6 Equivalent Data Types

Certain data types are considered equivalent with respect to their representation in memory and in a CDF. Table 2.9 shows the groups of equivalent data types.

| | | | | | |
|---|---|---|---|---|---|
| CDF_CHAR<br>CDF_UCHAR<br><br>CDF_INT1<br>CDF_UINT1<br>CDF_BYTE | CDF_INT2<br>CDF_UINT2 | CDF_INT4<br>CDF_UINT4 | CDF_INT8<br>CDF_TIME_TT 2000 | CDF_REAL4<br>CDF_FLOAT | CDF_REAL8<br>CDF_DOUBLE<br><br>CDF_EPOCH<br>CDF_EPOCH16 |

**Table 2.9 Equivalent Data Types**

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.

## 2.6 Compression Algorithms

Several compression algorithms are supported by the CDF library. Selecting the proper algorithm to use will depend on the characteristics of the data being compressed. Experimentation with the available algorithms on the CDF or variable being compressed will also be necessary. The following sections describe each compression algorithm, any associated parameters, and the types of data for which they are appropriate.

### 2.6.1 Run-Length Encoding

The run-length encoding compression algorithm, RLE_COMPRESSION, takes advantage of repeating bytes in the data. Currently, only the run-length encoding of zeros (0's) is supported. RLE_COMPRESSION has one parameter that must be set to RLE_OF_ZEROs. The notation RLE.0 is used for this type of RLE compression.

### 2.6.2 Huffman

The Huffman compression algorithm, HUFF_COMPRESSION, takes advantage of the frequency at which certain byte values occur in the data. A sequence of bytes that contain a high percentage of a limited number of byte values will compress better than if each byte value occurs with equal probability. HUFF_COMPRESSION has one parameter that must be set to OPTIMAL_ENCODING_TREES.[30] The notation HUFF.0 is used for this type of HUFF compression.

### 2.6.3 Adaptive Huffman

The adaptive Huffman compression algorithm, AHUFF_COMPRESSION, also takes advantage of the frequency at which certain byte values occur in the data. AHUFF_COMPRESSION is very similar to HUFF_COMPRESSION and generally provides slightly better compression. AHUFF_COMPRESSION has one parameter that must be set to OPTIMAL_ENCODING_TREES. The notation AHUFF.0 is used for this type of AHUFF compression.

### 2.6.4 GZIP

The Gnu ZIP compression algorithm, GZIP_COMPRESSION, uses the Lempel-Ziv coding (LZ77) taking advantage of common substrings within the data. Significant compression occurs over a wide variety of data sets. GZIP_COMPRESSION has one parameter which may be set to a level value in the range from 1 (one) to 9 (nine). 1 provides the least amount of compression and executes the fastest. 9 provides the most compression but executes the slowest. Levels between 1 and 9 allow for a trade-off between compression and execution speed. The notation GZIP.<level> is used for GZIP compression where <level> is a value from 1 to 9. For example, GZIP.7 specifies a level of 7.

Tests have shown that GZIP compression always provides a much better compression ratio. To balance the execution speed and compression ratio, a compression level of **six** (6), the default level for Unix/Linux gzip command, is suggested.

From CDF **V3.5.0**, the open source **ZLIB** package by **Jean-loup Gailly** and **Mark Adler** is used as the sole source code for GZIP compression/decompression. The original CDF library code, which was modified from the zip and unzip code by the same authors, is no longer used. Only needed source codes from ZLIB papackage are extracted and distributed with CDF. Please refer to **ACKNOWLEDGMENT.txt** in src/lib/zlib directory of the source package for information.

## 2.7 TT2000 and Leap Seconds

The CDF_TIME_TT2000 data type is defined as an 8-byte signed integer with a fixed Time_Base=J2000 (Julian date 2451545.0 TT or 2000 January 1, 12h TT), Resolution=nanoseconds, Time_Scale=Terrestrial Time (TT), Units=nanoseconds, Reference_Position=rotating Earth Geoid, with leap seconds included. Given a current list of leap seconds, conversion between TT and UTC is straightforward (TT = UTC + deltaAT + 32.184s, where deltaAT is the sum of the leap seconds since 1960; for example, for 2009, deltaAT = 34s). Use of an 8-byte integer provides time with nanosecond resolution for the next 280 years, so data providers will no longer need 16-byte CDF_EPOCH16 variables to carry their highest time resolution (and in half the storage space).

The leap seconds are determined by CDF through a leap second table. The text-formed table, a part of the CDF distribution since CDF V3.3.2, is also available from CDF site. The table is accessed by the CDF library **externally** through an environment variable, **CDF_LEAPSECONDSTABLE** on Unix/Windows (**CDF$LEAPSECONDSTABLE** on VMS). If the environment variable is not defined or the table file is not found, the **internal** table within the CDF library is used. If the CDF version is released after the last leap second was added, the internal table should be identical to external table. The tool program, CDFleapsecondsinfo at Section 3.15, will show how the table is used.

[30] OPTIMAL ENCODING TREES causes each buffer of data to be scanned for the best possible compression. An alternative method would be to scan the first buffer being compressed and then use the same byte value frequencies for subsequent buffers.

When a new leap second is added, the table will be updated and a new CDF library version will be released. Using the updated, external table with existing CDF version will continue to produce correct TT2000 data. Using a un-updated external/internal table will cause a problem if the epoch data is after the date the new leap second is added.

# Chapter 3

# 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 describe how to use the CDF tools in the toolkit. Two versions of the toolkit (command-line version and GUI-version) are included as part of the standard CDF distribution package, and the CDF tools described in this chapter are the command-line version. The Graphical User Interface (GUI) version of the CDF tools are not described here since they are self-explanatory.

The Graphical User Interface (GUI) version of the CDF toolkit (written in Java) is available starting with CDF 2.7, and a complete set of the toolkit is available for Unix and Macintosh OS X systems. The Windows operating system has its own complete set of GUI-based toolkit in CDFfsi.exe and CDFso.exe programs. In addition, a Java version of CDFedit and CDFexport programs are also included in the Windows distribution package. The Java version of CDFedit and CDFexport is recommended over the ones included in CDFfsi.exe since they are much more intuitive and easier to use.

### 3.1.1 VMS and UNIX (including Mac OS X)

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 that 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 that 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, 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 string will be necessary in the file names (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 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.

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

10. On some UNIX systems, it may be necessary to execute "stty tab3" before running CDFedit or CDFexport.

11. Some of the toolkit programs have a "paging" qualifier. Paging is not allowed if the output of the program has been directed to a file.

12. Most toolkit programs have an "about" qualifier that can be used to determine the CDF distribution from which the program came. On the Macintosh, an "about" selection is available on the "apple" pull-down menu.

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 systems you should have source'd (or equivalent) the script file named definitions.<shell-type> file located in the <cdf_install_dir>/bin directory where <shell-type> is the type of shell you are using: C for the C-shell (csh) and tcsh, K for the Korn (ksh), BASH, and POSIX shells, and B for the Bourne shell (sh). This will define the necessary environment variables and aliases. Your system administrator knows the location of definitions.<shell-type>.

### 3.1.2 How to Invoke the GUI Toolkit for Macintosh OS X

A complete set of the GUI toolkit is available in a file named CDFToolsDriver.jar. To invoke any of the CDF utilities (e.g. CDFedit, CDFexport, etc.) in the toolkit, do one of the following:

Double-click the CDFToolsDriver.jar icon on the Desktop

OR

Go to the directory where the CDF library is installed and double-click the CDFToolsDriver.jar file located under the <cdf_install_dir>/bin directory.

OR

Open a Terminal session and type "java CDFToolsDriver" at the operating system prompt.

Users will be presented with a main menu containing all the available CDF Java tools from which a desired tool can be selected with a single click.

### 3.1.3 How to Invoke the GUI Toolkit for Windows NT/95/98/2000/XP

Two executable programs (CDFfsi.exe & CDFso.exe) are included as part of the standard distribution package, and each program contains the following CDF utilities/tools:

| CDFfsi.exe | CDFso.exe |
|---|---|
| CDFedit | CDFcompare |
| CDFexport | CDFconvert |
| | CDFinquire |
| | CDFdump |
| | CDFstats |
| | SkeletonCDF |
| | SkeletonTable |

A CDF utility/tool can be invoked by running CDFfsi.exe or CDFso.exe and selecting the tool listed under the File menu. For example, the SkeletonCDF utility can be invoked by running the CDfso.exe program and then selecting the SkeletonCDF option under the File menu.

A Java version of CDFedit and CDFexport is also available in CDFToolsDriver.jar, and it is recommended over the ones in CDFfsi.exe since they are much more intuitive and easier to use. To invoke either program, do one of the following:

Double-click the CDFToolsDriver.jar icon on the Desktop

OR

Go to the directory where the CDF library is installed and double-click the CDFToolsDriver.jar program.

OR

Open a Command Prompt session (i.e. C:\) and type "java CDFToolsDriver" at the command prompt.

Users will be presented with a main menu containing two programs (CDFedit and CDFexport) from which a desired tool can be selected with a single click.

### 3.1.4 How to Invoke the GUI Toolkit for Unix

Java version of the CDF toolkit is available starting with CDF 2.7. A desired CDF tool can be invoked by typing "java CDFToolsDriver" at the system prompt and selecting the tool of interest from the main menu with a single click.

There are two environment variables that must be set prior to invoking the toolkit program (CDFToolsDriver.jar): CLASSPATH and LD_LIBRARY_PATH. Follow the instructions in the README.install file located under the <cdf_installed_dir>/cdfjava directory.

### 3.1.5 Special Attributes

There is a set of vAttributes that have special meaning to some of the CDF toolkit programs.[1] Your CDFs do not have to use these special attributes. The CDF toolkit programs will function properly whether or not these special attributes are present in a CDF. How the entries of each vAttribute are used for the corresponding variables is as follows:

| | |
|---|---|
| FORMAT | A Fortran or C format specification that is used when displaying a variable value. |
| VALIDMIN | The minimum valid value for a variable. |
| VALIDMAX | The maximum valid value for a variable. |
| FILLVAL | The value used for missing or invalid variable values.[2] |
| MONOTON | The monotonicity of a variable: INCREASE (strictly increasing values), DECREASE (strictly decreasing values), or FALSE (not monotonic). Monotonicity only applies to NRV variables that vary along one dimension and RV variables that vary along no dimensions. |
| SCALEMIN | The minimum value for scaling a variable when graphically displaying its values. |
| SCALEMAX | The maximum value for scaling a variable when graphically displaying its values. In the description of each CDF toolkit program, the special attributes that may affect that program's operation are defined. Note that most of the CDF toolkit programs can be instructed to ignore these special attributes. |

### 3.1.6 Special Qualifier

There is a special qualifier applied to all toolkit programs. This qualifier, as "-about" on all platforms except Macintosh, will show version, release and increment information of the distribution that the toolkit program is based on. This special qualifier, if present, supersedes all other qualifiers and parameters.

## 3.2 CDFedit

### 3.2.1 Introduction

The CDFedit program allows the display and/or modification of practically all of the contents of a CDF by way of a text-mode-full-screen interface. It is also possible to run CDFedit in a browse-only mode in order to prevent accidental

[1] These special attributes originated as part of the NSSDC standard for CDFs. The NSSDC standard is no longer used.
[2] Note that the FILLVAL attribute is not the same as the pad value for a variable although their values will often be the same. The pad value is used by the CDF library. The FILLVAL attribute is optionally used by a CDF toolkit program or by your applications.

modifications.[3] CDFedit can also be used to create a new CDF file if a CDF does not exist with the provided file path. The newly created CDF file can be of either default version, V3.*, or a backward version, i.e., V2.7. If the environment variable CDF_FILEBACKWARD on Unix or Windows or CDF$FILEBACKWARD on OpenVMS is set to TRUE, the new file is then automatically a V2.7 file. If this environment variable is not set or set to anything other than TRUE, then there is an option to choose for the file version when the program is executed.

### 3.2.2 Special Attribute Usage

The special attribute FORMAT is used by CDFedit (depending on the setting of the "format" qualifier) when displaying variable values.

### 3.2.3 Executing the CDFedit Program

**Usage:**

VMS:

```
$ CDFEDIT  [/[NO]BROWSE] [/ZMODE=<mode>] [/[NO]FORMAT] [/[NO]PROMPT]
           [/[NO]NEG2POSFP0] [/REPORT=(<types>)] [/CACHE=(<sizes>)]
           [/[NO]STATISTICS] [/[NO]GWITHENTRIES] [/[NO]VWITHENTRIES]
           [/ABOUT] <cdf-spec>
```

UNIX (including Mac OS X):

```
% cdfedit  [-[no]browse] [-zmode <mode>] [-[no]format] [-[no]prompt]
           [-[no]neg2posfp0] [-report "<types>"] [-cache "<sizes>"]
           [-[no]statistics] [-[no]gwithentries] [-[no]vwithentries]
           [-about] <cdf-spec>
```

**Parameter(s):**

<cdf-spec>

The specification of the CDF(s) to edit. (Do not specify an extension.) This may be either a single CDF file name 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).

