CDF
User's Guide

Version 2.3.0, October 1, 1992

National Space Science Data Center

Copyright © 1990, 1991, 1992 NASA/GSFC/NSSDC

National Space Science Data Center
NASA - Goddard Space Flight Center
Greenbelt, Maryland 20771 (USA)

NSInet — NSSDCA::CDFSUPPORT
Internet — CDFSUPPORT@NSSDCA.GSFC.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

Preface **1**

1 CDF Primer **3**

1.1 Introduction . . . 3

1.2 Why Use CDF? . . . 3

1.3 Conceptual Organization . . . 3

1.4 Features of the CDF Library . . . 4

1.4.1 File Format Options . . . 4

1.4.2 Data Encoding Options . . . 5

1.4.3 Variable Data Access Options . . . 8

1.4.4 CDF Library Interface Options . . . 8

1.5 Organizing Your Data in a CDF . . . 9

1.5.1 Variables . . . 9

1.6 Attributes . . . 13

1.7 CDF Toolkit . . . 13

1.8 Library Interface Routines . . . 14

1.8.1 Standard Interface . . . 15

1.8.2 Internal Interface . . . 15

1.9 Examples . . . 16

1.9.1 Creating a CDF, the hard way (but not that hard) . . . 16

1.9.2 Creating a CDF, an easier way . . . 24

2 General CDF Concepts 31

2.1 CDF Interfaces . . . 31

2.1.1 Standard Interface . . . 31

2.1.2 Internal Interface . . . 31

2.1.3 Obsolete Interface, VMS Systems Only . . . 31

2.2 Access to CDFs . . . 33

2.2.1 Version 2 CDFs . . . 33

2.2.2 Version 1 CDFs . . . 33

2.3 CDF Format . . . 33

2.3.1 Multi-file CDFs . . . 33

2.3.2 Single-file CDFs . . . 34

2.3.3 Performance . . . 34

2.4 CDF Encoding . . . 34

2.4.1 Host Encoding . . . 35

2.4.2 Network Encoding . . . 35

2.4.3 Performance . . . 35

2.5 CDF Data Types . . . 35

2.5.1 Integer Data Types . . . 36

2.5.2 Floating Point Data Types . . . 36

2.5.3 Character Data Types . . . 36

2.5.4 EPOCH Data Type . . . 36

2.5.5 Equivalent Data Types . . . 37

2.6 CDF Limits . . . 37

2.7 Variables . . . 38

2.7.1 Access . . . 38

2.7.2 Naming . . . 38

2.7.3 Numbering . . . 38
2.7.4 Dimensionality . . . 38
2.7.5 Records . . . 39
2.7.6 Data Specification . . . 41
2.7.7 Record Variance . . . 42
2.7.8 Dimension Variance . . . 42
2.7.9 Majority . . . 43
2.7.10 Hyper Access . . . 44
2.7.11 Sequential Access . . . 46
2.7.12 Variable Pad Values . . . 47
2.8 Attributes . . . 48
2.8.1 Naming . . . 48
2.8.2 Numbering . . . 48
2.8.3 Attribute Scopes . . . 48
2.8.4 Attribute Entries . . . 49
2.9 NSSDC Standard . . . 50
2.9.1 Standard Attributes . . . 50
2.9.2 Standard Variables . . . 51

3 CDF Toolkit Reference 53

3.1 Introduction . . . 53
3.2 CDFdir . . . 54
3.2.1 Introduction . . . 54
3.2.2 Executing the CDFdir program . . . 54
3.2.3 Output from the CDFdir program . . . 55
3.3 CDFcompare . . . 55
3.3.1 Introduction . . . 55
3.3.2 Executing the CDFcompare program . . . 55

3.3.3 Output from the CDFcompare program 57
3.4 CDFconvert 57
3.4.1 Introduction 57
3.4.2 Executing the CDFconvert program 57
3.4.3 Output from the CDFconvert program 60
3.5 CDFscope 60
3.5.1 Introduction 60
3.5.2 Executing the CDFscope program 60
3.5.3 Output from the CDFscope program 61
3.6 CDFstats 62
3.6.1 Introduction 62
3.6.2 Executing the CDFstats program 62
3.6.3 Output from the CDFstats program 64
3.7 CDFinquire 66
3.7.1 Introduction 66
3.7.2 Executing the CDFinquire program 66
3.7.3 Output from the CDFinquire program 68
3.8 SkeletonTable 71
3.8.1 Introduction 71
3.8.2 Executing the SkeletonTable program 72
3.8.3 Output from the SkeletonTable program 73
3.9 CDFskeleton 73
3.9.1 Introduction 73
3.9.2 Executing the CDFskeleton program 74
3.9.3 Creating the skeleton table 75
3.10 CDFbrowse 76
3.10.1 Introduction 76

3.10.2 Executing the CDFbrowse program . . . 76
3.10.3 Interaction with CDFbrowse . . . 77
3.11 CDFlist . . . 78
3.11.1 Introduction . . . 78
3.11.2 Executing the CDFlist program . . . 78
3.11.3 Interaction with CDFlist . . . 79
3.12 CDFwalk . . . 81
3.12.1 Introduction . . . 81
3.12.2 Executing the CDFwalk program . . . 81
3.12.3 Interaction with CDFwalk . . . 82

4 CDF C PROGRAMMING REFERENCE 85

4.1 Introduction . . . 85
4.2 Compiling . . . 85
4.2.1 Specifying `cdf.h` location in the compile command . . . 85
4.2.2 Specifying `cdf.h` location in the source file . . . 87
4.3 Linking . . . 87
4.3.1 VMS systems . . . 87
4.3.2 UNIX systems . . . 87
4.3.3 MS-DOS systems . . . 88
4.4 Linking, shared CDF library . . . 89
4.5 Item Referencing . . . 89
4.6 CDF Parameters — Predefined Constants and Types . . . 90
4.6.1 Defined Types . . . 90
4.6.2 CDFstatus Constants . . . 90
4.6.3 CDF Formats . . . 90
4.6.4 CDF Datatypes . . . 90
4.6.5 Data Encodings . . . 91

4.6.6 Variable Majorities . . . 92
4.6.7 Record/Dimension Variances . . . 92
4.6.8 Attribute Scopes . . . 93
4.6.9 Operational Limits . . . 93
4.6.10 Limits of Names and Other Character Strings . . . 93
4.7 C Function Reference . . . 94
4.7.1 CDFcreate (Standard Interface) . . . 94
4.7.2 CDFopen (Standard Interface) . . . 95
4.7.3 CDFdoc (Standard Interface) . . . 96
4.7.4 CDFinquire (Standard Interface) . . . 98
4.7.5 CDFclose (Standard Interface) . . . 99
4.7.6 CDFdelete (Standard Interface) . . . 100
4.7.7 CDFerror (Standard Interface) . . . 101
4.7.8 CDFattrCreate (Standard Interface) . . . 102
4.7.9 CDFattrNum (Standard Interface) . . . 103
4.7.10 CDFattrRename (Standard Interface) . . . 104
4.7.11 CDFattrInquire (Standard Interface) . . . 104
4.7.12 CDFattrEntryInquire (Standard Interface) . . . 106
4.7.13 CDFattrPut (Standard Interface) . . . 107
4.7.14 CDFattrGet (Standard Interface) . . . 109
4.7.15 CDFattrCorrectScope (Standard Interface) . . . 111
4.7.16 CDFvarCreate (Standard Interface) . . . 112
4.7.17 CDFvarNum (Standard Interface) . . . 114
4.7.18 CDFvarRename (Standard Interface) . . . 115
4.7.19 CDFvarInquire (Standard Interface) . . . 116
4.7.20 CDFvarPut (Standard Interface) . . . 118
4.7.21 CDFvarGet (Standard Interface) . . . 119

4.7.22 CDFvarHyperPut (Standard Interface) . . . 121
4.7.23 CDFvarHyperGet (Standard Interface) . . . 123
4.7.24 CDFlib (Internal Interface) . . . 124
4.8 Interpreting CDF Status Codes . . . 164
4.9 `CDF_EPOCH` Values . . . 165
4.9.1 `computeEPOCH` . . . 166
4.9.2 `EPOCHbreakdown` . . . 166
4.10 Custom C Functions . . . 166

5 CDF FORTRAN PROGRAMMING REFERENCE 169

5.1 Introduction . . . 169
5.2 Compiling . . . 169
5.2.1 VMS systems . . . 170
5.2.2 UNIX systems . . . 170
5.2.3 MS-DOS systems . . . 170
5.3 Linking . . . 171
5.3.1 VMS systems . . . 171
5.3.2 UNIX systems . . . 171
5.3.3 MS-DOS systems . . . 171
5.4 Linking, shared CDF library . . . 172
5.5 Argument Passing . . . 173
5.6 Item Referencing . . . 173
5.7 CDF Parameters — Predefined Constants and Types . . . 174
5.7.1 Status Code Constants . . . 174
5.7.2 CDF Formats . . . 174
5.7.3 CDF Datatypes . . . 174
5.7.4 Data Encodings . . . 175
5.7.5 Variable Majorities . . . 175

5.7.6 Record/Dimension Variances . . . 176
5.7.7 Attribute Scopes . . . 176
5.7.8 Operational Limits . . . 177
5.7.9 Limits of Names and Other Character Strings . . . 177
5.8 FORTRAN Function Reference . . . 177
5.8.1 CDF_create (Standard Interface) . . . 177
5.8.2 CDF_open (Standard Interface) . . . 179
5.8.3 CDF_doc (Standard Interface) . . . 180
5.8.4 CDF_inquire (Standard Interface) . . . 182
5.8.5 CDF_close (Standard Interface) . . . 183
5.8.6 CDF_delete (Standard Interface) . . . 184
5.8.7 CDF_error (Standard Interface) . . . 185
5.8.8 CDF_attr_create (Standard Interface) . . . 186
5.8.9 CDF_attr_num (Standard Interface) . . . 187
5.8.10 CDF_attr_rename (Standard Interface) . . . 188
5.8.11 CDF_attr_inquire (Standard Interface) . . . 189
5.8.12 CDF_attr_entry_inquire (Standard Interface) . . . 191
5.8.13 CDF_attr_put (Standard Interface) . . . 193
5.8.14 CDF_attr_get (Standard Interface) . . . 194
5.8.15 CDF_attr_correct_scope (Standard Interface) . . . 196
5.8.16 CDF_var_create (Standard Interface) . . . 197
5.8.17 CDF_var_num (Standard Interface) . . . 199
5.8.18 CDF_var_rename (Standard Interface) . . . 201
5.8.19 CDF_var_inquire (Standard Interface) . . . 202
5.8.20 CDF_var_put (Standard Interface) . . . 203
5.8.21 CDF_var_get (Standard Interface) . . . 205
5.8.22 CDF_var_hyper_put (Standard Interface) . . . 206

5.8.23 CDF_var_hyper_get (Standard Interface) . . . 208
5.8.24 CDF_lib (Internal Interface) . . . 210
5.9 Interpreting CDF Status Codes . . . 253
5.10 `CDF_EPOCH` Values . . . 255
5.10.1 `compute_EPOCH` . . . 255
5.10.2 `EPOCH_breakdown` . . . 255

A Skeleton Table Format **257**

A.1 Introduction . . . 257
A.2 Header Section . . . 258
A.3 gAttributes Section . . . 260
A.4 vAttributes Section . . . 262
A.5 rVariables Section . . . 263
A.6 zVariables Section . . . 266
A.7 End Section . . . 269
A.8 Example Skeleton Table . . . 269

B CDF's IDL Support **275**

B.1 Introduction . . . 275
B.2 Using CDF's IDL Interface . . . 275
B.3 CDF "Include" Files . . . 276
B.4 Available Functions/Procedures . . . 276
B.5 Example IDL Session . . . 278
B.6 Example IDL Function . . . 279

C CDF Status Codes **283**

C.1 Introduction . . . 283
C.2 Success Code and Message . . . 283
C.3 Informational Codes and Messages . . . 283

C.4 Warning Codes and Messages 284
C.5 Error Codes and Messages 284

D C Programming Summary **289**

D.1 Introduction 289
D.2 Standard Interface 289
D.3 Internal Interface 292

E FORTRAN Programming Summary **297**

E.1 Introduction 297
E.2 Standard Interface 297
E.3 Internal Interface 301

F Glossary **305**

G CDF Release Notes **307**

G.1 Introduction 307
G.2 Supported Systems 307
G.3 Changes for CDF V2.3 308

Index **309**

List of Figures

1.1 Conceptual View of a CDF, 0-Dimensional rVariables 5

1.2 Conceptual View of a CDF, 2-Dimensional rVariables 6

1.3 Conceptual View of a CDF, zVariables 7

1.4 Multi-File Format 8

1.5 Single-File Format 9

2.1 Physical vs. Virtual Records 40

2.2 Physical vs. Virtual Dimensions 43

List of Tables

1.1 Example Data Set - "Flat" Representation (0-dimensional) 10

1.2 Example CDF - 2-Dimensional Representation (Conceptual) 11

1.3 Example CDF - Specification for 2-Dimensional Representation 11

1.4 Example CDF - 2-Dimensional Representation (Physical) 12

1.5 vAttribute rEntries for the `Temperature` rVariable 14

2.1 Standard Interface Routines 32

2.2 Internal Interface Routines 32

2.3 NSSDC Standard CDF Attributes 50

Preface

About this document

This document is intended to be 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 and programmer. However, it does not serve as a cookbook for the proper methods of architecturing 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 multi-dimensional 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 towards 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 manipulating it. 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 catagories. 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 entires (metadata) describing the CDF in global terms or specifically for a single variable. This dual function of CDF is what provides it's "data set independence". Both the data dictionary (attributes) and the data objects (variables) are combined into an integrated data set. An important element of the CDF conceptual data abstraction is the "virtual" dimensional layer that allows data objects that share a subset of the overall CDF dimensionality to be projected into the full dimensional space. This capability is made available through the use of logical dimensional variances which indicate the subset of CDF dimensions that are applicable. A few simple examples of this...

1. Making time ordered measurements on a grid of data. Since the grid is fixed, it needs to be stored only once while still being projected at different time instances.
2. A two dimensional spectrogram with nonlinear axes. In this case the two one-dimensional vectors

describing the nonlinear gridding can be expressed as projections of the two-dimensional spectrogram. This allows easier intercomparison of data objects within the CDF while maintaining efficient storage of the actual data.

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 it's 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 (eg. NSSDC Graphics System, NGS).

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

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

The programming layer (CDF library) provides the essential framework on which graphical and data analysis packages can be created. The CDF library allows developers of CDF-based systems to easily create applications which permit users to slice data across multi-dimensional subspaces, access entire structures of data, perform sub-sampling of data, and access one data element independently regardless of its relationship to any other data element. CDF data sets are portable across any platform supported by CDF. These currently consist of: VAX (VMS), Sun (SunOS), DECstation (ULTRIX), Silicon Graphics Iris and Power Series (IRIX), IBM RS/6000 series (AIX), HP 9000 series (HP-UX), NeXT (MaCH), and IBM PC (MS-DOS).

How is CDF useful to me?

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

Chapter 1

CDF 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. 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.

1.2 Why Use CDF?

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

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

The term "CDF" is also used to refer to the physical files which the CDF Library generates.

1.3 Conceptual Organization

An important feature of CDF is that it can handle data sets which are inherently multi-dimensional in addition to data sets which are scalar. To do this, CDF groups data by "variables" whose values are

conceptually organized into arrays. The dimensionality of these variable arrays depends upon the data and is specified by the user when the CDF is created. For scalar data, as an example, the array of values would degenerate to 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 quite often does, contain multiple CDF records. This is useful for data where there are repeated observations at different times, altitudes, or pressure levels.

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

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

1.4 Features of the CDF Library

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

1.4.1 File Format Options

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

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

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

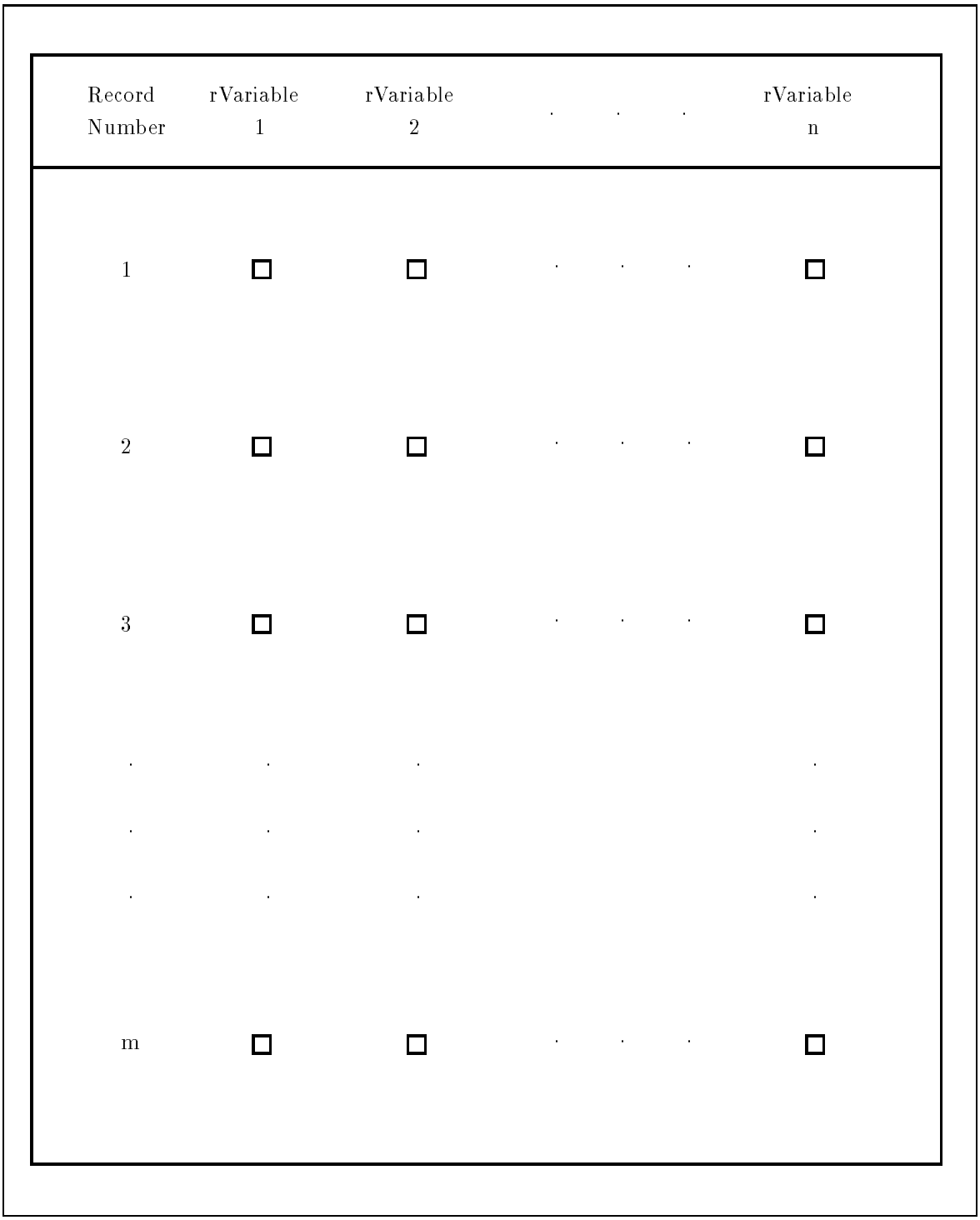

Figure 1.1: Conceptual View of a CDF, 0-Dimensional rVariables

1.4.2 Data Encoding Options

When creating a CDF, the programmer has the option to choose from one of two data encoding schemes. The first is the host encoding scheme which is also the default. Using the host encoding scheme, the data in the

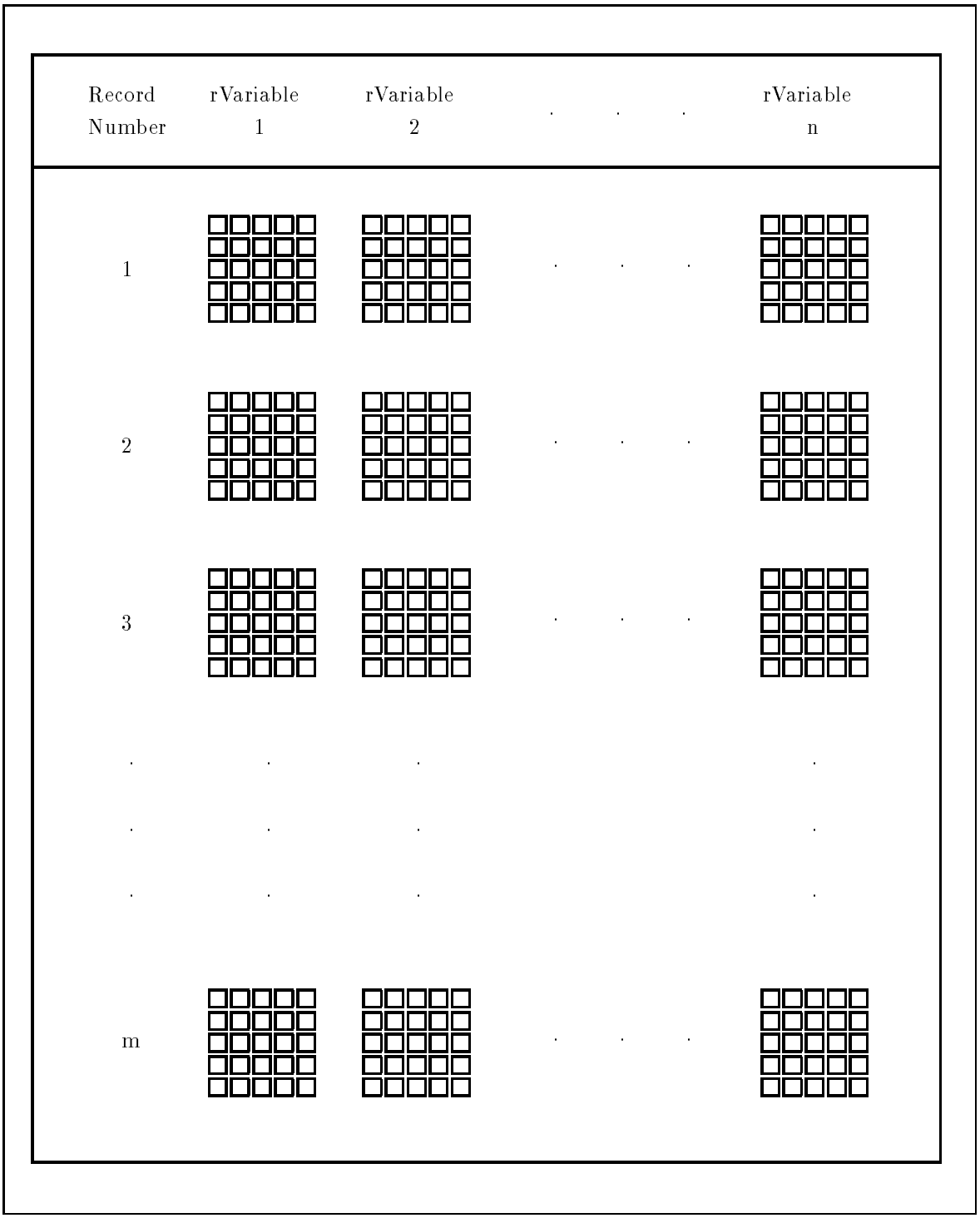

Figure 1.2: Conceptual View of a CDF, 2-Dimensional rVariables

CDF will be stored in the physical representation used by the machine on which the CDF library is running (VAX, Sun, SGi Personal Iris and Power Series, DECstation, IBM-RS/6000 series, HP 9000 series, NeXT,

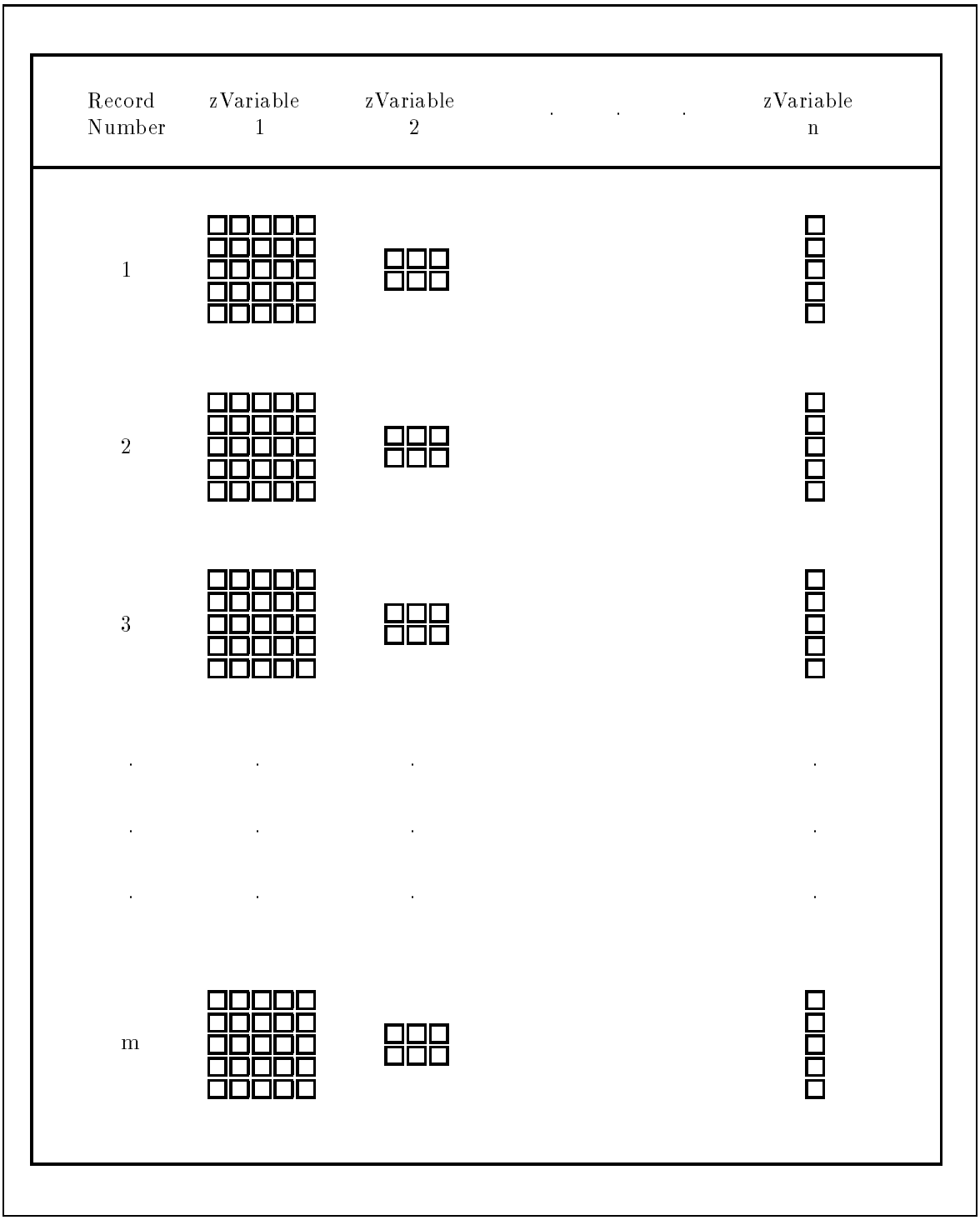

Figure 1.3: Conceptual View of a CDF, zVariables

or IBM PC). The second encoding scheme is network encoding. When using the network encoding scheme, the data values are transparently converted from the host representation to the XDR data representation

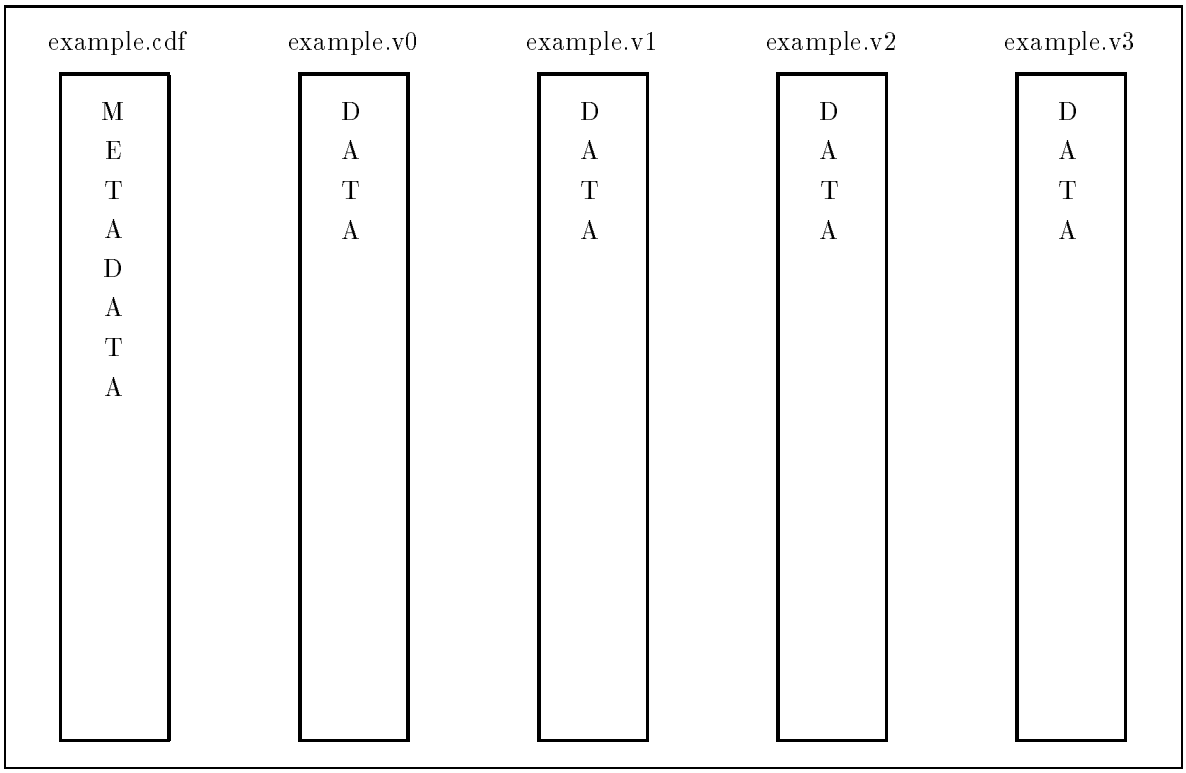

Figure 1.4: Multi-File Format

when writing the data and from the XDR representation to the host representation when reading the data. This gives the user the ability to transfer the CDFs to other platforms on which the CDF library runs and to read the data without having to be concerned with the data conversion.

1.4.3 Variable Data Access Options

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

1.4.4 CDF Library Interface Options

The CDF library supports two programming interfaces, the Standard Interface and the Internal Interface. The Standard Interface is similar to the interface provided with Version 1 of CDF (which is now known as the Obsolete Interface) with several additions for new features. The Internal Interface is provided to allow additional functionality to be added to the CDF library without the need to modify the Standard Interface. Those features not available using the Standard Interface are made available using the Internal Interface (eg. all access to zVariables). The Internal Interface makes CDF an easily extensible software package.

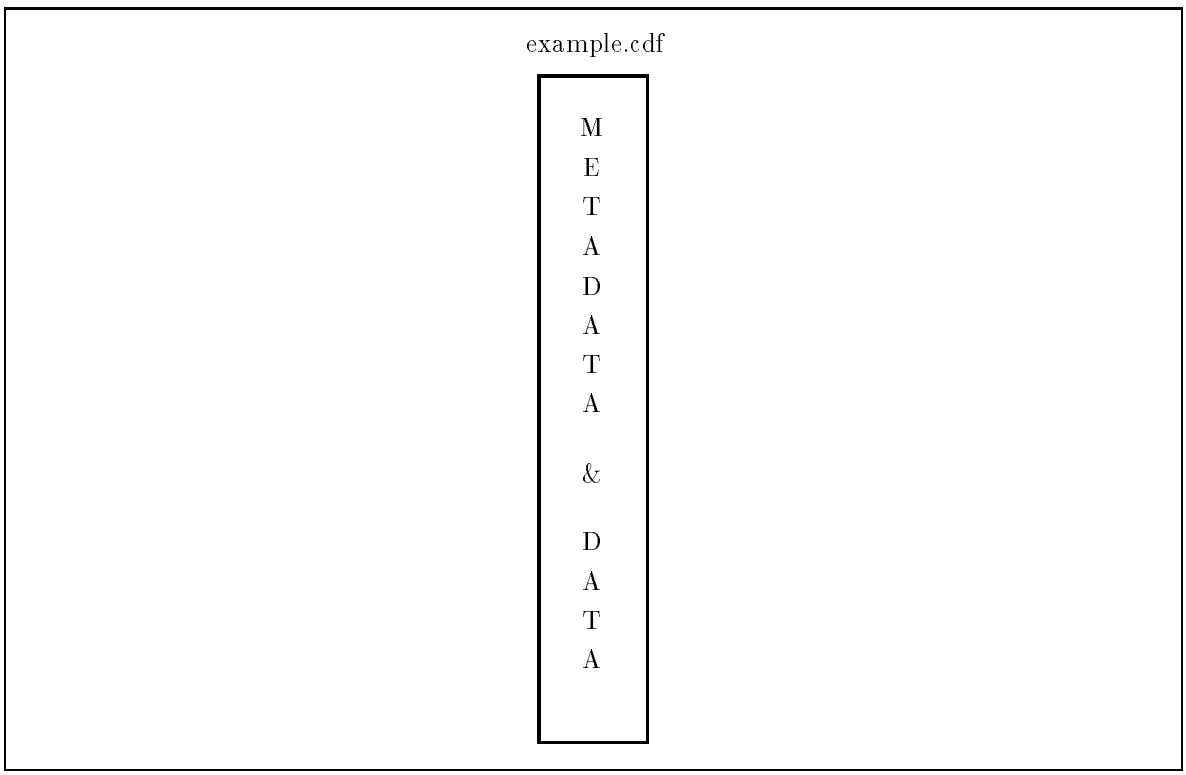

Figure 1.5: Single-File Format

The Standard and Internal interfaces are callable from both C and FORTRAN (the Obsolete Interface is callable only from FORTRAN).