**Qualifier(s):**

/[NO]BROWSE (VMS)
-[no]browse (UNIX)

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)

Specifies which zMode should be used. The zMode may be one of the following:

0 Indicates that zMode should be disabled.

[3] Running CDFedit in a browse-only mode provides the same functionality as CDFbrowse once did.

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)

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)

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).

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.)

/[NO]NEG2POSFP0 (VMS)
-[no]neg2posfp0 (UNIX)

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

/REPORT=(<types>) (VMS)
-report "<types>" (UNIX)

Specifies the types of return status codes from the CDF library that should be reported/displayed. The <types> option 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=(<sizes>) (VMS)
-cache "<sizes>" (UNIX)

Specifies the cache sizes to be used by the CDF library for the dotCDF file and the various scratch files. The <sizes> option is a comma-separated list of <size><type> pairs where <size> is a cache size and <type> is the type of file as follows: d for the dotCDF file, s for the staging scratch file, and c for the compression scratch file. For example, 200d,100s specifies a cache size of 200 for the dotCDF file and a cache size of 100 for the staging scratch file. The dotCDF file cache size can also be specified without the d file type for compatibility with older CDF releases (e.g., 200,100s). Note that not all of the file types must be specified. Those not specified will receive a default cache size chosen by the CDF library. A cache size is the number of 512-byte buffers to be used. Section 2.1.5 explains the caching scheme used by the CDF library.

/[NO]STATISTICS (VMS)
-[no]statistics (UNIX)

Specifies whether or not caching statistics are displayed when a CDF is closed.

/[NO]GWITHENTRIES (VMS)
-[no]gwithentries (UNIX)

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)

Specifies whether or not rEntries/zEntries are displayed with the vAttributes or on separate menus (with one menu per vAttribute).

/ABOUT (VMS) –about (UNIX)

Shows the library version that was used to create this tool program

.

**Example(s):**

VMS:

```
$ CDFEDIT [.SAMPLES]
$ CDFEDIT/ZMODE=2/NOFORMAT/CACHE=(10D,100S,200C) GISS_WETLX
$ CDFEDIT/BROWSE/PROMPT/REPORT=(ERRORS)
```

UNIX:

```
% cdfedit samples
% cdfedit -zmode 2 -noformat -cache "10d,100s,200c" giss_wetl
% cdfedit -browse -prompt -report "errors"
```

### 3.2.4 Interaction with CDFedit

Interaction with CDFedit is through a series of menus and windows. Extensive online help is provided and will not be repeated here.[4] The online help does refer to the sections of a window by name. Figure 3.1 illustrates the various sections of the possible types of windows.

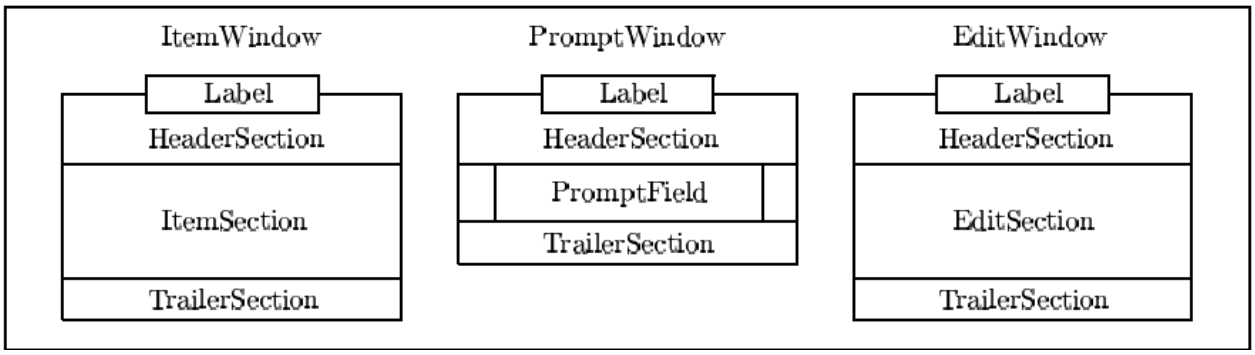


**Figure 3.1 Window Sections, CDFedit**

[4] It is our intention that the use of CDFedit be as intuitive as possible. You may not even need the online help. We're sure you'll let us know.

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.

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 used to display online help and to edit gAttribute character string entries as if they were a text file.

## 3.3 CDFexport

### 3.3.1 Introduction

CDFexport allows the contents of a CDF to be exported to the terminal screen, a text file, or another CDF. The variables to be exported can be selected along with a filter range for each variable which allows a subset of the CDF to be generated. When exporting to another CDF, a new compression and sparseness can be specified for each variable. When exporting to the terminal screen or a text file, the format of the output can be tailored as necessary. When exporting the output to a CDF, if the environment variable CDF_FILEBACKWARD on Unix or Windows or CDF$FILEBACKWARD on OpenVMS is set to TRUE, the output file is then automatically a V2.7 file. If this environment variable is not set or set to anything other than TRUE, then there is an option to choose for the file version when the program is executed.

### 3.3.2 Special Attribute Usage

CDFexport uses the following special attributes:

| | |
|---|---|
| FORMAT | Used as the initial value in a variable's Format field. |
| VALIDMIN | Used as the initial filter value in a variable's Minimum field. |
| VALIDMAX | Used as the initial filter value in a variable's Maximum field. |
| FILLVAL | Used as the initial value in a variable's FillValue field. |
| MONOTON | Used as the initial setting in a variable's Monotonicity field. |

These fields are described in the online help for the appropriate menu. The values of these fields can be changed at any time. The special attributes are simply used to provide initial values. Note also that the usage of these special attributes can be controlled by the options selected with the "initial" qualifier.

### 3.3.3 Executing the CDFexport Program

**Usage:**

VMS:

```
$ CDFEXPORT  [/INITIAL=(<options>)] [/[NO]PROMPT] [/ZMODE=<mode>]
             [/REPORT=(<types>)] [/[NO]STATISTICS] [/[NO]NEG2POSFP0]
             [/CACHE=(<sizes>)] [/[NO]SIMPLE] [/BATCH=<mode>] [/CDF=<path>]
             [/TEXT=<path>] [/SETTINGS=<path>] [/ABOUT]
             [/INCLUDE|EXCLUDE=[<vars>|<varsfile=file>]
             [/EPOCHRANGE|RECORDRANGE=<ranges>]
             [/CONTROLSETTINGS] <cdf-spec>
```

UNIX (including Mac OS X):

```
% cdfexport  [-initial "<options>"] [-[no]prompt] [-zmode <mode>]
             [-report "<types>"] [-[no]statistics] [-  [no]neg2posfp0]
             [-cache "<sizes>"] [-[no]simple] [-batch <mode>] [-cdf <path>]
             [-text <path>] [-settings <path>] [-about]
             [-include|exclude [<vars>|<varsfile=file>]
             [-epochrange|recordrange <ranges>]
             [-controlsettings] <cdf-spec>
```

## Parameter(s):

<cdf-spec>

The specification of the CDF(s) from which to export. Do not specify an extension. This may be either a single CDF file name or a directory/wildcard path. Wildcards are allowed in the CDF name but not in the directory path.

## Qualifier(s):

/[NO]PROMPT (VMS)
-[no]prompt (UNIX)

Specifies whether or not a prompt is issued for the CDF(s) specification. If this qualifier is not specified, the CDF(s) specification must be entered on the command line and is automatically opened.

If a CDF(s) specification was entered on the command line, that CDF(s) specification will initially appear at the prompt. Otherwise, the current directory will appear at the prompt.

/INITIAL=(<defaults>) (VMS)
-initial "<defaults>" (UNIX)

The default settings that are initially in affect when a CDF is opened. 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.

<defaults> is a comma-separated list of settings consisting of one or more of the options in the list that follows.

[NO]FILTER (VMS)
[no]filter (UNIX)

Whether or not each item/variable is initially filtered.

[NO]FILLS (VMS)
[no]fills (UNIX)

Whether or not the use of fill values is enabled.

[NO]FORMAT (VMS)
[no]format (UNIX)

Specifies whether or not a variable's FORMAT attribute entry is used as its initial "format" field.

[NO]FILLVAL (VMS)
[no]fillval (UNIX)

Specifies whether or not a variable's FILLVAL attribute entry is used as its initial "fill value" field.

[NO]VALIDMIN (VMS)
[no]validmin (UNIX)

Specifies whether or not a variable's VALIDMIN attribute entry is used as its initial minimum filter value.

[NO]VALIDMAX (VMS)
[no]validmax (UNIX)

Specifies whether or not a variable's VALIDMAX attribute entry is used as its initial maximum filter value.

[NO]MONOTON (VMS)
[no]monoton (UNIX)

Specifies whether or not a variable's MONOTON attribute entry is used as its initial monotonicity.

[NO]RECORD (VMS)
[no]record (UNIX)

Specifies whether or not the Record item will be present.

[NO]INDICES (VMS)
[no]indices (UNIX)

Specifies whether or not the Indices item will be present.

[NO]EXCLUSIVE (VMS)
[no]exclusive (UNIX)

Specifies whether or not exclusive filters are allowed.

[NO]OUTPUT (VMS)
[no]output (UNIX)

Specifies whether or not each item/variable is initially output.

[NO]DELETE (VMS)
[no]delete (UNIX)

Specifies the initial setting of whether or not an existing CDF will be deleted when a new CDF is created with the same name.

[NO]PREALLOCATE (VMS)
[no]preallocate (UNIX)

Specifies the initial setting of whether or not variable records are to be preallocated when creating a new CDF.

SINGLE or MULTI (VMS)
single or multi (UNIX)

Specifies the initial setting of whether single-file or multi-file CDFs are created.

HOST or NETWORK (VMS)
host or network (UNIX)

Specifies the initial setting of whether host-encoded or network-encoded CDFs are created.

ROW or COLUMN (VMS)
row or column (UNIX)

Specifies the initial setting of whether row-major, column-major, or input-major CDFs/listings are generated. Input-majority is the majority of the input CDF.

Input-majority is selected by specifying neither row-majority nor column-majority.

EPOCH, EPOCH1, EPOCH2, EPOCH3, ISO8601, EPOCHf or EPOCHx (VMS)
epoch, epoch1, epoch2, epoch3, iso8601, epochf or epochx (UNIX)

Specifies the initial EPOCH encoding style.

HORIZONTAL or VERTICAL (VMS)
horizontal or vertical (UNIX)

Specifies the initial setting of whether horizontal or vertical listings are generated.

Note that these options can be changed at any time after the CDF has been opened. If this qualifier is not specified, each of these options has a default setting. These default settings are also used for options not specified with this qualifier.

/ZMODE=<mode> (VMS)
-zmode <mode> (UNIX)

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]NEG2POSFP0 (VMS)
-[no]neg2posfp0 (UNIX)

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

/REPORT=(<types>) (VMS)
-report "<types>" (UNIX)

Specifies the types of return status codes from the CDF library that should be reported/displayed. The <types> option 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=(<sizes>) (VMS) -cache "<sizes>" (UNIX)

Specifies the cache sizes to be used by the CDF library for the dotCDF file and the various scratch files. The <sizes> option is a comma-separated list of <size><type> pairs where <size> is a cache size and <type> is the type of file as follows: d for the dotCDF file, s for the staging scratch file, and c for the compression scratch file. For example, 200d,100s specifies a cache size of 200 for the dotCDF file and a cache size of 100 for the staging scratch file. The dotCDF file cache size can also be specified without the d file type for compatibility with older CDF releases (e.g., 200,100s). Note that not all of the file types must be specified. Those not specified will receive a default cache size chosen by the CDF library. A cache size is the number of 512-byte buffers to be used. Section 2.1.5 explains the caching scheme used by the CDF library.

/[NO]STATISTICS (VMS)
-[no]statistics (UNIX)

Specifies whether or not caching statistics are displayed when a CDF is closed.

/[NO]SIMPLE (VMS)
-[no]simple (UNIX)

Specifies if a simplified version of CDFexport should be executed. The following conditions apply to simple mode:

- Only text listings can be generated (to the screen or a file).
- No filtering is available.
- When listing to a text file, FORMAT attribute entries are ignored and standard formats are used instead.
- Only a limited set of the options for the `initial' qualifier may be specified.
- zMode/2 is used by default.
- Horizontal listings are created by default.

/BATCH=<mode> (VMS)
-batch <mode> (UNIX)

Specifies if CDFexport should execute in a non-interactive batch mode. The mode option may be either "text" to generate a text file listing or "cdf" to output to a new CDF. A settings file will be used if one exists with the default name in the current directory or is explicitly specified with the `settings' qualifier. The settings file contains the parameters necessary to specify how the output CDF or text file should be generated. If a settings file is not available, default parameters will be used. CDFexport must be used interactively to create a settings file.