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 which can be organized into arrays. Two types of variables are supported: rVariables and zVariables.

rVariables

rVariables all have the same dimensionality (number of dimensions and dimension sizes). An example of the type of data set which may be stored in a CDF's rVariables is shown in Table 1.1. Each record holds one value for each of the four variables: `Time`, `Longitude`, `Latitude`, and `Temperature`. CDF can store scalar data in a "flat" (0-dimensional) representation such as this, but storage in this manner may hide fundamental relationships among the data values. Consistent repetitions found in the data for this example suggest another way to organize the data set. Note that records one and five are observations of the same

point on earth at two 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 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 which defines a dimension generates the size of that dimension. For example, `Longitude` has two unique values so the dimension defined by `Longitude` has a size of two.

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

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

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

Although conceptually there is a 2-dimensional array structure for each rVariable in each record of the CDF, this would not be an efficient way to store the data. For instance, the time for each record need only be stored once as opposed to being stored four times as shown in each 2-dimensional array (Table 1.2). This problem is circumvented by specifying "variances". For each rVariable there are variances associated with the array

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

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

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.

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

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

zVariables

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

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

Attempting to store the images and the palette using only rVariables would result in one of two undesirable situations. If the CDF's rVariables are 2-dimensional with sizes `[1024,1024]` (to store the images) the palette would have to be stored in a 1024 by 1024 array which does not make sense logically and would waste disk space regardless of how the dimension variances are set. If the CDF's rVariables are 3-dimensional with sizes `[1024,1024,256]`, the images could be stored in an rVariable having dimension

variances `[TRUE,TRUE,false]` and the palette could be stored in an rVariable having dimension variances `[false,false,TRUE]`. This would not waste any disk space but is not the intuitive way to store the data — nothing in the data set is 3-dimensional.

Using a zVariable to store the palette would solve both problems. The CDF's rVariables would be defined as being 2-dimensional with sizes `[1024,1024]`. The images would be stored in an rVariable having dimension variances `[TRUE,TRUE]` but the palette would be stored in a zVariable defined to be 1-dimensional with a size of `[256]` (and dimension variance of `[TRUE]`). This would waste no disk space and logically makes sense. Note that the images could also have been stored in a zVariable defined as 2-dimensional with sizes `[1024,1024]` and dimension variances `[TRUE,TRUE]`.

1.6 Attributes

The second component of a CDF is the metadata. Metadata consists of user-supplied descriptive information about the CDF and the variables in the CDF by way of attributes and attribute entries. Attributes can be divided into two categories: attributes of global scope (gAttributes) and attributes of variable scope (vAttributes). gAttributes describe the CDF as a whole, while vAttributes describe some property of each variable (rVariables and zVariables) in the CDF. Any number of attributes may be stored in a single CDF. Section 2.9 describes standard CDF attributes used by the NSSDC. The term "attribute" is used when describing a property which applies to both gAttributes and vAttributes.

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

vAttributes further describe the individual variables and their values. Examples of vAttributes would include such things as a field name for the variable, the valid minimum and maximum, the units in which the variable data values are stored, the format in which the data values are to be displayed, a fill value for errant or missing data, and a description of the expected order of data values: increasing or decreasing (monotonicity). The entries of a vAttribute correspond to the variables in the CDF. Each rEntry corresponds to an rVariable and each zEntry corresponds to a zVariable. Sample vAttribute rEntries for the `Temperature` rVariable from the example above are shown in Table 1.5.

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

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...

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

Table 1.5: vAttribute rEntries for the `Temperature` rVariable

`CDFbrowse`	Allows the display of attribute and variable data.
`CDFlist`	Allows a sequential "flat" view of filtered variable data. `CDFlist` also allows the subsetting of a CDF (creating a new, smaller CDF).
`CDFwalk`	Allows direct and sequential access to related variable values.
`CDFconvert`	Converts CDFs created by previous releases of the CDF library to be updated to the current release. `CDFconvert` also allows the format, encoding, and majority of a CDF to be changed. In all cases, a new CDF is created (the original CDF is not modified).
`CDFskeleton`	Reads a specially formatted text file (called a skeleton table) and creates a CDF. The created 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 `CDFskeleton` to create a new CDF.
`CDFinquire`	Produces a report on the contents of a CDF.
`CDFstats`	Produces a report containing various statistics about the variables in a CDF.
`CDFcompare`	Reports the differences between two CDFs.
`CDFscope`	Corrects the attributes in a CDF that have an assumed scope.
`CDFdir`	Produces a directory listing of a CDF's files. For a multi-file CDF, the variable files are listed in ascending numerical order.

1.8 Library Interface Routines

The CDF software library allows a user to create CDFs and access data randomly or sequentially using the fundamental concepts described previously without knowledge of low-level file format details. Two interfaces to the CDF library are provided: the Standard Interface and the Internal Interface. Both interfaces are callable from C and FORTRAN.

1.8.1 Standard Interface

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

The general CDF functions are:

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

The rVariable functions are:

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

The attribute functions are:

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

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

1.8.2 Internal Interface

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

1.9 Examples

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

1.9.1 Creating a CDF, the hard way (but not that hard)

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

```
/******************************************************************************
*
*  NSSDC/CDF           Create an example CDF (without using a skeleton table).
*
*  Version 1.1, 10-Feb-92, Hughes STX
*
*  Modification history:
*
*   V1.0  13-Nov-91, J Love    Original version (for CDF V2.1 Primer chapter).
*   V1.1  10-Feb-92, J Love    Typo city.
*
******************************************************************************/

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

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

#include "cdf.h"

/******************************************************************************
* Function prototypes.
******************************************************************************/

void StatusHandler ();

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

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

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

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

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

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

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

long TimeValidMin = 0;          /* 'Time' valid minimum (0000). */
long TimeValidMax = 2359;       /* 'Time' valid maximum (2359). */
long TimeScaleMin = 100;        /* 'Time' scale minimum (0100). */
long TimeScaleMax = 200;        /* 'Time' scale maximum (0200). */
long TimeFillValue = 9999;      /* 'Time' fill value. */

float LonValidMin = -180.0;     /* 'Longitude' valid minimum. */
float LonValidMax = 180.0;      /* 'Longitude' valid maximum. */
float LonScaleMin = -165.0;     /* 'Longitude' scale minimum. */
float LonScaleMax = -150.0;     /* 'Longitude' scale maximum. */
float LonFillValue = -999.9;    /* 'Longitude' fill value. */

float LatValidMin = -90.0;      /* 'Latitude' valid minimum. */
float LatValidMax = 90.0;       /* 'Latitude' valid maximum. */
float LatScaleMin = 30.0;       /* 'Latitude' scale minimum. */
float LatScaleMax = 40.0;       /* 'Latitude' scale maximum. */
float LatFillValue = -999.9;    /* 'Latitude' fill value. */

float TmpValidMin = -40.0;      /* 'Temperature' valid minimum. */
float TmpValidMax = 50.0;       /* 'Temperature' valid maximum. */
float TmpScaleMin = 17.0;       /* 'Temperature' scale minimum. */
float TmpScaleMax = 24.0;       /* 'Temperature' scale maximum. */
float TmpFillValue = -999.9;    /* 'Temperature' fill value. */

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

status = CDFcreate ("example1", numDims, dimSizes, NETWORK_ENCODING, ROW_MAJOR,
                    &id);
```

```
if (status != CDF_OK) StatusHandler (status);

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

dimVarys[0] = NOVARY;
dimVarys[1] = NOVARY;
status = CDFvarCreate (id, "Time", CDF_INT4, 1, VARY, dimVarys,
                       &TimeVarNum);
if (status != CDF_OK) StatusHandler (status);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

status = CDFattrPut (id, CDFattrNum(id,"FIELDNAM"),
                     CDFvarNum(id,"Time"), CDF_CHAR, 20,
                     "Time of observation ");
if (status != CDF_OK) StatusHandler (status);

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

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

status = CDFattrPut (id, CDFattrNum(id,"SCALEMIN"),
                     CDFvarNum(id,"Time"), CDF_INT4, 1, &TimeScaleMin);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"SCALEMAX"),
                     CDFvarNum(id,"Time"), CDF_INT4, 1, &TimeScaleMax);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"UNITS"),
                     CDFvarNum(id,"Time"), CDF_CHAR, 20,
                     "Hour/Minute         ");
if (status != CDF_OK) StatusHandler (status);
```

```
status = CDFattrPut (id, CDFattrNum(id,"FORMAT"),
                     CDFvarNum(id,"Time"), CDF_CHAR, 8, "I4      ");
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"MONOTON"),
                     CDFvarNum(id,"Time"), CDF_CHAR, 8, "INCREASE");
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"FILLVAL"),
                     CDFvarNum(id,"Time"), CDF_INT4, 1, &TimeFillValue);
if (status != CDF_OK) StatusHandler (status);

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

status = CDFattrPut (id, CDFattrNum(id,"FIELDNAM"),
                     CDFvarNum(id,"Longitude"), CDF_CHAR, 20,
                     "Longitude           ");
if (status != CDF_OK) StatusHandler (status);

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

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

status = CDFattrPut (id, CDFattrNum(id,"SCALEMIN"),
                     CDFvarNum(id,"Longitude"), CDF_REAL4, 1, &LonScaleMin);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"SCALEMAX"),
                     CDFvarNum(id,"Longitude"), CDF_REAL4, 1, &LonScaleMax);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"UNITS"),
                     CDFvarNum(id,"Longitude"), CDF_CHAR, 20,
                     "Degrees east        ");
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"FORMAT"),
                     CDFvarNum(id,"Longitude"), CDF_CHAR, 8, "F6.1    ");
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"MONOTON"),
                     CDFvarNum(id,"Longitude"), CDF_CHAR, 8, "INCREASE");
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"FILLVAL"),
```

```
                    CDFvarNum(id,"Longitude"), CDF_REAL4, 1, &LonFillValue);
if (status != CDF_OK) StatusHandler (status);

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

status = CDFattrPut (id, CDFattrNum(id,"FIELDNAM"),
                    CDFvarNum(id,"Latitude"), CDF_CHAR, 20,
                    "Latitude            ");
if (status != CDF_OK) StatusHandler (status);

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

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

status = CDFattrPut (id, CDFattrNum(id,"SCALEMIN"),
                    CDFvarNum(id,"Latitude"), CDF_REAL4, 1, &LatScaleMin);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"SCALEMAX"),
                    CDFvarNum(id,"Latitude"), CDF_REAL4, 1, &LatScaleMax);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"UNITS"),
                    CDFvarNum(id,"Latitude"), CDF_CHAR, 20,
                    "Degrees north       ");
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"FORMAT"),
                    CDFvarNum(id,"Latitude"), CDF_CHAR, 8, "F6.1    ");
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"MONOTON"),
                    CDFvarNum(id,"Latitude"), CDF_CHAR, 8, "DECREASE");
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"FILLVAL"),
                    CDFvarNum(id,"Latitude"), CDF_REAL4, 1, &LatFillValue);
if (status != CDF_OK) StatusHandler (status);

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

status = CDFattrPut (id, CDFattrNum(id,"FIELDNAM"),
                    CDFvarNum(id,"Temperature"), CDF_CHAR, 20,
```

```
                    "Temperature value   ");
if (status != CDF_OK) StatusHandler (status);

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

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

status = CDFattrPut (id, CDFattrNum(id,"SCALEMIN"),
                    CDFvarNum(id,"Temperature"), CDF_REAL4, 1, &TmpScaleMin);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"SCALEMAX"),
                    CDFvarNum(id,"Temperature"), CDF_REAL4, 1, &TmpScaleMax);
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"UNITS"),
                    CDFvarNum(id,"Temperature"), CDF_CHAR, 20,
                    "Degrees celsius     ");
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"FORMAT"),
                    CDFvarNum(id,"Temperature"), CDF_CHAR, 8, "F4.1    ");
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"MONOTON"),
                    CDFvarNum(id,"Temperature"), CDF_CHAR, 8, "FALSE   ");
if (status != CDF_OK) StatusHandler (status);

status = CDFattrPut (id, CDFattrNum(id,"FILLVAL"),
                    CDFvarNum(id,"Temperature"), CDF_REAL4, 1, &TmpFillValue);
if (status != CDF_OK) StatusHandler (status);

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

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

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

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

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

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

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

fclose (fp);

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

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

return;
}

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

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

if (status < CDF_WARN) {
  printf ("An error has occurred, halting...\n");
  CDFerror (status, message);
  printf ("%s\n", message);
```

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

1.9.2 Creating a CDF, an easier way

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

```
! Skeleton table for the "example" CDF.
! Generated: Mon 10 Feb 1992 10:53:57

#header

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

! Variables G.Attributes V.Attributes Records  Dims  Sizes
! --------- ------------ ------------ -------  ----  -----
     4/0           1            9       1/z      2    2 2


#GLOBALattributes

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

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

```
#VARIABLEattributes

  "FIELDNAM"
  "VALIDMIN"
  "VALIDMAX"
  "SCALEMIN"
  "SCALEMAX"
  "UNITS"
  "FORMAT"
  "MONOTON"
  "FILLVAL"


#variables

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

  "Time"            CDF_INT4        1          T         F F

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

    "FIELDNAM"      CDF_CHAR     { "Time of observation " }
    "VALIDMIN"      CDF_INT4     { 0 }
    "VALIDMAX"      CDF_INT4     { 2359 }
    "SCALEMIN"      CDF_INT4     { 100 }
    "SCALEMAX"      CDF_INT4     { 200 }
    "UNITS"         CDF_CHAR     { "Hour/Minute         " }
    "FORMAT"        CDF_CHAR     { "I4      " }
    "MONOTON"       CDF_CHAR     { "INCREASE" }
    "FILLVAL"       CDF_INT4     { 9999 } .


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

  "Longitude"       CDF_REAL4       1          F         T F

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

    "FIELDNAM"      CDF_CHAR     { "Longitude           " }
    "VALIDMIN"      CDF_REAL4    { -180.0 }
    "VALIDMAX"      CDF_REAL4    { 180.0 }
```

```
    "SCALEMIN"      CDF_REAL4    { -165.0 }
    "SCALEMAX"      CDF_REAL4    { -150.0 }
    "UNITS"         CDF_CHAR     { "Degrees east        " }
    "FORMAT"        CDF_CHAR     { "F6.1    " }
    "MONOTON"       CDF_CHAR     { "INCREASE" }
    "FILLVAL"       CDF_REAL4    { -999.9 } .

  ! 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
  ! --------          ----       -----

    "FIELDNAM"      CDF_CHAR     { "Latitude            " }
    "VALIDMIN"      CDF_REAL4    { -90.0 }
    "VALIDMAX"      CDF_REAL4    { 90.0 }
    "SCALEMIN"      CDF_REAL4    { 30.0 }
    "SCALEMAX"      CDF_REAL4    { 40.0 }
    "UNITS"         CDF_CHAR     { "Degrees north       " }
    "FORMAT"        CDF_CHAR     { "F6.1    " }
    "MONOTON"       CDF_CHAR     { "DECREASE" }
    "FILLVAL"       CDF_REAL4    { -999.9 } .

  ! 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
  ! --------          ----       -----

    "FIELDNAM"      CDF_CHAR     { "Temperature value   " }
    "VALIDMIN"      CDF_REAL4    { -40.0 }
```

```
    "VALIDMAX"      CDF_REAL4    { 50.0 }
    "SCALEMIN"      CDF_REAL4    { 17.0 }
    "SCALEMAX"      CDF_REAL4    { 24.0 }
    "UNITS"         CDF_CHAR     { "Degrees celsius     " }
    "FORMAT"        CDF_CHAR     { "F4.1    " }
    "MONOTON"       CDF_CHAR     { "FALSE   " }
    "FILLVAL"       CDF_REAL4    { -999.9 } .

#end
```

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

```
        PROGRAM exampleSKT

C------------------------------------------------------------------------------
C
C  NSSDC/CDF                    Create an example CDF (using skeleton table).
C
C  Version 1.0, 13-Nov-91, Hughes STX
C
C  Modification history:
C
C   V1.0  13-Nov-91, J Love    Original version (for CDF V2.1 Primer chapter).
C
C------------------------------------------------------------------------------

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

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

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

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

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

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

        DATA lun/1/

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

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

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

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

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

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

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

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

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

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

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

        CLOSE (lun, ERR=99)

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

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

        STOP

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

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

        END



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

        SUBROUTINE StatusHandler (status)
        INTEGER*4 status

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

        CHARACTER message*(CDF_ERRTEXT_LEN)

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

```
  END IF
END IF

RETURN
END
```

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

Chapter 2

General CDF Concepts

2.1 CDF Interfaces

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

2.1.1 Standard Interface

The Standard interface[1] provides a standard set of routines with which to access a CDF. Not all CDF features are available with the Standard Interface. The Internal Interface must be used to perform operations not available with the Standard Interface routines (eg. access to zVariables). The Standard Interface is callable from both C and FORTRAN applications. Table 2.1 lists the routines available when using the Standard Interface. Each routine is described in detail in the corresponding programming reference chapter.

2.1.2 Internal Interface

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

2.1.3 Obsolete Interface, VMS Systems Only

FORTRAN applications written for CDF Version 1 on VMS systems may still be used, without modification, via the Obsolete Interface. While we don't encourage the use of the Obsolete Interface, it is provided for use while existing applications are being updated for the new Standard Interface routines. To use the Obsolete

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

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

Table 2.1: Standard Interface Routines

Callable from C	Callable from FORTRAN	Purpose
CDFlib ()	CDF_lib ()	Performs all available operations.

Table 2.2: Internal Interface Routines

Interface, simply relink the application with the Obsolete CDF library (LIBCDF_OBSOLETE.OLB). An example of this would be as follows...

```
$ LINK file1, file2, ...fileN, CDF$LIB:LIBCDF_OBSOLETE/LIBRARY
```

where file1, file2, ...fileN are the application's object modules. CDF$LIB is defined by executing the definitions file, DEFINITIONS.COM, provided with the CDF distribution.

There are several conditions which apply to the use of the Obsolete Interface.

1. The functions (and their associated arguments) available for use are exactly the same as in CDF Version 1 (refer to the CDF Version 1 Implementer's Guide for details). New CDF features are not available.

2. CDFs created will be Version 2 CDFs. To create a Version 1 CDF you must use the CDF Version 1 library.

3. Version 2 CDFs may be accessed except in those cases where the Obsolete Interface does not support a new Version 2 feature. In those cases an error code indicating the incompatibility will be returned. These incompatibilities deal mainly with attribute entries (in Version 1 the data type was specified for the attribute — in Version 2 the data type is specified per entry).
4. Version 1 CDFs may be accessed but the only modification allowed is the updating of existing variable values. No other modifications are allowed to Version 1 CDFs (with the Version 2 CDF library).

2.2 Access to CDFs

Both Version 1 and Version 2 CDFs may be accessed with the Version 2 CDF library. Restrictions are described in the following sections.

2.2.1 Version 2 CDFs

Any operation on a Version 2 CDF may be performed with the Standard and Internal interfaces.

On VMS systems, when using the Obsolete Interface, all operations are allowed except when an incompatibility exists between the contents of the Version 2 CDF and the capabilities of the Obsolete Interface (see Section 2.1.3).

2.2.2 Version 1 CDFs

All types of read access are allowed on Version 1 CDFs using either the Standard or Internal Interface. The only write access allowed is the modification of existing variable values.

2.3 CDF 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 a CDF.

A CDF's format is normally specified when the CDF is created. In a user application, the Internal Interface must be used to specify a single-file CDF (the default using the Standard Interface is multi-file). The format of an existing CDF can be changed (with the Internal Interface) if no variables have been created in the CDF. If the `CDFskeleton` toolkit program is used to create a CDF, the format is specified in the skeleton table (see Section 3.9).

2.3.1 Multi-file CDFs

A multi-file CDF (`MULTI_FILE`) consists of one file (with extension `.cdf`) containing control information and attribute entry data and a separate file for each variable defined in the CDF (with extensions `.v0`,`.v1`,... for

rVariables and `.z0`,`.z1`,... for zVariables). Each variable file contains the data values for the corresponding variable (the control information for each variable is stored in the `.cdf` file).

2.3.2 Single-file CDFs

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

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

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

2.4 CDF Encoding

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

A CDF's encoding is specified when the CDF is created. The encoding of an existing CDF may be changed with the Internal Interface if no variable values or attribute entries have been written (variables and attributes may exist however). If the `CDFskeleton` toolkit program is used to create a CDF the encoding is specified in the skeleton table (see Section 3.9).

The encoding specified when creating a CDF may be either the host encoding of the computer begin used or network encoding. A CDF can not be created with the host encoding of a different computer. A CDF with any encoding, however, may be read on any computer.[2]

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

2.4.1 Host Encoding

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

`VAX_ENCODING`	VAX and microVAX computers.
`SUN_ENCODING`	Sun computers.
`MIPSEB_ENCODING`	Silicon Graphics Iris and Power Series computers.
`MIPSEL_ENCODING`	DECstation computers.
`IBMRS_ENCODING`	IBM RS-6000 series computers.
`HP_ENCODING`	HP 9000 series computers.
`IBMPC_ENCODING`	IBM PC personal computers.
`NeXT_ENCODING`	NeXT computers.

2.4.2 Network Encoding

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

2.4.3 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 because no encoding or decoding has to be performed by the CDF library. A CDF which 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 (on a slow machine if that is all that is available) and then transfer it to a faster machine for processing or conversion to network encoding. Obviously there are trade-offs as to which encoding should be used in any one particular case. Keep in mind that a CDF can always be converted to the host encoding of the machine being used (with `CDFconvert`) before being accessed.

2.5 CDF 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.

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.

2.5.2 Floating Point Data Types

CDF_REAL4	4-byte, single precision floating point.
CDF_FLOAT	4-byte, single precision floating point.
CDF_REAL8	8-byte, double precision floating point.
CDF_DOUBLE	8-byte, double precision floating point.

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.

2.5.4 EPOCH Data Type

CDF_EPOCH	8-byte, double precision floating point.

The EPOCH data type is used to store time values referenced from a particular epoch. For NSSDC applications that epoch is `01-Jan-0000 00:00:00.000`.[3] `CDF_EPOCH` values are the number of milliseconds since the epoch. CDF toolkit programs will display and expect `CDF_EPOCH` values in the format...

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

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

where

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

Functions exist which compute and breakdown CDF_EPOCH values. These functions are descibed in Section 4.9 for C applications and in Section 5.10 for FORTRAN applications.

2.5.5 Equivalent Data Types

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

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

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

2.6 CDF Limits

Limits within the CDF library are defined in the appropriate include files: cdf.h for C applications and cdf.inc for FORTRAN applications. The following limits exist.[4]

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

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

[4] Previous releases limited the number of variables a CDF could contain. That limit has been eliminated.

2.7 Variables

CDF variables are the mechanism for storing data (attributes are used to store metadata). A new variable may be created in a CDF at any time. Two types of variables are supported: rVariables and zVariables.[5] The main difference is that all rVariables in a CDF have the same dimensionality whereas zVariables can have differing dimensionalities. In the following sections, differences between the two types will be noted where appropriate.

2.7.1 Access

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

2.7.2 Naming

Each variable in a CDF has a unique name. This applies to rVariables and zVariables together (ie. an rVariable can not have the same name as a zVariable). Variable names are case sensitive regardless of the operating system being used and may consist of any 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. To allow CDF Version 2 applications to read Version 1 CDFs without having to be concerned with the trailing blanks, the trailing blanks are ignored when comparing variable names. The trailing blanks are returned as part of the name, however, when a variable is inquired by an application program.

2.7.3 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.

2.7.4 Dimensionality

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

Each rVariable in a CDF has the same dimensionality. An array of values exists for each rVariable at each record in a CDF. The values may not be physically stored but may be virtual (see Sections 2.7.5, 2.7.7, and 2.7.8).

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

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

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

2.7.5 Records

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

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

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

Record variance is described in Section 2.7.7.

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 through the Internal Interface.

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

Note that a variable's records do not have to be written sequentially starting at the first record. The records may be written in any order. For a variable with `VARY` record variance, if a new record more than one record

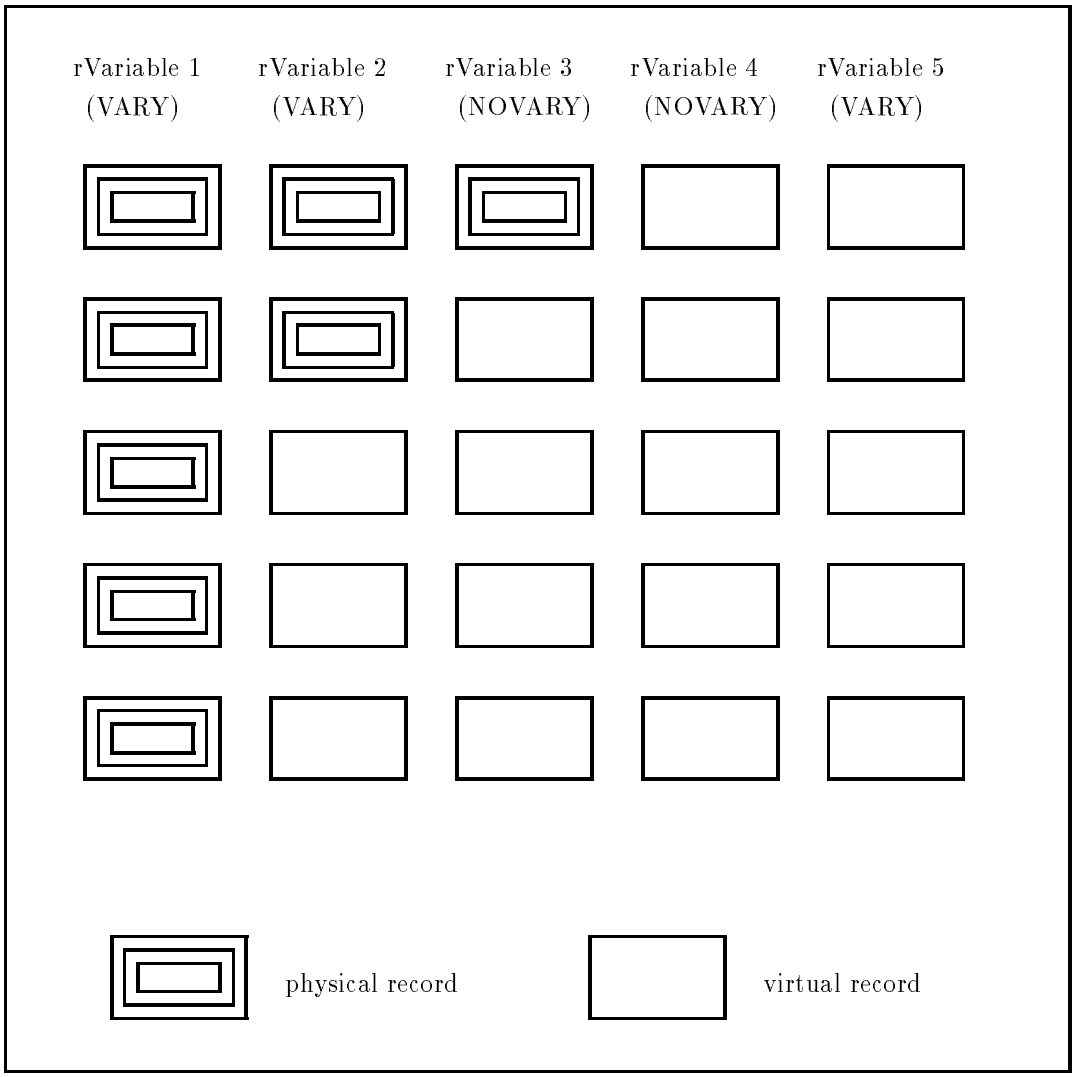

Figure 2.1: Physical vs. Virtual Records

beyond the current maximum record for a variable is written, the intervening records will be physically written and contain the pad value for that variable. Also, when one or more values are written to a new physical record, the entire record is physically written with the pad value for the variable being used for the unspecified values. Variable pad values are described in Section 2.7.12.

Numbering

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

Initial Records

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

Specifying a number of initial records for a variable would usually only be done for a CDF with the single-file format. Because the records would be allocated contiguously within the CDF file, the indexing scheme would require fewer entries making the access to that variable more efficient.

The number of initial records specified would in most cases be the number of records planned for a variable. Note that additional records may be added to a variable at any time. After the initial records are written with the pad value, the actual data values would be written by the application.

Extend Records

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

For single-file CDFs, however, the new records are allocated at the end of the `.cdf` file. The new records may not be located contiguously with the other records for the variable. This results in a new entry being created in the indexing scheme used to locate variable records. The indexing scheme is more efficient if fewer entries are needed (resulting from larger blocks of contiguous records being allocated). The CDF library attempts to optimize the indexing scheme by allocating more records than are actually needed with the assumption that they will eventually be used. A minimum number of records to allocate is maintained for each variable.

Because an application knows how records will be added to a variable more so than the CDF library, the Internal Interface may be used to explicitly specify the number of records to extend a variable when necessary. This would usually be done only for CDFs with the single-file format.

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

2.7.6 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 are true.

1. Values have not yet been written to the variable (including an explicitly written pad value, see Section 2.7.12).

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 string data types this is the number of characters in each string (a variable value consists of the entire character string). The character string can be thought of as an array of characters. For non-character string data types, this must always be one (1).[6] An array of elements per variable value is not allowed for non-character string data types.

2.7.7 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 (ie. 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 2.7.5 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).

2.7.8 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...

[6] A future release of CDF may relax this restriction. Note that for attribute entries, more than one element is allowed for any data type.

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

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

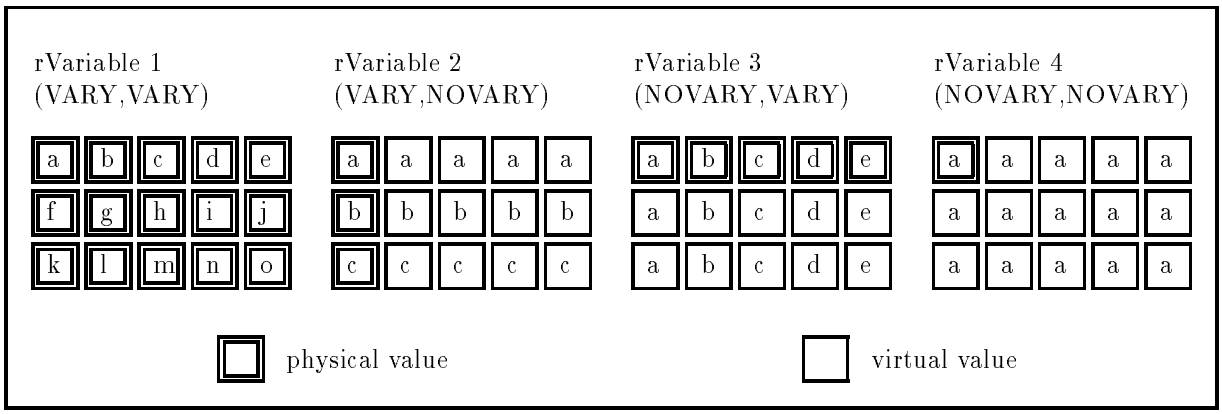

Figure 2.2: Physical vs. Virtual Dimensions

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).

2.7.9 Majority

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

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

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

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

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

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

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

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

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

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

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

3. When single value reads/writes are performed, the majority should have little or no effect. The CDF library uses a caching scheme to optimize the random access to variable values. If all of the values of a record are to be read/written, there may be a slight increase in performance if the values are accessed with (rather than against) the majority.[7] For example, if the majority is `ROW_MAJOR`, increment the last index the fastest.