/CDF=<cdf> (VMS)
-cdf <cdf> (UNIX)

Specifies an output CDF file name to be used when exporting to a CDF in batch mode. Do not include an extension. When executing interactively, this file name will initially appear at the output CDF prompt. If this qualifier is not specified, the default CDF name is "default" (in the current directory).

/TEXT=<path> (VMS)

-text <path> (UNIX)

Specifies a file name to be used when exporting to a text file listing in batch mode. When executing interactively this file name will initially appear at the text file prompt. If this qualifier is not specified, the default text file name is "default.lis" (in the current directory).

/SETTINGS=<path> (VMS) -settings <path> (UNIX)

Specifies a settings file name to be used when executing in batch mode. When executing interactively this file name will initially appear at the settings file prompt when saving/restoring the current settings. The default settings file is "simple.set" if executing in simple mode and "export.set" otherwise (with each being in the current directory).

/INCLUDE | EXCLUDE=<vars | varsfile=<file>> (VMS)
–include | exclude <vars | varsfile=<file>>(UNIX)

Includes or excludes only those specified variables in the CDF for export. The variables can be specified in "vars", a comma-separated variable list. Alternatively, if the list gets too long, a text file, containing the variables, one variable per line, can be used.The text file option is identified by "varsfile". This a simple and easier way of exporting variables without settings file, nor data filtering. A warning is shown if a variable(s) specified is not in the CDF. Record numbers and indices will be displayed, if applicable, for exporting to text. "include" and "exclude" are mutually exclusive. This option and "settings" option are also mutually exclusive. As "settings" is not allowed, the default settings, if not overridden, will be used for the exports. This option works in the batch mode.

/EPOCHRANGE | RECORDRANGE=<ranges> (VMS)
-epochrange | recordrange <ranges> (UNIX)

Selects a range of variable data for exporting from a CDF. The range can be based on epoch, specified by "epochrange" or record, by "recordrange". "ranges" should be a pair of values for starting and ending value, separated by a comma. Epoch includes data types of CDF_EPOCH, CDF_EPOCH16 and CDF_TIME_TT2000. For the starting and ending epoch, they must be one of the following format pairs:
"dd-mm-yyyy hh:mm:ss.msc" for CDF_EPOCH,
"dd-mm-yyyy hh:mm:ss.mmm.uuu.nnn.ppp" for CDF_EPOCH16, and
“yyyy-mm-ddThh:mm:ss.mmmuuunnn” for CDF_TIME_TT2000, respectively.
All date fields should be numeric. For the record numbers, they start from record one (1). If no data is found from the exported variables, nothing is outputted. "epochrange" and "recordrange" are mutually exclusive. This option works in the batch mode.

/CONTROLSETTINGS (VMS) –controlsettings (UNIX)

A batch mode option that uses the settings file to control what variables and their order for export. CDFexport, by default, exports variables in a sequence based on their order in the source CDF file, no matter how settings file is defined. This option requires the settings file to be present, either specified at the command line or the default file "export.set" (or "simple.set" for simple mode) residing in the current directory. Only those variables specified in settings file are considered for export. The file also dictates the output sequence of variables in the export.

/ABOUT (VMS) –about (UNIX)

Shows the library version that was used to create this tool program

.

**Example(s):**

VMS:

```
$ CDFEXPORT [.SAMPLES]
$ CDFEXPORT/ZMODE=2/CACHE=(50d,100s) GISS_WETLX
$ CDFEXPORT/PROMPT/REPORT=(W,E)/INITIAL=(EXCLUSIVE,NOFORMAT)
$ CDFEXPORT/SIMPLE/BATCH=TEXT/TEXT=FLUX.OUT FLUX1996
$ CDFEXPORT/BATCH=CDF/CDF=MYCDF1/INCLUDE="var1,var2" test
$ CDFEXPORT/BATCH=CDF/CDF=MYCDF2/EXCLUDE="var1,var2" test
$ CDFEXPORT/BATCH=TEXT/TEXT=MYTEXT/EXCLUDE="varsfile='/home/cdf/varslist.txt'" test
$  cdfexport/BATCH=CDF/CDF=MYCDF/EPOCHRANGE="01-10-2005  00:00:00.000,01-10-2005
10:00:00.000" test
$ CDFEXPORT/BATCH=TEXT/TEXT=MYOUTPUT.txt/CONTROLSETTINGS/SETTINGS=EXPORT.set test
```

UNIX:

```
% cdfexport samples
% cdfexport -zmode 2 -cache "50d,100s" giss_wetl
% cdfexport -prompt -report "w,e" -initial "exclusive,noformat"
% cdfexport -simple -batch text -text flux.out flux1996
% cdfexport -batch cdf -cdf mycdf1 -include "var1,var2" test
% cdfexport -batch cdf -cdf mycdf2 -exclude "var1,var2" test
% cdfexport -batch text -text mytext-exclude "varsfile='/home/cdf/varslist.txt'" test
% cdfexport -batch cdf -cdf mycdf –epochrange "01-10-2005 00:00:00.000,01-10-2005 10:00:00.000" test
% cdfexport -batch text -text myoutput -controlsettings -settings export.set test
```

### 3.3.4 Interaction with CDFexport

Interaction with CDFexport is through a 4-part SelectionWindow, an ActionMenu, an OptionMenu, numerous prompt windows, and several screen listing windows. Detailed online help is available for each window so only a brief description of each will be given here. After selecting a CDF from which to export, part 1 of the SelectionWindow will be loaded with a line for the <Record> item, the <Indices> item, and each variable. The <Record> item allows the record number to be included in a screen/file listing and/or filtering on the record number for any type of output. The <Indices> item allows the dimension indices to be included in a screen/file listing and/or filtering on the dimension indices for any type of output. Each variable line allows that variable to be included and/or filtered when generating any type of output. The KeyDefinitions window displays the available functions and their corresponding keys for a given window/prompt. The MessageBuffer displays errors/instructions as necessary.

Cycling through the four parts of the SelectionWindow allows the selection of the output to be generated. The online help explains the purpose of each field in the four parts of the SelectionWindow. The OptionMenu allows additional selections affecting the output. The ActionMenu is then used to generate the desired type of output (as well as some other miscellaneous operations).

The easiest way to learn how to use CDFexport is to read through the online help while generating the various types of output using a CDF with which you are familiar.

## 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. The format of the CDF may be changed (see Section 2.2.7).
2. The data encoding of the CDF may be changed (see Section 2.2.8).
3. The variable majority of the CDF may be changed (see Section 2.3.15).
4. The compression of the CDF (see Section 2.2.10) or the CDF's variables (see Section 2.3.14) may be changed.
5. The sparseness of the CDF's variables may be changed (see Sections 2.3.12 and 2.3.13).
6. The file version may be changed to be backward compatible.
7. The checksum method may be changed.

### 3.4.2 Executing the CDFconvert Program

**Usage:**

VMS:

```
$ CDFCONVERT  [/SKELETON=<skt-cdf-path>] [/[NO]LOG] [/[NO]PERCENT]
              [/REPORT=(<types>)] [/CACHE=(<sizes>)] [/[NO]PAGE] [/[NO]STATISTICS]
              [/ABOUT] <src-cdf-spec>
                  [/ZMODE=<mode>] [/[NO]NEG2POSFP0]
              <dst-cdf-spec>
                  [/SINGLE | /MULTI] [/ROW | /COLUMN] [/[NO]DELETE]
                  [/ENCODING=<encoding> | /HOST | /NETWORK]
                   [/COMPRESSION=(<types>)] [/SPARSENESS=(<types>)]
                   [/BACKAWRD] [/CHECKSUM=<mode>]
                   [/EPOCH2TT2000 | /TT20002EPOCH | /TT20002DEPOCH]
```

UNIX (including Mac OS X):

```
% cdfconvert  [-skeleton <skt-cdf-path>] [-[no]log]  [-[no]percent]
              [-report "<types>"] [-cache "<sizes>"] [-[no]page] [-[no]statistics]
              [-about] <src-cdf-spec>
                  [-zmode <mode>] [-[no]neg2posfp0]
              <dst-cdf-spec>
                  [-single | -multi] [-row | -column] [-[no]delete]
                  [-encoding <encoding> | -host | -network]
                  [-compression <types>] [-sparseness <types>]
                  [-backward] [-checksum <mode>]
                  [-epoch2tt2000 | -tt20002epoch | -tt20002depoch]
```

**Parameter(s):**

<src-cdf-spec>

The source CDF(s). This can be either a single CDF file name 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.

<dst-cdf-spec>

The destination of the converted CDF(s). This may be a single CDF file name 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.

**Qualifier(s):**

/SKELETON=<skt-cdf-path> (VMS)
-skeleton <skt-cdf-path> (UNIX)

The file name 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.

/[NO]LOG (VMS)
-[no]log (UNIX)

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

/[NO]PAGE (VMS)
-[no]page (UNIX)

Specifies whether or not the output is displayed a page at a time. A prompt for the RETURN key will be issued after each page. A page is generally 22 lines of output.

/[NO]PERCENT (VMS)
-[no]percent (UNIX)

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)

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

/SINGLE | /MULTI (VMS)
-single | -multi (UNIX)

The format of the destination CDF(s).

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.

/ROW | /COLUMN (VMS)
-row | -column (UNIX)

The variable majority of the destination CDF(s).

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.

/ENCODING=<encoding> | /HOST | /NETWORK (VMS)
-encoding <encoding> | -host | -network (UNIX)
Source/Host/Network/Sun...Vax radio buttons (Macintosh, Java/UNIX & Windows NT/95/98)

The data encoding of the destination CDF(s).

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).

/COMPRESSION=(<types>) (VMS)
-compression <types> (UNIX)

Specifies the types of compression to be used for the CDF and/or variables. The <types> option consists of a comma-separated list of the following. . .

| | |
|---|---|
| cdf:<cT> | CDF's compression. |
| vars:<cT> | Compression for all variables. |
| vars:<cT>:<bF> | Compression for all variables with a blocking factor specified. |
| vars:<cT>:<bF>:<r%> | Compression for all variables with a blocking factor and reserve percentage specified. |
| var:<name>:<cT> | Compression for one particular variable. |
| var:<name>:<cT>:<bF> | Compression for one particular variable with a blocking factor specified. |
| var:<name>:<cT>:<bF>:<r%> | Compression for one particular variable with a blocking factor and reserve percentage specified. |

Where <cT> is one of the following compressions: none, rle.0, huff.0, ahuff.0, or gzip.<level>; <bF> is a blocking factor; <r%> is a reserve percentage; and <name> is a delimited, case-sensitive variable name with the following syntax:

<delim><char1><char2>...<charN><delim>

In general, do not use single or double quote marks as delimiters. VMS: The entire delimited variable name must be enclosed in double quote marks (to preserve case-sensitivity).

For the gzip compression, <level> must be in the range from 1 (fastest compression) to 9 (best compression).

For compressions not specified the compression in the source CDF will be used. Specifying a variable compression using var:…overrides a compression specified with vars:. . .

/SPARSENESS=(<types>) (VMS) -sparseness <types> (UNIX)

Specifies the types of sparseness to be used for the variables. The <types> option consists of a comma-separated list of the following. . .

| | |
|---|---|
| vars:<sT> | Sparseness for all variables. |
| var:<name>:<sT> | Sparseness for one particular variable. |

Where <sT> is one of the following: srecords.no, srecords.pad, or srecords.prev; and <name> is a delimited, case-sensitive variable name with the following syntax:

<delim><char1><char2>...<charN><delim>

In general, do not use single or double quote marks as delimiters. VMS: The entire delimited variable name must be enclosed in double quote marks (to preserve case-sensitivity).

For sparsenesses not specified the sparseness in the source CDF will be used. Specifying a variable sparseness using var:. . . overrides a sparseness specified with vars:.. .

/ZMODE=<mode> (VMS)
-zmode <mode> (UNIX)

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)

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

/REPORT=(<types>) (VMS)
-report "<types>" (UNIX)

Specifies the types of return status codes from the CDF library that should be reported/displayed. The <types> option 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=(<sizes>) (VMS) -cache "<sizes>" (UNIX)

Specifies the cache sizes to be used by the CDF library for the dotCDF file and the various scratch files. The <sizes> option is a comma-separated list of <size><type> pairs where <size> is a cache size and <type> is the type of file as follows: d for the dotCDF file, s for the staging scratch file, and c for the compression scratch file. For example, 200d,100s specifies a cache size of 200 for the dotCDF file and a cache size of 100 for the staging scratch file. The dotCDF file cache size can also be specified without the d file type for compatibility with older CDF releases (e.g., 200,100s). Note that not all of the file types must be specified. Those not specified will receive a default cache size chosen by the CDF library. A cache size is the number of 512-byte buffers to be used. Section 2.1.5 explains the caching scheme used by the CDF library.

/[NO]STATISTICS (VMS)
-[no]statistics (UNIX)

Specifies whether or not caching statistics are displayed when a CDF is closed.