A CDF's majority is specified when the CDF is created. 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).

2.7.10 Hyper Access

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

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

[7] Hyper reads/writes being the most efficient of course.

`dimension intervals` For each dimension, the interval between values being accessed. An interval of three (3) would indicate that every third value is to be accessed.

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

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

Hyper access is sensitive to the record and dimension variances of a variable. For instance, if a variable has a record variance of `NOVARY` 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 which 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.

2.7.11 Sequential Access

Sequential access provides a way to sequentially read/write the values physically stored for a variable. To use sequential access, a starting point must be selected by specifying a record number and dimension indices (which selects the current position). A sequential read will return the value at the current position and then automatically increment the current position to the next value. Likewise, a sequential write will store a value at the current position and then increment the current position 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). Sequential reading will increment to the beginning of the next physical record if necessary. Sequential writing can be used to extend the physical records for a variable (as well as to overwrite existing values).

Example (FORTRAN application)

Assume a 2-dimensional array with sizes `[2,3]`, column majority, a record variance of `VARY`, dimension variances of `[VARY,VARY]`, nine (9) physical records written, and that the current sequential position 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 seqential position caused physical value `7(1,2)` to actually be selected (because the first dimension variance is `NOVARY`).

2.7.12 Variable Pad Values

Variable pad[8] values are used in several situations.

1. When the first value is written to a new record, the other values in that record will contain the pad value. This also applies to hyper writes if less than the entire record is written (the unwritten values will contain the pad value).

2. When a new record is written which is more than one record beyond the last record already written, the intervening records will contain pad values. This will only happen if the record variance is `VARY` (the variable values vary from record to record).

3. When reading a record beyond the last record for a variable, pad values will be returned.

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 values for each data type are as follows...[9]

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

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

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

2.8 Attributes

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

2.8.1 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 any 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. To allow CDF Version 2 applications to read Version 1 CDFs without having to be concerned with the trailing blanks, the trailing blanks are ignored when comparing attribute names. The trailing blanks are returned as part of the name, however, when an attribute is inquired by an application program.

2.8.2 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.

2.8.3 Attribute Scopes

Attribute scopes declare the intended purpose of an attribute. gAttributes (global scope) describe some aspect of the entire CDF. vAttributes (variable scope) describe some property of each variable (one entry per rVariable and one entry per zVariable will normally exist for a vAttribute).

The CDF library places no restrictions on the operations which may be performed on an attribute of a certain scope. An attribute's scope only affects the interpretation of its entries by CDF toolkit programs and user applications (eg. entries of a vAttribute should correspond to variables).

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. However, when the Version 2 CDF library reads a Version 1 CDF it cannot always determine the scope of an attribute. When this occurs the Version 2 CDF library makes a best guess and returns `GLOBAL_SCOPE_ASSUMED` or `VARIABLE_SCOPE_ASSUMED` as the scope of the attribute. When a user application reading a Version 1 CDF encounters an assumed scope, it should take the appropriate action to determine the correct scope. However, when the `CDFconvert` toolkit program (see Section 3.4) encounters an assumed scope it cannot always determine the correct scope. In those cases it will create the attribute in the new CDF with the assumed scope. The assumed scopes may be corrected by using the `CDFscope` toolkit program (see Section 3.5). The assumed scopes may also be corrected by a user application via the CDF library.

2.8.4 Attribute Entries

Attribute entries are used to actually store metadata values. 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".

Access

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 meaing only to the application.

The entry numbers used need not be contiguous (as are variable and attribute numbers). An application may chose 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 Type

Each entry for an attribute has a specified data type. The supported data types are described in Section 2.5. The entries for an attribute may have any combination of data types.

Number of Elements

In addition to a data type, each attribute entry has a number of elements. This refers to the number of elements of the specified data type.

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

For non-character string data types this is the number of elements in 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).

2.9 NSSDC Standard

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

2.9.1 Standard Attributes

There are several attributes which are required to exist in a CDF if it is to conform to the NSSDC standard. These required attributes are shown in Table 2.3. Several optional attributes are also shown.

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

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

Table 2.3: NSSDC Standard CDF Attributes

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

TITLE A title for the CDF data set. Only one gEntry should be written. (gEntry number 0 for C applications, gEntry number 1 for FORTRAN applications).

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

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

VALIDMAX The maximum valid value for an rVariable. Values above this maximum are treated as invalid or fill data. One rEntry per rVariable should be written.

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

2.9.2 Standard Variables

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

The optional rVariable EPOCH is used by many NSSDC applications to store date/time values. Section 2.5.4 defines an EPOCH value. The record variance should be VARY, each dimension variance should be NOVARY, the data type should be CDF_EPOCH, and the number of elements should be one (1). If a CDF contains an EPOCH variable, the following variable names should not be used: TIME, YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, MSEC, IYEAR, IMONTH, IDAY, IHOUR, IMINUTE, ISECOND, and IMSEC.

Chapter 3

CDF Toolkit Reference

3.1 Introduction

The CDF toolkit is a set of utility programs which allow the creation, access, and modification of CDFs. 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 <>s.

2. All qualifiers are optional.

3. Qualifiers can be abbreviated as long as the abbreviation is unique.

4. Optional parts of a command are shown in []s.

5. 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.

   NOTE: On UNIX systems, if a CDF specification is to contain a wildcard character, the entire specification must be enclosed in single quote marks (eg. `'/disk3/sst*'`).

6. On VMS systems, qualifiers begin with a '/'. On UNIX and MS-DOS systems, qualifiers begin with a '-'. When describing qualifiers, both notations may be shown (eg. `/[NO]LOG`, `-[no]log`). Use only the notation appropriate for the operating system begin used.

If a program allows message logging to be disabled (using `/NOLOG`,`-nolog`), INFO and WARNING messages will not be displayed. ERROR messages are always displayed.

3.2 CDFdir

3.2.1 Introduction

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

3.2.2 Executing the CDFdir program

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

Usage:

VMS:

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

UNIX:

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

MS-DOS:

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

Parameter(s):

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

Example(s):

VMS:

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

UNIX:

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

MS-DOS:

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

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

3.2.3 Output from the CDFdir program

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

3.3 CDFcompare

3.3.1 Introduction

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

3.3.2 Executing the CDFcompare program

The command line syntax for `CDFcompare` is as follows.

Usage:

VMS:

```
$ CDFCOMPARE [/[NO]LOG] [/[NO]ATTR] [/[NO]VAR] [/[NO]NUMBER] [/[NO]PERCENT]
             <cdf-spec-1> <cdf-spec-2>
```

UNIX:

```
% cdfcompare [-[no]log] [-[no]attr] [-[no]var] [-[no]number] [-[no]percent]
             <cdf-spec-1> <cdf-spec-2>
```

MS-DOS:

```
> cdfcompare [-[no]log] [-[no]attr] [-[no]var] [-[no]number] [-[no]percent]
             <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 pathname specifying a single CDF or a directory/wildcard path specifying more than one CDF. Wildcards are allowed in the CDF name but not in the directory path.

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

Qualifier(s):

`/[NO]LOG, -[no]log` Specifies whether or not messages about the progress of each comparison are displayed. Message logging is disabled by default.

`/[NO]PERCENT, -[no]percent`

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. This qualifier is disabled by default.

`/[NO]ATTR, -[no]attr` Specifies whether or not attributes (and their entries) are to be compared. Attributes are compared by default.

`/[NO]VAR, -[no]var` Specifies whether or not variables are to be compared. Variables are compared by default. Note that an rVariable will never be compared with a zVariable.

`/[NO]NUMBER, -[no]number`

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

Example(s):

VMS:

```
$ CDFCOMPARE GISS_WETL GISS_WETL1
$ CDFCOMPARE/LOG/NOATTR/NUMBER GISS_WETL CDF$SMPL:GISS_WETL
$ CDFCOMPARE/NOVAR NCDS$SMPL: NCDS$DATA:
```

UNIX:

```
% cdfcompare giss_wetl giss_wetl1
% cdfcompare -log -noattr -number giss_wetl ../../giss_wetlx
% cdfcompare -novar /user5/CDFs /user6/CDFs
```

MS-DOS:

```
> cdfcompare gisswetl c:\gisswetl
> cdfcompare -log -noattr -number gisswetl ..\..\giswetlx
> cdfcompare -novar a:\cdfs c:\cdf\cdfs
```

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

3.3.3 Output from the CDFcompare program

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

3.4 CDFconvert

3.4.1 Introduction

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

1. A CDF created with an earlier release of the CDF library may be converted to the current library release.
2. The format of the CDF may be changed (see Section 2.3).
3. The data encoding of the CDF may be changed (see Section 2.4).
4. The variable majority of the CDF may be changed (see Section 2.7.9).

3.4.2 Executing the CDFconvert program

The command line syntax for `CDFconvert` is as follows.

Usage:

VMS:

```
$ CDFCONVERT [/SKELETON=<cdf-path>] [/[NO]LOG] [/[NO]PERCENT] [/[NO]DELETE]
             <src-cdf-spec> <dst-cdf-spec>
             [/SINGLE or /MULTI] [/ROW or /COLUMN] [/HOST or /NETWORK]
```

UNIX:

```
% cdfconvert [-skeleton <cdf-path>] [-[no]log] [-[no]percent] [-[no]delete]
             <src-cdf-spec> <dst-cdf-spec>
             [-single or -multi] [-row or -column] [-host or -network]
```

MS-DOS:

```
> cdfconvert [-skeleton <cdf-path>] [-[no]log] [-[no]percent] [-[no]delete]
             <src-cdf-spec> <dst-cdf-spec>
             [-single or -multi] [-row or -column] [-host or -network]
```

Parameter(s):

`<src-cdf-spec>` The source CDF(s). This can be either a single CDF pathname or a directory/wildcard path in which case all CDFs which 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 pathname only if `<src-cdf-spec>` is a single CDF pathname. If the directory paths are the same, then a different CDF name must be specified. If `<src-cdf-spec>` 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=<cdf-path>, -skeleton <cdf-path>`

`<cdf-path>` is the pathname of a skeleton CDF to be used during the conversions (do not enter an extension). The skeleton CDF is used in the following cases.

1. If an assumed attribute scope is found in a source CDF, then the scope of that attribute in the skeleton CDF (if the attribute exists) will be used in the destination CDF.
2. If a format for the destination CDF was not specified, then the format of the skeleton CDF will be used.
3. If a majority for the destination CDF was not specified, then the majority of the skeleton CDF will be used.
4. If an encoding for the destination CDF was not specified, then the encoding of the skeleton CDF will be used.

Specifying a skeleton CDF is optional.

`/[NO]LOG, -[no]log` Specifies whether or not messages about the progress of each conversion are displayed. Message logging is disabled by default.

`/[NO]PERCENT`, `-[no]percent`

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. This qualifier is disabled by default.

`/[NO]DELETE`, `-[no]delete`

Specifies whether or not a destination CDF is deleted if it already exists. This qualifier is disabled by default.

`/SINGLE`, `-single` or `/MULTI`, `-multi`

The format of the destination CDF(s). This overrides the format of the skeleton CDF (if specified). If neither this qualifier nor a skeleton CDF is specified, then the format of the destination CDF(s) will be the same as that of the source CDF(s).

`/ROW`, `-row` or `/COLUMN`, `-column`

The majority of the destination CDF(s). This overrides the majority of the skeleton CDF (if specified). If neither this qualifier nor a skeleton CDF is specified, then the majority of the destination CDF(s) will be the same as that of the source CDF(s).

`/HOST`, `-host` or `/NETWORK`, `-network`

The encoding of the destination CDF(s). This overrides the encoding of the skeleton CDF (if specified). If neither this qualifier nor a skeleton CDF is specified, then the encoding of the destination CDF(s) will be the same as that of the source CDF(s).

Example(s):

VMS:

```
$ CDFCONVERT CDF$SMPL:TEMPLATE0 TEMPLATE0X
$ CDFCONVERT/LOG CDF$SMPL: USER_DISK:[USER.CDF]
$ CDFCONVERT CAC_SST_BLENDED CAC_SST_BLENDEDX/SINGLE/NETWORK
$ CDFCONVERT/SKELETON=CDF$SMPL:TEMPLATE3 CAC_SST_BLENDED* [USER.CDF]
```

UNIX:

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

MS-DOS:

```
> cdfconvert ..\samples\tplate0 tplate0x
> cdfconvert -log ..\samples c:\dir4\user\cdf
```

```
> cdfconvert cac_sst cac_sst1 -single -network
> cdfconvert -skeleton tplate3 a:\cdf\cac_sst dir5\cdf
```

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 CDFscope

3.5.1 Introduction

The `CDFscope` program is used to correct assumed attribute scopes found in a CDF. Section 2.8.3 explains attribute scopes and how assumed scopes can arise.

NOTE: This program is not a permanent one — it may be deleted from a future release of the CDF distribution. Version 1 CDFs should be converted to Version 2 CDFs as soon as possible. Any assumed attribute scopes should then be corrected using this program.

3.5.2 Executing the CDFscope program

The command line syntax for `CDFscope` is as follows.

Usage:

VMS:

```
$ CDFSCOPE [/[NO]CONFIRM] [/[NO]LOG] <cdf-path>
```

UNIX:

```
% cdfscope -[no]confirm -[no]log <cdf-path>
```

MS-DOS:

```
> cdfscope -[no]confirm -[no]log <cdf-path>
```

Parameter(s):

`<cdf-path>`	The pathname of the CDF to be corrected (do not specify an extension).

Qualifier(s):

`/[NO]CONFIRM, -[no]confirm`

Specifies whether or not `CDFscope` prompts for confirmation before correcting a scope. The valid responses to the confirmation prompts are as follows.

`Y/y`	Yes, correct the scope (assumed to definite).
`N/n`	No, don't correct the scope.
`G/g`	Correct the scope to global.
`V/v`	Correct the scope to variable.

Confirming is disabled by default (all assumed scopes are corrected to the corresponding definite scopes).

`/[NO]LOG, -[no]log` Specifies whether or not message logging is enabled. Message logging is disabled by default.

Example(s):

VMS:

```
$ CDFSCOPE DUA5:[CDF.TEST]TOGA_ECMWF
$ CDFSCOPE/CONFIRM/NOLOG TOGA_ECMWF
```

UNIX:

```
% cdfscope /disk5/cdf/toga_ecmwf
% cdfscope -confirm -nolog toga_ecmwf
```

MS-DOS:

```
> cdfscope a:\dir5\cdf\toga
> cdfscope -confirm -nolog toga
```

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

3.5.3 Output from the CDFscope program

If message logging is enabled, the number of scopes corrected will be 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 analysed. For each variable, it determines the actual minimum and maximum values (in all of the variable records), the minimum and maximum values within the valid range (if the `VALIDMIN` and `VALIDMAX` vAttributes and corresponding entries are present in the CDF), and the monotonicity. An option exists to allow fill values (specified by the `FILLVAL` vAttribute) to be filtered out when collecting statistics.

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

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

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

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

3.6.2 Executing the CDFstats program

The command line syntax for `CDFstats` is as follows.

Usage:

VMS:

```
$ CDFSTATS [/[NO]RANGE] [/[NO]FILL] [/OUTPUT=<file-path>] [/[NO]FORMAT]
           [/[NO]PAGE] [/[NO]UPDATE_VALIDS] [/[NO]UPDATE_SCALES] <cdf-path>
```

UNIX:

```
% cdfstats [-[no]range] [-[no]fill] [-output <file-name>] [-[no]format]
           [-[no]page] [-[no]update_valids] [-[no]update_scales] <cdf-path>
```

MS-DOS:

```
> cdfstats [-[no]range] [-[no]fill] [-output <file-name>] [-[no]format]
```

```
[-[no]page] [-[no]update_valids] [-[no]update_scales] <cdf-path>
```

Parameter(s):

`<cdf-path>`	The pathname of the CDF to check (do not specify an extension).

Qualifier(s):

`/[NO]RANGE, -[no]range`	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. Range checking is enabled by default.
`/[NO]FILL, -[no]fill`	Specifies whether or not fill values are to be filtered out when collecting statistics. To filter out fill values, the CDF must contain the `FILLVAL` attribute. A variable must also have an entry for the `FILLVAL` attribute in order for fill value filtering to be performed on that variable. Fill value filtering is enabled by default.
`/OUTPUT=<file-path>, -output <file-path>`	Redirects the output to a file. The file created will be named `<file-path>` (if `<file-path>` does not have an extension, `.sts` is appended automatically). If this qualifier is not specified, the output is displayed at the terminal.
`/[NO]FORMAT, -[no]format`	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). The `FORMAT` attribute is used by default.
`/[NO]PAGE, -[no]page`	Specifies whether or not the output is displayed a page (screen) at a time. If the output is directed to a file, paging is not allowed. Paging is disabled by default.
`/[NO]UPDATE_VALIDS, -[no]update_valids`	Specifies whether or not the `VALIDMIN` and `VALIDMAX` attribute entry values are updated for each variable based on the actual mininum and maximum values found (with fill values being ignored if `/FILL,-fill` was specified). If the `VALIDMIN` and `VALIDMAX` attributes do not exist, they are created. The `VALIDMIN` and `VALIDMAX` attribute entries are not updated by default.
`/[NO]UPDATE_SCALES, -[no]update_scales`	Specifies whether or not the `SCALEMIN` and `SCALEMAX` attribute entry values are updated for each variable based on the actual mininum and maximum values found (with fill values being ignored if `/FILL,-fill` was specified). If the `SCALEMIN` and `SCALEMAX` attributes do not exist, they are created. The `SCALEMIN` and `SCALEMAX` attribute entries are not updated by default.

Example(s):

VMS:

```
$ CDFSTATS GISS_SOIL
$ CDFSTATS/NOFILL/OUTPUT=TEMPLATE3/NORANGE CDF$SMPL:TEMPLATE3
```

UNIX:

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

MS-DOS:

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

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

3.6.3 Output from the CDFstats program

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

For each variable (rVariables and zVariables),

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

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

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

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

        monotonic: <monotonicity>
```

If range checking and/or fill value filtering are 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 (eg. 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 varys along more than one "dimension" (if records are considered another "dimension").

3.7 CDFinquire

3.7.1 Introduction

The `CDFinquire` program produces a report on a CDF's contents. It may be used to inquire the following.

1. The version, release, and increment numbers of the creating CDF library.
2. The copyright notice of the creating CDF library.
3. Header information — the CDF's data encoding, variable majority, number of records, number of gAttributes and vAttributes, number of variables (rVariables and zVariables), and number of dimensions and dimension sizes for the rVariables.
4. Global scope attribute definitions — the name, number, and number of entries for each gAttribute.
5. Global scope attribute entries — for each gAttribute entry, the data type, number of elements, and value(s).
6. Variable scope attribute definitions — the name, number, and number of entries for each vAttribute.
7. Variable definitions — the name, number, data type, number of elements, record variance, and dimension variances for each variable. For zVariables, the number of dimensions and dimension sizes are also inquired.
8. Variable scope attribute entries — for each vAttribute entry, the data type, number of elements, and value(s).

3.7.2 Executing the CDFinquire program

The command line syntax for `CDFinquire` is as follows.

Usage:

VMS:

```
$ CDFINQUIRE [/DISPLAY=<mask>] [/OUTPUT=<file-path>] [/[NO]PAGE] [/ID]
             <cdf-path>
```

UNIX:

```
% cdfinquire [-display <mask>] [-output <file-path>] [-[no]page] [-id]
            <cdf-path>
```

MS-DOS:

```
> cdfinquire [-display <mask>] [-output <file-path>] [-[no]page] [-id]
            <cdf-path>
```

Parameter(s):

`<cdf-path>` The pathname of the CDF to inquire (do not specify an extension).

Qualifier(s):

`/DISPLAY=<mask>`, `-display <mask>`

The type of inquiry to be done. The `<mask>` consists of one or more of the following.

`i`	CDF copyright and version/release/increment.
`h`	Header information.
`g`	gAttribute definitions.
`g+`	gAttribute definitions & entries.
`a`	vAttribute definitions.
`v`	rVariable definitions.
`v+`	rVariable definitions & vAttribute rEntries.
`z`	zVariable definitions.
`z+`	zVariable definitions & vAttribute zEntries.

Do not specify both `g` and `g+`, both `v` and `v+`, or both `z` and `z+`. If this qualifier is not specified, the default `<mask>` is `ihg+av+z+`.

`/OUTPUT=<file-path>`, `-output <file-path>`

Redirects the output to a file. The file created will be named `<file-path>` (if `<file-path>` does not have an extension, `.inq` is appended automatically). If this qualifier is not specified, the output is displayed at the terminal.

`/[NO]PAGE`, `-[no]page` Specifies whether or not the output is displayed a page (screen) at a time. If the output is directed to a file, paging is not allowed. Paging is disabled by default.

`/ID`,`-id` Causes the version of the CDF distribution (not the CDF data set) to be displayed. All other parameters/qualifiers are ignored. This qualifier would be used to determine which version of the CDF distribution you are using.

Example(s):

VMS:

```
$ CDFINQUIRE GISS_VEG
$ CDFINQUIRE/DISPLAY=ihv/OUTPUT=GISS_VEG GISS_VEG
```

UNIX:

```
% cdfinquire giss_veg
% cdfinquire -display ihv -output giss_veg giss_veg
```

MS-DOS:

```
> cdfinquire giss_veg
> cdfinquire c:\cdfs\giss\giss_veg
> cdfinquire -display ihv -output ..\giss_veg giss_veg
```

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

3.7.3 Output from the CDFinquire program

The format of the output from `CDFinquire` is as follows (assuming a display mask of `ihg+av+z+`).

```
INFORMATION BLOCK
-----------------

<copyright>

Created by CDF library version: <version>.<release>.<increment>


HEADER BLOCK
------------

  CDF name.............. <cdf-name>
  Encoding.............. <encoding>
  Majority.............. <majority>
  Format................ <format>
  rRecords.............. <num-r-records>
  Attributes............ <num-attributes> (<num-global-scope> global scope,
                                          <num-variable-scope> variable scope)
  rVariables............ <num-r-variables>
  rDimensions........... <num-dimensions>, sizes: [<dimension-sizes>]
                                                          (rVariables)
  zVariables............ <num-z-variables>


GLOBAL SCOPE ATTRIBUTES
```

```
-----------------------

  <attr-num>.  Name...... <attr-name>
               Entries... <num-g-entries>

                  Number...... <entry-num>
                  Data type... <entry-data-type> (<entry-num-elems> elements)
                  Value....... <entry-value(s)>

                     .
                     .
                     .

      .
      .
      .


VARIABLE SCOPE ATTRIBUTES
-------------------------

  <attr-num>.  Name.............. <attr-name>
               rEntries.......... <num-r-entries>
               zEntries.......... <num-z-entries>
       .
       .
       .


rVARIABLES
----------

  <var-num>.  Name........ <var-name>
              Data type... <var-data-type> (<var-num-elems> elements)
              Variances... <rec-variance>/<dim-variances>

                 Attribute name... <attr-name>
                 Data type........ <entry-data-type>
                                              (<entry-num-elems> elements)
                 Value............ <entry-value(s)>

                    .
                    .
                    .

    .
    .
    .

zVARIABLES
----------
```

```
<var-num>.  Name......... <var-name>
            Data type.... <var-data-type> (<var-num-elems> elements)
            Dimensions... <num-dimensions>, sizes: <dimension-sizes>
            Variances.... <rec-variance>/<dim-variances>

               Attribute name... <attr-name>
               Data type........ <entry-data-type>
                                          (<entry-num-elems> elements)
               Value............ <entry-value(s)>

                  .
                  .
                  .

   .
   .
   .
```

Each field is described as follows.

`<attr-name>`	The attribute name.
`<attr-num>`	The attribute number.
`<cdf-name>`	The name of the CDF.
`<copyright>`	The CDF copyright notice.
`<dim-variances>`	The variable dimension variances. For each dimension, a `T` if values vary along that dimension (`VARY`) or an `F` if they do not (`NOVARY`). Dimension variances are described in Section 2.7.8. If there are zero (0) dimensions, this field is omitted.
`<dimension-sizes>`	The dimension sizes for the rVariables or a zVariable. If there are zero (0) dimensions, this field is omitted.
`<encoding>`	The data encoding of the CDF. Data encodings are described in Section 2.4.
`<entry-data-type>`	The attribute entry data type. Data types are described in Section 2.5.
`<entry-num>`	The attribute entry number.
`<entry-num-elems>`	The number of elements of the data type. The number of elements for an attribute entry is explained in Section 2.8.4.
`<entry-value(s)>`	The attribute entry value(s).
`<format>`	The format of the CDF. CDF formats are described in Section 2.3.
`<increment>`	The increment number of the CDF library which created/modified the CDF.

`<majority>`	The variable majority of the CDF. Variable majorities are described in Section 2.7.9.
`<num-attributes>`	The number of attributes in the CDF.
`<num-dimensions>`	The number of dimensions for the rVariables or a zVariable.
`<num-g-entries>`	The number of gEntries for the gAttribute.
`<num-global-scope>`	The number of gAttributes in the CDF.
`<num-r-entries>`	The number of rEntries for the vAttribute.
`<num-r-records>`	Of all the rVariables, the maximum number of records written (zVariables are not considered).
`<num-r-variables>`	The number of rVariables in the CDF.
`<num-variable-scope>`	The number of vAttributes in the CDF.
`<num-z-entries>`	The number of zEntries for the vAttribute.
`<num-z-variables>`	The number of zVariables in the CDF.
`<rec-variance>`	The variable record variance. A `T` if values do vary from record to record (`VARY`) or an `F` if they do not (`NOVARY`). Record variances are described in Section 2.7.7.
`<release>`	The release number of the CDF library which created/modified the CDF.
`<var-data-type>`	The variable data type. Data types are described in Section 2.5.
`<var-name>`	The variable name.
`<var-num>`	The variable number.
`<var-num-elems>`	The number of elements of the data type. The number of elements for a variable is explained in Section 2.7.6.
`<version>`	The version number of the CDF library which created/modified the CDF.

If a display mask specifying less than `ihg+av+z+` is used, the information not requested will be omitted from the output.

3.8 SkeletonTable

3.8.1 Introduction

The `SkeletonTable` program is used to create an ASCII text file called a skeleton table containing information about a given CDF. It reads a CDF and writes into the skeleton table the following information.

1. Data encoding, majority, and format.

2. Number of dimensions and dimension sizes for the rVariables.
3. gAttribute definitions (and gEntry values).
4. rVariable and zVariable definitions and vAttribute definitions (and rEntry/zEntry values).
5. Non-record-variant (NRV) data for all of the CDF's NRV variables (if requested). NRV variables are described in Section 2.7.7

The above information is written in a format which can be "understood" by the `CDFskeleton` program (see Section 3.9). `CDFskeleton` reads a skeleton table and creates a new CDF.

3.8.2 Executing the SkeletonTable program

The command line syntax for `SkeletonTable` is as follows.

Usage:

VMS:

```
$ SKELETONTABLE [/SKELETON=<skeleton-path>] [/[NO]LOG]
                [/NONRV or /NRVTABLE] <cdf-path>
```

UNIX:

```
% skeletontable [-skeleton <skeleton-path>] [-[no]log]
                [-noNRV or -NRVtable] <cdf-path>
```

MS-DOS:

```
> skeletontable [-skeleton <skeleton-path>] [-[no]log]
                [-noNRV or -NRVtable] <cdf-path>
```

Parameter(s):

`<cdf-path>` The pathname of the CDF to read (do not enter an extension). It is from this CDF that the skeleton table will be created.

Qualifier(s):

`/SKELETON=<skeleton-path>`, `-skeleton <skeleton-path>`

`<skeleton-path>` is the pathname of the skeleton table to be created. Any valid pathname may be specified (do not enter an extension, `.skt` is appended automatically). If this qualifier is not specified, the default skeleton table name is `<cdf-name>.skt` in the default/current directory (where `<cdf-name>` is the name portion of `<cdf-path>`).

`/NONRV`, `-noNRV` or `/NRVTABLE`, `-NRVtable`

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

`/NONRV`, `-noNRV`

Ignore NRV data (the values are not placed in the skeleton table).

`/NRVTABLE`, `-NRVtable`

Put NRV data values in the skeleton table

`/[NO]LOG`, `-[no]log` Specifies whether or not messages are displayed as the program executes. Message logging is disabled by default.

Example(s):

VMS:

```
$ SKELETONTABLE/NOLOG FGGE3B
$ SKELETONTABLE/SKELETON=FGGE3B/noNRV FGGE3B
```

UNIX:

```
% skeletontable -nolog fgge3b
% skeletontable -skeleton fgge3b -noNRV ../cdfs/fgge3b
```

MS-DOS:

```
> skeletontable -nolog fgge3b
> skeletontable -skeleton fgge3b -noNRV c:\temp\fgge3b
```

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

3.8.3 Output from the SkeletonTable program

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

3.9 CDFskeleton

3.9.1 Introduction

The `CDFskeleton` program is used to make a fully structured CDF by reading an ASCII text file called a skeleton table. The skeleton table contains the information necessary to create a CDF which is complete in

all respects except for record-variant (RV) variable values (RV variables vary from record to record). RV values are then written to the CDF by execution of an application program. The `CDFskeleton` 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 those variables that are non-record-variant (NRV). NRV variables do not vary from record to record.

The created CDF is often referred to as a skeleton CDF.

3.9.2 Executing the CDFskeleton program

The command line syntax for the `CDFskeleton` program is as follows.

Usage:

VMS:

```
$ CDFSKELETON [/CDF=<cdf-path>] [/[NO]LOG] [/[NO]DELETE] <skeleton-path>
```

UNIX:

```
% cdfskeleton [-cdf <cdf-path>] [-[no]log] [-[no]delete] <skeleton-path>
```

MS-DOS: UNIX:

```
> cdfskeleton [-cdf <cdf-path>] [-[no]log] [-[no]delete] <skeleton-path>
```

Parameter(s):

`<skeleton-path>`	The pathname of the skeleton table (do not specify an extension). This skeleton table will be read and its information used to create a new CDF.

Qualifier(s):

`/CDF=<cdf-path>, -cdf <cdf-path>`

`<cdf-path>` is the pathname of the CDF that will be created (overriding the pathname in the skeleton table). If this qualifier is not specified, the CDF pathname in the skeleton table is used. Do not specify an extension in the pathname.

`/[NO]LOG, -[no]log` Specifies whether or not messages are displayed as the program executes. Message logging is disabled by default.

`/[NO]DELETE, -[no]delete`

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

Example(s):

VMS:

```
$ CDFSKELETON FGGE3B
$ CDFSKELETON/NOLOG/CDF=[-.TEMP]FGGE3B_X FGGE3B
```

UNIX:

```
% cdfskeleton fgge3b
% cdfskeleton -nolog -cdf ../fgge3b_x fgge3b
```

MS-DOS:

```
> cdfskeleton fgge3b
> cdfskeleton -nolog -cdf ..\fgge3b_x a:\fgge3b
```

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

3.9.3 Creating the skeleton table

A skeleton table is an ASCII 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 which is similar to the CDF you want to create. Then edit the created skeleton table to meet your needs and use `CDFskeleton` 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.

3.10 CDFbrowse

3.10.1 Introduction

The `CDFbrowse` program allows the user to browse through a CDF's metadata and data.[1] More specifically, it allows the user to browse the attribute definitions, variable definitions, attribute entry values, and variable data.

3.10.2 Executing the CDFbrowse program

The command line syntax for `CDFbrowse` is as follows.

Usage:

VMS:

```
$ CDFBROWSE [<cdf-spec>]
```

UNIX:

```
% cdfbrowse [<cdf-spec>]
```

MS-DOS:

```
> cdfbrowse [<cdf-spec>]
```

Parameter(s):

`<cdf-spec>`	The initial specification of the CDF(s) to browse (do not specify an extension). This may be either a single CDF pathname or a directory/wildcard path. Wildcards are allowed in the CDF name but not in the directory path. This parameter is optional.

Example(s):

VMS:

```
$ CDFBROWSE
$ CDFBROWSE NCDS$DATA:GISS_WETL_CLIMATOLOGY
$ CDFBROWSE FGGE3B
```

UNIX:

[1] `CDFbrowse` does not yet (and may never) support zVariables.

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

MS-DOS:

```
> cdfbrowse
> cdfbrowse ..\cac_sst
> cdfbrowse b:\cdfs\giss_wetl_climatology
```

3.10.3 Interaction with CDFbrowse

If not specified on the command line, the first window that appears prompts the user for the CDF and/or directory at which to begin browsing. Entering this results in a window displaying the CDF(s) in that location. The user can scroll upwards or downwards (using the up and down arrow keys), to select the CDF to be browsed. Selecting a CDF and pressing the enter key, results in two windows being displayed. The first window displays the header information and the second window displays a menu with the following selections.

```
Attribute Definitions
Variable Definitions
Variable Values
```

The `Attribute Definition` option, when selected, displays a window containing a list of all the attributes, their scopes (global or variable), and the maximum entry number for each. Selecting an attribute from the list causes a window with the entries for that attribute to be displayed. For each entry, the data type, number of elements, and entry value are displayed. If the attribute is variable in scope, the name of the corresponding variable will also be displayed. For non-character string data types with more than one element, `{...}` will be displayed after the value to indicate that there are more elements. Selecting that entry causes a window with the additional elements to be displayed.