/[BACKWARD] (VMS)
-backward (UNIX)

Specifies whether or not to make the converted file as a backward compatible file (i.e., V 2.7) instead of the default V.3.* file. If the environment variable CDF_FILEBACKWARD on Unix or Windows or CDF$FILEBACKWARD on OpenVMS is set to TRUE, the converted file is then automatically a V2.7 file whether this option is set. If this environment variable is not set or set to anything other than TRUE, then use this option to create a backward file.

/[CHECKSUM=<MODE>] (VMS)
-checksum <mode> (UNIX)

By default, whether to set the checksum mode on the converted file is based on the environment variable CDF_CHECKSUM (or CDF$CHECKSUM on OpenVMS) or the source file. If the environment variable is set to a valid method, the checksum bit for the converted file will be set accordingly. If the environment variable is not set, then the source file's checksum mode is used for the converted file. Alternatively, to force the checksum mode to be (or not to be) set, this option can be used. If it is specified, then it will overwrite the environment variable and the mode from the source file. Currently, the possible values for the checksum option are `none', `md5' or `source'.

/[EPOCH2TT2000 | TT20002EPOCH | TT20002DEPOCH] (VMS)
-[epoch2tt2000 | tt20002epoch | tt20002depoch] (UNIX)

By default, the epoch/tt2000 data type is reserved when the file is converted. However, this option will convert the data in EPOCH/EPOCH16 to TT2000 or vice verse. As both EPOCH and EPOCH16 data types do not have leap seconds, leap seconds occur in TT2000 will be lost. “epoch2tt2000” will convert source data in either EPOCH or EPOCH16 to TT2000. Also, as TT2000 and EPOCH/EPOCH16 have slightly different time resolution, sub-millisecond data may get lost or filled after conversion. “tt20002epoch” will convert source data in TT2000 to EPOCH, while “tt20002depoch” convert data in TT2000 to EPOCH16.

/[ABOUT](VMS) –about (UNIX)

Shows the library version that was used to create this tool program

.

**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]
$ CDFCONVERT SOURCE DESTINATION /BACKWARD
$ CDFCONVERT FILEIN FILEOUT /CHECKSUM=MD5
```

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
% cdfconvert  source destination -backward
% cdfconvert  fileIn fileOut -checksum md5
```

VMS, UNIX**:**

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 CDFcompare

### 3.5.1 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).

### 3.5.2 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=(<sizes>)] [/[NO]PAGE]
              [/[NO]STATISTICS] [/[NO]PERCENT] [/[NO]VALUE]
              [/TOLERANCE=(<F:tolerance1>,<D:tolerance1>)]
              [/ABOUT] <cdf-spec-1> <cdf-spec-2>
```

UNIX (including Mac OS X):

```
% cdfcompare  [-[no]log] [-[no]attr] [-[no]var] [-[no]number] [-[no]etc]
              [-[no]neg2posfp0] [-zmodes "<mode1>,<mode2>"] [-[no]location]
              [-report "<types>"] [-cache "<sizes>"] [-[no]page]
```

```
[-[no]statistics] [-[no]percent] [-[no]value]
[-tolerance "<f:tolerance1>,<d:tolerance2>"]
[-about] <cdf-spec-1> <cdf-spec-2>
```

**Parameter(s):**

<cdf-spec-1> <cdf-spec-2>

The specifications of the CDFs to be compared. (Do not enter extensions.) These can be either a file name 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 file name 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 file names are specified, the CDFs are compared. (This is the only way to compare two CDFs having different names.)

**Qualifier(s):**

/[NO]LOG (VMS)
-[no]log (UNIX)

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

/[NO]PERCENT (VMS)
-[no]percent (UNIX)

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)

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

/[NO]VAR (VMS)
-[no]var (UNIX)

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)

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)

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)

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)

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

/[NO]PAGE (VMS)
-[no]page (UNIX)

Specifies whether or not the output is displayed a page at a time. A prompt for the RETURN key will be issued after each page. A page is generally 22 lines of output.

/[NO]LOCATION (VMS)
-[no]location (UNIX)

Specifies whether or not the locations of variable value differences are displayed. The locations are displayed in the form:

<record-number>:[<index1>,<index2>,...,<indexN>]

/[NO]VALUE (VMS)
-[no]value (UNIX)

Specifies whether or not the values are displayed when a difference is detected between variable values or attribute entries. Note that for variable values to be displayed, the display of the locations of the differences must also be enabled.

/REPORT=(<types>) (VMS)
-report "<types>" (UNIX)

Specifies the types of return status codes from the CDF library that should be reported/displayed. The <types> option 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=(<sizes>) (VMS)
-cache "<sizes>" (UNIX)

Specifies the cache sizes to be used by the CDF library for the dotCDF file and the various scratch files. The <sizes> option is a comma-separated list of <size><type> pairs where <size> is a cache size and <type> is the type of file as follows: d for the dotCDF file, s for the staging scratch file, and c for the compression scratch

file. For example, 200d,100s specifies a cache size of 200 for the dotCDF file and a cache size of 100 for the staging scratch file. The dotCDF file cache size can also be specified without the d file type for compatibility with older CDF releases (e.g., 200,100s). Note that not all of the file types must be specified. Those not specified will receive a default cache size chosen by the CDF library. A cache size is the number of 512-byte buffers to be used. Section 2.1.5 explains the caching scheme used by the CDF library.

/TOLERANCE=(<F:TOLERANCE1,D:TOLERANCE2>) (VMS)
-tolerance "<f:tolerance1,d:tolerance2>" (UNIX)

Specifies the tolerance(s) that is used to check the equality between two single/double-precision floating-point values. The default option is no tolerance. It means that two values are considered unequal if their data representations in common encoding are different. If a tolerance(s) is provided, it is used against the difference between the two unequal values. If their difference is within the tolerance, they are considered to be technically equal. Either one or both of these two tolerances, one for 4-byte single-precision floating-point data and the other for 8-byte double-precision floating-point data, respectively, can be specified.

If the given tolerance is positive, the following formula is used to check their equality:
abs(value1-value2) > tolerance

If the given tolerance is negative, the following formula is applied:
abs(value1-value2) > abs(tolerance)*max(abs(value1),abs(value2))

tolerance1, used for the single-precision floating-point data, may be in one of the two forms: "default" or a value. Using "default" indicates that the default value, 1.0E-06, is used for the tolerance check for any single-precision floating-point data. Or, the specified value is used for the tolerance check. This field applies to data types of CDF_REAL4 and CDF_FLOAT. "def" can be used to substitute for "default".

tolerance2, used for the double-precision floating-point data, may be in one of the two forms: "default" or a value. Using "default" indicates that the default value, 1.0E-09, is used for the tolerance check for any double-precision floating-point data. Or, the specified value is used for the tolerance check. This field applies to data types of CDF_REAL8, CDF_DOUBLE and CDF_EPOCH. "default" can be abbreviated as "def".

/[NO]STATISTICS (VMS)
-[no]statistics (UNIX)

Specifies whether or not caching statistics are displayed when a CDF is closed.

/ABOUT (VMS) –about (UNIX)

Shows the library version that was used to create this tool program

.

## Example(s):

VMS:

```
$ CDFCOMPARE GISS_WETL GISS_WETL1
$ CDFCOMPARE/LOG/TOLERANCE=(F:DEF,D:1.0E-12)/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 -tolerance "f:def,d:1.0e-12" -noattr
              -number -report "errors" giss_wetl ../giss_wetlx
```

```
% cdfcompare  -novar -noetc -zmodes "1,2" /user5/CDFs /user6/CDFs
```

**VMS, UNIX:** Command line help is displayed when CDFcompare is executed without any arguments.

### 3.5.3 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.

## 3.6 CDFstats

### 3.6.1 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 a valid range of values (with illegal/fill values being ignored), and the variable's monotonicity.

Monotonicity refers to whether or not a variable's 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.

| rVariable | Record Variance | Dimension Variances | Check Monotonicity? |
|---|---|---|---|
| EPOCH | VARY | NOVARY,NOVARY | Yes |
| LATITUDE | NOVARY | VARY,NOVARY | Yes |
| LONGITUDE | NOVARY | NOVARY,VARY | Yes |
| ELEVATION | NOVARY | VARY,VARY | No |
| TEMPERATURE | VARY | VARY,VARY | No |

**Table 3.1 Example rVariables, CDFstats Monotonicity Checking**

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

### 3.6.2 Special Attribute Usage

CDFstats uses the following special attributes:

FORMAT — Used when displaying a variable statistic (e.g., minimum variable value).

VALIDMIN — If range checking is enabled, used as the minimum valid value for a variable. For a variable with a non-character data type, only the first element of its VALIDMIN attribute entry is used. Also, if requested, the VALIDMIN attribute entry for a variable will be updated with the actual minimum value found. Again, if the variable has a non-character data type the VALIDMIN attribute entry will be updated to have just one element.

VALIDMAX — If range checking is enabled, used as the maximum valid value for a variable. For a variable with a non-character data type, only the first element of its VALIDMAX attribute

entry is used. Also, if requested, the VALIDMAX attribute entry for a variable will be updated with the actual maximum value found. Again, if the variable has a non-character data type the VALIDMAX attribute entry will be updated to have just one element.

FILLVAL — If fill value usage is enabled, used as the value that is ignored while collecting statistics for a variable.

MONOTON — If requested, the MONOTON attribute entry for a variable will be updated with the actual monotonicity found. The possible values for the MONOTON attribute entry are described in Section 3.1.5.

SCALEMIN — If requested, the SCALEMIN attribute entry for a variable will be updated with the actual minimum value found.

SCALEMAX — If requested, the SCALEMAX attribute entry for a variable will be updated with the actual maximum value found.

The usage of these special attributes can be controlled with command line qualifiers.

### 3.6.3 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=(<sizes>)] [/[NO]STATISTICS]
            [/ABOUT] <cdf-path>
```

UNIX (including Mac OS X):

```
% 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 "<sizes>"] [-[no]statistics]
            [-about] <cdf-path>
```

**Parameter(s):**

<cdf-path>

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

**Qualifier(s):**

/[NO]RANGE (VMS)
-[no]range (UNIX)

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. Note that for variables having a non-character data type only the first element of the VALIDMIN and VALIDMAX attribute entries are used.

/[NO]FILL (VMS)
-[no]fill (UNIX)

Specifies whether or not fill values are ignored when collecting statistics. The FILLVAL attribute entry for a variable (if it exists) is used as the fill value.

/OUTPUT=<file-path> (VMS)
-output <file-path> (UNIX)

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.

/[NO]FORMAT (VMS)
-[no]format (UNIX)

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)

Specifies whether or not the output is displayed a page at a time. A prompt for the RETURN key will be issued after each page. A page is generally 22 lines of output.

/[NO]UPDATE VALIDS (VMS)
-[no]update valids (UNIX)

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)

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)

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)

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)

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

/REPORT=(<types>) (VMS)
-report "<types>" (UNIX)

Specifies the types of return status codes from the CDF library that should be reported/displayed. The <types> option 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=(<sizes>) (VMS)
-cache "<sizes>" (UNIX)

Specifies the cache sizes to be used by the CDF library for the dotCDF file and the various scratch files. The <sizes> option is a comma-separated list of <size><type> pairs where <size> is a cache size and <type> is the type of file as follows: d for the dotCDF file, s for the staging scratch file, and c for the compression scratch file. For example, 200d,100s specifies a cache size of 200 for the dotCDF file and a cache size of 100 for the staging scratch file. The dotCDF file cache size can also be specified without the d file type for compatibility with older CDF releases (e.g., 200,100s). Note that not all of the file types must be specified. Those not specified will receive a default cache size chosen by the CDF library. A cache size is the number of 512-byte buffers to be used. Section 2.1.5 explains the caching scheme used by the CDF library.

/[NO]STATISTICS (VMS)
-[no]statistics (UNIX)

Specifies whether or not caching statistics are displayed when a CDF is closed.

/ABOUT (VMS) –about (UNIX)

Shows the library version that was used to create this tool program

.

**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
```

**VMS, UNIX:** Command line help is displayed when CDFstats is executed without any arguments.

### 3.6.4 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"). |

## 3.7 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 to 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.

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 Special Attribute Usage

The special attribute FORMAT is used by SkeletonTable (depending on the setting of the "format" qualifier) when writing variable values in a skeleton table.

### 3.7.3 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=(<sizes>)] [/[NO]PAGE] [/[NO]STATISTICS]
                 [/ABOUT] <cdf-path>
```

UNIX (including Mac OS X):

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

**Parameter(s):**

<cdf-path>

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

**Qualifier(s):**

/SKELETON=<skeleton-path> (VMS)
-skeleton <skeleton-path> (UNIX)

The file name of the skeleton table to be created. (Do not enter an extension because .skt 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).