The `Variable Definitions` option, when selected, displays a window containing a list of all the variables, their data types (and number of elements if a character string data type), and their record and dimension variances. Selecting one of the variables causes a window to be displayed containing the variable scope attribute entries associated with that variable. Each entry would include the attribute name, entry data type and number of elements, and the value (attribute entry values are displayed as described for the `Attribute Definitions` option).

The `Variable Values` option, when selected, displays a window containing a list of all the variables as described for the `Variable Definitions` section. Selecting a variable results in a window prompting for the record number to display. Entering the record number causes a window to be displayed containing the values for the variable at that record. Each value is displayed along with the corresponding record number and dimension indices. If the values do not change from record to record, an asterisk is displayed as the record number. Likewise, if the values do not change along a dimension, an asterisk is displayed as the index (and the value is only displayed once).

`Ctrl-K` is used to close a window. `Ctrl-E` may be used to exit `CDFbrowse` at any point. When in a window, the up and down keys are used to move from line to line. If the window contains more items than can be

displayed at one time, the items are automatically scrolled when the up arrow key is used at the top of the window and when the down arrow key is used at the bottom of the window. Also, `d` (the "`d`" key) may be used to move down more than one line at a time within a window (automatically scrolling the window if necessary). Likewise, `u` may be used to move up more than one line at a time. The number of lines moved using `d` or `u` is dependent on the window size.

The user can, at any instant, invoke the "Help" windows. Two types of Help windows are available. `Ctrl-I` brings up the "Explain" window which provides information on the current window. `Ctrl-L` brings up the "Main" help window. The "Main" help window contains a listing of the following topics.

```
Explain the current menu or form.
Explain menus in general.
Show menu key usage.
Explain input forms in general.
Show form key usage.
Show copyright text in CDF V2 header.
Show a map of CDFbrowse's menu structure.
```

Selecting any of the above options brings up a window containing detailed information on the associated topic.

`Ctrl-W` is used to refresh the terminal display. This is sometimes necessary on certain terminals after particular operations.

3.11 CDFlist

3.11.1 Introduction

The `CDFlist` program allows you to selectively list all (or a subset of) the rVariable data values from a CDF.[2] The values can be displayed at the screen, output to a text file, or used to create a new CDF (the new CDF may be a subset of the original CDF).

3.11.2 Executing the CDFlist program

The command line syntax for `CDFlist` is as follows.

Usage:

VMS:

```
$ CDFLIST [<cdf-spec>]
```

UNIX:

[2] `CDFlist` does not yet support zVariables

```
% cdflist [<cdf-spec>]
```

MS-DOS:

```
> cdflist [<cdf-spec>]
```

Parameter(s):

`<cdf-spec>`	The initial specification of the CDF(s) to list (do not specify an extension). This may be either a single CDF pathname or a directory/wildcard path. Wildcards are allowed in the CDF name but not in the directory path. This parameter is optional.

Example(s):

VMS:

```
$ CDFLIST
$ CDFLIST GISS_WETL
$ CDFLIST NCDS$DATA:FGGE3B
```

UNIX:

```
% cdflist
% cdflist ../cac_sst_blended
% cdflist ~/CDFs
```

MS-DOS:

```
> cdflist
> cdflist ..\cac_sst
> cdflist a:\cdfs
```

3.11.3 Interaction with CDFlist

The user interface to `CDFlist` is contained on a single screen which is divided into several fixed windows and pop-up windows. The `Action Menu` controls the execution of the program. This menu allows you to move to a different window, start the data output, browse a help file, or exit the program. You must return to this menu each time you want to select a new variable, output option, listing type, or input CDF. This menu may be displayed at any time by entering `Ctrl-F`. Use the up and down arrow keys to highlight a choice in the `Action Menu` and then enter a carriage return to execute that function. Two very important windows are the `Keypad Definition` and `Message` windows. They contain information on keypad functionality and program status/error messages.

When CDFlist is executed (and a CDF and/or directory is not entered on the command line), you must first specify the CDF you wish to list. If you enter a full disk/directory specification (without a CDF name), all of the CDFs in the specified directory will appear in a separate pop-up window. Use the up and down arrow keys to choose a CDF and then enter a carriage return to list that CDF.

The `Output` window will then be displayed with the valid choices. Use the up and down arrow keys to scroll up and down within the menu and then enter a carriage return to select the desired output option. If output to an ASCII file or CDF is selected, you will be prompted for a file/CDF name. You may enter any file/CDF name valid for the operating system being used or accept the default file/CDF name that appears. Do not enter a file extension (it is automatically appended). The file extension for a generated text file is `.dat`.

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

There are three fields on each line in the `Variable Selection` window: variable name, minimum filter value, and maximum filter value. You can move forward to the next field by entering a carriage return or backwards to the previous field with a `Ctrl-B`. You may also use the up and down arrow keys to move vertically through the fields. When the cursor is positioned in a variable name field, entering `Ctrl-P` will display the variables in the CDF in a pop-up window. Use the arrow keys to highlight a variable and then enter a carriage return to select it. You can quit the pop-up window without selecting a variable by entering `q` (quit). You may also use the `u` and `d` keys to page up and down in the pop-up window. The first line in this pop-up menu contains `**SELECT ALL**`. Selecting this item causes all of the variables in the CDF to be selected.

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

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

When all of the desired variables and filters have been selected, enter `Ctrl-F` to display the `Action Menu`. Selecting `Generate Output` will cause the listing or new CDF to be generated. When `CDFlist` completes the generation of the output, the `Action Menu` is redisplayed. It may then be used to select a new CDF to list, respecify the output option, select new variables/filters, get online help, select special options, or exit the program.

NOTE: On MS-DOS systems, a listing may be terminated early by entering `q`.

When on the `Action Menu`, selecting `Special Options` causes a menu to be displayed with the following options...

1. The format used when displaying `CDF_EPOCH` values may be selected (the `FORMAT` attribute is always ignored when displaying `CDF_EPOCH` values).
2. Whether or not to use the `FORMAT` attribute when displaying values may be selected (the `FORMAT` attribute and corresponding entries must exist in the CDF of course).
3. Whether or not to use the `FILLVAL` attribute when filtering may be selected (assuming the `FILLVAL` attribute exists). If the `FILLVAL` attribute is to be used, fill values are considered valid when determining whether or not to filter out a set of variable values (ie. fill values outside the filter range will appear in the output).
4. Whether or not to ignore all of the filters when generating the output. This is a more convenient way to turn off filtering than to do so manually for each variable on the `Variable Selection` window.

NOTE: If you are running `CDFlist` under UNIX, your terminal output will be limited to 132 chars per line. If too many variables have been selected, the output to the terminal screen will be truncated. Output to a file or CDF, however, will not be affected (as many variables as desired may be selected).

3.12 CDFwalk

3.12.1 Introduction

The `CDFwalk` program allows direct access to rVariable data values.[3] The data values of up to 27 variables may be displayed at once (the CDF can have more variables but only 27 may be viewed at any one time). The data value for each variable at a given record/indices location within the CDF conceptual view is displayed. The user can "walk" from one record/indices location to another sequentially (in any "direction") or a particular record/indices location can be "jumped" to directly. Variable values may also be used to directly "jump" to a record/indices location.

3.12.2 Executing the CDFwalk program

The command line syntax for `CDFwalk` is as follows.

Usage:

VMS:

```
$ CDFWALK [<cdf-spec>]
```

UNIX:

[3] `CDFwalk` does not yet support zVariables.

```
% cdfwalk [<cdf-spec>]
```

MS-DOS:

```
> cdfwalk [<cdf-spec>]
```

Parameter(s):

`<cdf-spec>`	The initial specification of the CDF(s) at which to begin "walking" (do not specify an extension). This may be either a single CDF pathname or a directory/wildcard path. Wildcards are allowed in the CDF name but not in the directory path. This parameter is optional.

Example(s):

VMS:

```
$ CDFWALK
$ CDFWALK [-.TEMP]NOAA7
```

UNIX:

```
% cdfwalk
% cdfwalk ../cac_sst
% cdfwalk /disk3/CDFs/ssclimate
```

MS-DOS:

```
> cdfwalk
> cdfwalk ..\cac_sst
> cdfwalk c:\cdfs
```

3.12.3 Interaction with CDFwalk

The user interface to `CDFwalk` is contained on a single screen which is divided into several fixed windows and pop-up windows. The `Action Menu` controls the execution of the program. This menu allows you to "walk" through a CDF, select a new CDF through which to "walk", browse online help, or exit the program. This menu may be displayed at any time by entering `Ctrl-F`. Use the up and down arrow keys to highlight a choice in the `Action Menu` and then enter a carriage return to execute that function. Two very important windows are the `Keypad Definition` and `Message` windows. They contain information on keypad functionality and program status/error messages.

When `CDFwalk` is executed (and a CDF and/or directory is not entered on the command line), you must first specify the CDF through which you wish to "walk". If you enter a full disk/directory specification (without

a CDF name), all of the CDFs in the specified directory will appear in a separate pop-up window. Use the up and down arrow keys to choose a CDF and then enter a carriage return to "walk" throught that CDF.

After the input CDF is selected, the `Record/Indices` window appears. The size of this window depends on the dimensionality of the input CDF. Vertically the window will contain one row for the record number and one row for each dimension of the CDF. Horizontally, each line of the window contains three fields. The first field contains a variable name, the second field contains a data value for the variable, and the third field contains a record number or dimension index.

The variable selected for the record number line will be one that only varies from record to record (not varying along any of the dimensions). If more than one variable of that type exists in the CDF, a pop-up window will be displayed containing the list of varibles. Select one of the variables keeping in mind that you will have an opportunity to use it for positioning by value to a desired record. If a variable does not exist meeting the requirements mentioned above, only the record number field will be used.

The variable selected for each dimension line will be one that only varys along the corresponding dimension (and does not vary from record to record). Again, if more than one variable of that type exists in the CDF, a pop-up window will be displayed to allow you to select the variable to use for that dimension. Keep in mind that you will be able to use it to position by value to a desired index along that dimension. If, for a particular dimension, a variable does not exist meeting the requirements mentioned above, only the dimension index field will be used.

After the `Record/Indices` window has been specified (which will usually occur automatically), the `Variable Values` window will be displayed. It contains up to 27 variable names and the associated data values for the record/indices shown in the `Record/Indices` window.

Initially the cursor will be positioned at the data value field of the record number line in the `Record/Indices` window (assuming that a variable was found meeting the above requirements - if not, the cursor will start at the record number field). You may now begin to "walk" through the CDF by using the right arrow to increment to and the left arrow to decrement to a new record. For each new record, the data for each variable is updated in both the `Record/Indices` and `Variable Values` windows. If the CDF contains many records, you may want to quickly advance to a desired record. You may do this by entering the `i` key. `CDFwalk` enters an input mode and prompts for the value to which to advance. If the exact value is not in the CDF for the variable, the record number is changed to correspond to the closest possible value. Another method of advancing to a desired value is by entering `Ctrl-P`. This will cause a pop-up window containing all of the values for the variable to be displayed. You may select the value desired as you would with any other pop-up window and `CDFwalk` will automatically position the record number to the corresponding value.

After positioning to a desired record (and the CDF is multi-dimensional) you may now wish to "walk" through the dimension indices. This is done by entering the up and down arrow keys to position the cursor to the desired dimension line. "Walking" through dimensions is done in the same way as "walking" through the record numbers.

You may also wish to "walk" through the CDF by using the actual record number or dimension indices rather than data values (this would also be necessary if variables meeting the above requirements were not found in the CDF). This is done by entering a carriage return which causes the cursor to advance to the next field (ie. from the data value field to the record number/indices field). Entering `Ctrl-B` causes the cursor to move back to the previous field. "Walking" through the CDF using the record number/indices fields is the same as using the data value fields but with a few exceptions. First, no `Ctrl-P` option is available with these fields. Also, when in input mode (after entering `i`), you may input either an absolute record number/index to which to move or a relative offset for the record number/index. Prefixing the value with a `+/-` will increment/decrement the record number/index by that amount.

If the CDF contains more than 27 variables and a particular variable is not currently displayed in the `Variable Values` window, an option is provided to scroll new variables into that window. This is done by entering a `v` key. A small pop-up window is displayed in which you may use the up and down arrow keys to scroll in or scroll out variables. If the CDF contains a very large number of variables, entering a `u` or `d` key will display a new set of 27 variables (moving up or down through the variables in the CDF). When the desired set of variables is displayed, enter a carriage return to continue "walking" thru the CDF.

When you are done "walking" through the CDF, enter `Ctrl-F` to display the `Action Menu`. You may input a new CDF, resume "walking" through the current CDF, select special options, read online help, or exit the program.

When on the `Action Menu`, selecting `Special Options` causes a menu to be displayed with the following options...

1. Whether or not to use the `FORMAT` attribute when displaying values may be selected (the `FORMAT` attribute and corresponding entries must exist in the CDF of course).
2. Stay tuned...

Chapter 4

CDF C PROGRAMMING REFERENCE

4.1 Introduction

This chapter provides a reference for using the CDF library from C applications. A separate reference exists for writing CDF applications in FORTRAN (Chapter 5).

4.2 Compiling

Each source file which calls the CDF library or references CDF parameters must include `cdf.h`. On VMS systems a logical name, `CDF$INC`, which specifies the location of `cdf.h` is defined in the definitions file, `DEFINITIONS.COM`, provided with the CDF distribution. On UNIX systems an environment variable, `CDF_INC`, which serves the same purpose is defined in the definitions file `definitions.<shell>` (where `<shell>` is the shell being used). This section assumes that you are using the appropriate definitions file.

One of two methods may be used to include `cdf.h`.

4.2.1 Specifying `cdf.h` location in the compile command

The first method involves including the following line at the top of each source file...

```
#include "cdf.h"
```

Since the pathname of the disk/directory containing `cdf.h` was not specified, it must be specified when the source file is compiled.

VMS systems

An example of the command to compile a source file on VMS systems would be as follows...

```
$ CC/INCLUDE_DIRECTORY=CDF$INC <source-file(s)>
```

where `<source-file(s)>` is the name of the source file(s) being compiled.

UNIX systems

An example of the command to compile a source file on UNIX systems would be as follows...

```
% cc -c -I${CDF_INC} <source-file(s)>.c
```

where `<source-file(s)>` is the name of the source file(s) being compiled.

The `-c` option specifies that only an object module(s) is to be produced (the link step is described in Section 4.3).

MS-DOS systems

An example of the command to compile a source file on MS-DOS systems would be as follows (for Microsoft C, Borland C, and Turbo C respectively)...

```
> CL /c /AL /I<inc-path> <source-file(s)>.c
> BCC -c -ml -I<inc-path> <source-file(s)>.c
> TCC -c -ml -I<inc-path> <source-file(s)>.c
```

where `<source-file(s)>` is the name of the source file(s) being compiled and `<inc-path>` is the pathname of the directory containing `cdf.h`. You will need to know where on your system `cdf.h` has been installed. You may also need to specify the location of system include files. Consult the documentation for the compiler being used for more information.

The `/c`,`-c` option specifies that only an object module(s) is to be produced (the link step is described in Section 4.3).

The `/AL`,`-ml` option specifies that the object module(s) is to be compiled using the large memory model. The huge memory model could also be used if necessary. The CDF library can be built using either the large or huge memory model. All of your application source files must be compiled using the same memory model as that used for the CDF library.

Most C compilers for MS-DOS systems have some type of environment in which you can develop your applications (Integrated Developers Environment in Borland C and Turbo C, Programmer's Workbench in Microsoft C, etc.). The options shown above for the command line compilers are specified in the development environment being used. Consult the documentation for the development environment you are using for the steps needed to compile (and link) your application.

4.2.2 Specifying cdf.h location in the source file

The second method involves specifying the pathname of the directory containing **cdf.h** in the actual source file. The following line would be included at the top of each source file...

```
#include "<inc-path>cdf.h"
```

where **<inc-path>** is the pathname of the directory containing **cdf.h**. The source file would then be compiled as shown in Section 4.2.1 but without specifying the location of **cdf.h** on the command line.

On VMS systems, **CDF$INC:** may be used for **<inc-path>**. On UNIX and MS-DOS systems, **<inc-path>** must be a relative or absolute pathname (an environment variable may not be used for **<inc-path>** on UNIX systems). Check with your system manager to determine where on your system **cdf.h** has been installed.

4.3 Linking

Your applications must be linked with the CDF library (**LIBCDF.OLB** on VMS systems, **libcdf.a** on UNIX systems, and **libcdf.lib** on MS-DOS systems).[1] Both the Standard and Internal interfaces for C applications are built into the CDF library. On VMS systems a logical name, **CDF$LIB**, which specifies the location of the CDF library is defined in the definitions file, **DEFINITIONS.COM**, provided with the CDF distribution. On UNIX systems an environment variable, **CDF_LIB**, which serves the same purpose is defined in the definitions file **definitions.<shell>** (where **<shell>** is the shell being used). This section assumes that you are using the appropriate definitions file.

4.3.1 VMS systems

An example of the command to link your application with the CDF library on VMS systems would be as follows...

```
$ LINK <object-file(s)>, CDF$LIB:LIBCDF/LIBRARY
```

where **<object-file(s)>** is your application's object module(s). The name of the executable created will be the name part of the (first) object file with **.EXE** appended.

4.3.2 UNIX systems

An example of the command to link your application with the CDF library on UNIX systems would be as follows...

```
% cc <object-file(s)>.o ${CDF_LIB}/libcdf.a
```

[1] A shared version of the CDF library is also available on VMS and some flavors of UNIX. Its use is described in Section 4.4.

where <object-file(s)> is your application's object module(s). The name of the executable created will be a.out by default. It may also be explicitly specified using the -o option. Some UNIX systems may also require that -lm (the math library) and/or -ldl (the dynamic linker library) be specified. This may depend on the release of the operating system begin used (eg. SunOS).

Combining the compile and link

On UNIX systems the compile and link may be combined into one step as follows...

```
% cc -I${CDF_INC} <source-file(s)>.c ${CDF_LIB}/libcdf.a
```

4.3.3 MS-DOS systems

An example of the command to link your application with the CDF library on MS-DOS systems would be as follows (for Microsoft C, Borland C, and Turbo C respectively)...

```
> LINK <object-file(s)>.obj,,nul.map,<lib-path>libcdf.lib
> TLINK /x <object-file(s)>.obj,,,<lib-path>libcdf.lib
> TLINK /x <object-file(s)>.obj,,,<lib-path>libcdf.lib
```

where <object-file(s)> is your application's object module(s) and <lib-path> is the pathname of the directory containing the CDF library. You will need to know where on your system the CDF library has been installed.

For Microsoft C, the omitted parameter (indicated by ,,) is the name of the executable to be created. The name of the executable will be the name part of the (first) object file with .exe appended. A different name for the executable may be explicitly specified if desired. A map file is created by default unless the special name nul.map is used (as shown). If a map file is desired, the map file parameter should be omitted (in which case the name of the map file will be the name part of the executable file with .map appended) or a map file should be explicitly specified.

For Borland C and Turbo C, the two omitted parameters (indicated by ,,,) are the name of the executable to be created and the name of a map file to be created. The name of the executable will be the name part of the (first) object file with .exe appended. A different name for the executable may be explicitly specified if desired. A map file is created by default unless the /x option is used (as shown). In either case, the name of the map file will be the name part of the executable file with .map appended (unless a name for the map file is explicitly specified).

You may also need to specify the location of system startup modules and system libraries. Consult the documentation for the compiler/linker being used for more information.

NOTE: The same memory model must have been used to compile your application's source files and the CDF library. The CDF library must have been compiled with the large or huge memory model.

Most C compilers for MS-DOS systems have some type of environment in which you can develop your applications (Integrated Developers Environment in Borland C and Turbo C, Programmer's Workbench in

Microsoft C, etc.). The options shown above for the command line linkers are specified in the development environment being used. Consult the documentation for the development environment you are using for the steps needed to link (and compile) your application.

Combining the compile and link

On MS-DOS systems the compile and link may be combined into one step as follows (for Microsoft C, Borland C, and Turbo C respectively)...

```
> CL -AL /I<inc-path> <source-file(s)>.c <lib-path>libcdf.lib
> BCC -ml -I<inc-path> <source-file(s)>.c <lib-path>libcdf.lib
> TCC -ml -I<inc-path> <source-file(s)>.c <lib-path>libcdf.lib
```

The name of the executable created will be the name part of the (first) source file with `.exe` appended. Additional options/parameters may be required. Consult the documentation for the compiler/linker being used for more information.

4.4 Linking, shared CDF library

A shareable version of the CDF library is also available on VMS systems and some flavors of UNIX. The shared version is put in the same directory as the non-shared version and is named as follows...

Machine/Operating System	Shared CDF Library
VMS	`LIBCDF.EXE`
UNIX, Sun/SunOS	`libcdf.so`
UNIX, HP9000/HP-UX	`libcdf.sl`
UNIX, IBM RS6000/AIX	`libcdf.o`

The commands necessary to link to a shared library vary from operating system to operating system. Please consult the documentation for your operating system for details.

NOTE: On VMS systems, the shared CDF library may also be installed in `SYS$SHARE`.

4.5 Item Referencing

For C applications, all items are referenced starting at zero (0). These include variable, attribute, and attribute entry numbers, record numbers, dimensions, and dimension indices. Note that both rVariables and zVariables are numbered starting at zero (0).

4.6 CDF Parameters — Predefined Constants and Types

The following sections describe constants and types defined for use by all CDF application programs written in C. These constants and types are defined in `cdf.h`. `cdf.h` should be included in all application source files referencing CDF functions/parameters.

4.6.1 Defined Types

The following typedef's are provided. They should be used when declaring or defining the corresponding items.

`CDFstatus`	All CDF functions except `CDFvarNum` and `CDFattrNum` are of type `CDFstatus`. They return a status code indicating the completion status of the function. The `CDFerror` function can be used to inquire the meaning of any status code. Appendix C lists the possible status codes along with their explanations. Section 4.8 describes how to interpret status codes.
`CDFid`	An identifier (or handle) for a CDF which must be used when referring to a CDF. A new `CDFid` is established whenever a CDF is created or opened, establishing a connection to that CDF on disk. The `CDFid` is used in all subsequent operations on a particular CDF. The `CDFid` must not be altered by an application.

4.6.2 CDFstatus Constants

These constants are of type `CDFstatus`.

`CDF_OK`	A status code indicating the normal completion of a CDF function.
`CDF_WARN`	Threshold constant for testing severity of non-normal CDF status codes.

Section 4.8 describes how to use these constants to interpret status codes.

4.6.3 CDF Formats

`SINGLE_FILE`	The CDF consists of only one file.
`MULTI_FILE`	The CDF consists of one header file for control and attribute data and one additional file for each variable in the CDF.

4.6.4 CDF Datatypes

One of the following constants must be used when specifying a CDF data type for an attribute entry or variable.

CDF_BYTE	1-byte, signed integer.
CDF_CHAR	1-byte, signed character.
CDF_INT1	1-byte, signed integer.
CDF_UCHAR	1-byte, unsigned character.
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_REAL4	4-byte, floating point.
CDF_FLOAT	4-byte, floating point.
CDF_REAL8	8-byte, floating point.
CDF_DOUBLE	8-byte, floating point.
CDF_EPOCH	8-byte, floating point.

CDF_CHAR and CDF_UCHAR are considered character string data types. These are significant because only variables of these data types may have more than one element per value (where each element is a character).

4.6.5 Data Encodings

A CDF's data encoding affects how its attribute entry and variable data values are stored (on disk). Attribute entry and variable values in memory are always in the host machine's native encoding.

HOST_ENCODING	Indicates host machine data representation (native). This encoding will provide the greatest performance but will not allow the CDF to be transported to other machines that have different encodings.
NETWORK_ENCODING	Indicates network transportable data representation (XDR). This encoding allows the CDF to be transported to other machines but with (possibly) reduced performance.
VAX_ENCODING	Indicates VAX data representation.
SUN_ENCODING	Indicates SUN data representation.
MIPSEB_ENCODING	Indicates MIPSEB data representation (Silicon Graphics Iris and Power Series).
MIPSEL_ENCODING	Indicates MIPSEL data representation (DECstation).
IBMRS_ENCODING	Indicates IBMRS data representation (IBM RS/6000 series).
HP_ENCODING	Indicates HP data representation (HP 9000 series).

IBMPC_ENCODING	Indicates IBM PC data representation.
NeXT_ENCODING	Indicates NeXT data representation.

When creating a CDF, you must specify NETWORK_ENCODING, HOST_ENCODING, or the specific encoding for the host machine (eg. VAX_ENCODING). Specifying the host machine's encoding explicitly has the same effect as specifying HOST_ENCODING.

When inquiring the encoding of a CDF, either NETWORK_ENCODING or the specific machine encoding will be returned (HOST_ENCODING is never returned).

4.6.6 Variable Majorities

A CDF's variable majority determines the order in which variable values (within the variable arrays) are stored in the CDF file(s). The majority is the same for rVariable and zVariables.

ROW_MAJOR	C-like array ordering for variable storage. The first dimension in each variable array varies the slowest.
COL_MAJOR	FORTRAN-like array ordering for variable storage. The first dimension in each variable array varies the fastest.

For C applications, the compiler defined majority for arrays is row major. The first dimension of multi-dimensional arrays varies the slowest in memory.

Knowing the variable majority of a CDF is necessary when performing hyper reads and writes. During a hyper read, the CDF library will place the variable data values into the memory buffer in the same majority as that of the CDF. The buffer must then be processed according to that majority. Likewise, during a hyper write, the CDF library will expect to find the variable data values in the memory buffer in the same majority as that of the CDF.

4.6.7 Record/Dimension Variances

Record and dimension variances affect how variable data values are physically stored.

VARY	True record or dimension variance.
NOVARY	False record or dimension variance.

If a variable has a record variance of VARY, then each record for that variable is physically stored. If the record variance is NOVARY, then only one record is physically stored (all virtual records contain the same values).

If a variable has a dimension variance of VARY, then each value/subarray along that dimension is physically stored. If the dimension variance is NOVARY, then only one value/subarray along that dimension is physically stored (all virtual values/subarrays along that dimension are the same).

4.6.8 Attribute Scopes

Attribute scopes are simply a way to explicitly declare the intended use of an attribute by user applications (and the CDF toolkit).

GLOBAL_SCOPE	Indicates that an attribute's scope is global (applies to the CDF as a whole).
VARIABLE_SCOPE	Indicates that an attribute's scope is by-variable (each rEntry or zEntry corresponds to an rVariable or zVariable respectively).
GLOBAL_SCOPE_ASSUMED	Indicates that an attribute's scope is assumed global.
VARIABLE_SCOPE_ASSUMED	Indicates that an attribute's scope is assumed by-variable.

GLOBAL_SCOPE and VARIABLE_SCOPE are definite scopes. GLOBAL_SCOPE_ASSUMED and VARIABLE_SCOPE_ASSUMED are assumed scopes.

Section 2.8.3 explains attribute scopes and how assumed scopes can arise in a CDF.

4.6.9 Operational Limits

These are limits within the CDF library. If you reach one of these limits please contact CDF User Support.

CDF_MAX_DIMS	Maximum number of dimensions for the rVariables or a zVariable.

The CDF library imposes no limit on the number of variables, attributes, or attribute entries that a CDF may have. On the IBM PC, however, the number of rVariables and zVariables will be limited to 100 of each in a multi-file CDF because of the 8.3 naming convention imposed by MS-DOS.

4.6.10 Limits of Names and Other Character Strings

CDF_PATHNAME_LEN	Maximum length of a CDF pathname (excluding the NUL[2] terminator and the .cdf or .vnn appended by the CDF library to construct file names). A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating systems being used (including logical names on VMS systems and environment variables on UNIX systems). **NOTE:** On UNIX systems pathnames are case sensitive.
CDF_VAR_NAME_LEN	Maximum length of a variable name (excluding the NUL terminator).
CDF_ATTR_NAME_LEN	Maximum length of an attribute name (excluding the NUL terminator).
CDF_COPYRIGHT_LEN	Maximum length of the CDF copyright text (excluding the NUL terminator).

[2] The ASCII null character, 0 octal.

CDF_STATUSTEXT_LEN	Maximum length of the explanation text for a status code (excluding the NUL terminator).

4.7 C Function Reference

The following sections describe the CDF library functions callable from C applications. Most functions return a status code of type CDFstatus (see Section 4.8). Each function except CDFlib is part of the Standard Interface — CDFlib is part of the Internal Interface. An application can use both interfaces when necessary. Note that zVariables and vAttribute zEntries are only accessible via the Internal Interface.

Each section begins with an ANSI-style function prototype.

4.7.1 CDFcreate (Standard Interface)

```
CDFstatus CDFcreate(
char  *CDFname,        /* in  -- CDF pathname. */
long  numDims,         /* in  -- Number of dimensions, rVariables. */
long  dimSizes[],      /* in  -- Dimension sizes, rVariables. */
long  encoding,        /* in  -- Data encoding. */
long  majority,        /* in  -- Variable majority. */
CDFid *id);            /* out -- CDF identifier. */
```

CDFcreate creates a CDF as defined by the arguments. A CDF cannot be created if it already exists (the existing CDF will not be overwritten). If you want to overwrite an existing CDF, you must first open it with CDFopen, delete it with CDFdelete, and then recreate it with CDFcreate. If the existing CDF is corrupted, the call to CDFopen will fail (an error code will be returned). In this case you must delete the CDF at the command line. Delete the .cdf file and if the CDF has the multi-file format, delete all of the .v0,.v1,... files and .z0,.z1,... files.

The arguments to CDFcreate are defined as follows.

CDFname	The pathname of the CDF to create (do not specify an extension). This may be at most CDF_PATHNAME_LEN characters (excluding the NUL). A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on VMS systems and environment variables on UNIX systems). **NOTE:** On UNIX systems pathnames are case sensitive.
numDims	Number of dimensions the rVariables in the CDF are to have. This may be as few as zero (0) and at most CDF_MAX_DIMS.
dimSizes	The size of each dimension. dimSizes is a 1-dimensional array containing numDims elements. Each element of dimSizes specifies the corresponding dimension size. Each size must be greater then zero (0). If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

encoding	The encoding for variable data and attribute entry data. Specify one of the encodings described in Section 4.6.5.
majority	The majority for variable data. Specify one of the majorities described in Section 4.6.6.
id	The identifier for the created CDF. This identifier must be used in all subsequent operations on the CDF.

When a CDF is created, both read and write access are allowed. The default format for a CDF created with CDFcreate is multi-file (MULTI_FILE). The CDFlib function (Internal Interface) must be used to change a CDF's format to single-file (SINGLE_FILE).

CDFclose MUST be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk (see Section 4.7.5).

Example(s)

The following example will create a CDF named test1 with network encoding and row majority.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                                 /* CDF identifier. */
CDFstatus   status;                             /* Returned status code. */
static long numDims = 3;                        /* Number of dimensions,
                                                   rVariables. */
static long dimSizes[3] = {180,360,10};         /* Dimension sizes,
                                                   rVariables. */
static long majority = ROW_MAJOR;               /* Variable majority. */
.
.
status = CDFcreate ("test1", numDims, dimSizes, NETWORK_ENCODING,
                    majority, &id);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

ROW_MAJOR and NETWORK_ENCODING are defined in cdf.h.

4.7.2 CDFopen (Standard Interface)

```
CDFstatus CDFopen(
char  *CDFname,        /* in  -- CDF pathname. */
CDFid *id);            /* out -- CDF identifier. */
```

CDFopen opens an existing CDF. The CDF is initially opened with only read access. This allows multiple applications to read the same CDF simultaneously. When an attempt to modify the CDF is made, it is automatically closed and reopened with read/write access (the function will fail if the application does not have or can not get write access to the CDF).

The arguments to CDFopen are defined as follows.

CDFname	The pathname of the CDF to open (do not specify an extension). This may be at most CDF_PATHNAME_LEN characters (excluding the NUL). A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on VMS systems and environment variables on UNIX systems). **NOTE:** On UNIX systems pathnames are case sensitive.
id	The identifier for the opened CDF. This identifier must be used in all subsequent operations on the CDF.

CDFclose MUST be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk (see Section 4.7.5).

Example(s)

The following example will open a CDF named NOAA1.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                         /* CDF identifier. */
CDFstatus   status;                     /* Returned status code. */
static char CDFname[] = { "NOAA1" };    /* Name of CDF. */
.
.
status = CDFopen (CDFname, &id);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.3 CDFdoc (Standard Interface)

```
CDFstatus CDFdoc(
CDFid id,                             /* in  -- CDF identifier. */
long  *version,                       /* out -- Version number. */
long  *release,                       /* out -- Release number. */
char  copyRight[CDF_DOCUMENT_LEN+1]); /* out -- Copyright. */
```

CDFdoc is used to inquire general documentation about a CDF. The version/release of the CDF library which created the CDF is provided (eg. CDF V2.3 is version 2, release 3) along with the CDF copyright notice.

The arguments to CDFdoc are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate or CDFopen.
version	The version number of the CDF library which created the CDF.
release	The release number of the CDF library which created the CDF.
copyRight	The copyright notice of the CDF library which created the CDF. This character string must be large enough to hold CDF_COPYRIGHT_LEN + 1 characters (including the NUL terminator). This string will contain a newline character after each line of the copyright notice.

The copyright notice is formatted for printing without modification. The version and release are used together (eg. CDF V2.3 is version 2, release 3).

Example(s)

The following example will inquire and display the version/release and copyright notice.

```
.
.
#include "cdf.h"
.
.
CDFid     id;                              /* CDF identifier. */
CDFstatus status;                          /* Returned status code. */
long      version;                         /* CDF version number. */
long      release;                         /* CDF release number. */
char      copyRight[CDF_COPYRIGHT_LEN+1]; /* Copyright notice -- +1 for
                                             NUL terminator. */
.
.
status = CDFdoc (id, &version, &release, copyRight);

if (status < CDF_OK)                    /* INFO status codes ignored */
  UserStatusHandler (status);
else {
  printf ("CDF V%d.%d\n", version, release);
  printf("%s\n", copyRight);
}
.
.
```

4.7.4 CDFinquire (Standard Interface)

```
CDFstatus CDFinquire(
CDFid id,                          /* in  -- CDF identifier */
long  *numDims,                    /* out -- Number of dimensions, rVariables. */
long  dimSizes[CDF_MAX_DIMS],      /* out -- Dimension sizes, rVariables. */
long  *encoding,                   /* out -- Data encoding. */
long  *majority,                   /* out -- Variable majority. */
long  *maxRec,                     /* out -- Maximum record number in the
                                          CDF, rVariables. */
long  *numVars,                    /* out -- Number of rVariables in
                                          the CDF. */
long  *numAttrs);                  /* out -- Number of attributes in the CDF. */
```

`CDFinquire` inquires the basic characteristics of a CDF. An application needs to know the number of dimensions and their sizes before it can access variable data. Knowing the variable majority can be used to optimize performance and is necessary to properly access the buffer of data values used with the hyper functions — `CDFvarHyperPut` and `CDFvarHyperGet`.

The arguments to `CDFinquire` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`numDims`	The number of dimensions for the rVariables in the CDF.
`dimSizes`	The dimension sizes of the rVariables in the CDF. `dimSizes` is a 1-dimensional array containing one element per dimension. Each element of `dimSizes` receives the corresponding dimension size. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
`encoding`	The encoding of the variable data and attribute entry data. The encodings are defined in Section 4.6.5.
`majority`	The majority of the variable data. The majorities are defined in Section 4.6.6.
`maxRec`	The maximum record number written to an rVariable in the CDF. Note that the maximum record number written is also kept separately for each rVariable in the CDF. `maxRec` is the largest of these. Some rVariables may have fewer records actually written. `CDFlib` (Internal Interface) may be used to inquire the maximum record written for an individual rVariable (see Section 4.7.24).
`numVars`	The number of rVariables in the CDF.
`numAttrs`	The number of attributes in the CDF.

Example(s)

The following example will inquire the basic information about a CDF.

.

```
.
#include "cdf.h"
.
.
CDFid id;                       /* CDF identifier. */
CDFstatus status;               /* Returned status code. */
long numDims;                   /* Number of dimensions, rVariables. */
long dimSizes[CDF_MAX_DIMS];    /* Dimension sizes, rVariables
                                   (allocate to allow the maximum number
                                   of dimensions). */
long encoding;                  /* Data encoding. */
long majority;                  /* Variable majority. */
long maxRec;                    /* Maximum record number, rVariables. */
long numVars;                   /* Number of rVariables in CDF. */
long numAttrs;                  /* Number of attributes in CDF. */
.
.
status = CDFinquire (id, &numDims, dimSizes, &encoding, &majority,
                     &maxRec, &numVars, &numAttrs);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.5 CDFclose (Standard Interface)

```
CDFstatus CDFclose(
CDFid id);              /* in -- CDF identifier. */
```

`CDFclose` closes the specified CDF. The CDF's data buffers are flushed, the CDF's open file is closed (or files in the case of a multi-file CDF), and the CDF identifier is made available for reuse.

You MUST close a CDF with `CDFclose` to guarantee that all modifications you have made will actually be written to the CDF's file(s). If your program exits, normally or otherwise, without a successful call to `CDFclose`, the CDF's data buffers are left unflushed. If you have only read from a CDF, and not written to it, failure to close it is of no consequence. Of course, it is always good practice to close your CDFs before exiting your application.

The arguments to `CDFclose` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.

Example(s)

The following example will close an open CDF.

```
.
```

```
.
#include "cdf.h"
.
.
CDFid id;                       /* CDF identifier. */
CDFstatus status;               /* Returned status code. */
.
.
status = CDFclose (id);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.6 CDFdelete (Standard Interface)

```
CDFstatus CDFdelete(
CDFid id);              /* in -- CDF identifier. */
```

`CDFdelete` deletes the specified CDF. The CDF files deleted include the `.cdf` file and if the CDF consists of multiple files, the `.v0`,`.v1`,... files and `.z0`,`.z1`,... files.

You must open a CDF before you are allowed to deleted it. If you have no privilege to delete the CDF files, they will not be deleted. If the CDF is corrupted and cannot be opened, the CDF file(s) must be deleted at the command line.

The arguments to `CDFdelete` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.

Example(s)

The following example will open and then delete an existing CDF.

```
.
.
#include "cdf.h"
.
.
CDFid id;               /* CDF identifier. */
CDFstatus status;       /* Returned status code. */
.
.
status = CDFopen ("test2", &id);
if (status < CDF_OK)               /* INFO status codes ignored. */
  UserStatusHandler (status);
```

```
else {
  status = CDFdelete (id);
  if (status != CDF_OK) UserStatusHandler (status);
}
.
.
```

4.7.7 CDFerror (Standard Interface)

```
CDFstatus CDFerror(
CDFstatus status,                          /* in  -- Status code. */
char      message[CDF_STATUSTEXT_LEN+1]);  /* out -- Explanation text for
                                                    the status code. */
```

`CDFerror` is used to inquire the explanation of a given status code (not just error codes). Section 4.8 explains how to interpret status codes and Appendix C lists all of the possible status codes.

The arguments to `CDFerror` are defined as follows.

`status`	The status code to check.
`message`	The explanation of the status code. This character string must be large enough to hold `CDF_STATUSTEXT_LEN + 1` characters (including the NUL terminator).

Example(s)

The following example displays the explanation text if an error code is returned from a call to `CDFopen`.

```
.
.
#include "cdf.h"
.
.
CDFid     id;                    /* CDF identifier. */
CDFstatus status;                /* Returned status code. */
char text[CDF_STATUSTEXT_LEN+1]; /* Explanation text.  +1 added for
                                    NUL terminator. */
.
.
status = CDFopen ("giss_wetl", &id);
if (status < CDF_WARN) {              /* INFO and WARNING codes ignored. */
  CDFerror (status, text);
  printf ("ERROR> %s\n", text);
}
.
.
```

4.7.8 CDFattrCreate (Standard Interface)

```
CDFstatus CDFattrCreate(
CDFid id,             /* in  -- CDF identifier. */
char  *attrName,      /* in  -- Attribute name. */
long  attrScope,      /* in  -- Scope of attribute. */
long  *attrNum);      /* out -- Attribute number. */
```

`CDFattrCreate` creates an attribute in the specified CDF. An attribute with the same name must not already exist in the CDF.

The arguments to `CDFattrCreate` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrName`	The name of the attribute to create. This may be at most `CDF_ATTR_NAME_LEN` characters (excluding the NUL terminator). Attribute names are case-sensitive.
`attrScope`	The scope of the new attribute. Specify one of the scopes described in Section 4.6.8.
`attrNum`	The number assigned to the new attribute. This number must be used in subsequent CDF function calls when referring to this attribute. An existing attribute's number may be determined with the `CDFattrNum` function.

Example(s)

The following example creates two attributes. The `TITLE` attribute is created with global scope — it applies to the entire CDF (most likely the title of the data set stored in the CDF). The `Units` attribute is created with variable scope — each entry describes some property of the corresponding variable (in this case the units for the data).

```
.
.
#include "cdf.h"
.
.
CDFid     id;                               /* CDF identifier. */
CDFstatus status;                           /* Returned status code. */
static char UNITSattrName[] = {"Units"};    /* Name of "Units" attribute. */
long        UNITSattrNum;                   /* "Units" attribute number. */
long        TITLEattrNum;                   /* "TITLE" attribute number. */
static long TITLEattrScope = GLOBAL_SCOPE;  /* "TITLE" attribute scope. */
.
.
status = CDFattrCreate (id, "TITLE", TITLEattrScope, &TITLEattrNum);
if (status != CDF_OK) UserStatusHandler (status);
```

```
status = CDFattrCreate (id, UNITSattrName, VARIABLE_SCOPE, &UNITSattrnum);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.9 CDFattrNum (Standard Interface)

```
long CDFattrNum(
CDFid id,              /* in -- CDF id */
char  *attrName);      /* in -- attribute name */
```

`CDFattrNum` is used to determine the attribute number associated with a given attribute name. If the attribute is found, `CDFattrNum` returns its number — which will be equal to or greater than zero (0). If an error occurs (eg. the attribute name does not exist in the CDF), an error code (of type `CDFstatus`) is returned. Error codes are less than zero (0).

The arguments to `CDFattrNum` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrName`	The name of the attribute for which to search. This may be at most `CDF_ATTR_NAME_LEN` characters (excluding the NUL terminator). Attribute names are case-sensitive.

`CDFattrNum` may be used as an embedded function call when an attribute number is needed.

Example(s)

In the following example the attribute named `pressure` will be renamed `PRESSURE` with `CDFattrNum` being used as an embedded function call. Note that if the attribute `pressure` did not exist in the CDF, the call to `CDFattrNum` would have returned an error code. Passing that error code to `CDFattrRename` as an attribute number would have resulted in `CDFattrRename` also returning an error code. `CDFattrRename` is described in Section 4.7.10.

```
.
.
#include "cdf.h"
.
.
CDFid     id;       /* CDF identifier. */
CDFstatus status;   /* Returned status code. */
.
.
status = CDFattrRename (id, CDFattrNum(id,"pressure"), "PRESSURE");
if (status != CDF_OK) UserStatusHandler (status);
```

4.7.10 CDFattrRename (Standard Interface)

```
CDFstatus CDFattrRename(
CDFid id,             /* in -- CDF identifier. */
long  attrNum,        /* in -- Attribute number. */
char  *attrName);     /* in -- New attribute name. */
```

`CDFattrRename` is used to rename an existing attribute. An attribute with the new name must not already exist in the CDF.

The arguments to `CDFattrRename` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrNum`	The number of the attribute to rename. This number may be determined with a call to `CDFattrNum` (see Section 4.7.9).
`attrName`	The new attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters (excluding the NUL terminator). Attribute names are case-sensitive.

Example(s)

In the following example the attribute named `LAT` is renamed `LATITUDE`.

```
.
.
#include "cdf.h"
.
.
CDFid     id;       /* CDF identifier. */
CDFstatus status;   /* Returned status code. */
.
.
status = CDFattrRename (id, CDFattrNum(id,"LAT"), "LATITUDE");
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.11 CDFattrInquire (Standard Interface)

```
CDFstatus CDFattrInquire(
CDFid id,             /* in  -- CDF identifier. */
long  attrNum,        /* in  -- Attribute number. */
char  *attrName,      /* out -- Attribute name. */
long  *attrScope,     /* out -- Attribute scope. */
long  *maxEntry);     /* out -- Maximum entry number. */
```

`CDFattrInquire` is used to inquire about the specified attribute. To inquire about a specific attribute entry, use `CDFattrEntryInquire` (Section 4.7.12).

The arguments to `CDFattrInquire` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrNum`	The number of the attribute to inquire. This number may be determined with a call to `CDFattrNum` (see Section 4.7.9).
`attrName`	The attribute's name. This character string must be large enough to hold `CDF_ATTR_NAME_LEN + 1` characters (including the NUL terminator).
`attrScope`	The scope of the attribute. Attribute scopes are defined in Section 4.6.8.
`maxEntry`	For gAttributes this is the gEntry number used. For vAttributes this is the maximum rEntry number used. In either case, this may not correspond with the number of entries (if some entry numbers were not used). The number of entries actually used may be inquired with the `CDFlib` function (Internal Interface, see Section 4.7.24).

Example(s)

The following example displays the name of each attribute in a CDF. The number of attributes in the CDF is first determined using the function `CDFinquire`. Note that attribute numbers start at zero (0) and are consecutive.

```
.
.
#include "cdf.h"
.
.
CDFid     id;                       /* CDF identifier. */
CDFstatus status;                   /* Returned status code. */
long numDims;                       /* Number of dimensions. */
long dimSizes[CDF_MAX_DIMS];        /* Dimension sizes (allocate to allow
                                       the maximum number of dimensions). */
long encoding;                      /* Data encoding. */
long majority;                      /* Variable majority. */
long maxRec;                        /* Maximum record number in CDF. */
long numVars;                       /* Number of variables in CDF. */
long numAttrs;                      /* Number of attributes in CDF. */
long attrN;                         /* Attribute number. */
char attrName[CDF_ATTR_NAME_LEN+1]; /* Attribute name -- +1 for NUL
                                       terminator. */
long attrScope;                     /* Attribute scope. */
long maxEntry;                      /* Maximum entry number. */
.
.
```

```
status = CDFinquire (id, &numDims, dimSizes, &encoding, &majority,
                     &maxRec, &numVars, &numAttrs);
if (status != CDF_OK) UserStatusHandler (status);

for (attrN = 0; attrN < numAttrs; attrN++) {
   status = CDFattrInquire (id, attrN, attrName, &attrScope, &maxEntry);
   if (status < CDF_OK)             /* INFO status codes ignored. */
     UserStatusHandler (status);
   else
     printf ("%s\n", attrName);
}
.
.
```

4.7.12 CDFattrEntryInquire (Standard Interface)

```
CDFstatus CDFattrEntryInquire(
CDFid id,             /* in  -- CDF identifier. */
long  attrNum,        /* in  -- Attribute number. */
long  entryNum,       /* in  -- Entry number. */
long  *dataType,      /* out -- Data type. */
long  *numElements);  /* out -- Number of elements (of the data type). */
```

`CDFattrEntryInquire` is used to inquire about a specific attribute entry. To inquire about the attribute in general, use `CDFattrInquire` (Section 4.7.11). `CDFattrEntryInquire` would normally be called before calling `CDFattrGet` in order to determine the data type and number of elements (of that data type) for an entry. This would be necessary to correctly allocate enough memory to receive the value read by `CDFattrGet`.

The arguments to `CDFattrEntryInquire` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrNum`	The attribute number for which to inquire an entry. This number may be determined with a call to `CDFattrNum` (see Section 4.7.9).
`entryNum`	The entry number to inquire. If the attribute is global in scope, this is simply the entry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).
`dataType`	The data type of the specified entry. The data types are defined in Section 4.6.4.
`numElements`	The number of elements of the data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string (an array of characters). For all other data types, this is the number of elements in an array of that data type.

Example(s)

The following example inquires each entry for an attribute. Note that entry numbers need not be consecutive — not every entry number between zero (0) and the maximum entry number must exist. For this reason, NO_SUCH_ENTRY is an expected error code. Note also that if the attribute has variable scope, the entry numbers are actually rVariable numbers.

```
.
.
#include "cdf.h"
.
.
CDFid     id;                              /* CDF identifier. */
CDFstatus status;                          /* Returned status code. */
long      attrN;                           /* Attribute number. */
long      entryN;                          /* Entry number. */
char      attrName[CDF_ATTR_NAME_LEN+1];   /* Attribute name, +1 for NUL
                                              terminator. */
long      attrScope;                       /* Attribute scope. */
long      maxEntry;                        /* Maximum entry number used. */
long      dataType;                        /* Data type. */
long      numElems;                        /* Number of elements (of the
                                              data type). */
.
.
attrN = CDFattrNum (id, "TMP");
if (attrN < 0) UserStatusHandler (attrN);  /* If less than zero (0), then
                                              it must be a warning/error
                                              code. */

status = CDFattrInquire (id, attrN, attrName, &attrScope, &maxEntry);
if (status != CDF_OK) UserStatusHandler (status);

for (entryN = 0; entryN <= maxEntry; entryN++) {
   status = CDFattrEntryInquire (id, attrN, entryN, &dataType, &numElems);
   if (status < CDF_OK) {
     if (status != NO_SUCH_ENTRY) UserStatusHandler (status);
   }
   else {
     /* process entries */
     .
     .
   }
}
```

4.7.13 CDFattrPut (Standard Interface)

```
CDFstatus CDFattrPut(
```

```
CDFid id,             /* in -- CDF identifier. */
long  attrNum,        /* in -- Attribute number. */
long  entryNum,       /* in -- Entry number. */
long  dataType,       /* in -- Data type of this entry. */
long  numElements,    /* in -- Number of elements (of the data type). */
void  *value);        /* in -- Value. */
```

`CDFattrPut` is used to write an attribute entry to a CDF. The entry may or may not already exist. If it does exist, it is overwritten. The data type and number of elements (of that data type) may be changed when overwriting an existing entry.

The arguments to `CDFattrPut` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrNum`	The attribute number. This number may be determined with a call to `CDFattrNum` (see Section 4.7.9).
`entryNum`	The entry number. If the attribute is global in scope, this is simply the entry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).
`dataType`	The data type of the specified entry. Specify one of the data types defined in Section 4.6.4.
`numElements`	The number of elements of the data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string (an array of characters). For all other data types, this is the number of elements in an array of that data type.
`value`	The value(s) to write. The entry value is written to the CDF from memory address `value`.

`numElements` elements of the data type `dataType` will be written to the CDF starting from memory address `value`.

Example(s)

The following example writes two attribute entries. The first is to gEntry number zero (0) of the gAttribute `TITLE`. The second is to the variable scope attribute `VALIDs` for the rEntry which corresponds to the rVariable `TMP`.

```
.
.
#include "cdf.h"
.
.
```

```
#define TITLE_LEN   10                /* Length of CDF title. */
.
.
CDFid        id;                      /* CDF identifier. */
CDFstatus    status;                  /* Returned status code. */
long         entryNum;                /* Entry number. */
long         numElements;             /* Number of elements (of data
                                         type). */
static char  title[TITLE_LEN+1]
                   = {"CDF title."};  /* Value of TITLE attribute,
                                         entry number 0. */
static short TMPvalids = {15,30};     /* Value(s) of VALIDs attribute,
                                         rEntry for rVariable TMP. */
.
.
entryNum = 0;
status = CDFattrPut (id, CDFattrNum(id,"TITLE"), entryNum, CDF_CHAR,
                     TITLE_LEN, title);
if (status != CDF_OK) UserStatusHandler (status);
.
.
numElements = 2;
status = CDFattrPut (id, CDFattrNum(id,"VALIDs"), CDFvarNum(id,"TMP"),
                     CDF_INT2, numElements, TMPvalids);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.14 CDFattrGet (Standard Interface)

```
CDFstatus CDFattrGet(
CDFid id,             /* in  -- CDF identifier. */
long  attrNum,        /* in  -- Attribute number. */
long  entryNum,       /* in  -- Entry number. */
void  *value);        /* out -- Value. */
```

`CDFattrGet` is used to read an attribute entry from a CDF. In most cases it will be necessary to call `CDFattrEntryInquire` before calling `CDFattrGet` in order to determine the data type and number of elements (of that data type) for the entry.

The arguments to `CDFattrGet` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrNum`	The attribute number. This number may be determined with a call to `CDFattrNum` (see Section 4.7.9).
`entryNum`	The entry number. If the attribute is global in scope, this is simply the entry number and has meaning only to the application. If the attribute is variable

in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).

`value` The value read. This buffer must be large enough to hold the value. `CDFattrEntryInquire` would be used to determine the entry data type and number of elements (of that data type). The value is read from the CDF and placed into memory at address `value`.

Example(s)

The following example displays the value of the `UNITS` attribute for the rEntry corresponding to the `PRES_LVL` rVariable (but only if the data type is `CDF_CHAR`). Note that the CDF library does not automatically NUL terminate character data (when the data type is `CDF_CHAR` or `CDF_UCHAR`) for attribute entries (or variable values).

```
.
.
#include "cdf.h"
.
.
CDFid     id;             /* CDF identifier. */
CDFstatus status;         /* Returned status code. */
long      attrN;          /* Attribute number. */
long      entryN;         /* Entry number. */
long      dataType;       /* Data type. */
long      numElems;       /* Number of elements (of data type). */
void      *buffer;        /* Buffer to receive value. */
.
.
attrN = CDFattrNum (id, "UNITS");
if (attrN < 0) UserStatusHandler (attrN);  /* If less than zero (0), then
                                              it must be a warning/error
                                              code. */

entryN = CDFvarNum (id, "PRES_LVL");         /* The rEntry number
                                                is the rVariable
                                                number. */
if (entryN < 0) UserStatusHandler (entryN);  /* If less than zero (0), then
                                                it must be a warning/error
                                                code. */

status =  CDFattrEntryInquire (id, attrN, entryN, &dataType, &numElems);
if (status != CDF_OK) UserStatusHandler (status);

if (dataType == CDF_CHAR) {
  buffer = (char *) malloc (numElems + 1);
  if (buffer == NULL)...

  status = CDFattrGet (id, attrN, entryN, buffer);
```

```
    if (status != CDF_OK) UserStatusHandler (status);

    buffer[numElems] = '\0';          /* NUL terminate. */

    printf ("Units of PRES_LVL variable: %s\n", buffer);

    free (buffer);
  }
  .
  .
```

4.7.15 CDFattrCorrectScope (Standard Interface)

```
CDFstatus CDFattrCorrectScope(
CDFid id,             /* in -- CDF identifier. */
long  attrNum,        /* in -- Attribute number. */
long  attrScope)      /* in -- New scope of attribute. */
```

`CDFattrCorrectScope` is used to correct the scope of an attribute. Section 2.8.3 describes attribute scopes and the need for this function.

The arguments to `CDFattrCorrectScope` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrNum`	The attribute number. This number may be determined with a call to `CDFattrNum` (see Section 4.7.9).
`attrScope`	The new scope for the attribute. Specify one of the definite scopes defined in Section 4.6.8.

IMPORTANT NOTE

This function is not a permanent one — it may be deleted from a future release of the CDF library. Version 1 CDFs should be converted to Version 2 CDFs as soon as possible. Any assumed attribute scopes should then be corrected by using this function or the `CDFscope` toolkit program.

Example(s)

The following example will correct the scope of every attribute in a CDF. Assumed scopes will be corrected to the corresponding definite scope. Note that it may not always be the case that an assumed scope should be corrected to the corresponding definite scope (see Section 2.8.3). The toolkit program `CDFscope` could also be used to correct attribute scopes.

```
  .
```

```
.
#include "cdf.h"
.
.
CDFid     id;                       /* CDF identifier. */
CDFstatus status;                   /* Returned status code. */
long numDims;                       /* Number of dimensions. */
long dimSizes[CDF_MAX_DIMS];        /* Dimension sizes (allocate to allow
                                       the maximum number of dimensions). */
long encoding;                      /* Data encoding. */
long majority;                      /* Variable majority. */
long maxRec;                        /* Maximum rVariable record number in
                                       CDF. */
long numVars;                       /* Number of rVariables in CDF. */
long numAttrs;                      /* Number of attributes in CDF. */
long attrN;                         /* Attribute number. */
char attrName[CDF_ATTR_NAME_LEN+1]; /* Attribute name -- +1 for NUL
                                       terminator. */
long attrScope;                     /* Attribute scope. */
long maxEntry;                      /* Maximum entry number. */
.
.
status = CDFinquire (id, &numDims, dimSizes, &encoding, &majority,
                     &maxRec, &numVars, &numAttrs);
if (status != CDF_OK) UserStatusHandler (status);

for (attrN = 0; attrN < numAttrs; attrN++) {
   status = CDFattrInquire (id, attrN, attrName, &attrScope, &maxEntry);
   if (status != CDF_OK) UserStatusHandler (status);

   switch (attrScope) {
     case GLOBAL_SCOPE_ASSUMED:
       status = CDFattrCorrectScope (id, attrN, GLOBAL_SCOPE);
       if (status != CDF_OK) UserStatusHandler (status);
       break;
     case VARIABLE_SCOPE_ASSUMED:
       status = CDFattrCorrectScope (id, attrN, VARIABLE_SCOPE);
       if (status != CDF_OK) UserStatusHandler (status);
       break;
     default:
       /* Do nothing --- definite scope. */
   }
}
.
.
```

4.7.16 CDFvarCreate (Standard Interface)

```
CDFstatus CDFvarCreate(
```

```
CDFid id,              /* in --  CDF identifier. */
char  *varName,        /* in --  rVariable name. */
long  dataType,        /* in --  Data type. */
long  numElements,     /* in --  Number of elements (of the data type). */
long  recVariance,     /* in --  Record variance. */
long  dimVariances[],  /* in --  Dimension variances. */
long  *varNum);        /* out -- rVariable number. */
```

CDFvarCreate is used to create a new rVariable in a CDF. A variable (rVariable or zVariable) with the same name must not already exist in the CDF.

The arguments to CDFvarCreate are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate or CDFopen.
varName	The name of the rVariable to create. This may be at most CDF_VAR_NAME_LEN characters (excluding the NUL terminator). Variable names are case-sensitive.
dataType	The data type of the new rVariable. Specify one of the data types defined in Section 4.6.4.
numElements	The number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (each value consists of the entire string). For all other data types, this must always be one (1) — multiple elements at each value are not allowed for non-character string data types.
recVariance	The rVariable's record variance. Specify one of the variances defined in Section 4.6.7.
dimVariances	The rVariable's dimension variances. dimVariances is a 1-dimensional array containing one element per rVariable dimension. Each element of dimVariances specifies the corresponding dimension variance. For each dimension, specify one of the variances defined in Section 4.6.7. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
varNum	The number assigned to the new rVariable. This number must be used in subsequent CDF function calls when referring to this rVariable. An existing rVariables's number may be determined with the CDFvarNum function.

Example(s)

The following example will create several rVariables in a CDF whose rVariables are 2-dimensional. In this case, EPOCH, LAT, and LON are independent rVariables and TMP is a dependent rVariable.

```
.
.
#include "cdf.h"
.
```

```
.
CDFid     id;                                   /* CDF identifier. */
CDFstatus status;                               /* Returned status code. */
static long EPOCHrecVary = {VARY};              /* EPOCH record variance. */
static long LATrecVary = {NOVARY};              /* LAT record variance. */
static long LONrecVary = {NOVARY};              /* LON record variance. */
static long TMPrecVary = {VARY};                /* TMP record variance. */
static long EPOCHdimVarys = {NOVARY,NOVARY};    /* EPOCH dimension
                                                    variances. */
static long LATdimVarys = {NOVARY,VARY};        /* LAT dimension variances. */
static long LONdimVarys = {VARY,NOVARY};        /* LON dimension variances. */
static long TMPdimVarys = {VARY,VARY};          /* TMP dimension variances. */
long        EPOCHvarNum;                        /* EPOCH variable number. */
long        LATvarNum;                          /* LAT rVariable number. */
long        LONvarNum;                          /* LON rVariable number. */
long        TMPvarNum;                          /* TMP rVariable number. */
.
.
status = CDFvarCreate (id, "EPOCH", CDF_EPOCH, 1,
                       EPOCHrecVary, EPOCHdimVarys, &EPOCHvarNum);
if (status != CDF_OK) UserStatusHandler (status);

status = CDFvarCreate (id, "LATITUDE", CDF_INT2, 1,
                       LATrecVary, LATdimVarys, &LATvarNum);
if (status != CDF_OK) UserStatusHandler (status);

status = CDFvarCreate (id, "LONGITUDE", CDF_INT2, 1,
                       LONrecVary, LONdimVarys, &LONvarNum);
if (status != CDF_OK) UserStatusHandler (status);

status = CDFvarCreate (id, "TEMPERATURE", CDF_REAL4, 1,
                       TMPrecVary, TMPdimVarys, &TMPvarNum);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.17 CDFvarNum (Standard Interface)

```
long CDFvarNum(
CDFid id,             /* in -- CDF identifier. */
char  *varName);      /* in -- rVariable name. */
```

`CDFvarNum` is used to determine the rVariable number associated with a given rVariable name. If the rVariable is found, `CDFvarNum` returns its number — which will be equal to or greater than zero (0). If an error occurs (eg. the rVariable does not exist in the CDF), an error code (of type `CDFstatus`) is returned. Error codes are less than zero (0).

The arguments to `CDFvarNum` are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate or CDFopen.
varName	The name of the rVariable for which to search. This may be at most CDF_VAR_NAME_LEN characters (excluding the NUL terminator). Variable names are case-sensitive.

CDFvarNum may be used as an embedded function call when an rVariable number is needed.

Example(s)

In the following example CDFvarNum is used as an embedded function call when inquiring about an rVariable.

```
.
.
#include "cdf.h"
.
.
CDFid     id;                           /* CDF identifier. */
CDFstatus status;                       /* Returned status code. */
char      varName[CDF_VAR_NAME_LEN+1];  /* rVariable name. */
long      dataType;                     /* Data type of the rVariable. */
long      numElements;                  /* Number of elements (of the
                                           data type). */
long      recVariances;                 /* Record variance. */
long      dimVariances[CDF_MAX_DIMS];   /* Dimension variances. */
.
.
status = CDFvarInquire (id, CDFvarNum(id,"LATITUDE"), varName, &dataType,
                        &numElements, &recVariance, dimVariances);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

In this example the rVariable named LATITUDE was inquired. Note that if LATITUDE did not exist in the CDF, the call to CDFvarNum would have returned an error code. Passing that error code to CDFvarInquire as an rVariable number would have resulted in CDFvarInquire also returning an error code. Also note that the name written into varName is already known (LATITUDE). In some cases the rVariable names will be unknown — CDFvarInquire would be used to determine them. CDFvarInquire is described in Section 4.7.19.

4.7.18 CDFvarRename (Standard Interface)

```
CDFstatus CDFvarRename(
CDFid id,             /* in -- CDF identifier. */
long  varNum,         /* in -- rVariable number. */
char  *varName);      /* in -- New name. */
```

CDFvarRename is used to rename an existing rVariable. A variable (rVariable or zVariable) with the same name must not already exist in the CDF.

The arguments to CDFvarRename are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate or CDFopen.
varNum	The number of the rVariable to rename. This number may be determined with a call to CDFvarNum (see Section 4.7.17).
varName	The new rVariable name. This may be at most CDF_VAR_NAME_LEN characters (excluding the NUL terminator). Variable names are case-sensitive.

Example(s)

In the following example the rVariable named TEMPERATURE is renamed TMP (if it exists). Note that if CDFvarNum returns a value less than zero (0), that value is not an rVariable number but rather a warning/error code.

```
.
.
#include "cdf.h"
.
.
CDFid     id;       /* CDF identifier. */
CDFstatus status;   /* Returned status code. */
long      varNum;   /* rVariable number. */
.
.
varNum = CDFvarNum (id, "TEMPERATURE");
if (varNum < 0) {
  if (varNum != NO_SUCH_VAR) UserStatusHandler (varNum);
}
else {
  status = CDFvarRename (id, varNum, "TMP");
  if (status != CDF_OK) UserStatusHandler (status);
}
.
.
```

4.7.19 CDFvarInquire (Standard Interface)

```
CDFstatus CDFvarInquire(
CDFid id,                          /* in  -- CDF identifier. */
long  varNum,                      /* in  -- rVariable number. */
char  varName,                     /* out -- rVariable name. */
long  *dataType,                   /* out -- Data type. */
```

```
long  *numElements,                 /* out -- Number of elements (of the
                                               data type). */
long  *recVariance,                 /* out -- Record variance. */
long  dimVariances[CDF_MAX_DIMS]);  /* out -- Dimension variances. */
```

`CDFvarInquire` is used to inquire about the specified rVariable. This function would normally be used before reading rVariable values (with `CDFvarGet` or `CDFvarHyperGet`) to determine the data type and number of elements (of that data type).

The arguments to `CDFvarInquire` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`varNum`	The number of the rVariable to inquire. This number may be determined with a call to `CDFvarNum` (see Section 4.7.17).
`varName`	The rVariable's name. This character string must be large enough to hold `CDF_VAR_NAME_LEN + 1` characters (including the NUL terminator).
`dataType`	The data type of the rVariable. The data types are defined in Section 4.6.4.
`numElements`	The number of elements of the data type at each rVariable value. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string (each value consists of the entire string). For all other data types, this will always be one (1) — multiple elements at each value are not allowed for non-character string data types.
`recVariance`	The record variance. The record variances are defined in Section 4.6.7.
`dimVariances`	The dimension variances. `dimVariances` is a 1-dimensional array containing one element per rVariable dimension in the CDF. Each element of `dimVariances` receives the corresponding dimension variance. The dimension variances are defined in Section 4.6.7. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

Example(s)

The following example inquires about an rVariable named `HEAT_FLUX` in a CDF. Note that the rVariable name returned by `CDFvarInquire` will be the same as that passed in to `CDFvarNum`.