/VALUES=<values> | /NRVTABLE | /NONRV (VMS)
-values <values> | -nrvtable | -nonrv (UNIX)

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

/VALUES=<values> (VMS)
-values <values> (UNIX)
No values/.../Selected values radio buttons (Macintosh, Java/UNIX & Windows NT/95/98)

VMS, UNIX: The <values> option specifies which variable values should be put in the skeleton table. Select one of the options from the list which follows.

None (VMS)

No values radio button (UNIX)
Off radio button (Macintosh, Java/UNIX & Windows NT/95/98)

No variable values should be put in the skeleton table.

Nrv (VMS)
NRV values radio button (UNIX)
NRV radio button (Macintosh, Java/UNIX & Windows NT/95/98)

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

Rv (VMS)
RV values radio button (UNIX)
RV radio button (Macintosh, Java/UNIX & Windows NT/95/98)

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

All (VMS)
All values radio button (UNIX)
All radio button (Macintosh, Java/UNIX & Windows NT/95/98)

All variable values should be put in the skeleton table.

<named> (VMS)
Selected values radio button (UNIX)
named radio button (Macintosh, Java/UNIX & Windows NT/95/98)

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

**VMS, UNIX:** <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.

/NONRV (VMS)
-nonrv (UNIX)

Ignore NRV data. (No values are placed in the skeleton table.)

/NRVTABLE (VMS)
-nrvtable (UNIX)

Put NRV variable data values in the skeleton table.

**VMS, UNIX:** 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)

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

/ZMODE=<mode> (VMS)
-zmode <mode> (UNIX)

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)

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)

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

/REPORT=(<types>) (VMS)
-report "<types>" (UNIX)

Specifies the types of return status codes from the CDF library that should be reported/displayed. The <types> option 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=(<sizes>) (VMS) -cache "<sizes>" (UNIX)

Specifies the cache sizes to be used by the CDF library for the dotCDF file and the various scratch files. The <sizes> option is a comma-separated list of <size><type> pairs where <size> is a cache size and <type> is the type of file as follows: d for the dotCDF file, s for the staging scratch file, and c for the compression scratch file. For example, 200d,100s specifies a cache size of 200 for the dotCDF file and a cache size of 100 for the staging scratch file. The dotCDF file cache size can also be specified without the d file type for compatibility with older CDF releases (e.g., 200,100s). Note that not all of the file types must be specified. Those not specified will receive a default cache size chosen by the CDF library. A cache size is the number of 512-byte buffers to be used. Section 2.1.5 explains the caching scheme used by the CDF library.

/[NO]STATISTICS (VMS)
-[no]statistics (UNIX)

Specifies whether or not caching statistics are displayed when a CDF is closed.

/[NO]SCREEN (VMS)
-[no]screen (UNIX)

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)

Specifies whether or not the output is displayed a page at a time. A prompt for the RETURN key will be issued after each page. A page is generally 22 lines of output.

/ABOUT (VMS) –about (UNIX)

Shows the library version that was used to create this tool program

.

**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  -screen -values "'Var1','Var2'"
```

**VMS, UNIX:** Command line help is displayed when SkeletonTable is executed without any arguments.

### 3.7.4 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[5] program is used to make a fully structured CDF, called a skeleton CDF, by reading a text file called a skeleton table. 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.

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 any or all of the variables.

The created CDF is referred to as a skeleton CDF.

[5] 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.

### 3.8.2 Executing the SkeletonCDF Program

**Usage:**

VMS:

```
$ SKELETONCDF  [/CDF=<cdf-path>] [/[NO]LOG] [/[NO]DELETE] [/[NO]FILLVAL]
               [/REPORT=(<types>)] [/[NO]NEG2POSFP0] [/CACHE=(<sizes>)]
               [/ZMODE=<mode>] [/BACKWARD] [/ABOUT] <skeleton-path>
```

UNIX (including Mac OS X):

```
% skeletoncdf  [-cdf <cdf-path>] [-[no]log] [-[no]delete] [-[no]fillval]
               [-report "<types>"] [-[no]neg2posfp0] [-cache "<sizes>"]
               [-zmode <mode>] [-backward] [-about] <skeleton-path>
```

**Parameter(s):**

<skeleton-path>

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

**Qualifier(s):**

/CDF=<cdf-path> (VMS)
-cdf <cdf-path> (UNIX)

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

/[NO]LOG (VMS)
-[no]log (UNIX)

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

/[NO]NEG2POSFP0 (VMS)
-[no]neg2posfp0 (UNIX)

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

/[NO]DELETE (VMS)
-[no]delete (UNIX)

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

/[NO]FILLVAL (VMS)
-[no]fillval (UNIX)

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.

/CACHE=(<sizes>) (VMS) -cache "<sizes>" (UNIX)

Specifies the cache sizes to be used by the CDF library for the dotCDF file and the various scratch files. The <sizes> option is a comma-separated list of <size><type> pairs where <size> is a cache size and <type> is the type of file as follows: d for the dotCDF file, s for the staging scratch file, and c for the compression scratch file. For example, 200d,100s specifies a cache size of 200 for the dotCDF file and a cache size of 100 for the staging scratch file. The dotCDF file cache size can also be specified without the d file type for compatibility with older CDF releases (e.g., 200,100s). Note that not all of the file types must be specified. Those not specified will receive a default cache size chosen by the CDF library. A cache size is the number of 512-byte buffers to be used. Section 2.1.5 explains the caching scheme used by the CDF library.

/ZMODE=<mode> (VMS)
-zmode <mode> (UNIX)

Specifies the zMode that should be used with the skeleton table. If zMode is enabled, zVariables will be created from the definitions in the rVariables section. 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 F [false] are removed.

/BACKWARD (VMS)
-backward (UNIX)

Specifies whether an older, backward compatible version of file (in V2.7) is to be created. The default will create a file of the same version as the underlying CDF library.

/REPORT=(<types>) (VMS)
-report "<types>" (UNIX)

Specifies the types of return status codes from the CDF library that should be reported/displayed. The <types> option 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).

/ABOUT (VMS) –about (UNIX)

Shows the library version that was used to create this tool program

.

**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
```

**VMS, UNIX:** 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 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 CDFinquire

### 3.9.1 Introduction

The CDFinquire program displays the version of the CDF distribution being used, most configurable parameters, and the default toolkit qualifiers.

### 3.9.2 Executing the CDFinquire Program

**Usage:**

VMS:

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

UNIX:

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

**Parameter(s):**

None

**Qualifier(s):**

/ID (VMS)
-id (UNIX)

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)

Specifies whether or not the output is displayed a page at a time. A prompt for the RETURN key will be issued after each page. A page is generally 22 lines of output.

/ABOUT (VMS) –about (UNIX)

Shows the library version that was used to create this tool program

.

**Example(s):**

VMS:

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

UNIX:

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

**VMS, UNIX:** Command line help is displayed when CDFinquire is executed without any arguments.

### 3.9.3 Output from the CDFinquire Program

The version of your CDF distribution is displayed first followed by the configurable parameters and then the default toolkit qualifiers (in the style of the system being used).

## 3.10 CDFdir

### 3.10.1 Introduction

The CDFdir utility is used to display a directory listing of a CDF's files.[6] The dotCDF 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.

### 3.10.2 Executing the CDFdir Program

The command line syntax for CDFdir is as follows:

**Usage:**

VMS:

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

[6] CDFdir is not available on Windows systems. It's also not available in Java version pf the CDF toolkit.

UNIX (including Mac OS X):

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

**NOTE:** This tool is not supported by Windows.

**Parameter(s):**

| | |
|---|---|
| <cdf-path> | The file name of the CDF for which to display a directory listing (do not specify an extension). |

**Example(s):**

VMS:

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

UNIX:

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

Help is displayed when CDFdir is executed without any arguments.

### 3.10.3 Output from the CDFdir Program

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

## 3.11 CDFmerge

### 3.11.1 Introduction

The CDFmerge utility merges two or more CDF files into a single file. Input file names and output file name can be specified either from the command line or in a text file. If there are many CDF files to merge, it may be more convenient to specify the input file names and output file name in a text file with the '-file' flag.

### 3.11.2 Executing the CDFmerge Program

The command line syntax for CDFmerge is as follows:

**Usage:**

VMS:

```
$ CDFMERGE [ [/NOPREFIX] | [/PREFIXES=<prefix1,><prefix2,>...<prefixN>] ]
           [-[NO]LOG] [-[NO]DATAONLY] [/ABOUT]
           <<cdf-path1> <cdf-path2>...<cdf-pathN>> <out-cdf>> | /FILE=<filename>
```

UNIX (including Mac OS X):
Windows:

```
% cdfmerge [ [-noprefix] | [-prefixes <prefix1,><prefix2,>...<prefixN>] ]
           [-[no]log] [-[no]dataonly] [-about]
           <<cdf-path1> <cdf-path2>...<cdf-pathN> <out-cdf>> | [-file filename]
```

## Parameter(s):

<cdf-path1>
<cdf-path2>
….
<cdf-pathN> The pathnames of the input CDF files to be merged. Two input files must be specified at a minimum.

<out-cdf> The pathname of the output CDF. This is the last one in the parameter list. Any rVariables in the source files will be converted to zVariables. The first source file dictates how the merged file is to be created: its majority (row/column), encoding, format (single/multiple) and compression are used for the output file.

/FILE=<file name> (VMS)
-file <file name> (Unix/Windows)
Specifies the name of the file that contains the names of the source CDFs and the output, merged file. This option provides an alternative way of entering a long list of files at the command line. Each input file name and the output file name should be specified on a separate line. User-defined prefix, if desired, should be added at the end of the input file name separated by one or more blank spaces. Suppose you want to merge five files (a.cdf, b.cdf, c.cdf, d.cdf, and e.cdf) into a file called merged_ab.cdf and provide the names of the input and output CDFs in a text file called my_merge.txt. Then the contents of my_merge.txt should contain the following:

```
a.cdf
b.cdf
c.cdf
d.cdf
e.cdf
merged_ab.cdf
```

## Qualifier(s):

/NOPREFIX (VMS)
-noprefix (Unix/Windows)
This option specifies not to add the system default prefix to the beginning of each variable name in the merged file. If this option is specified, it's assumed that all the variable names in the source CDFs are unique. If a duplicate variable name is encountered, the cdfmerge program will abort. This option cannot be used with the -prefixes option.

/PREFIXES=<prefix1,><prefix2,>...,<prefixN> (VMS)
-prefixes <prefix1,><prefix2,>...,<prefixN> (Unix/Windows)

This option allows to specify the user-provided prefixes (optional) to be used when naming the variables in the merged CDF file. Prefixes should be separated by a comma, and the number of prefixes must match the number of source CDFs. The first prefix corresponds to the first input file, the second prefix corresponds to the second input file, and so on. Prefix can be any text (combination of letters and numbers) that describes the file. User-defined prefix followed by a period (.) is added at the beginning of each variable name in the merged file. Suppose there are two files to be merged (a.cdf and b.cdf) and each file contains two variables (var1 and var2). The following command will merge these two files into a file called merged_ab.cdf that contains four variables named p1.var1, p1.var2, p2.var1, and p2.var2.

cdfmerge –prefix p1,p2 a.cdf b.cdf merged_ab.cdf

If the –prefix option is not specified in the above example, merged_ab.cdf will contain four variables named file1.var1, file1.var2, file2.var1, and file2.var2.

This option cannot be used with the -noprefix option

/[NO]DATAONLY (VMS)
-[no]dataonly (Unix/Windows)

This option allows to specify to copy the variable data only. If this option is selected, the merged file will not have a separate variable for each variable in the source CDFs. Suppose there are two files to be merged (a.cdf and b.cdf) and each file contains two variables (var1 and var2). The following command will merge these two files into a file called merged_ab.cdf that contains two variables named var1 and var2:

cdfmerge a.cdf b.cdf merged_ab.cdf

The merged file will have the metadata, i.e., global and variable attributes, from that of the first source CDF. The variable data of the same name are combined in the same sequence as the source CDFs are presented. Arrange the source files in a proper sequence if they are sequence-sensitive. The default is nodataonly.

/[NO]LOG (VMS)
-[no]log (Unix/Windows)

Specifies whether or not messages are displayed indicating the progress of CDF merging. The default is nolog.

/ABOUT (VMS)
-about (Unix/Windows)

Shows the library version that was used to create this tool program.

## Example(s):

VMS:

```
$ CDFMERGE /LOG test1.cdf test2.cdf outtest.cdf
$ CDFMERGE /PREFIXES=ID1,ID2,ID3 cdffile1 cdffile2 cdffile3 mergedFile
$ CDFMERGE /NOPREFIX cdffile1 cdffile2 cdffile3 mergedFile
$ CDFMERGE /LOG /FILE=filelist /DATAONLY
```

UNIX and Windows:

```
% cdfmerge -log test1.cdf test2.cdf outtest.cdf
% cdfmerge -prefixes ID1,ID2,ID3 cdffile1 cdffile2 cdffile3 mergedFile
```

```
% cdfmerge -noprefix cdffile1 cdffile2 cdffile3 mergedFile
% cdfmerge -log -file filelist -dataonly
```

Help is displayed when CDFmerge is executed without any arguments.

## 3.12 CDFdump

### 3.12.1 Introduction

The CDFdump program displays or extracts the contents of a CDF file to a screen (default) or text file. This program extracts and displays one variable at a time while the CDFexport programs extracts and displays the contents of a CDF file in a table format, each column representing a variable. CDFdump extracts the value of the variable attributes, but CDFexport does not extract the value of the variable attributes.

### 3.12.2 Executing the CDFdump Program

**Usage:**

VMS:

```
$ CDFDUMP [/[NO]FORMAT] [/DUMP=<option>] [/OUTPUT=<file-path>]
          [/VARS=<var1,var2,...varN>] [/ABOUT] [/NOHEADER]
          [/RECORDRANGE=<STARTREC,ENDREC>] cdf-path>
```

UNIX (including Mac OS X):
Windows:

```
% cdfdump [-[no]format] [-dump <option>] [-output <file-path>]
         [-vars <var1,var2,...varN>] [-about] [-noheader]
         [-recordrange <startrec,endrec>] <cdf-path>
```

**Parameter(s):**

<cdf-path> The pathname of the CDF file to be dumped.

**Qualifier(s):**

`[/[NO]FORMAT]` (VMS)
`[-[no]format]` (Unix/Windows)
Specifies whether or not the FORMAT attribute is used when displaying or extracting variable values (if the FORMAT attribute exists and an entry exists for the variable). The default is to use the format.

/DUMP=<option> (VMS)
`[-dump <option>]` (Unix/Windows)
Specifies how the program should produce a dump. Valid options are "all", "data", "metadata". The "all" option (the default) output includes detailed information about the CDF, not just only the metadata and variable data. The "data" option output includes variable data and minimum information about the variable. The "metadata" option output only includes the global attributes and the variable attributes.

/OUTPUT=<file-path>
-output <file-path> (Unix/Windows)
By default, the contents of file is displayed on the screen. This option redirects the output to a designated file indicated by <file-path>. If <file-path> does not have an extension, `.txt' is appended at the end of <file-path>. If this qualifier is entered as "source", then the source CDF pathname is used as its output name with its extension of ".cdf" being replaced by ".txt".

/VARS=<[var1,][var2,]...[varN]> (VMS)
-vars <[var1,][var2,]...[varN]> (Unix/Windows)
Specifies which variables in the CDF should be dumped. By default, all the variables. In the CDF are dumped. Variable names must be separated by a comma.

/RECORDRANGE=<startrec[,endrec]> (VMS)
-recordrange <startrec[,endrec]> (Unix/Windows)
Specifies the record range for variables in the CDF to be dumped. By default, all the variables' records in the CDF are dumped. If only one record number is provided, then all records after that number are dumped.

/ABOUT (VMS)
-about (Unix/Windows)
Shows the library version that was used to create this tool program.

/NOHEADER (VMS)
-noheader (Unix/Windows)
Whether to show the 1-line header "Dumping cdf from ...." from the dump output. No display if "-noheader" is specified..

**Example(s):**

VMS:

```
$ CDFDUMP my_data.cdf
```

UNIX and Windows:

```
% cdfdump my_data.cdf
```

Help is displayed when CDFdump is executed without any arguments.

## 3.13 CDFirsdump

### 3.13.1 Introduction

The CDFirsdump program displays the statistics of CDF Internal Records (IRs) within a file to a screen (default) or text file. This program can also dump, in hex form, each internal record. CDFirsdump can be used to show how a CDF is constructed: whether it's more compacted or very fragmented. Compacted CDF files provide much better performance as less IRs are visited to acquire data. If r/zVDR (r/zVaribale Descriptor Record) counts are quite fewer than VXR (Variable Index Record) or/and VVR (Variabale Value Record), a CDF file has become more fragmented. CDFconvert can use be to reconstruct CDF files to make them compact. See CDF Internal Format Description for information about various internal records in a CDF.

### 3.13.2 Executing the CDFirsdump Program

**Usage:**

VMS:

```
$ CDFirsDUMP [/OUTPUT=<file-path>]  [-BRIEF | -FULL]  [-[NO]PAGE]
             [-[NO]SUMMARY] [/ABOUT]
             <cdf-path>
```

UNIX (including Mac OS X):
Windows:

```
% cdfirsdump [-output <file-path>]  [-brief | -full]  [-[no]page]
            [-[no]summary] [-about]
            <cdf-path>
```

**Parameter(s):**

<cdf-path> The pathname of the CDF file to be dumped.

**Qualifier(s):**

/OUTPUT=<file-path>
-output <file-path> (Unix/Windows)
By default, the contents of file is displayed on the screen. This option redirects the output to a designated file indicated by <file-path>. If <file-path> does not have an extension, `.dmp' is appended at the end of <file-path>.

/[[NO]PAGE] (VMS)
[-[no]page] (Unix/Windows)
Specifies whether or not a page breaker is on when displaying the data on the screen. The default is no page breaker.

[/BRIEF | /FULL] > (VMS)
[-brief | -full]] (Unix/Windows)
Specifies how the program should produce a dump, whether it will display the full hex dump of all internal records, or just the summary. The default is full dump.

/[[NO]SUMMARY] (VMS)
[-[no]summary] (Unix/Windows)
Specifies whether or not the summary is to be displayed. The default is to display the summary.

/ABOUT (VMS)
-about (Unix/Windows)
Shows the library version that was used to create this tool program.

**Example(s):**

VMS:

```
$ CDFirsDUMP my_data.cdf
```

UNIX and Windows:

```
% cdfirsdump my_data.cdf
```

Help is displayed when CDFirsdump is executed without any arguments.

## 3.14 CDFvalidate

### 3.14.1 Introduction

The CDFvalidate program optionally performs sanity checks on certain data in the CDF files. The program goes through the Internal Records (IRs), which construct a CDF file, and tries to detect if the file is compromised. The relevant data fields in IRs are checked against its range and predefined values. The variable data values, however, are not checked as there are no criteria to check against. The program can show where/what an offending field is when such an anomaly is detected.

### 3.14.2 Executing the CDFvalidate Program

**Usage:**

VMS:

```
$ CDFVALIDATE [-VALIDATE | -NOVALIDATE]  [-DEBUG]
              [/ABOUT]
              <cdf-path1> <cdf-path2> …
```

UNIX (including Mac OS X):
Windows:

```
% cdfvalidate [-validate | -novalidate]  [-debug]
              [-about]
              <cdf-path1> <cdf-path2> …
```

**Parameter(s):**

<cdf-path1> The pathnames of the CDF files to be validated.
<cdf-path2>
…

**Qualifier(s):**

[/VALIDATE | /NOVALIDATE] > (VMS)
`[-validatre | -novalidate]` (Unix/Windows)
Specifies whether or not the file(s) is to be validated. The default is to validate. A CDF file will only go through the normal opening process when the "novalidate" option is specified.

[`/DEBUG`] (VMS)
`[-debug]` (Unix/Windows)

Specifies whether or not the debugging information is to be displayed when the data validation is being performed. This option is applicable to the previous "validate" option.

[/ABOUT ] (VMS)
[-about ] (Unix/Windows)
Shows the library version that was used to create this tool program.

**Example(s):**

VMS:

```
$ CDFVALIDATE my_data.cdf
$ CDFVALIDATE /DEBUG file1 file2
```

UNIX and Windows:

```
% cdfvalidate my_data.cdf
% cdfvalidate -debug file1 file2
```

Help is displayed when CDFvalidate is executed without any arguments.

## 3.15 CDFleapsecondsinfo

### 3.15.1 Introduction

The CDFleapsecondsinfo program displays the information of the leap seconds table that the CDF uses for processing the epoch data in CDF_TIME_TT2000 type.

### 3.15.2 Executing the CDFleapsecondsinfo Program

**Usage:**

VMS:

```
$ CDFLEAPSEONDSINFO [-DUMP | -NODUMP] [/ABOUT]
```

UNIX (including Mac OS X):
Windows:

```
% cdfleapsecondinfo [-dump | -nodump]  [-about]
```

**Parameter(s):**

**Qualifier(s):**

[/DUMP | /NODUMP] > (VMS)
`[-dump | -nodump]` (Unix/Windows)
Specifies whether or not the table contents of the leap seconds is to be displayed.

[/ABOUT ] (VMS)
[-about ] (Unix/Windows)
Shows the library version that was used to create this tool program.

**Example(s):**

VMS:

```
$ CDFLEAPSECONDSINFO /NODUMP
$ CDFLEAPSECONDSINFO /DUMP
```

UNIX and Windows:

```
% cdfvalidate -nodump
% cdfvalidate -dump
```

Help is displayed when CDFleapsecondsinfo is executed without any arguments.

# 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       Size
!   -----------          ---------------  ---------------  ----------   -------    -----
    <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 file name specified). When SkeletonCDF reads a skeleton table, this will be the name of the CDF created unless a CDF file name is specified on the command line. If the CDF name in the skeleton table is to be used, a full file name 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, ALPHAVMSi, SUN, SGi, DECSTATION, ALPHAOSF1, IBMRS, HP, PC, 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 2.3.15.

<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     Size
!     -----------   ---------------  ---------------  ----------  -------  ----------
        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     Size
!     -----------   ---------------  ---------------  ----------  -------  -----
        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_INT8, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_EPPCH16, CDF_TIME_TT2000, 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 data types (CDF_CHAR and CDF_UCHAR), each character is the element of a string. 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 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 data type) is defined in Section 2.5.4. 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:                       { 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 rVariable 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_INT8, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_EPOCH16, CDF_TIME_TT2000, CDF_CHAR, or CDF_UCHAR.

<n-elems> The number of elements of the data type. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in each string. For non-character 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_INT8, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_EPOCH16, CDF_TIME_TT2000, 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 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 data type. For non-character 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 2.5.4.

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             F             F  F

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

   "scale"          CDF_REAL4         {  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_lv1"         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 zVariable 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_INT8, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_EPOCH16, CDF_TIME_TT2000, CDF_CHAR, or CDF_UCHAR. |
| <n-elems> | The number of elements of the data type. For character data types (CDF_CHAR and CDF_UCHAR) this is the number of characters in each string. For non-character 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_INT8, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_EPOCH16, CDF_TIME_TT2000, 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 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 data type. For non-character 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 2.5.4.

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 CDF/IDL Interface and Legacy Applications

In addition to the built-in CDF functions (e.g. CDF_CREATE, CDF_OPEN, etc.) in IDL, the CDF distribution package prior to CDF 3.0 used to include its own set of IDL functions/procedures (e.g. CDFcreate, CDFopen, etc.) that are functionally equivalent to the ones that are built into IDL. The CDF office had to supply its own routines (hereafter referred to as the CDF/IDL interface) for manipulating CDF files in IDL because IDL originally didn't include support for CDF. Research Systems, Inc. (the developers of IDL) later implemented an interface to CDF as part of the IDL product. It differs from the interface provided with the CDF/IDL interface distribution in that it is intended more for the non-programmer (and is functionally similar to other interfaces IDL provide).

The CDF/IDL interface was always included as part of the CDF standard distribution package for Unix (albeit they are redundant with IDL's built-in CDF routines) up until CDF 2.7.2 to support legacy applications that utilities the CDF/IDL interface. Those users who must run applications that are based on the old CDF/IDL interface SHOULD NOT upgrade to CDF 3.0 or a later version. For those IDL applications that utilize the CDF/IDL interface, it's highly recommended to port these applications to use the IDL's built-in CDF interface. The migration is relatively easy since IDL's built-in CDF functions are very similar to the ones in the CDF/IDL interface.

## B.2 CDF Version 3.x and IDL

The advent of CDF 3.0 introduced, among many other things, an ability to create files bigger than 2G bytes and a new data type (CDF_EPOCH16) to address the limitation of the highest timestamp resolution offered by the CDF_EPOCH data type. Although the maximum timestamp resolution of CDF_EPOCH (milliseconds, 10**3) is adequate for many users, there are some users who need a finer timestamp. As a result, a new data type CDF_EPOCH16 was introduced to accommodate a finer timestamp that address up to picoseconds (10**12).

IDL 6.2 or earlier versions understand CDF 2.7.2 or earlier versions, but not CDF 3.0 or later versions. This means that if you need to take advantage of any of the new CDF 3.0 features (e.g. ability to create a CDF file bigger than 2 GB) or need to manipulate CDF files that were created with CDF 3.0 or later in IDL, you'll have to wait until RSI (manufacturer of IDL) incorporates the CDF 3.1 library into IDL 6.3, scheduled for release in late 1Q, 2006. In order to address this problem, as an interim solution, the CDF office obtained a copy of the IDL's built-in CDF functions (e.g. CDF_CREATE, CDF_OPEN, etc.) from RSI and extended it to support CDF Version 3.x's new file structure and data type. If you now need to use any of the CDF 3.0's new features or manipulate CDF files that were created with CDF 3.0 or a later version in IDL 6.2 or earlier, please contact the CDF support office at gsfc-cdf-support@lists.nasa.gov for a binary copy of the IDL CDF system routines.