```
.
.
#include "cdf.h"
.
.
CDFid     id;                           /* CDF identifier. */
CDFstatus status;                       /* Returned status code. */
char      varName[CDF_VAR_NAME_LEN+1];  /* rVariable name, +1
                                           for NUL terminator. */
```

```
long      dataType;                    /* Data type of the rVariable. */
long      numElems;                    /* Number of elements (of data
                                          type). */
long      recVary;                     /* Record variance. */
long      dimVarys[CDF_MAX_DIMS];      /* Dimension variances (allocate
                                          to allow the maximum number of
                                          dimensions). */
.
.
status = CDFvarInquire (id, CDFvarNum(id,"HEAT_FLUX"), varName, &dataType,
                        &numElems, &recVary, dimVarys);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.20 CDFvarPut (Standard Interface)

```
CDFstatus CDFvarPut(
CDFid id,             /* in -- CDF identifier. */
long  varNum,         /* in -- rVariable number. */
long  recNum,         /* in -- Record number. */
long  indices[],      /* in -- Dimension indices. */
void  *value);        /* in -- Value. */
```

`CDFvarPut` is used to write a single value to an rVariable. `CDFvarHyperPut` may be used to write more than one rVariable value with a single call (see Section 4.7.22).

The arguments to `CDFvarPut` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`varNum`	The number of the rVariable to which to write. This number may be determined with a call to `CDFvarNum` (see Section 4.7.17).
`recNum`	The record number at which to write.
`indices`	The array indices within the specified record at which to write. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` specifies the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
`value`	The value to write. The value is written to the CDF from memory address `value`.

Example(s)

The following example writes values to the rVariable named `LATITUDE` in a CDF whose rVariables are 2-dimensional with dimension sizes `[360,181]`. For `LATITUDE`, the record variance is `NOVARY`, the dimension

variances are [NOVARY,VARY], and the data type is CDF_INT2.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                       /* CDF identifier. */
CDFstatus   status;                   /* Returned status code. */
short       lat;                      /* Latitude value. */
long        varN;                     /* rVariable number. */
static long recNum = 0;               /* Record number. */
static long indices[2] = {0,0};       /* Dimension indices. */
.
.
varN = CDFvarNum (id, "LATITUDE");
if (varN < 0) UserStatusHandler (varN);  /* If less than zero (0), not a
                                            rVariable number but
                                            rather a warning/error code. */

for (lat = -90; lat <= 90; lat += 1) {
   indices[1] = 90 + lat;
   status = CDFvarPut (id, varN, recNum, indices, &lat);
   if (status != CDF_OK) UserStatusHandler (status);
}
.
.
```

Since the record variance is NOVARY, the record number (recNum) is set to zero (0). Also note that because the dimension variances are [NOVARY,VARY], only the second dimension is varied as values are written (the values are "virtually" the same at each index of the first dimension).

4.7.21 CDFvarGet (Standard Interface)

```
CDFstatus CDFvarGet(
CDFid id,             /* in  -- CDF identifier. */
long  varNum,         /* in  -- rVariable number. */
long  recNum,         /* in  -- Record number. */
long  indices[],      /* in  -- Dimension indices. */
void  *value);        /* out -- Value. */
```

CDFvarGet is used to read a single value from an rVariable. CDFvarHyperGet may be used to read more than one rVariable value with a single call (see Section 4.7.23).

The arguments to CDFvarGet are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate or CDFopen.

`varNum`	The number of the rVariable from which to read. This number may be determined with a call to `CDFvarNum` (see Section 4.7.17).
`recNum`	The record number at which to read.
`indices`	The array indices within the specified record at which to read. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` specifies the corresponding dimension index. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
`value`	The value read. This buffer must be large enough to hold the value. `CDFvarInquire` would be used to determine the rVariable's data type and number of elements (of that data type) at each value. The value is read from the CDF and placed at memory address `value`.

Example(s)

The following example will read and hold an entire record of data from an rVariable. The CDF's rVariables are 3-dimensional with sizes `[180,91,10]`. For this rVariable, the record variance is `VARY`, the dimension variances are `[VARY,VARY,VARY]`, and the data type is `CDF_REAL4`.

```
.
.
#include "cdf.h"
.
.
CDFid     id;                /* CDF identifier. */
CDFstatus status;            /* Returned status code. */
float     tmp[180][91][10];  /* Temperature values. */
long      indices[3];        /* Dimension indices. */
long      varN;              /* rVariable number. */
long      recNum;            /* Record number. */
long      d0, d1, d2;        /* Dimension index values. */
.
.
varN = CDFvarNum (id, "Temperature");
if (varN < 0) UserStatusHandler (varN);  /* If less than zero (0), then
                                            it is actually a warning/error
                                            code. */

recNum = 13;

for (d0 = 0; d0 < 180; d0++) {
   indices[0] = d0;
   for (d1 = 0; d1 < 91; d1++) {
      indices[1] = d1;
      for (d2 = 0; d2 < 10; d2++) {
         indices[2] = d2;
         status = CDFvarGet (id, varN, recNum, indices, &tmp[d0][d1][d2]);
         if (status != CDF_OK) UserStatusHandler (status);
```

```
        }
      }
   }
   .
   .
```

4.7.22 CDFvarHyperPut (Standard Interface)

```
CDFstatus CDFvarHyperPut(
CDFid id,             /* in -- CDF identifier. */
long  varNum,         /* in -- rVariable number. */
long  recStart,       /* in -- Starting record number. */
long  recCount,       /* in -- Number of records. */
long  recInterval,    /* in -- Interval between records. */
long  indices[],      /* in -- Dimension indices of starting value. */
long  counts[],       /* in -- Number of values along each dimension. */
long  intervals[],    /* in -- Interval between values along each dimension. */
void  *buffer);       /* in -- Buffer of values. */
```

`CDFvarHyperPut` is used to write a buffer of one or more values to an rVariable. It is important to know the variable majority of the CDF before using `CDFvarHyperPut` because the values in the buffer to be written must be in the same majority. `CDFinquire` is used to determine the variable majority of a CDF. Section 2.7.9 describes the variable majorities.

The arguments to `CDFvarHyperPut` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`varNum`	The number of the rVariable to which to write. This number may be determined with a call to `CDFvarNum` (see Section 4.7.17).
`recStart`	The record number at which to start writing.
`recCount`	The number of records to write.
`recInterval`	The interval between records for subsampling[3] (eg. an interval of 2 means write to every other record).
`indices`	The indices (within each record) at which to start writing. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` specifies the corresponding dimension index. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
`counts`	The number of values along each dimension to write. `counts` is a 1-dimensional array containing one element per rVariable dimension. Each element of `count` specifies the corresponding dimension count. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

[3] "Subsampling" is not the best term to use when writing data but you should know what we mean.

`intervals`	For each dimension, the interval between values for subsampling[4] (eg. an interval of 2 means write to every other value). `intervals` is a 1-dimensional array containing one element per rVariable dimension. Each element of `intervals` specifies the corresponding dimension interval. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
`buffer`	The buffer of values to write. The majority of the values in this buffer must be the same as that of the CDF. The values starting at memory address `buffer` are written to the CDF.

Example(s)

The following example writes values to the rVariable `LATITUDE` of a CDF whose rVariables are 2-dimensional with dimension sizes `[360,181]`. For `LATITUDE`, the record variance is `NOVARY`, the dimension variances are `[NOVARY,VARY]`, and the data type is `CDF_INT2`. This example is similar to the example in Section 4.7.20 except that is uses a single call to `CDFvarHyperPut` rather than numerous calls to `CDFvarPut`.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                       /* CDF identifier. */
CDFstatus   status;                   /* Returned status code. */
short       lat;                      /* Latitude value. */
short       lats[181];                /* Buffer of latitude values. */
long        varN;                     /* rVariable number. */
long recStart = 0;                    /* Record number. */
long recCount = 1;                    /* Record counts. */
long recInterval = 1;                 /* Record interval. */
static long indices[2] = {0,0};       /* Dimension indices. */
static long counts[2] = {1,181};      /* Dimension counts. */
static long intervals[2] = {1,1};     /* Dimension intervals. */
.
.
varN = CDFvarNum (id, "LATITUDE");
if (varN < 0) UserStatusHandler (varN);  /* If less than zero (0), not an
                                            rVariable number but rather a
                                            warning/error code. */

for (lat = -90; lat <= 90; lat += 1) lats[90+lat] = lat;

status = CDFvarHyperPut (id, varN, recStart, recCount, recInterval,
                         indices, counts, intervals, lats);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

[4] Again, not the best term.

4.7.23 CDFvarHyperGet (Standard Interface)

```
CDFstatus CDFvarHyperGet(
CDFid id,             /* in --  CDF identifier. */
long  varNum,         /* in --  rVariable number. */
long  recStart,       /* in --  Starting record number. */
long  recCount,       /* in --  Number of records. */
long  recInterval,    /* in --  Subsampling interval between records. */
long  indices[],      /* in --  Dimension indices of starting value. */
long  counts[],       /* in --  Number of values along each dimension. */
long  intervals[],    /* in --  Subsampling intervals along each dimension. */
void  *buffer);       /* out -- Buffer of values. */
```

`CDFvarHyperGet` is used to read a buffer of one or more values from an rVariable. It is important to know the variable majority of the CDF before using `CDFvarHyperGet` because the values placed into the buffer will be in that majority. `CDFinquire` is used to determine the variable majority of a CDF. Section 2.7.9 describes the variable majorities.

The arguments to `CDFvarHyperGet` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`varNum`	The number of the rVariable from which to read. This number may be determined with a call to `CDFvarNum` (see Section 4.7.17).
`recStart`	The record number at which to start reading.
`recCount`	The number of records to read.
`recInterval`	The interval between records for subsampling (eg. an interval of 2 means read every other record).
`indices`	The indices (within each record) at which to start reading. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` specifies the corresponding dimension index. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
`counts`	The number of values along each dimension to write. `counts` is a 1-dimensional array containing one element per rVariable dimension. Each element of `counts` specifies the corresponding dimension count. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
`intervals`	For each dimension, the interval between values for subsampling (eg. an interval of 2 means read every other value). `intervals` is a 1-dimensional array containing one element per rVariable dimension. Each element of `intervals` specifies the corresponding dimension interval. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
`buffer`	The buffer of values read. The majority of the values in this buffer will be the same as that of the CDF. This buffer must be large to hold the values. `CDFvarInquire` would be used to determine the rVariable's data type and number of elements (of that data type) at each value. The values are read from the

CDF and placed into memory starting at address `buffer`.

Example(s)

The following example will read an entire record of data from an rVariable. The CDF's rVariables are 3-dimensional with sizes `[180,91,10]` and CDF's variable majority is `ROW_MAJOR`. For the rVariable, the record variance is `VARY`, the dimension variances are `[VARY,VARY,VARY]`, and the data type is `CDF_REAL4`. This example is similar to the example in Section 4.7.21 except that it uses a single call to `CDFvarHyperGet` rather than numerous calls to `CDFvarGet`.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                         /* CDF identifier. */
CDFstatus   status;                     /* Returned status code. */
float       tmp[180][91][10];           /* Temperature values. */
long        varN;                       /* rVariable number. */
long recStart = 13;                     /* Record number. */
long recCount = 1;                      /* Record counts. */
long recInterval = 1;                   /* Record interval. */
static long indices[3] = {0,0,0};       /* Dimension indices. */
static long counts[3] = {180,91,10};    /* Dimension counts. */
static long intervals[3] = {1,1,1};     /* Dimension intervals. */
.
.
varN = CDFvarNum (id, "Temperature");
if (varN < 0) UserStatusHandler (varN);  /* If less than zero (0), then
                                            it is actually a warning/error
                                            code. */

status = CDFvarHyperGet (id, varN, recStart, recCount, recInterval,
                         indices, counts, intervals, tmp);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

Note that if the CDF's variable majority had been `COL_MAJOR`, the `tmp` array would have been declared `float tmp[10][91][180]` for proper indexing.

4.7.24 CDFlib (Internal Interface)

```
CDFstatus CDFlib (...);
```

The Internal Interface consists of only one function, `CDFlib`. `CDFlib` can be used to perform all possible operations on a CDF. (In fact, all of the Standard Interface functions are implemented using the Internal

Interface). CDFlib must be used to perform operations not possible with the Standard Interface functions. These operations would involve CDF features added after the Standard Interface functions had been defined (eg. specifying a single-file format for a CDF, accessing zVariables, or specifying a pad value for an rVariable or zVariable). Note that CDFlib can also be used to perform certain operations more efficiently than with the Standard Interface functions.

CDFlib takes a variable number of arguments which specify one or more operations to be performed. The operations are performed according to the order of the arguments. Each operation is performed on an item. An item may be either an object (eg. a CDF, variable, or attribute) or a state (eg. a CDF's format or a variable's data type/number of elements). The possible operations and corresponding items (on which to perform those operations) are described in detail later in this section.

Example(s)

The easiest way to explain how to use CDFlib would be to start with a few examples. The following example will show how a CDF would be created with the single-file format.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                         /* CDF identifier (handle). */
CDFstatus   status;                     /* Status returned from CDF
                                           library. */
static char CDFname[] = {"test1"};      /* Name of the CDF. */
long        numDims = 2;                /* Number of dimensions. */
static long dimSizes[2] = {100,200};    /* Dimension sizes. */
long        encoding = HOST_ENCODING;   /* Data encoding. */
long        majority = ROW_MAJOR;       /* Variable data majority. */
long        format = SINGLE_FILE;       /* Format of CDF. */
.
.
status = CDFcreate (CDFname, numDims, dimSizes, encoding, majority, &id);
if (status != CDF_OK) UserStatusHandler (status);

status = CDFlib (SELECT_, CDF_, id,
                 PUT_, CDF_FORMAT_, format,
                 NULL_);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

The call to CDFcreate created the CDF as expected but the default format is MULTI_FILE (a separate file for each variable in the CDF). The call to CDFlib is then used to change the format to SINGLE_FILE (which must be done before any variables are created in the CDF).

The arguments to CDFlib in this example are explained as follows.

SELECT_	The first operation to be performed. In this case an item is going to be selected. SELECT_ is defined in cdf.h (as are all CDF constants).
CDF_	The item to be selected — the current CDF (the CDF on which subsequent operations are to be performed).
id	The identifier of the CDF being selected.
PUT_	The next operation to be performed. In this case an item is going to be put (a new CDF format).
CDF_FORMAT_	The item to be put. In this case it is the CDF's format.
format	The actual format for the CDF. Depending on the item being put, one or more arguments would have been necessary. In this case only one argument is necessary. format contains the value SINGLE_FILE.
NULL_	This argument could have been one of two things. It could have been another item to put (followed by the arguments required for that item) or it could have been a new operation to perform. In this case it is a new operation to perform — the NULL_ operation. NULL_ indicates the end of the call to CDFlib. Specifying NULL_ at the end of the argument list is required because not all compilers/operating systems provide the ability for a called function to determine how many arguments were passed in by the calling function.

The next example shows how the same CDF could have been created using only one call to CDFlib (the declarations would be the same).

```
.
.
status = CDFlib (CREATE_, CDF_, CDFname, numDims, dimSizes, &id,
                 PUT_, CDF_ENCODING_, encoding,
                       CDF_MAJORITY_, majority,
                       CDF_FORMAT_, format,
                 NULL_);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

The purpose of each argument is as follows.

CREATE_	The first operation to be performed. In this case something will be created.
CDF_	The item to be created — a CDF in this case. There are four required arguments which must follow. When a CDF is created (with CDFlib), the format defaults to multi-file (MULTI_FILE), the data encoding defaults to host (HOST_ENCODING), and the variable majority default to row major (ROW_MAJOR).
CDFname	The name of the CDF.
numDims	The number of dimensions in the CDF.
dimSizes	The dimension sizes.

`id`	The identifier to be used when referencing the created CDF in subsequent operations.
`PUT_`	This argument could have been one of two things. Another item to create or a new operation to perform. In this case it is another operation to perform — something will be put.
`CDF_ENCODING_`	The item to be put — in this case the CDF's encoding. Note that the CDF did not have to be selected. It was automatically selected as the current CDF when it was created.
`encoding`	The encoding to be put to the CDF.
`CDF_MAJORITY_`	This argument could have been one of two things. Another item to put or a new operation to perform. In this case it is another item to put — the CDF's majority.
`majority`	The majority to be put to the CDF.
`CDF_FORMAT_`	Once again this argument could have been either another item to put or a new operation to perform. It is another item to put — the CDF's format.
`format`	The format to be put to the CDF.
`NULL_`	This argument could have been either another item to put or a new operation to perform. Here it is another operation to perform — the `NULL_` operation which ends the call to `CDFlib`.

Note that the operations are performed in the order that they appear in the argument list. The CDF had to be created before the encoding, majority, and format could be specified (put).

Current Objects/States (Items)

The use of `CDFlib` requires that an application be aware of the current objects/states maintained by the CDF library. The following current objects/states are used by the CDF library when performing operations...

CDF	A CDF operation is always performed on the current CDF. The current CDF is implicitly selected whenever a CDF is opened or created. The current CDF may be explicitly selected using the `<SELECT_,CDF_>`[5] operation. There is no current CDF until one is opened or created (which implicitly selects it) or until one is explicitly selected. **NOTE:** The current CDF is not maintained by the CDF library from one call to the next of `CDFlib`. The current CDF must be selected in each call to `CDFlib` (if an operation is to be performed on a CDF).
rVariable	An rVariable operation is always performed on the current rVariable in the current CDF. For each open CDF a current rVariable is maintained. This current rVariable is implicitly

[5] This notation is used to specify an operation to be performed on an item. The syntax is: `<operation,item>`.

	selected when an rVariable is created (in the current CDF) or it may be explicitly selected with the <SELECT_,rVAR_> or <SELECT_,rVAR_NAME_> operations. There is no current rVariable in a CDF until one is created (which implicitly selects it) or until one is explicitly selected.
zVariable	A zVariable operation is always performed on the current zVariable in the current CDF. For each open CDF a current zVariable is maintained. This current zVariable is implicitly selected when a zVariable is created (in the current CDF) or it may be explicitly selected with the <SELECT_,zVAR_> or <SELECT_,zVAR_NAME_> operations. There is no current zVariable in a CDF until one is created (which implicitly selects it) or until one is explicitly selected.
attribute	An attribute operation is always performed on the current attribute in the current CDF. For each open CDF a current attribute is maintained. This current attribute is implicitly selected when an attribute is created (in the current CDF) or it may be explicitly selected with the <SELECT_,ATTR_> or <SELECT_,ATTR_NAME_> operations. There is no current attribute in a CDF until one is created (which implicitly selects it) or until one is explicitly selected.
gEntry/rEntry number	An attribute gEntry/rEntry operation is always performed on the current gEntry/rEntry number in the current CDF for the current attribute in that CDF. The current gEntry/rEntry number has a dual purpose. For gAttributes it is the gEntry number at which to perform an operation. For vAttributes it is the rEntry number at which to perform an operation. For each open CDF a current gEntry/rEntry number is maintained. This current gEntry/rEntry number must be explicitly selected with the <SELECT_,gENTRY_> operation (for gAttributes) or <SELECT_,rENTRY_ operation (for vAttributes) — there is no implicit or default selection of the current gEntry/rEntry number for a CDF. Note that the current gEntry/rEntry number is maintained for the CDF (not each attribute) — it applies to all of the attributes in that CDF.
zEntry number	A vAttribute zEntry operation is always performed on the current zEntry number in the current CDF for the current attribute in that CDF. For each open CDF a current zEntry number is maintained. This current zEntry number must be explicitly selected with the <SELECT_,zENTRY_> operation (there is no implicit or default selection of the current zEntry number for a CDF). Note that the current zEntry number is maintained for the CDF (not each attribute) — it applies to all of the attributes in that CDF.
record number, rVariables	An rVariable read or write operation is always performed at (for single reads and writes) or starting at (for hyper reads and writes) the current record number for the rVariables in the current CDF. When a CDF is opened or created the current record

number for its rVariables is initialized to zero (0). It may then be explicitly selected using the <SELECT_,rVARs_RECNUMBER_> operation. Note that the current record number for rVariables is maintained for a CDF (not each rVariable) — it applies to all of the rVariables in that CDF.

record count, rVariables

An rVariable hyper read or write operation is always performed using the current record count for the rVariables in the current CDF. When a CDF is opened or created the current record count for its rVariables is initialized to one (1). It may then be explicitly selected using the <SELECT_,rVARs_RECCOUNT_> operation. Note that the current record count for rVariables is maintained for a CDF (not each rVariable) — it applies to all of the rVariables in that CDF.

record interval, rVariables

An rVariable hyper read or write operation is always performed using the current record interval for the rVariables in the current CDF. When a CDF is opened or created the current record interval for its rVariables is initialized to one (1). It may then be explicitly selected using the <SELECT_,rVARs_RECINTERVAL_> operation. Note that the current record interval for rVariables is maintained for a CDF (not each rVariable) — it applies to all of the rVariables in that CDF.

dimension indices, rVariables

An rVariable read or write operation is always performed at (for single reads and writes) or starting at (for hyper reads and writes) the current dimension indices for the rVariables in the current CDF. When a CDF is opened or created the current dimension indices for its rVariables are initialized to zeroes (0,0,...). They may then be explicitly selected using the <SELECT_,rVARs_DIMINDICES_> operation. Note that the current dimension indices for rVariables are maintained for a CDF (not each rVariable) — they apply to all of the rVariables in that CDF. For 0-dimensional rVariables the current dimension indices are not applicable.

dimension counts, rVariables

An rVariable hyper read or write operation is always performed using the current dimension counts for the rVariables in the current CDF. When a CDF is opened or created the current dimension counts for its rVariables are initialized to the dimension sizes of the rVariables (which specifies the entire array). They may then be explicitly selected using the <SELECT_,rVARs_DIMCOUNTS_> operation. Note that the current dimension counts for rVariables are maintained for a CDF (not each rVariable) — they apply to all of the rVariables in that CDF. For 0-dimensional rVariables the current dimension counts are not applicable.

dimension intervals, rVariables

An rVariable hyper read or write operation is always performed using the current dimension intervals for the rVariables in the current CDF. When a CDF is opened or created the current dimension intervals for its rVariables are initial-

ized to ones (1,1,...). They may then be explicitly selected using the <SELECT_,rVARs_DIMINTERVALS_> operation. Note that the current dimension intervals for rVariables are maintained for a CDF (not each rVariable) — they apply to all of the rVariables in that CDF. For 0-dimensional rVariables the current dimension intervals are not applicable.

record number, zVariable

A zVariable read or write operation is always performed at (for single reads and writes) or starting at (for hyper reads and writes) the current record number for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened) the current record number for that zVariable is initialized to zero (0). It may then be explicitly selected using the <SELECT_,zVAR_RECNUMBER_> operation (which only affects the current zVariable in the current CDF). Note that a current record number is maintained for each zVariable in a CDF.

record count, zVariable

A zVariable hyper read or write operation is always performed using the current record count for the current zVariable in the current CDF. When a zVariable created (or for each zVariable in a CDF being opened) the current record count for that zVariable is initialized to one (1). It may then be explicitly selected using the <SELECT_,zVAR_RECCOUNT_> operation (which only affects the current zVariable in the current CDF). Note that a current record count is maintained for each zVariable in a CDF.

record interval, zVariable

A zVariable hyper read or write operation is always performed using the current record interval for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened) the current record interval for that zVariable is initialized to one (1). It may then be explicitly selected using the <SELECT_,zVAR_RECINTERVAL_> operation (which only affects the current zVariable in the current CDF). Note that a current record interval is maintained for each zVariable in a CDF.

dimension indices, zVariable

A zVariable read or write operation is always performed at (for single reads and writes) or starting at (for hyper reads and writes) the current dimension indices for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened) the current dimension indices for that zVariable are initialized to zeroes (0,0,...). They may then be explicitly selected using the <SELECT_,zVAR_DIMINDICES_> operation (which only affects the current zVariable in the current CDF). Note that current dimension indices are maintained for each zVariable in a CDF. For 0-dimensional zVariables the current dimension indices are not applicable.

dimension counts, zVariable

A zVariable hyper read or write operation is always performed using the current dimension counts for the current

	zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened) the current dimension counts for that zVariable are initialized to the dimension sizes of that zVariable (which specifies the entire array). They may then be explicitly selected using the `<SELECT_,zVAR_DIMCOUNTS_>` operation (which only affects the current zVariable in the current CDF). Note that current dimension counts are maintained for each zVariable in a CDF. For 0-dimensional zVariables the current dimension counts are not applicable.
dimension intervals, zVariable	A zVariable hyper read or write operation is always performed using the current dimension intervals for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened) the current dimension intervals for that zVariable are initialized to ones (1,1,...). They may then be explicitly selected using the `<SELECT_,zVAR_DIMINTERVALS_>` operation (which only affects the current zVariable in the current CDF). Note that current dimension intervals are maintained for each zVariable in a CDF. For 0-dimensional zVariables the current dimension intervals are not applicable.
status code	When inquiring the explanation of a CDF status code, the text returned is always for the current status code. One current status code is maintained for the entire CDF library (regardless of the number of open CDFs). The current status code may be selected using the `<SELECT_,CDF_STATUS_>` operation. There is no default current status code. Note that the current status code is NOT the status code from the last operation performed). **NOTE:** The current status code is not maintained by the CDF library from one call to the next of `CDFlib`.

Returned Status

`CDFlib` returns a status code of type `CDFstatus`. Since more than one operation may be performed with a single call to `CDFlib`, the following rules apply.

1. The first error detected aborts the call to `CDFlib` and the corresponding status code is returned.
2. In the absence of any errors, the status code for the last warning detected is returned.
3. In the absence of any errors or warnings, the status code for the last informational condition is returned.
4. In the absence of any errors, warnings, or informational conditions, `CDF_OK` is returned.

Section 4.8 explains how to interpret status codes and Appendix C lists the possible status codes.

Indentation/Style

Indentation should be used to make calls to `CDFlib` readable. The following example shows a call to `CDFlib` using proper indentation.

```
status = CDFlib (CREATE_, CDF_, CDFname, numDims, dimSizes, &id,
                 PUT_, CDF_FORMAT_, format,
                       CDF_MAJORITY_, majority,
                 CREATE_, ATTR_, attrName, scope, &attrNum,
                          rVAR_, varName, dataType, numElements,
                                 recVary, dimVarys, &varNum,
                 NULL_);
```

Note that the operations (`CREATE_`, `PUT_`, and `NULL_`) are indented the same and that the items (`CDF_`, `CDF_FORMAT_`, `CDF_MAJORITY_`, `ATTR_`, and `rVAR_`) are indented the same under their corresponding operations.

The following example shows the same call to `CDFlib` without the proper indentation.

```
status = CDFlib (CREATE_, CDF_, CDFname, numDims, dimSizes, &id, PUT_,
                 CDF_FORMAT_, format, CDF_MAJORITY_, majority, CREATE_,
                 ATTR_, attrName, scope, &attrNum, rVAR_, varName, dataType,
                 numElements, recVary, dimVarys, &varNum, NULL_);
```

The need for proper indentation to ensure the readability of your applications should be obvious.[6]

Syntax

`CDFlib` takes a variable number of arguments. There must always be at least one argument. The maximum number of arguments is not limited by CDF but rather the C compiler and operating system being used. Under normal circumstances that limit would never be reached (or even approached). Note also that a call to `CDFlib` with a large number of arguments can always be broken up into two or more calls to `CDFlib` with fewer arguments.

Operations/items/arguments...

The syntax for `CDFlib` is as follows...

```
status = CDFlib (op1, item1, arg1, arg2, ...argN,
                      item2, arg1, arg2, ...argN,
                      .
                      .
                      itemN, arg1, arg2, ...argN,
                 op2, item1, arg1, arg2, ...argN,
```

[6] If it isn't, call us, we can help you.

```
                    item2, arg1, arg2, ...argN,
                    .
                    .
                    itemN, arg1, arg2, ...argN,
               .
               .
               .
               opN, item1, arg1, arg2, ...argN,
                    item2, arg1, arg2, ...argN,
                    .
                    .
                    itemN, arg1, arg2, ...argN,
               NULL_);
```

where opx is an operation to perform, itemx is the item on which to perform the operation, and argx is a required argument for the operation. The NULL_ operation must be used to end the call to CDFlib. The completion status is returned by CDFlib.

The supported operations are as follows.

CREATE_	Used to create an item.
OPEN_	Used to open an item.
DELETE_	Used to delete an item.
CLOSE_	Used to close an item.
SELECT_	Used to select the value of an item.
CONFIRM_	Used to confirm the value of an item.
PUT_	Used to put (write) something to an item.
GET_	Used to get (read) something from an item.

For each operation, the supported items and required arguments are as follows.

CREATE_ The following items may be created.

CDF_ A new CDF will be created. It is illegal to create a CDF that already exists. The created CDF implicitly becomes the current CDF. Required arguments are...

in: char *CDFname

Pathname of the CDF to be created. This can be at most CDF_PATHNAME_LEN characters (excluding the NUL).

in: long numDims

Number of dimensions for the rVariables. This can be as few as zero (0) and at most CDF_MAX_DIMS.

in: long dimSizes[]

Dimension sizes for the rVariables. dimSizes is a 1-dimensional array containing numDims elements. Each element of dimSizes specifies the corresponding dimension size. Each dimension

size must be greater than zero (0). For 0-dimensional rVaribles this argument is ignored (but a place holder is necessary).

`out: CDFid *id`

CDF identifier to be used in subsequent operations on the CDF.

A CDF is created with a default format of multi-file (`MULTI_FILE`), a default data encoding of host (`HOST_ENCODING`), and a default variable majority of row major (`ROW_MAJOR`). These defaults can be changed with the corresponding `PUT_` operations (of `CDF_FORMAT_`, `CDF_ENCODING_` and `CDF_MAJORITY_` respectively).

A CDF must be closed with `<CLOSE_,CDF_>` to ensure that the CDF will be correctly written to disk.

`rVAR_` A new rVariable will be created in the current CDF. A variable (rVariable or zVariable) with the same name must not already exist in the CDF. The created rVariable implicitly becomes the current rVariable (in the current CDF). Required arguments are...

`in: char *varName`

Name of the rVariable to be created. This can be at most `CDF_VAR_NAME_LEN` characters (excluding the NUL). Variable names are case-sensitive.

`in: long dataType`

Data type of the new rVariable. Specify one of the data types described in Section 4.6.4.

`in: long numElements`

Number of elements of the data type at each value. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in each string (an array of characters). A string exists at each value. For the non-character string data types this must be one (1) — arrays of values are not allowed for non-character string data types.

`in: long recVary`

Record variance. Specify one of the variances described in Section 4.6.7.

`in: long dimVarys[]`

Dimension variances. `dimVarys` is a 1-dimensional array containg one element per rVarible dimension. Each element of `dimVarys` specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 4.6.7. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).

`out: long *varNum`

Number assigned to the new rVariable. This number must be used in subsequent CDF function calls when referring to this rVariable. An existing rVariable's number may also be

determined with the <GET_,rVAR_NUMBER_> operation.

zVAR_ A new zVariable will be created in the current CDF. A variable (rVariable or zVariable) with the same name must not already exist in the CDF. The created zVariable implicitly becomes the current zVariable (in the current CDF). Required arguments are...

in: char *varName

Name of the zVariable to be created. This can be at most CDF_VAR_NAME_LEN characters (excluding the NUL). Variable names are case-sensitive.

in: long dataType

Data type of the new zVariable. Specify one of the data types described in Section 4.6.4.

in: long numElements

Number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in each string (an array of characters). A string exists at each value. For the non-character string data types this must be one (1) — arrays of values are not allowed for non-character string data types.

in: long numDims

Number of dimensions for the zVariable. This may be as few as zero and at most CDF_MAX_DIMS.

in: long dimSizes[]

The dimension sizes. dimSizes is a 1-dimensional array containing numDims elements. Each element of dimSizes specifies the corresponding dimension size. Each dimension size must be greater than zero (0). For a 0-dimensional zVariable this argument is ignored (but a place holder is necessary).

in: long recVary

Record variance. Specify one of the variances described in Section 4.6.7.

in: long dimVarys[]

Dimension variances. dimVarys is a 1-dimensional array containg one element per dimension. Each element of dimVarys specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 4.6.7. For a 0-dimensional zVariable this argument is ignored (but a place holder is necessary).

out: long *varNum

Number assigned to the new zVariable. This number must be used in subsequent CDF function calls when referring to this zVariable. An existing zVariable's number may also be determined with the <GET_,zVAR_NUMBER_> operation.