## B.3 Backward File Compatibility with CDF 2.7

By default, a CDF file created by IDL 6.3 (scheduled for release in late 1Q, 2006) or later cannot be read by IDL 6.2 or earlier, or by CDF version 2.7.2 or earlier. However, IDL 6.3 or later versions can read CDF files by IDL 6.2 or earlier, or by CDF version 2.7.2 or earlier. The file incompatibility problem is due to the use of 64-bit file offset in CDF 3.0 and later versions to allow for creation of files bigger than 2G bytes. Note that IDL 6.3 uses CDF 3.1.

If you wish to create and share CDF files with colleagues who access CDF files using IDL 6.2 or earlier, or CDF version 2.7.2 or earlier, there's an IDL procedure called CDF_SET_CDF27_BACKWARD_FILE_COMPATIBLE that allow users of IDL version 6.3 or later to create a CDF file that can be read by IDL 6.2 or earlier, or by CDF version 2.7.2 or earlier. This procedure must be called prior to creating a CDF file with CDF_CREATE. If a file is created in the CDF 2.7 format, the maximum file size is 2G bytes. If you can't wait until IDL 6.3 is released and now need to use any of the CDF 3.0's new features or manipulate CDF files that were created with CDF 3.0 or later in IDL 6.2 or earlier, please contact the CDF support office at gsfc-cdf-support@lists.nasa.gov for a binary copy of the IDL CDF system routines.

# 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_BLOCKING_FACTOR[1] | An illegal blocking factor was specified. Blocking factors must be at least zero (0). [Error] |
| BAD_CACHESIZE | An illegal number of cache buffers was specified. The value must be at least zero (0). [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_COMPRESSION_PARM | An illegal compression parameter was specified. [Error] |
| BAD_DATA_TYPE | An unknown data type was specified or encountered. The CDF data types are defined in cdf.h for C applications and in cdf.inc for Fortran applications. [Error] |
| BAD_DECODING | An unknown decoding was 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 interval must not specify an element beyond the end of the dimension. [Error] |

[1] The status code BAD BLOCKING FACTOR was previously named BAD_EXTEND RECS.

| | |
|---|---|
| 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_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 variables with a non-character data type, 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_SCRATCH_DIR | An illegal scratch directory was specified. The scratch directory must be writeable and accessible (if a relative path was specified) from the directory in which the application has been executed. [Error] |
| BAD_SPARSEARRAYS_PARM | An illegal sparse arrays parameter was specified. [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_ALLOCATE RECORDS | Records cannot be allocated for the given type of variable (e.g., a compressed variable). [Error] |
| CANNOT_CHANGE | Because of dependencies on the value, it cannot be changed. Some possible causes of this error follow:<br>1. Changing a CDF's data encoding after a variable value (including a pad value) or an attribute entry has been written.<br>2. Changing a CDF's format after a variable has been created or if a compressed single-file CDF.<br>3. Changing a CDF's variable majority after a variable value (excluding a pad value) has been written.<br>4. Changing a variable's data specification after a value (including the pad value) has been written to that variable or after records have been allocated for that variable.<br>5. Changing a variable's record variance after a value (excluding the pad value) has been written to that variable or after records have been allocated for that variable.<br>6. Changing a variable's dimension variances after a value (excluding the pad value) has been written to that variable or after records have been allocated for that variable. |

7. Writing "initial" records to a variable after a value (excluding the pad value) has already been written to that variable.

8. Changing a variable's blocking factor when a compressed variable and a value (excluding the pad value) has been written or when a variable with sparse records and a value has been accessed.

9. Changing an attribute entry's data specification where the new specification is not equivalent to the old specification.

CANNOT_COMPRESS — The CDF or variable cannot be compressed. For CDFs, this occurs if the CDF has the multi-file format. For variables, this occurs if the variable is in a multi-file CDF, values have been written to the variable, or if sparse arrays have already been specified for the variable. [Error]

CANNOT_SPARSEARRAYS — Sparse arrays cannot be specified for the variable. This occurs if the variable is in a multi-file CDF, values have been written to the variable, records have been allocated for the variable, or if compression has already been specified for the variable. [Error]

CANNOT_SPARSERECORDS — Sparse records cannot be specified for the variable. This occurs if the variable is in a multi-file CDF, values have been written to the variable, or records have been allocated for the variable. [Error]

CDF_CLOSE_ERROR — Error detected while trying to close CDF. Check that sufficient disk space exists for the dotCDF file and that it has not been corrupted. [Error]

CDF_CREATE_ERROR — Cannot create the CDF specified - error from file system. Make sure that sufficient privilege exists to create the dotCDF file in the disk/directory location specified and that an open file quota has not already been reached. [Error]

CDF_DELETE_ERROR — Cannot delete the CDF specified - error from file system. Insufficient privileges exist the delete the CDF file(s). [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 file name 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 the CDF specified - error from file system. Check that the dotCDF 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 the CDF file - error from file system. Check that the dotCDF file is not corrupted. [Error] |
| CDF_WRITE_ERROR | Failed to write the CDF file - error from file system. Check that the dotCDF file is not corrupted. [Error] |
| CHECKSUM_ERROR | The data integrity verification through the checksum failed. [Error] |
| CHECKSUM_NOT_ALLOWED | The checksum is not allowed for old versioned files. [Error] |
| COMPRESSION_ERROR | An error occurred while compressing a CDF or block of variable records. This is an internal error in the CDF library. Contact CDF User Support. [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] |
| DECOMPRESSION_ERROR | An error occurred while decompressing a CDF or block of variable records. The most likely cause is a corrupted dotCDF file. [Error] |
| DID_NOTCOMPRESS | For a compressed variable, a block of records did not compress to smaller than their uncompressed size. They have been stored uncompressed. This can result if the blocking factor is set too low or if the characteristics of the data are such that the compression algorithm chosen is unsuitable. [Informational] |
| EMPTY_COMPRESSED_CDF | The compressed CDF being opened is empty. This will result if a program which was creating/modifying the CDF abnormally terminated. [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] |
| FORCED_PARAMETER | A specified parameter was forced to an acceptable value (rather than an error being returned). [Warning] |
| IBM_PC_OVERFLOW | An operation involving a buffer greater than 64k bytes in size has been specified for PCs running 16-bit DOS/Windows 3.*. [Error] |
| ILLEGAL_EPOCH_VALUE | The time or date value supplied for the CDF_EPOCH or CDF_EPOCH16 data type is invalid. [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] |

| | |
|---|---|
| ILLEGAL_TT2000_VALUE | The time or date value supplied for the CDF_TIME_TT2000 data type is invalid. [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] |
| NA_FOR_VARIABLE | The attempted operation is not applicable to the given variable. [Warning] |
| 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, an attempt was made to save the changes made prior to the severe error. In any event, the CDF should still be closed. [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 file name specified is correct. [Error] |
| NO_SUCH_ENTRY | No such entry for specified attribute. [Error] |
| NO_SUCH_RECORD | The specified record does not exist for the given variable. [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. This can also occur if an older CDF distribution is being used to read a CDF created by a more recent CDF distribution. Contact CDF User Support if you are sure that the specified file is a CDF that should be readable by the CDF distribution being used. CDF is backward compatible but not forward compatible. [Error] |
| PRECEEDING_RECORDS_ALLOCATED | Because of the type of variable, records preceding the range of records being allocated were automatically allocated as well. [Informational] |
| 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] |
| SCRATCH_CREATE_ERROR | Cannot create a scratch file - error from file system. If a scratch directory has been specified, ensure that it is writable. [Error] |
| SCRATCH_DELETE_ERROR | Cannot delete a scratch file - error from file system. [Error] |
| SCRATCH_READ_ERROR | Cannot read from a scratch file - error from file system. [Error] |
| SCRATCH_WRITE_ERROR | Cannot write to a scratch file - error from file system. [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] |
| SOME_ALREADY_ALLOCATED | Some of the records being allocated were already allocated. [Informational] |
| TOO_MANY_PARMS | A type of sparse arrays or compression was encountered having too many parameters. This could be causes by a corrupted CDF or if the CDF was created/modified by a CDF distribution more recent than the one being used. [Error] |
| TOO_MANY_VARS | A multi-file CDF on a 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] |
| UNKNOWN_COMPRESSION | An unknown type of compression was specified or encountered. [Error] |
| UNKNOWN_SPARSENESS | An unknown type of sparseness was specified or encountered. [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_DELETE_ERROR | An error occurred while deleting a variable file in a multi-file CDF. Check that sufficient privilege exist to delete the CDF files. [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 Supported Systems

CDF V3.2 is currently supported on the following computers/operating systems.

1. VAX (OpenVMS & POSIX shell)
2. Sun (Solaris)
3. DECstation (ULTRIX)
4. Silicon Graphics Iris & Power Series (IRIX)
5. IBM RS6000 series (AIX)[1]
6. HP 9000 series (HP-UX)[1]
7. PC (MS-DOS, Windows NT/95/98/2000/XP, Linux, & QNX)
8. NeXT (Mach)[1]
9. DEC Alpha (OSF/1 & OpenVMS)
10. Macintosh (Mac OS X)

## D.2 Compatibility with CDF 2.7.2 and Earlier Versions

CDF V3.1 is backward compatible with the previous versions of CDF, and it can read CDF files that were created with CDF 3.0 or CDF 2.7.2 or earlier. If a file was created with CDF 2.7 and read and modified by CDF 3.1, the resultant file will be saved in the CDF 2.7 format, not CDF 3.1. The same principle applies to files that were created with CDF 2.5 and 2.6. CDF files that are created from scratch with CDF V3.1 are compatible with CDF 3.0, but not compatible (due to a 64-bit file offsets used in CDF 3.0 or later versions) with CDF 2.7.2 or earlier, and an attempt to read CDF 3.0 or 3.1 files from CDF 2.7.2 or earlier will produce an error.

---

[1] Due to lack of user's interest and hardware, this operating system is not tested. If you need to run the CDF V3.1 library on either HP-UX or IBM's AIX operating system, please contact the CDF support office at gsfc-cdf-support@lists.nasa.gov.

Users of CDF 3.0 or later versions will be able to create CDF files that can be read by CDF 2.7.2 or earlier by using the CDFsetFileBackward function (in C) or CDF_set_FileBackward subroutine (in Fortran), or using the CDF_FILE_BACKWARD environment variable. See section 4.18 of the CDF C Reference Manual and the CDF Fortran Reference Manual for details on how to create CDF 2.7-compatible files.

The command-line version of the CDFedit, CDFexport and CDFconvert utility programs can now create CDF files that can be read by CDF 2.7.2 or earlier.

The <GET_, CDF_INFO_> routine now returns the data type of 64-bit off_t (or __int64 on Windows) for the compressed file size (cSize) and uncompressed file size (uSize) parameters in V3.1 while they used to return as 32-bit long integer in V2.5, 2.6 or 2.7. Thus, if you have a legacy application that calls the <GET_, CDF_INFO_> routine, you MUST change the data type of the cSize and uSize parameters to off_t (or __int64 on Windows) from *long* to access files that were created with V3.*. If the file accessed was created with V2.5, V2.6, or V2.7, you should always use 'long' instead of off_t to get the correct results. Using off_t for non-V3.* files from V3.* library may or may not return the correct results depending upon what operating system it is executed under.

## D.3 Changes

The following features have been added to from CDF V3.0 to V3.2:

1. Add checksum feature to allow verification of data integrity in a CDF file upon its opening.

2. Ability to create a CDF file that is compatible with CDF 2.7.2 or earlier.

3. Addition of the easy-to-use Extended Standard Interface that allows almost all of the CDF operations which were previously only available through the Internal Interface.

4. Retrofit of CDF tools to include the option of the new checksum feature while creating a new CDF.

5. Retrofit of CDFedit and CDFexport to create a CDF file that is compatible with CDF 2.7.2 or earlier.

6. Miscellaneous bug fixes.

7. Increase in cache buffer size from 512 to 10240 bytes.

8. When read-only mode is set, all metadata is read into memory where requests for metadata are then directed. This improves metadata access performance in most situations.

# Appendix E Glossary

| | |
|---|---|
| AHUFF | The Adaptive Huffman compression algorithm. |
| allocated records | For uncompressed variables in a single-file CDF it is possible for an application to allocate records before they are written. This has the advantage of reducing the indexing overhead in the dotCDF file which will improve performance when accessing a variable. An application would generally then write to the records that were allocated. |
| 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. |
| blocking factor | For a standard variable (in a single-file CDF), the blocking factor is the minimum number of records actually allocated when a new record is written. More records may be allocated than are actually needed in order to keep the variable's records as contiguous as possible (with the assumption that the records will eventually be written).<br><br>For a compressed variable in a single-file CDF, the blocking factor is the maximum number of records per compressed block.<br><br>For an uncompressed variable having sparse records in a single-file CDF, the blocking factor is the number of records allocated in the staging scratch file. For this type of variable the staging scratch file is used to optimize the indexing in the dotCDF file by storing sequential records contiguously when possible.<br><br>Blocking factors are not applicable to variables in multi-file CDFs. |
| 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 or writing them to disk. |
| CDF | This term is used in more than one way. . .<br><br>1. The actual files that contain your data/metadata. For example: The CDF library must be used to create a "CDF."<br><br>2. The software distribution containing the CDF library, include files, and toolkit. For example: We like using "CDF" to store our data. |

| | |
|---|---|
| CDF base name | The file name of a CDF minus the extension (or extensions if a multi-file CDF). |
| 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. |
| CDFedit | A CDF toolkit program that allows the display and modification of a CDF's contents. |
| CDFexport | A CDF toolkit program that allows the (possibly filtered) contents of a CDF to be exported to the terminal screen, a text file, or another CDF. |
| CDFstats | A CDF toolkit program that generates a report containing various statistics about a CDF's variables. |
| CDFcompare | A CDF toolkit program that reports any differences between two CDFs. |
| CDFconvert | A CDF toolkit program that allows various overall properties of a CDF to be changed (in a newly created CDF). |
| CDFinquire | A CDF toolkit program that displays the version of the CDF distribution being used, many of the configurable parameters, and the default CDF toolkit qualifiers/options. |
| 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. |
| cdfdf.inc | An include file used in Digital Visual Fortran applications. |
| cdfdvf2.inc | An include file used in Digital Visual Fortran applications. |
| cdfdvf3.inc | An include file used in Digital Visual Fortran applications. |
| cdfdvf.inc | An include file used in Digital Visual 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. |
| Compression | The process of encoding a group of bytes into a smaller group of bytes, storing the smaller group of bytes, and then decoding the smaller group of bytes back to the original group of bytes. CDF allows both a CDF and/or individual variables to be compressed when stored. |
| 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. |
| DLLCDF.DLL | The dynamic CDF library for Windows NT/95/98 systems. |
| dimension variance | The property of a variable that specifies whether or not the values along a dimension change or stay the same. |
| Dimensionality | The number of dimensions and the dimension sizes for the rVariables or a zVariable. |
| dotCDF file | A file having an extension of .cdf (or .CDF if the operating system being used prefers uppercase). For a single-file CDF this will be the only file. For a multi-file CDF this file will exist along with zero or more variable files (depending on the number of variables in the CDF). |
| 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. |
| error code | A status code indicating that a fatal condition was encountered. The operation was aborted. |
| 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. |
| GZIP | The Gnu ZIP compression algorithm. |
| host decoding | The decoding of the computer currently being used. |

| | |
|---|---|
| host encoding | The encoding of the computer currently being used. |
| HUFF | The Huffman compression algorithm. |
| 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 a function is performed. |
| libcdf.a | The static CDF library on UNIX systems. |
| libcdf.sl | The dynamic CDF library on HP-UX systems. |
| libcdf.so | The dynamic CDF library on UNIX (other than HP-UX). |
| LIBCDF.LIB | The static CDF library on MS-DOS or Windows NT/95/98 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. |
| Metadata | Data about data. A CDF stores metadata using attributes and attribute entries. |
| 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. |
| 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, a function 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. |
| record variance | The property of a variable that specifies whether or not its values change from record to record. |
| reserve percentage | For a compressed variable, the reserve percentage specifies how much additional space to allocate in the dotCDF file when a compressed block of records is initially written. A value of 0 (zero) causes no reserve space to be allocated. Values from 1 to 100 cause at least that percentage of the uncompressed size to be allocated. Values greater than 100 cause that percentage of the compressed size to be allocated (but not exceeding the uncompressed size). |
| eEntry | An entry for a vAttribute corresponding to an rVariable. |
| RLE | A run-length encoding compression algorithm. Currently, the only type of RLE compression supported is the run-length encoding of bytes containing zero. |
| 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. |
| scratch directory | The directory in which the CDF library creates scratch files. This directory may be specified by a user or an application. |
| scratch files | Temporary files used by the CDF library to minimize core memory usage. |
| 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, and NRV variable values. |
| skeleton table | A text file containing the control, metadata, and traditionally only the NRV variable values of a CDF. RV variable values may now also be included in a skeleton table. A skeleton table is read by the SkeletonCDF toolkit program which then creates the corresponding skeleton CDF (or complete CDF if the RV variable values also existed in the skeleton table). The SkeletonTable toolkit program can be used to create a skeleton table from a CDF. |
| SkeletonCDF | A CDF toolkit program which creates a skeleton CDF based on a skeleton table. A complete CDF may also be created if the skeleton table contained RV variable values in addition to NRV variable values. |
| SkeletonTable | A CDF toolkit program which creates a skeleton table from a CDF. |
| sparse arrays | A property assigned to a variable indicating that only those values written to a record should be stored. Because the values of a variable record can be written in any order this allows gaps of missing values to occur. |
| sparse records | A property assigned to a variable indicating that only those records written to the variable should be stored. Because the records of a variable can be written in any order this allows gaps of missing records to occur. |
| 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). |
| standard variable | A variable in a single-file CDF that is not compressed nor has sparse records or arrays. |
| 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 file | In a multi-file CDF, these are the files containing the data values for each variable (in one file per variable). These files are named using the CDF's base name with extensions of `.v0', `.v1', and so on for rVariables and `.z0', `.z1', and so on for zVariables. |
| variable scope | Variable scope indicates that an attribute describes some property of each variable. |
| variance (dimension) | The property of a variable that specifies whether or not the values along a dimension change or stay the same. |
| variance (record) | The property of a variable that specifies whether or not its values change from record to record. |
| 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


-0.0 to 0.0 Mode, 25
Adaptive Huffman compression, 59
allocated records, 43
assumed scope, 54
attributes, 16, 54
  creating, 54
  deleting, 55
  entries, 17, 55
    accessing, 55
    data specification, 55
      data type, 56
      number of elements, 56
    deleting, 56
    gEntry, 17, 54
    numbering, 55
    rEntry, 17, 54
  FILLVAL, 64, 85
  naming, 54
    case sensitivity, 54
    trailing blanks, 54
  numbering, 54, 82
    assigning, 54
  SCALEMAX, 64, 85
  SCALEMIN, 64, 85
  scopes, 54
    assumed, 54
      converting, 55
      correcting, 55
    global, 54
    purpose, 54
    restrictions, 54
    variable, 54
  special, 64
    usage, 65, 68, 85, 90
  VALIDMAX, 64, 85
  VALIDMIN, 64, 85
  vAttributes, 17, 54
big-endian, 32
blocking factor, 44
caching scheme, files, 26
CDF, 5
  definition, 5
  deleting, 71, 77, 95
CDF Java Interface, 19
CDF library, 7, 23
  caching scheme, 26
    selecting, 66, 72, 79, 83, 87, 93, 95
  interfaces, 18, 23
  limits, 26
    open CDFs, 26
  modes, 24
    -0.0 to 0, 25
    decoding, 34
    performance considerations, 35
  read-only, 24
  zMode, 24, 66, 71, 78, 82, 87, 92, 95
    example, 24
    selecting, 24
    zMode/1, 24
    zMode/2, 24
  scratch files, 26
CDF toolkit, 17, 61
  CDFcompare, 81
  CDFconvert, 75
  CDFdir, 98
  CDFdump, 101
  CDFedit, 65
  CDFexport, 68
  CDFinquire, 97
  CDFirsdump, 103
  CDFleapsecondsinfo, 106
  CDFmerge, 99
  CDFstats, 84
  CDFvalidate, 104
  command line syntax, 61
  default settings, 62
  Java version, 63
  Macintosh OS X, 62
  SkeletonCDF, 94
  SkeletonTable, 90
  Windows NT/95/98/2000/XP, 63
CDF_ATTR_NAME_LEN256, 35
CDF_EPOCH, 57
CDF_EPOCH16, 57
CDF_error, 127
CDF_TIME_TT2000, 58
CDF_VAR_NAME_LEN256, 35
CDFcompare, 81
  executing, 81
  output, 84
CDFconvert, 75
  executing, 75
  output, 80
CDFdir, 98
  executing, 98
  output, 99
CDFdump, 101
  executing, 102
cdfedit, 19, 65
  executing, 65
  interaction with, 67
CDFerror, 127
CDFexport, 68
  executing, 69
  interaction with, 74

CDFinquire, 97
executing, 97
output, 98
CDFirsdump, 103
executing, 103
CDFleapsecondsinfo, 106
executing, 106
CDFmerge, 99
executing, 99
CDFs, 28
accessing, 28
backward compatible, 75
browsing, 65
checksum
changing, 75
closing, 28
comparing, 81
compression, 12, 35
algorithms, 58
changing, 75
conceptual organization, 6
converting, 75
creating, 28
editing/modifying, 65
encoding, 12, 31
changing, 31, 75
equivalent, 32
host, 31
network, 32
performance considerations, 33
exporting, 68
file extension, 30
file format, 7, 29
changing, 29
default, 29
multi-file, 30
performance considerations, 30
single-file, 30
filtering, 68
leap seconds, 60
limits, 26, 35
listing, 68
naming
wildcards, 62
naming, 29, 35
trailing blanks, 29
opening, 28
statistics, 84
subsetting, 68
TT2000, 60
verifying, 81
CDFstats, 84
executing, 85
output, 88
CDFvalidate, 104
executing, 105
compression, 12
algorithms, 58
CDF file(s), 12, 35
variable(s), 12, 45
conceptual organization, 6
data specification, 38
attribute entry, 55
variable, 38
data types, 56
character, 57
EPOCH, 57
EPOCH16, 57
equivalent data types, 58
floating point, 56
integer, 56
TT2000, 58
decoding, CDF, 34
definitions file, 62
dimensionality, variable, 38
encoding, CDF, 31
EPOCH, 57
syntax, 57
EPOCH16, 57
syntax, 57
examples, 19
cdfedit, 19
conceptual view, 14
data set, flat, 14
physical view, 15
skeleton table, 19, 120
FILLVAL attribute, 64, 85
FORMAT attribute, 64, 68, 85
format, CDF, 29
GZIP compression, 59
host decoding, 34
host encoding, 31
Huffman compression, 59
hyper access, variable, 48
IDL
CDF's interface, 24
IEEE 754, 25, 33, 142
indexing, variable records, 30
initial records, 43
interfaces
IDL, 24
internal, 23
standard, 23
Internal Interface, 23
limits, 26, 35
attribute name length, 35
CDF file name length, 36
dimensions, 35
open CDFs, 26
variable name length, 35
little-endian, 32
majority
variable, 46
MONOTON attribute, 64, 69, 85
multi-file format, 30
multiple variable access, 51
network encoding, 32
pad values, variable, 52
performance considerations
decoding, 30, 35
encoding, 33
format, 30
majority, 47
qualifier

special, 64
read-only mode, 24
reserve percentage, compression, 46
Run-Length encoding compression, 59
SCALEMAX attribute, 64, 85
SCALEMIN attribute, 64, 85
scope, attribute, 54
scratch files, 26
sequential access, variable, 47, 50
single-file format, 30
Skeleton CDF, 94
skeleton table, 90, 94
creating, 90, 96
example, 120
file extension, 96
SkeletonCDF, 18, 20, 94
executing, 94
SkeletonTable, 18, 90, 109
executing, 90
output, 94
sparseness
arrays, 12, 45
records, 12, 42
Standard Interface, 23
status codes, 127
categories, 127
trailing blanks
attribute name, 54
CDF file name, 29
variable name, 37
TT2000, 58
syntax, 58
VALIDMAX attribute, 64, 69, 85
VALIDMIN attribute, 64, 68, 85
variables, 6, 13, 36
accessing, 36
hyper read/write, 48
example, 49
reading, 47
writing, 47
multiple variable, 51
sequential values, 50
example, 50
single values, 47
arrays, 38
closing, 37
compression, 12, 45
algorithms, 58
reserve percentage, 46
data specification, 38
changing, 38
data type, 38
number of elements, 38
deleting, 37
dimensionality, 38
majority, 46
changing, 47
example, 47
naming, 37
case sensitivity, 37
trailing blanks, 37
non-record-variant (NRV), 39
numbering, 37
assigning, 37
opening, 37
pad values, 52
default, 53
usage, 52
records, 40
allocated, 43
blocking factor, 44
compression, 45
reserve percentage, 46
deleting, 45
indexing, 30
initial, 43
maximum, 40
numbering, 42
physical, 40
sparse, 42
virtual, 40
record-variant (RV), 39
reserve percentage, 46
rVariables, 13
sparse arrays, 45
sparse records, 42
zVariables, 16
variance
dimension, 39
record, 39
XDR, 32
zMode, 24