ATTR_ A new attribute will be created in the current CDF. An attribute with

the same name must not already exist in the CDF. The created attribute implicitly becomes the current attribute (in the current CDF). Required arguments are...

in: char *attrName

Name of the attribute to be created. This can be at most CDF_ATTR_NAME_LEN characters (excluding the NUL). Attribute names are case-sensitive.

in: long scope

Scope of the new attribute. Specify one of the scopes described in Section 4.6.8.

out: long *attrNum

Number assigned to the new attribute. This number must be used in subsequent CDF function calls when referring to this attribute. An existing attribute's number may also be determined with the <GET_,ATTR_NUMBER_> operation.

OPEN_ The follow item(s) may be opened.

CDF_ Opens the named CDF. The opened CDF implicitly becomes the current CDF. Required arguments are...

in: char *CDFname

Pathname of the CDF to be opened. This can be at most CDF_PATHNAME_LEN characters (excluding the NUL).

out: CDFid *id

CDF identifier to be used in subsequent operations on the CDF.

DELETE_ The following item(s) may be deleted.

CDF_ Deletes the current CDF. A CDF must be opened before it can be deleted. When a CDF is deleted, there is no longer a current CDF. There are no arguments.

CLOSE_ The following items may be closed.

CDF_ Closes the current CDF. When a CDF is closed, there is no longer a current CDF. A CDF must be closed to ensure that it will be properly written to disk. There are no arguments.

rVAR_ Closes the current rVariable (in the current CDF). There are no arguments.

zVAR_ Closes the current zVariable (in the current CDF). There are no arguments.

SELECT_ The following items may be selected.

CDF_ Explicitly selects the current CDF. Required arguments are...

in: CDFid id

Identifier of the CDF. This identifier must have been initialized by a successful <CREATE_,CDF_> or <OPEN_,CDF_> operation.

CDF_STATUS_ Selects the current status code. Required arguments are...

in: CDFstatus status

CDF status code.

rVAR_ Explicitly selects the current rVariable (in the current CDF) by number. Required arguments are...

in: long varNum

rVariable number.

rVAR_NAME_ Explicitly selects the current rVariable (in the current CDF) by name. Required arguments are...

in: char *varName

rVariable name. This may be at most CDF_VAR_NAME_LEN characters (excluding the NUL).

rVARs_RECNUMBER_ Selects the current record number for all rVariables in the current CDF. Required arguments are...

in: long recNum

Record number.

rVARs_RECCOUNT_ Selects the current record count for all rVariables in the current CDF. Required arguments are...

in: long recCount

Record count.

rVARs_RECINTERVAL_ Selects the current record interval for all rVariables in the current CDF. Required arguments are...

in: long recInterval

Record interval.

rVARs_DIMINDICES_ Selects the current dimension indices for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

in: long indices[]

Dimension indices. indices is a 1-dimensional array containing one element per dimension. Each element of indices specifies the corresponding dimension index.

rVARs_DIMCOUNTS_ Selects the current dimension counts for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

in: `long counts[]`

Dimension counts. `counts` is a 1-dimensional array containing one element per dimension. Each element of `counts` specifies the corresponding dimension count.

`rVARs_DIMINTERVALS_` Selects the current dimension intervals for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

in: `long intervals[]`

Dimension intervals. `intervals` is a 1-dimensional array containing one element per dimension. Each element of `intervals` specifies the corresponding dimension interval.

`rVAR_SEQPOS_` Selects the current position for sequential access for the current rVariable (in the current CDF). Note that a current position is maintained for each rVariable individually. Required arguments are...

in: `long recNum`

Record number.

in: `long indices[]`

Dimension indices. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` specifies the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).

`zVAR_` Explicitly selects the current zVariable (in the current CDF) by number. Required arguments are...

in: `long varNum`

zVariable number.

`zVAR_NAME_` Explicitly selects the current zVariable (in the current CDF) by name. Required arguments are...

in: `char *varName`

zVariable name. This may be at most `CDF_VAR_NAME_LEN` characters (excluding the NUL).

`zVAR_RECNUMBER_` Selects the current record number for the current zVariable in the current CDF. Required arguments are...

in: `long recNum`

Record number.

`zVAR_RECCOUNT_` Selects the current record count for the current zVariable in the current CDF. Required arguments are...

in: `long recCount`

Record count.

`zVAR_RECINTERVAL_` Selects the current record interval for the current zVariable in the current CDF. Required arguments are...

`in: long recInterval`

Record interval.

`zVAR_DIMINDICES_` Selects the current dimension indices for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`in: long indices[]`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` specifies the corresponding dimension index.

`zVAR_DIMCOUNTS_` Selects the current dimension counts for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`in: long counts[]`

Dimension counts. `counts` is a 1-dimensional array containing one element per dimension. Each element of `counts` specifies the corresponding dimension count.

`zVAR_DIMINTERVALS_` Selects the current dimension intervals for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`in: long intervals[]`

Dimension intervals. `intervals` is a 1-dimensional array containing one element per dimension. Each element of `intervals` specifies the corresponding dimension interval.

`zVAR_SEQPOS_` Selects the current position for sequential access for the current zVariable (in the current CDF). Note that a current position is maintained for each zVariable individually. Required arguments are...

`in: long recNum`

Record number.

`in: long indices[]`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` specifies the corresponding dimension index. For 0-dimensional zVariables this argument is ignored (but a place holder is necessary).

`ATTR_` Explicitly selects the current attribute (in the current CDF) by number. Required arguments are...

`in: long attrNum`

Attribute number.

ATTR_NAME_ Explicitly selects the current attribute (in the current CDF) by name. Required arguments are...

in: char *attrName

Attribute name. This may be at most CDF_ATTR_NAME_LEN characters (excluding the NUL).

gENTRY_ Selects the current gEntry/rEntry number for all attributes in the current CDF. **NOTE:** Only use this operation when dealing with gAttributes. Required arguments are...

in: long entryNum

gEntry/rEntry number.

rENTRY_ Selects the current gEntry/rEntry number for all attributes in the current CDF. **NOTE:** Only use this operation when dealing with vAttributes. Required arguments are...

in: long entryNum

gEntry/rEntry number.

zENTRY_ Selects the current zEntry number for all attributes in the current CDF. **NOTE:** Only use this operation when dealing with vAttributes. Required arguments are...

in: long entryNum

zEntry number.

CONFIRM_ The following items may be confirmed.

CDF_NAME_ Confirms the pathname of the current CDF. Required arguments are...

out: char *CDFname

Pathname of the CDF. This character string must be large enough to hold CDF_PATHNAME_LEN + 1 characters (including the NUL terminator).

rVAR_ Confirms the current rVariable (in the current CDF). Required arguments are...

out: long *varNum

rVariable number.

rVARs_RECNUMBER_ Confirms the current record number for all rVariables in the current CDF. Required arguments are...

out: long *recNum

Record number.

rVARs_RECCOUNT_ Confirms the current record count for all rVariables in the current CDF. Required arguments are...

`out: long *recCount`

Record count.

`rVARs_RECINTERVAL_` Confirms the current record interval for all rVariables in the current CDF. Required arguments are...

`out: long *recInterval`

Record interval.

`rVARs_DIMINDICES_` Confirms the current dimension indices for all rVariables in the current CDF. For 0-dimensional rVarialbes this operation is not applicable. Required arguments are...

`out: long indices[]`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` receives the corresponding dimension index.

`rVARs_DIMCOUNTS_` Confirms the current dimension counts for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

`out: long counts[]`

Dimension counts. `counts` is a 1-dimensional array containing one element per dimension. Each element of `counts` receives the corresponding dimension count.

`rVARs_DIMINTERVALS_` Confirms the current dimension intervals for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

`out: long intervals[]`

Dimension intervals. `intervals` is a 1-dimensional array containing one element per dimension. Each element of `intervals` receives the corresponding dimension interval.

`rVAR_SEQPOS_` Confirms the current position for sequential access for the current rVariable (in the current CDF). Note that a current position is maintained for each rVariable individually. Required arguments are...

`out: long *recNum`

Record number.

`out: long indices[]`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` receives the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).

`zVAR_` Confirms the current zVariable (in the current CDF). Required arguments are...

`out: long *varNum`

zVariable number.

`zVAR_RECNUMBER_` Confirms the current record number for the current zVariable in the current CDF. Required arguments are...

`out: long *recNum`

Record number.

`zVAR_RECCOUNT_` Confirms the current record count for the current zVariable in the current CDF. Required arguments are...

`out: long *recCount`

Record count.

`zVAR_RECINTERVAL_` Confirms the current record interval for the current zVariable in the current CDF. Required arguments are...

`out: long *recInterval`

Record interval.

`zVAR_DIMINDICES_` Confirms the current dimension indices for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`out: long indices[]`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` receives the corresponding dimension index.

`zVAR_DIMCOUNTS_` Confirms the current dimension counts for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`out: long counts[]`

Dimension counts. `counts` is a 1-dimensional array containing one element per dimension. Each element of `counts` receives the corresponding dimension count.

`zVAR_DIMINTERVALS_` Confirms the current dimension intervals for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`out: long intervals[]`

Dimension intervals. `intervals` is a 1-dimensional array containing one element per dimension. Each element of `intervals` receives the corresponding dimension interval.

`zVAR_SEQPOS_` Confirms the current position for sequential access for the current zVariable (in the current CDF). Note that a current position is maintained for each zVariable individually. Required arguments are...

`out: long *recNum`

Record number.

`out: long indices[]`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` receives the corresponding dimension index. For 0-dimensional zVariables this argument is ignored (but a place holder is necessary).

`ATTR_` Confirms the current attribute (in the current CDF). Required arguments are...

`out: long *attrNum`

Attribute number.

`gENTRY_` Confirms the current gEntry/rEntry number for all attributes in the current CDF. **NOTE:** Only use this operation when dealing with gAttributes. Required arguments are...

`out: long *entryNum`

gEntry/rEntry number.

`rENTRY_` Confirms the current gEntry/rEntry number for all attributes in the current CDF. **NOTE:** Only use this operation when dealing with vAttributes. Required arguments are...

`out: long *entryNum`

gEntry/rEntry number.

`zENTRY_` Confirms the current zEntry number for all attributes in the current CDF. **NOTE:** Only use this operation when dealing with vAttributes. Required arguments are...

`out: long *entryNum`

zEntry number.

`PUT_` The following items may be written (put).

`CDF_ENCODING_` Respecifies the data encoding of the current CDF. A CDF's data encoding may not be changed after any variable values (including the pad value) or attribute entries have been written. Required arguments are...

`in: long encoding`

New data encoding. Specify one of the encodings described in Section 4.6.5.

`CDF_MAJORITY_` Respecifies the variable majority of the current CDF. A CDF's variable majority may not be changed after any variable values have been written. Required arguments are...

in: long majority

New variable majority. Specify one of the majorities described in Section 4.6.6.

CDF_FORMAT_ Respecifies the format of the current CDF. A CDF's format may not be changed after any variables have been created. Required arguments are...

in: long format

New CDF format. Specify one of the formats described in Section 4.6.3.

rVAR_NAME_ Renames the current rVariable (in the current CDF). A variable (rVariable or zVariable) with the same name must not already exist in the CDF. Required arguments are...

in: char *varName

New name of the rVariable. This may consist of at most CDF_VAR_NAME_LEN characters (excluding the NUL).

rVAR_DATASPEC_ Respecifies the data specification (data type and number of elements) of the current rVariable (in the current CDF). An rVariable's data specification may not be changed if the new data specification is not equivalent to the old data specification and any values (including the pad value) have been written. Data specifications are considered equivalent if the data types are equivalent (see Section 2.5.5) and the number of elements are the same. Required arguments are...

in: long dataType

New data type. Specify one of the data types described in Section 4.6.4.

in: long numElements

Number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in each string (an array of characters). A string exists at each value. For the non-character string data types this must be one (1) — arrays of values are not allowed for non-character string data types.

rVAR_RECVARY_ Respecifies the record variance of the current rVariable (in the current CDF). An rVariable's record variance may not be changed if any values have been written (except for an explicit pad value — it may have been written). Required arguments are...

in: long recVary

New record variance. Specify one of the variances described in Section 4.6.7.

rVAR_DIMVARYS_ Respecifies the dimension variances of the current rVariable (in the current CDF). An rVariable's dimension variances may not be changed if any values have been written (except for an explicit pad value — it

may have been written). For 0-dimensional rVariables this operation is not applicable. Required arguments are...

in: `long dimVarys[]`

New dimension variances. `dimVarys` is a 1-dimensional array containing one element per dimension. Each element of `dimVarys` specifies the corresponding dimension variance. For each dimension, specify one of the variances described in Section 4.6.7.

`rVAR_PADVALUE_` Specifies the pad value for the current rVariable (in the current CDF). An rVariable's pad value may be specified (or respecified) at any time without affecting already written values (including where pad values were used). Section 2.7.12 describes variable pad values. Required arguments are...

in: `void *value`

Pad value. The pad value is written to the CDF from memory address `value`.

`rVAR_INITIALRECS_` Specifies the number of records to initially write to the current rVariable (in the current CDF). This may be specified only once per rVariable and before any other records have been written to that rVariable. If a pad value has not yet been specified, the default is used (see Section 2.7.12). If a pad value has been explicitly specified, that value is written to the records. Section 2.7.5 describes initial records. Required arguments are...

in: `long NinitialRecs`

Number of records to write.

`rVAR_EXTENDRECS_` Specifies the number of physical records to extend the current rVariable (in the current CDF) whenever that rVariable needs to be extended. Section 2.7.5 describes extend records. Required arguments are...

in: `long NextendRecs`

Number of records to extend.

`rVAR_DATA_` Writes one value to the current rVariable (in the current CDF). The value is written at the current record number and current dimension indices for the rVariables (in the current CDF). Required arguments are...

in: `void *value`

Value. The value is written to the CDF from memory address `value`.

`rVAR_HYPERDATA_` Writes one or more values to the current rVariable (in the current CDF). The values are written based on the current record number, current record count, current record interval, current dimension in-

dices, current dimension counts, and current dimension intervals for the rVariables (in the current CDF). Required arguments are...

in: void *buffer

Values. The values starting at memory address buffer are written to the CDF.

rVAR_SEQDATA_ Writes one value to the current rVariable (in the current CDF) at the current sequential position for that rVariable. After the write, the sequential position is automatically incremented to the next value (crossing a record boundary if necessary). If the sequential position is past the last record for the rVariable, the rVariable is extended as necessary. Required arguments are...

in: void *value

Value. The value is written to the CDF from memory address value.

zVAR_NAME_ Renames the current zVariable (in the current CDF). A variable (rVariable or zVariable) with the same name must not already exist in the CDF. Required arguments are...

in: char *varName

New name of the zVariable. This may consist of at most CDF_VAR_NAME_LEN characters (excluding the NUL).

zVAR_DATASPEC_ Respecifies the data specification (data type and number of elements) of the current zVariable (in the current CDF). A zVariable's data specification may not be changed if the new data specification is not equivalent to the old data specification and any values (including the pad value) have been written. Data specifications are considered equivalent if the data types are equivalent (see Section 2.5.5) and the number of elements are the same. Required arguments are...

in: long dataType

New data type. Specify one of the data types described in Section 4.6.4.

in: long numElements

Number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in each string (an array of characters). A string exists at each value. For the non-character string data types this must be one (1) — arrays of values are not allowed for non-character string data types.

zVAR_RECVARY_ Respecifies the record variance of the current zVariable (in the current CDF). A zVariable's record variance may not be changed if any values have been written (except for an explicit pad value — it may have been written). Required arguments are...

in: long recVary

New record variance. Specify one of the variances described in Section 4.6.7.

`zVAR_DIMVARYS_` Respecifies the dimension variances of the current zVariable (in the current CDF). A zVariable's dimension variances may not be changed if any values have been written (except for an explicit pad value — it may have been written). For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`in: long dimVarys[]`

New dimension variances. `dimVarys` is a 1-dimensional array containing one element per dimension. Each element of `dimVarys` specifies the corresponding dimension variance. For each dimension, specify one of the variances described in Section 4.6.7.

`zVAR_PADVALUE_` Specifies the pad value for the current zVariable (in the current CDF). A zVariable's pad value may be specified (or respecified) at any time without affecting already written values (including where pad values were used). Section 2.7.12 describes variable pad values. Required arguments are...

`in: void *value`

Pad value. The pad value is written to the CDF from memory address `value`.

`zVAR_INITIALRECS_` Specifies the number of records to initially write to the current zVariable (in the current CDF). This may be specified only once per zVariable and before any other records have been written to that zVariable. If a pad value has not yet been specified, the default is used (see Section 2.7.12). If a pad value has been explicitly specified, that value is written to the records. Section 2.7.5 describes initial records. Required arguments are...

`in: long NinitialRecs`

Number of records to write.

`zVAR_EXTENDRECS_` Specifies the number of physical records to extend the current zVariable (in the current CDF) whenever that zVariable needs to be extended. Section 2.7.5 describes extend records. Required arguments are...

`in: long NextendRecs`

Number of records to extend.

`zVAR_DATA_` Writes one value to the current zVariable (in the current CDF). The value is written at the current record number and current dimension indices for that zVariable (in the current CDF). Required arguments are...

`in: void *value`

Value. The value is written to the CDF from memory address `value`.

`zVAR_HYPERDATA_` Writes one or more values to the current zVariable (in the current CDF). The values are written based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals for that zVariable (in the current CDF). Required arguments are...

`in: void *buffer`

Values. The values starting at memory address `buffer` are written to the CDF.

`zVAR_SEQDATA_` Writes one value to the current zVariable (in the current CDF) at the current sequential position for that zVariable. After the write, the sequential position is automatically incremented to the next value (crossing a record boundary if necessary). If the sequential position is past the last record for the zVariable, the zVariable is extended as necessary. Required arguments are...

`in: void *value`

Value. The value is written to the CDF from memory address `value`.

`ATTR_SCOPE_` Respecifies the scope for the current attribute (in the current CDF). Required arguments are...

`in: long scope`

New attribute scope. Specify one of the scopes described in Section 4.6.8.

`ATTR_NAME_` Renames the current attribute (in the current CDF). An attribute with the same name must not already exist in the CDF. Required arguments are...

`in: char *attrName`

New attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters (excluding the NUL).

`gENTRY_DATA_` Writes a gEntry to the current attribute at the current gEntry/rEntry number (in the current CDF). An existing gEntry may be overwritten with a new gEntry having the same data specification (data type and number of elements) or a different data specification. **NOTE:** This operation should only be performed on gAttributes. Required arguments are...

`in: long dataType`

Data type of the gEntry. Specify one of the data types described in Section 4.6.4.

`in: long numElements`

Number of elements of the data type. This may be greater than one (1) for any of the supported data types. For char-

acter string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (an array of characters). For all other data types, this is the number of elements in an array of that data type.

in: void *value

Value(s). The entry value is written to the CDF from memory address value.

rENTRY_DATA_ Writes a rEntry to the current attribute at the current gEntry/rEntry number (in the current CDF). An existing rEntry may be overwritten with a new rEntry having the same data specification (data type and number of elements) or a different data specification. **NOTE:** This operation should only be performed on vAttributes. Required arguments are...

in: long dataType

Data type of the rEntry. Specify one of the data types described in Section 4.6.4.

in: long numElements

Number of elements of the data type. This may be greater than one (1) for any of the supported data types. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (an array of characters). For all other data types, this is the number of elements in an array of that data type.

in: void *value

Value(s). The entry value is written to the CDF from memory address value.

zENTRY_DATA_ Writes a zEntry to the current attribute at the current zEntry number (in the current CDF). An existing zEntry may be overwritten with a new zEntry having the same data specification (data type and number of elements) or a different data specification. **NOTE:** This operation should only be performed on vAttributes. Required arguments are...

in: long dataType

Data type of the zEntry. Specify one of the data types described in Section 4.6.4.

in: long numElements

Number of elements of the data type. This may be greater than one (1) for any of the supported data types. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (an array of characters). For all other data types, this is the number of elements in an array of that data type.

in: void *value

Value(s). The entry value is written to the CDF from memory address value.

GET_ The following items may be read (gotten).

rVARs_NUMDIMS_ Inquires the number of dimensions for the rVariables in the current CDF. Required arguments are...

out: long *numDims
Number of dimensions.

rVARs_DIMSIZES_ Inquires the size of each dimension for the rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

out: long dimSizes[]
Dimension sizes. dimSizes is a 1-dimensional array containing one element per dimension. Each element of dimSizes receives the corresponding dimension size.

zVAR_NUMDIMS_ Inquires the number of dimensions for the current zVariable in the current CDF. Required arguments are...

out: long *numDims
Number of dimensions.

zVAR_DIMSIZES_ Inquires the size of each dimension for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

out: long dimSizes[]
Dimension sizes. dimSizes is a 1-dimensional array containing one element per dimension. Each element of dimSizes receives the corresponding dimension size.

CDF_ENCODING_ Inquires the data encoding of the current CDF. Required arguments are...

out: long *encoding
Data encoding. The encodings are described in Section 4.6.5.

CDF_MAJORITY_ Inquires the variable majority of the current CDF. Required arguments are...

out: long *majority
Variable majority. The majorities are described in Section 4.6.6.

CDF_FORMAT_ Inquires the format of the current CDF. Required arguments are...

out: long *format
CDF format. The formats are described in Section 4.6.3.

CDF_COPYRIGHT_ Reads the copyright notice for the CDF library which created the current CDF. Required arguments are...

`out: char *copyRight`

CDF copyright text. This character string must be large enough to hold `CDF_COPYRIGHT_LEN + 1` characters (including the NUL terminator).

`LIB_COPYRIGHT_` Reads the copyright notice of the CDF library being used. Required arguments are...

`out: char *copyRight`

CDF library copyright text. This character string must be large enough hold `CDF_COPYRIGHT_LEN + 1` characters (including the NUL terminator).

`CDF_NUMrVARS_` Inquires the number of rVariables in the current CDF. Required arguments are...

`out: long *numVars`

Number of rVariables.

`CDF_NUMzVARS_` Inquires the number of zVariables in the current CDF. Required arguments are...

`out: long *numVars`

Number of zVariables.

`CDF_NUMATTRS_` Inquires the number of attributes in the current CDF. Required arguments are...

`out: long *numAttrs`

Number of attributes.

`CDF_NUMgATTRS_` Inquires the number of gAttributes in the current CDF. Required arguments are...

`out: long *numAttrs`

Number of gAttributes.

`CDF_NUMvATTRS_` Inquires the number of vAttributes in the current CDF. Required arguments are...

`out: long *numAttrs`

Number of vAttributes.

`rVARs_MAXREC_` Inquires the maximum record number of the rVariables in the current CDF Note that this is not the number of records but rather the maximum record number (which is one less than the number of records). A value of negative one (-1) indicates that the rVariables contain no records. The maximum record number for an individual rVariable may be inquired using `<GET_,rVAR_MAXREC_>`. Required arguments are...

`out: long *maxRec`

Maximum record number.

`CDF_VERSION_` Inquires the version number of the CDF library which created the current CDF. Required arguments are...

`out: long *version`

Version number.

`CDF_RELEASE_` Inquires the release number of the CDF library which created the current CDF. Required arguments are...

`out: long *release`

Release number.

`CDF_INCREMENT_` Inquires the incremental number of the CDF library which created the current CDF. Required arguments are...

`out: long *increment`

Incremental number.

`LIB_VERSION_` Inquires the version number of the CDF library being used. Required arguments are...

`out: long *version`

Version number.

`LIB_RELEASE_` Inquires the release number of the CDF library being used. Required arguments are...

`out: long *release`

Release number.

`LIB_INCREMENT_` Inquires the incremental number of the CDF library being used. Required arguments are...

`out: long *increment`

Incremental number.

`LIB_subINCREMENT_` Inquires the subincremental character of the CDF library being used. Required arguments are...

`out: char *subincrement`

Subincremental character.

`rVAR_NAME_` Inquires the name of the current rVariable (in the current CDF). Required arguments are...

`out: char *varName`

Name of the rVariable. This character string must be large enough to hold `CDF_VAR_NAME_LEN + 1` characters (including the NUL).

`rVAR_DATATYPE_` Inquires the data type of the current rVariable (in the current CDF). Required arguments are...

`out: long *dataType`

Data type. The data types are described in Section 4.6.4.

`rVAR_NUMELEMS_` Inquires the number of elements (of the data type) for the current rVariable (in the current CDF). Required arguments are...

`out: long *numElements`

Number of elements of the data type at each value. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string (each value consists of the entire string). For all other data types, this will always be one (1) — multiple elements at each value are not allowed for non-character string data types.

`rVAR_RECVARY_` Inquires the record variance of the current rVariable (in the current CDF). Required arguments are...

`out: long *recVary`

Record variance. The variances are described in Section 4.6.7.

`rVAR_DIMVARYS_` Inquires the dimension variances of the current rVariable (in the current CDF). For 0-dimensional rVariables this operation is not applicable. Required arguments are...

`out: long dimVarys[]`

Dimension variances. `dimVarys` is a 1-dimensional array containing one element per dimension. Each element of `dimVarys` receives the corresponding dimension variance. The variances are described in Section 4.6.7.

`rVAR_NUMBER_` Gets the number of the named rVariable (in the current CDF). Required arguments are...

`in: char *varName`

The rVariable name. This may be at most `CDF_VAR_NAME_LEN` characters (excluding the NUL).

`out: long *varNum`

The rVariable number.

`rVAR_PADVALUE_` Inquires the pad value of the current rVariable (in the current CDF). Required arguments are...

`out: void *value`

Pad value. This buffer must be large enough to hold the pad value. The pad value is read from the CDF and placed in memory at address `value`.

`rVAR_EXTENDRECS_` Inquires the number of extend records for the current rVariable (in the current CDF). Extend records are described in Section 2.7.5. Required arguments are...

`out: long *NextendRecs`

Number of records to extend.

`rVAR_MAXREC_` Inquires the maximum record number for the current rVariable (in the current CDF). For rVariables with a record variance of `NOVARY`, this will be at most zero (0). A value of negative one (-1) indicates that no records have been written. Required arguments are...

`out: long *varMaxRec`

Maximum record number.

`rVAR_DATA_` Reads a value from the current rVariable (in the current CDF). The value is read at the current record number and current dimension indices for the rVariables (in the current CDF). Required arguments are...

`out: void *value`

Value. This buffer must be large enough to hold the value. The value is read from the CDF and placed into memory at address `value`.

`rVAR_HYPERDATA_` Reads one or more values from the current rVariable (in the current CDF). The values are read based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals for the rVariables (in the current CDF). Required arguments are...

`out: void *buffer`

Values. This buffer must be large enough to hold the values. The values are read from the CDF and placed into memory starting at address `buffer`.

`rVAR_SEQDATA_` Reads one value from the current rVariable (in the current CDF) at the current sequential position for that rVariable. After the read, the sequential position is automatically incremented to the next value (crossing a record boundary if necessary). An error is returned if the sequential position is past the last record for the rVariable. Required arguments are...

`out: void *value`

Value. This buffer must be large enough to hold the value. The value is read from the CDF and placed into memory at address `value`.

`zVAR_NAME_` Inquires the name of the current zVariable (in the current CDF). Required arguments are...

`out: char *varName`

Name of the zVariable. This character string must be large enough to hold `CDF_VAR_NAME_LEN + 1` characters (including the NUL).

`zVAR_DATATYPE_` Inquires the data type of the current zVariable (in the current CDF). Required arguments are...

`out: long *dataType`

Data type. The data types are described in Section 4.6.4.

`zVAR_NUMELEMS_` Inquires the number of elements (of the data type) for the current zVariable (in the current CDF). Required arguments are...

`out: long *numElements`

Number of elements of the data type at each value. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string (each value consists of the entire string). For all other data types, this will always be one (1) — multiple elements at each value are not allowed for non-character string data types.

`zVAR_RECVARY_` Inquires the record variance of the current zVariable (in the current CDF). Required arguments are...

`out: long *recVary`

Record variance. The variances are described in Section 4.6.7.

`zVAR_DIMVARYS_` Inquires the dimension variances of the current zVariable (in the current CDF). For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`out: long dimVarys[]`

Dimension variances. `dimVarys` is a 1-dimensional array containing one element per dimension. Each element of `dimVarys` receives the corresponding dimension variance. The variances are described in Section 4.6.7.

`zVAR_NUMBER_` Gets the number of the named zVariable (in the current CDF). Required arguments are...

`in: char *varName`

The zVariable name. This may be at most `CDF_VAR_NAME_LEN` characters (excluding the NUL).

`out: long *varNum`

The zVariable number.

`zVAR_PADVALUE_` Inquires the pad value of the current zVariable (in the current CDF). Required arguments are...

`out: void *value`

Pad value. This buffer must be large enough to hold the pad value. The pad value is read from the CDF and placed in memory at address `value`.

`zVAR_EXTENDRECS_` Inquires the number of extend records for the current zVariable (in the

current CDF). Extend records are described in Section 2.7.5. Required arguments are...

`out: long *NextendRecs`

Number of records to extend.

`zVAR_MAXREC_` Inquires the maximum record number for the current zVariable (in the current CDF). For zVariables with a record variance of `NOVARY`, this will be at most zero (0). A value of negative one (-1) indicates that no records have been written. Required arguments are...

`out: long *varMaxRec`

Maximum record number.

`zVAR_DATA_` Reads a value from the current zVariable (in the current CDF). The value is read at the current record number and current dimension indices for that zVariable (in the current CDF). Required arguments are...

`out: void *value`

Value. This buffer must be large enough to hold the value. The value is read from the CDF and placed into memory at address `value`.

`zVAR_HYPERDATA_` Reads one or more values from the current zVariable (in the current CDF). The values are read based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals for that zVariable (in the current CDF). Required arguments are...

`out: void *buffer`

Values. This buffer must be large enough to hold the values. The values are read from the CDF and placed into memory starting at address `buffer`.

`zVAR_SEQDATA_` Reads one value from the current zVariable (in the current CDF) at the current sequential position for that zVariable. After the read, the sequential position is automatically incremented to the next value (crossing a record boundary if necessary). An error is returned if the sequential position is past the last record for the zVariable. Required arguments are...

`out: void *value`

Value. This buffer must be large enough to hold the value. The value is read from the CDF and placed into memory at address `value`.

`ATTR_SCOPE_` Inquires the scope of the current attribute (in the current CDF). Required arguments are...

`out: long *scope`

Attribute scope. The scopes are described in Section 4.6.8.

`ATTR_NAME_` Inquires the name of the current attribute (in the current CDF). Required arguments are...

`out: char *attrName`

Attribute name. This character string must be large enough to hold `CDF_ATTR_NAME_LEN + 1` characters (including the NUL).

`ATTR_NUMBER_` Gets the number of the named attribute (in the current CDF). Required arguments are...

`in: char *attrName`

Attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters (excluding the NUL).

`out: long *attrNum`

The attribute number.

`ATTR_MAXgENTRY_` Inquires the maximum gEntry number used for the current attribute (in the current CDF). This does not necessarily correspond with the number of gEntries for the attribute. **NOTE:** Use this operation with gAttributes only. Required arguments are...

`out: long *maxEntry`

The maximum gEntry number for the attribute.

`ATTR_NUMgENTRIES_` Inquires the number of gEntries for the current attribute (in the current CDF). This does not necessarily correspond with the maximum gEntry number used. **NOTE:** Use this operation with gAttributes only. Required arguments are...

`out: long *numEntries`

The number of gEntries for the attribute.

`ATTR_MAXrENTRY_` Inquires the maximum rEntry number used for the current attribute (in the current CDF). This does not necessarily correspond with the number of rEntries for the attribute. **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: long *maxEntry`

The maximum rEntry number for the attribute.

`ATTR_NUMrENTRIES_` Inquires the number of rEntries for the current attribute (in the current CDF). This does not necessarily correspond with the maximum rEntry number used. **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: long *numEntries`

The number of rEntries for the attribute.

`ATTR_MAXzENTRY_` Inquires the maximum zEntry number used for the current attribute (in the current CDF). This does not necessarily correspond with the number of zEntries for the attribute. **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: long *maxEntry`

The maximum zEntry number for the attribute.

`ATTR_NUMzENTRIES_` Inquires the number of zEntries for the current attribute (in the current CDF). This does not necessarily correspond with the maximum zEntry number used. **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: long *numEntries`

The number of zEntries for the attribute.

`gENTRY_DATATYPE_` Inquires the data type of the gEntry at the current gEntry/rEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with gAttributes only. Required arguments are...

`out: long *dataType`

Data type. The data types are described in Section 4.6.4.

`gENTRY_NUMELEMS_` Inquires the number of elements (of the data type) of the gEntry at the current gEntry/rEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with gAttributes only. Required arguments are...

`out: long *numElements`

Number of elements of the data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string (an array of characters). For all other data types, this is the number of elements in an array of that data type.

`gENTRY_DATA_` Reads the gEntry data value from the current attribute at the current gEntry/rEntry number (in the current CDF). **NOTE:** Use this operation with gAttributes only. Required arguments are...

`out: void *value`

Value. This buffer must be large to hold the value. The value is read from the CDF and placed into memory at address `value`.

`rENTRY_DATATYPE_` Inquires the data type of the rEntry at the current gEntry/rEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: long *dataType`

Data type. The data types are described in Section 4.6.4.

`rENTRY_NUMELEMS_` Inquires the number of elements (of the data type) of the rEntry at the current gEntry/rEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: long *numElements`

Number of elements of the data type. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (an array of characters). For all other data types, this is the number of elements in an array of that data type.

rENTRY_DATA_ Reads the rEntry data value from the current attribute at the current gEntry/rEntry number (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

out: void *value

Value. This buffer must be large to hold the value. The value is read from the CDF and placed into memory at address value.

zENTRY_DATATYPE_ Inquires the data type of the zEntry at the current zEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

out: long *dataType

Data type. The data types are described in Section 4.6.4.

zENTRY_NUMELEMS_ Inquires the number of elements (of the data type) of the zEntry at the current zEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

out: long *numElements

Number of elements of the data type. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (an array of characters). For all other data types, this is the number of elements in an array of that data type.

zENTRY_DATA_ Reads the zEntry data value from the current attribute at the current zEntry number (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

out: void *value

Value. This buffer must be large to hold the value. The value is read from the CDF and placed into memory at address value.

STATUS_TEXT_ Inquires the explanation text for the current status code. Note that the current status code is NOT the status from the last operation performed. Required arguments are...

out: char *text

Text explaining the status code. This string must be large enough to hold CDF_STATUSTEXT_LEN + 1 characters.

More Examples

Several more examples of the use of `CDFlib` follow.

rVariable creation In this example an rVariable will be created with a pad value being specified, initial records will be written, and the number of records to extend the rVariable when necessary being specified. Note that the pad value was specified before the initial records. This results in the specified pad value being written. Had the pad value not been specified first, the initial records would have been written with the default pad value.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                         /* CDF identifier (handle). */
CDFstatus   status;                     /* Status returned from CDF
                                           library. */
long        dimVarys[2];                /* Dimension variances. */
long        varNum;                     /* rVariable number. */
float       padValue = -999.9;          /* Pad value. */
.
.
dimVarys[0] = VARY;
dimVarys[1] = VARY;
status = CDFlib (SELECT_, CDF_, id,
                 CREATE_, rVAR_, "HUMIDITY", CDF_REAL4, 1, VARY,
                                 dimVarys, &varNum,
                 PUT_, rVAR_PADVALUE_, &padValue,
                       rVAR_INITIALRECS_, (long) 500,
                       rVAR_EXTENDRECS_, (long) 50,
                 NULL_);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

zVariable creation (character string data type) In this example a zVariable with a character string data type will be created with a pad value being specified.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                         /* CDF identifier (handle). */
CDFstatus   status;                     /* Status returned from CDF
                                           library. */
```

```
long        dimVarys[1];               /* Dimension variances. */
long        varNum;                    /* zVariable number. */
long        numDims = 1;               /* Number of dimensions. */
static long dimSizes[1] = { 20 };      /* Dimension sizes. */
long        numElems = 10;             /* Number of elements (characters
                                          in this case). */
static char padValue = "**********";   /* Pad value. */
.
.
dimVarys[0] = VARY;
status = CDFlib (SELECT_, CDF_, id,
                 CREATE_, zVAR_, "Station", CDF_CHAR, numElems, numDims,
                                 dimSizes, NOVARY, dimVarys, &varNum,
                 PUT_, zVAR_PADVALUE_, padValue,
                 NULL_);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

Hyper read with subsampling In this example an rVariable will be subsampled in a CDF whose rVariables are 2-dimensional and have dimension sizes `[100,200]`. The CDF is row major and the data type of the rVariable is `CDF_UINT2`.

```
.
.
#include "cdf.h"
.
.
CDFid          id;                  /* CDF identifier (handle). */
CDFstatus      status;              /* Status returned from CDF library. */
unsigned short values[50][100];     /* Buffer to receive values. */
long           recCount = 1;        /* Record count, one record per hyper
                                       get. */
long           recInterval = 1;     /* Record interval, set to one to
                                       indicate contiguous records (really
                                       meaningless since record count is
                                       one). */
static long    indices[2] = {0,0};  /* Dimension indices, start each read
                                       at 0,0 of the array. */
static long    counts[2] = {50,100}; /* Dimension counts, half of the
                                       values along each dimension will
                                       be read. */
static long    intervals[2] = {2,2}; /* Dimension intervals, every other
                                       value along each dimension will be
                                       read. */
long           recNum;              /* Record number. */
long           maxRec;              /* Maximum rVariable record
                                       number in the CDF - this was
                                       determined with a call to
```

```
                                        CDFinquire. */
   .
   .
   status = CDFlib (SELECT_, CDF_, id,
                            rVAR_NAME_, "BRIGHTNESS",
                            rVARs_RECCOUNT_, recCount,
                            rVARs_RECINTERVAL_, recInterval,
                            rVARs_DIMINDICES_, indices,
                            rVARs_DIMCOUNTS_, counts,
                            rVARs_DIMINTERVALS_, intervals,
                    NULL_);
   if (status != CDF_OK) UserStatusHandler (status);

   for (recNum = 0; recNum <= maxRec; recNum++) {
      status = CDFlib (SELECT_, CDF_, id,
                               rVARs_RECNUMBER_, recNum,
                       GET_, rVAR_HYPERDATA_, values,
                       NULL_);
      if (status != CDF_OK) UserStatusHandler (status);
      .
      .
      /* process values */
      .
      .
   }
   .
   .
```

Attribute renaming In this example the attribute named `Tmp` will be renamed `TMP`.

```
   .
   .
   #include "cdf.h"
   .
   .
   CDFid          id;                  /* CDF identifier (handle). */
   CDFstatus      status;              /* Status returned from CDF library. */
   .
   .
   status = CDFlib (SELECT_, CDF_, id,
                            ATTR_NAME_, "Tmp",
                    PUT_, ATTR_NAME, "TMP",
                    NULL_);
   if (status != CDF_OK) UserStatusHandler (status);
   .
   .
```

Sequential Access In this example the values for a zVariable will be averaged. The values will be read using the sequential access method (see Section 2.7.11). Each value in each record will be read and averaged.

It is assumed that the data type of the zVariable has been determined to be CDF_REAL4.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                 /* CDF identifier (handle). */
CDFstatus   status;             /* Status returned from CDF library. */
long        varNum;             /* zVariable number. */
long        recNum = 0;         /* Record number, start at first record. */
static long indices[2] = {0,0}; /* Dimension indices. */
float       value;              /* Value read. */
double      sum = 0.0;          /* Sum of all values. */
long        count = 0;          /* Number of values. */
float       ave;                /* Average value. */
.
.
status = CDFlib (SELECT_, CDF_, id,
                 GET_, zVAR_NUMBER_, "FLUX", &varNum,
                 NULL_);
if (status != CDF_OK) UserStatusHandler (status);

status = CDFlib (SELECT_, CDF_, id,
                          zVAR_, varNum,
                          zVAR_SEQPOS_, recNum, indices,
                 GET_, zVAR_SEQDATA_, &value,
                 NULL_);
while (status >= CDF_OK) {
  sum += value;
  count++;
  status = CDFlib (SELECT_, CDF_, id,
                   GET_, zVAR_SEQDATA_, &value,
                   NULL_);
}

if (status != END_OF_VAR) UserStatusHandler (status);

ave = sum / count;
.
.
```

Attribute rEntry writes In this example a set of attribute rEntries for a particular rVariable will be written.

```
.
.
#include "cdf.h"
.
```

```
.
CDFid          id;                        /* CDF identifier (handle). */
CDFstatus      status;                    /* Status returned from CDF
                                             library. */
static float   scale[2] = {-90.0,90.0};   /* Scale, minimum/maximum. */
.
.
status = CDFlib (SELECT_, CDF_, id,
                          rVAR_NAME_, "LATITUDE",
                          ATTR_NAME_, "FIELDNAM",
                 PUT_, rENTRY_DATA_, CDF_CHAR, (long) 20,
                                     "Latitude            ",
                 SELECT_, ATTR_NAME_, "SCALE",
                 PUT_, rENTRY_DATA_, CDF_REAL4, (long) 2, scale,
                 SELECT_, ATTR_NAME_, "UNITS",
                 PUT_, rENTRY_DATA_, CDF_CHAR, (long) 20,
                                     "Degrees north       ",
                 NULL_);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.8 Interpreting CDF Status Codes

Most CDF functions return a status code of type `CDFstatus`. The symbolic names for these codes are defined in `cdf.h` and should be used in your applications rather than using the true numeric values. Appendix C explains each status code. When testing the status code returned from a CDF function, the following rules apply.

`status > CDF_OK`	Indicates successful completion but some additional information is provided. These are informational codes.
`status = CDF_OK`	Indicates successful completion.
`CDF_WARN < status < CDF_OK`	Indicates that the function completed but probably not as expected. These are warning codes.
`status < CDF_WARN`	Indicates that the function did not complete. These are error codes.

The following example shows how you could check the status code returned from CDF functions.

```
CDFstatus status;
.
.
status = CDFfunction (...);   /* any CDF function returning CDFstatus */
if (status != CDF_OK) {
  UserStatusHandler (status, ...);
```

```
    .
    .
  }
```

In your own status handler you can take whatever action is appropriate to the application. An example status handler follows. Note that no action is taken in the status handler if the status is CDF_OK.

```
  #include <stdio.h>

  #include "cdf.h"

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

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

Explanations for all CDF status codes are available to your applications through the function CDFerror. CDFerror writes to a text string an explanation of a given status code.

4.9 CDF_EPOCH Values

Two functions exist which compute and breakdown CDF_EPOCH values. These functions should be used by applications using the CDF_EPOCH data type. Section 2.5.4 describes EPOCH values.

4.9.1 computeEPOCH

computeEPOCH calculates a CDF_EPOCH value given the individual components.

```
double computeEPOCH(     /* out -- CDF_EPOCH value returned. */
long year,               /* in  -- Year (AD, eg. 1991). */
long month,              /* in  -- Month (1-12). */
long day,                /* in  -- Day (1-31). */
long hour,               /* in  -- Hour (0-23). */
long minute,             /* in  -- Minute (0-59). */
long second,             /* in  -- Second (0-59). */
long msec);              /* in  -- Millisecond (0-999). */
```

computeEPOCH is a function in the CDF library.

4.9.2 EPOCHbreakdown

EPOCHbreakdown breaks down a CDF_EPOCH value into the individual components.

```
void EPOCHbreakdown(
double epoch,            /* in  -- The CDF_EPOCH value. */
long   *year,            /* out -- Year (AD, eg. 1991). */
long   *month,           /* out -- Month (1-12). */
long   *day,             /* out -- Day (1-31). */
long   *hour,            /* out -- Hour (0-23). */
long   *minute,          /* out -- Minute (0-59). */
long   *second,          /* out -- Second (0-59). */
long   *msec);           /* out -- Millisecond (0-999). */
```

EPOCHbreakdown is a function in the CDF library.

4.10 Custom C Functions

Most of the Standard Interface functions callable from C applications are implemented as C macros which call CDFlib (Internal Interface). For example, the CDFcreate function is actually defined as the following C macro...

```
#define CDFcreate(CDFname,numDims,dimSizes,encoding,majority,id) \
CDFlib (CREATE_, CDF_, CDFname, numDims, dimSizes, id, \
        PUT_, CDF_ENCODING_, encoding, \
              CDF_MAJORITY_, majority, \
        NULL_)
```

These macros are defined in `cdf.h`. Where your application calls `CDFcreate`, the C compiler expands the macro into the corresponding call to `CDFlib`.

The flexibility of `CDFlib` allows you to define your own custom CDF functions using C macros. For instance, a function which inquires the format of a CDF could be defined as follows...

```
#define CDFinquireFormat(id,format) \
CDFlib (SELECT_, CDF_, id, \
        GET_, CDF_FORMAT_, format, \
        NULL_)
```

Your application would call the function as follows...

```
.
.
CDFid     id;            /* CDF identifier. */
CDFstatus status;        /* Returned status code. */
long      format;        /* Format of CDF. */
.
.
status = CDFinquireFormat (id, &format);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

Chapter 5

CDF FORTRAN PROGRAMMING REFERENCE

5.1 Introduction

This chapter provides a reference for using the CDF library from FORTRAN applications. A separate reference exists for writing CDF applications in C (Chapter 4).

5.2 Compiling

Each program, subroutine, or function which calls the CDF library or references CDF parameters must include `cdf.inc`. On VMS systems a logical name, `CDF$INC`, which specifies the location of `cdf.inc` is defined in the definitions file, `DEFINITIONS.COM`, provided with the CDF distribution. On UNIX systems an environment variable, `CDF_INC`, which serves the same purpose is defined in the definitions file named `definitions.<shell>` (where `<shell>` is the shell being used). This section assumes that you are using the appropriate definitions file.

The following line would be included at the top of each file...

```
INCLUDE '<inc-path>cdf.inc'
```

where `<inc-path>` is the pathname of the directory containing `cdf.inc`. On VMS systems, `CDF$INC:` may be used for `<inc-path>`. On UNIX and MS-DOS systems, `<inc-path>` must be a relative or absolute pathname (an environment variable may not be used on UNIX systems).

On UNIX systems, a symbolic link to `cdf.inc` (using `ln -s`) could be created making `cdf.inc` appear to be in the same directory as the source file to be compiled. In that case, specifying `<inc-path>` would not be necessary.

On MS-DOS systems using some FORTRAN compilers, the location of `cdf.inc` may be specified on the

command line. In that case, specifying `<inc-path>` would not be necessary.

5.2.1 VMS systems

An example of the command to compile a source file on VMS systems would be as follows...

```
$ FORTRAN <source-file(s)>
```

where `<source-file(s)>` is the name of the source file(s) being compiled.

5.2.2 UNIX systems

An example of the command to compile a source file on UNIX systems would be as follows...

```
% f77 -c <source-file(s)>.f
```

where `<source-file(s)>` is the name of the source file(s) being compiled.

The `-c` option specifies that only an object module(s) is to be produced (the link step is described in Section 4.3).

5.2.3 MS-DOS systems

An example of the command to compile a source file on MS-DOS systems would be as follows (for Microsoft FORTRAN)...

```
> FL /c /AL <source-file(s)>.for
```

where `<source-file(s)>` is the name of the source file(s) being compiled.

The `/c` option specifies that only an object module(s) is to be produced (the link step is described in Section 5.3).

The `/AL` option specifies that the object module(s) is to be compiled using the large memory model. The huge memory model could also be used if necessary. The CDF library can be built using either the large or huge memory model. All of your application source files must be compiled using the same memory model as that used for the CDF library.

NOTE: The CDF library is written in C. The FORTRAN compiler used to compile your applications must be compatible with the C compiler used to build the CDF library (Microsoft C and Microsoft FORTRAN have been shown in the examples in this document). The linker used must also be configured to allow a FORTRAN application to call C routines (in the CDF library). Your FORTRAN applications, however, should not be concerned with calling functions written in C (they can assume that they are calling FORTRAN). The CDF library is written to handle the default FORTRAN calling conventions.

5.3 Linking

Your applications must be linked with the CDF library (`LIBCDF.OLB` on VMS systems, `libcdf.a` on UNIX systems, and `libcdf.lib` on MS-DOS systems).[1] Both the Standard and Internal interfaces for FORTRAN applications are built into the CDF library. On VMS systems a logical name, `CDF$LIB`, which specifies the location of the CDF library is defined in the definitions file, `DEFINITIONS.COM`, provided with the CDF distribution. On UNIX systems an environment variable, `CDF_LIB`, which serves the same purpose is defined in the definitions file `definitions.<shell>` (where `<shell>` is the shell being used). This section assumes that you are using the appropriate definitions file.

5.3.1 VMS systems

An example of the command used to link an application on a VMS system would be as follows...

```
$ LINK <object-file(s)>, CDF$LIB:LIBCDF/LIBRARY
```

where `<object-file(s)>` is your application's object module(s). The name of the executable created will be the name part of the (first) object file with `.EXE` appended.

5.3.2 UNIX systems

An example of the command used to link an application on a UNIX system would be as follows...

```
% f77 <object-file(s)>.o ${CDF_LIB}/libcdf.a
```

where `<object-file(s)>.o` is your application's object module(s). The name of the executable created will be `a.out` by default. It may also be explicitly specified using the `-o` option. Some UNIX systems may also require that `-lm` (the math library) and/or `-ldl` (the dynamic linker library) be specified.

Combining the compile and link

On UNIX systems the compile and link may be combined into one step as follows...

```
% f77 <source-file(s)>.f ${CDF_LIB}/libcdf.a
```

5.3.3 MS-DOS systems

An example of the command used to link an application on a MS-DOS system would be as follows (for Microsoft FORTRAN)...

[1] A shared version of the CDF library is also available on VMS and some flavors of UNIX. Its use is described in Section 5.4.

```
> LINK <object-file(s)>.obj,,nul.map,<lib-path>libcdf.lib
```

where `<object-file(s)>.o` is your application's object module(s) and `<lib-path>` is the pathname of the directory containing `libcdf.lib`. You will need to know where on your system `libcdf.lib` has been installed.

For Microsoft FORTRAN, the omitted parameter (indicated by `,,`) is the name of the executable to be created. The name of the executable will be the name part of the (first) object file with `.exe` appended. A different name for the executable may be explicitly specified if desired. A map file is created by default unless the special name `nul.map` is used (as shown). If a map file is desired, the map file parameter should be omitted (in which case the name of the map file will be the name part of the executable file with `.map` appended) or a map file should be explicitly specified.

You may also need to specify the location of system libraries. Consult the documentation for the compiler/linker being used for more information.

NOTE: The same memory model must have been used to compile your application's source files and the CDF library. The CDF library must have been compiled with the large or huge memory model.

Combining the compile and link

On MS-DOS systems the compile and link may be combined into one step as follows (for Microsoft FORTRAN)...

```
> FL /AL <source-file(s)>.for <lib-path>libcdf.lib
```

The name of the executable created will be the name part of the (first) source file with `.exe` appended. Additional options/parameters may be required. Consult the documentation for the compiler/linker being used for more information.

5.4 Linking, shared CDF library

A shareable version of the CDF library is also available on VMS systems and some flavors of UNIX. The shared version is put in the same directory as the non-shared version and is named as follows...

Machine/Operating System	Shared CDF Library
VMS	`LIBCDF.EXE`
UNIX, Sun/SunOS	`libcdf.so`
UNIX, HP9000/HP-UX	`libcdf.sl`
UNIX, IBM RS6000/AIX	`libcdf.o`

The commands necessary to link to a shared library vary from operating system to operating system. Please consult the documentation for your operating system for details.

NOTE: On VMS systems, the shared CDF library may also be installed in `SYS$SHARE`.

5.5 Argument Passing

The CDF library is written entirely in C. Most computer systems have FORTRAN and C compilers which allow a FORTRAN application to call a C function without being concerned that different programming languages are involved. The CDF library takes advantage of the mechanisms provided by these compilers so that your FORTRAN application can appear to be calling another FORTRAN subroutine/function (in actuality the CDF library written in C). Pass all arguments exactly as shown in the description of each CDF function. This includes character strings (ie. `%REF(...)` is not required). Be aware, however, that trailing blanks on variable and attribute names will be considered as part of the name. If the trailing blanks are not desired, pass only the part of the character string containing the name (eg. `VAR_NAME(1:8)`).

NOTE: Unfortunately the world is not a perfect place. The Microsoft C and Microsoft FORTRAN compilers on the IBM PC and the C and FORTRAN compilers on the NeXT computer do not provide the needed mechanism to pass character strings from FORTRAN to C without explicitly NUL-terminating the strings. Your FORTRAN application must place an ASCII NUL character after the last character of a CDF, variable, or attribute name. An example of this follows...

```
.
.
CHARACTER ATTR_NAME*9                ! Attribute name
.
.
ATTR_NAME(1:8) = 'VALIDMIN'          ! Actual attribute name
ATTR_NAME(9:9) = CHAR(0)             ! ASCII NUL character
.
.
```

`CHAR(0)` is an intrinsic FORTRAN function which returns the ASCII character for the numerical value passed in (0 is the numerical value for an ASCII NUL character). `ATTR_NAME` could then be passed to one of the CDF library functions.

When the CDF library passes out a character string on an IBM PC (using the Microsoft compilers) or on a NeXT computer, the number of characters written will be exactly as shown in the description of the function called. You must declare your FORTRAN variable to be exactly that size.

5.6 Item Referencing

For FORTRAN applications, all items are referenced starting at one (1). These include variable, attribute, and attribute entry numbers, record numbers, dimensions, and dimension indices. Note that both rVariables and zVariables are numbered starting at one (1).

5.7 CDF Parameters — Predefined Constants and Types

The following sections describe constants and types defined for use by all CDF application programs written in FORTRAN. These constants and types are defined in `cdf.inc`. `cdf.inc` should be included in all application source files referencing CDF functions/parameters.

5.7.1 Status Code Constants

`CDF_OK`	A status code indicating the normal completion of a CDF function.
`CDF_WARN`	Threshold constant for testing severity of non-normal CDF status codes.

Section 5.9 describes how to use these constants to interpret status codes.

5.7.2 CDF Formats

`SINGLE_FILE`	The CDF consists of only one file.
`MULTI_FILE`	The CDF consists of one header file for control and attribute data and one additional file for each variable in the CDF.

5.7.3 CDF Datatypes

One of the following constants must be used when specifying a CDF data type for an attribute entry or variable.

`CDF_BYTE`	1-byte, signed integer.
`CDF_CHAR`	1-byte, signed character.
`CDF_INT1`	1-byte, signed integer.
`CDF_UCHAR`	1-byte, unsigned character.
`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_REAL4`	4-byte, floating point.
`CDF_FLOAT`	4-byte, floating point.
`CDF_REAL8`	8-byte, floating point.

CDF_DOUBLE	8-byte, floating point.
CDF_EPOCH	8-byte, floating point.

CDF_CHAR and CDF_UCHAR are considered character string data types. These are significant because only variables of these data types may have more than one element per value (where each element is a character).

5.7.4 Data Encodings

A CDF's data encoding affects how its attribute entry and variable data values are stored (on disk). Attribute entry and variable values in memory are always in the host machine's native encoding.

HOST_ENCODING	Indicates host machine data representation (native). This encoding will provide the greatest performance but will not allow the CDF to be transported to other machines that have different encodings.
NETWORK_ENCODING	Indicates network transportable data representation (XDR). This encoding allows the CDF to be transported to other machines but with (possibly) reduced performance.
VAX_ENCODING	Indicates VAX data representation.
SUN_ENCODING	Indicates SUN data representation.
MIPSEB_ENCODING	Indicates MIPSEB data representation (Silicon Graphics Iris and Power Series).
MIPSEL_ENCODING	Indicates MIPSEL data representation (DECstation).
IBMRS_ENCODING	Indicates IBMRS data representation (IBM RS/6000 series).
HP_ENCODING	Indicates HP data representation (HP 9000 series).
IBMPC_ENCODING	Indicates IBM PC data representation.
NeXT_ENCODING	Indicates NeXT data representation.

When creating a CDF, you must specify NETWORK_ENCODING, HOST_ENCODING, or the specific encoding for the host machine (eg. VAX_ENCODING). Specifying the host machine's encoding explicitly has the same effect as specifying HOST_ENCODING.

When inquiring the encoding of a CDF, either NETWORK_ENCODING or the specific machine encoding will be returned (HOST_ENCODING is never returned).

5.7.5 Variable Majorities

A CDF's variable majority determines the order in which variable values (within the variable arrays) are stored in the CDF file(s). The majority is the same for rVariables and zVariables.

ROW_MAJOR	C-like array ordering for variable storage. The first dimension in each variable array varies the slowest.
COL_MAJOR	FORTRAN-like array ordering for variable storage. The first dimension in each variable array varies the fastest.

For FORTRAN applications, the compiler defined majority for arrays is column major. The first dimension of multi-dimensional arrays varies the fastest in memory.

Knowing the variable majority of a CDF is necessary when performing hyper reads and writes. During a hyper read the CDF library will place the variable data values into the memory buffer in the same majority as that of the CDF. The buffer must then be processed according to that majority. Likewise, during a hyper write the CDF library will expect to find the variable data values in the memory buffer in the same majority as that of the CDF.

5.7.6 Record/Dimension Variances

Record and dimension variances affect how variable data values are physically stored.

VARY	True record or dimension variance.
NOVARY	False record or dimension variance.

If a variable has a record variance of VARY, then each record for that variable is physically stored. If the record variance is NOVARY, then only one record is physically stored (all virtual records contain the same values).

If a variable has a dimension variance of VARY, then each value/subarray along that dimension is physically stored. If the dimension variance is NOVARY, then only one value/subarray along that dimension is physically stored (all virtual values/subarrays along that dimension are the same).

5.7.7 Attribute Scopes

Attribute scopes are simply a way to explicitly declare the intended use of an attribute by user applications (and the CDF toolkit).

GLOBAL_SCOPE	Indicates that an attribute's scope is global (applies to the CDF as a whole).
VARIABLE_SCOPE	Indicates that an attribute's scope is by-variable (each rEntry or zEntry corresponds to an rVariable or zVariable respectively).
GLOBAL_SCOPE_ASSUMED	Indicates that an attribute's scope is assumed global.
VARIABLE_SCOPE_ASSUMED	Indicates that an attribute's scope is assumed by-variable.

GLOBAL_SCOPE and VARIABLE_SCOPE are definite scopes. GLOBAL_SCOPE_ASSUMED and VARIABLE_SCOPE_ASSUMED are assumed scopes.

Section 2.8.3 explains attribute scopes and how assumed scopes can arise in a CDF.

5.7.8 Operational Limits

These are limits within the CDF library. If you reach one of these limits please contact CDF User Support.

`CDF_MAX_DIMS`	Maximum number of dimensions for the rVariables or a zVariable.

The CDF library imposes no limit on the number of variables, attributes, or attribute entries that a CDF may have. On the IBM PC, however, the number of rVariable and zVariables will be limited to 100 of each in a multi-file CDF because of the 8.3 naming convention imposed by MS-DOS.

5.7.9 Limits of Names and Other Character Strings

`CDF_PATHNAME_LEN`	Maximum length of a CDF pathname (excluding the `.cdf` or `.vnn` appended by the CDF library to construct filenames). A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on VMS systems and environment variables on UNIX systems). **NOTE:** On UNIX systems pathnames are case sensitive.
`CDF_VAR_NAME_LEN`	Maximum length of a variable name.
`CDF_ATTR_NAME_LEN`	Maximum length of an attribute name.
`CDF_COPYRIGHT_LEN`	Maximum length of the CDF copyright text.
`CDF_STATUSTEXT_LEN`	Maximum length of the explanation text for a status code.

5.8 FORTRAN Function Reference

The following sections describe the CDF library routines callable from FORTRAN applications. Most routines return a status code of type `INTEGER*4` (see Section 5.9). Each routine except `CDF_lib` is part of the Standard Interface — `CDF_lib` is part of the Internal Interface. An application can use both interfaces when necessary. Note that zVariables and vAttribute zEntries are only accessable via the Internal Interface.

5.8.1 CDF_create (Standard Interface)

```
SUBROUTINE CDF_create (CDF_name, num_dims, dim_sizes, encoding, majority,
1                      id, status)

CHARACTER CDF_name*(*)        ! in  -- CDF pathname.
INTEGER*4 num_dims            ! in  -- Number of dimensions, rVariables.
INTEGER*4 dim_sizes(*)        ! in  -- Dimension sizes, rVariables.
```

```
INTEGER*4 encoding            ! in  -- Data encoding.
INTEGER*4 majority            ! in  -- Variable majority.
INTEGER*4 id                  ! out -- CDF identifier.
INTEGER*4 status              ! out -- Completion status.
```

CDF_create creates a CDF as defined by the arguments. A CDF cannot be created if it already exists (the existing CDF will not be overwritten). If you want to overwrite an existing CDF, you must first open it with CDF_open, delete it with CDF_delete, and then recreate it with CDF_create. If the existing CDF is corrupted, the call to CDF_open will fail (an error code will be returned). In this case you must delete the CDF at the command line. Delete the .cdf file and if the CDF has the multi-file format, delete all of the .v0,.v1,... files and .z0,.z1,... files.

The arguments to CDF_create are defined as follows.

CDF_name	The pathname of the CDF to create (do not specify an extension). This may be at most CDF_PATHNAME_LEN characters. A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on VMS systems and environment variables on UNIX systems). **NOTE:** On UNIX systems pathnames are case sensitive.
num_dims	Number of dimensions the rVariables in the CDF are to have. This may be as few as zero (0) and at most CDF_MAX_DIMS.
dim_sizes	The size of each dimension. dim_sizes is a 1-dimensional array containing num_dims elements. Each element of dim_sizes specifies the corresponding dimension size. Each size must be greater than zero (0). If there are zero (0) dimensions this argument is ignored (but a place holder is necessary).
encoding	The encoding for variable data and attribute entry data. Specify one of the encodings described in Section 5.7.4.
majority	The majority for variable data. Specify one of the majorities described in Section 5.7.5.
id	The identifier for the created CDF. This identifier must be used in all subsequent operations on the CDF.
status	The completion status code. Section 5.9 explains how to interpret status codes.

When a CDF is created, both read and write access are allowed. The default format for a CDF created with CDF_create is multi-file (MULTI_FILE). The CDF_lib function (Internal Interface) must be used to change a CDF's format to single-file (SINGLE_FILE).

CDF_close MUST be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk (see Section 5.8.5).

Example(s)

The following example will create a CDF named test1 with network encoding and row majority.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id              ! CDF identifier.
INTEGER*4 status          ! Returned status code.
INTEGER*4 num_dims        ! Number of dimensions, rVariables.
INTEGER*4 dim_sizes(3)    ! Dimension sizes, rVariables.
INTEGER*4 majority        ! Variable majority.

DATA num_dims/3/, dim_sizes/180,360,10/, majority/ROW_MAJOR/
.
.
CALL CDF_create ('test1', num_dims, dim_sizes, NETWORK_ENCODING,
1                majority, id, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

`ROW_MAJOR` and `NETWORK_ENCODING` are defined in `cdf.inc`.

5.8.2 CDF_open (Standard Interface)

```
SUBROUTINE CDF_open (CDF_name, id, status)

CHARACTER CDF_name*(*)        ! in  -- CDF pathname.
INTEGER*4 id                  ! out -- CDF identifier.
INTEGER*4 status              ! out -- Completion status.
```

`CDF_open` opens an existing CDF. The CDF is initially opened with only read access. This allows multiple applications to read the same CDF simultaneously. When an attempt to modify the CDF is made, it is automatically closed and reopened with read/write access (the function will fail if the application does not have or can not get write access to the CDF).

The arguments to `CDF_open` are defined as follows.

`CDF_name`	The pathname of the CDF to open (do not specify an extension). This may be at most `CDF_PATHNAME_LEN` characters. A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on VMS systems and environment variables on UNIX systems). **NOTE:** On UNIX systems pathnames are case sensitive.
`id`	The identifier for the opened CDF. This identifier must be used in all subsequent operations on the CDF.
`status`	The completion status code. Section 5.9 explains how to interpret status codes.

CDF_close MUST be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk (see Section 5.8.5).

Example(s)

The following example will open a CDF named NOAA1.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                            ! CDF identifier.
INTEGER*4 status                        ! Returned status code.
CHARACTER CDF_name*(CDF_PATHNAME_LEN)   ! Name of CDF.

DATA CDF_name/'NOAA1'/
.
.
CALL CDF_open (CDF_name, id, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.3 CDF_doc (Standard Interface)

```
SUBROUTINE CDF_doc (id, version, release, copyright, status)

INTEGER*4 id,                           ! in  -- CDF identifier.
INTEGER*4 version,                      ! out -- Version number.
INTEGER*4 release,                      ! out -- Release number.
CHARACTER copyright*(CDF_DOCUMENT_LEN)  ! out -- Copyright.
```

CDF_doc is used to inquire general documentation about a CDF. The version/release of the CDF library which created the CDF is provided (eg. CDF V2.3 is version 2, release 3) along with the CDF copyright notice.

The arguments to CDF_doc are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
version	The version number of the CDF library which created the CDF.
release	The release number of the CDF library which created the CDF.
copyright	The copyright notice of the CDF library which created the CDF. This character string must be large enough to hold CDF_COPYRIGHT_LEN characters and will be

blank padded if necessary. This string will contain a linefeed character after each line of the copyright notice (it can be printed without modification but may contain many trailing blanks).

`status` The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example will inquire and display the version/release and copyright notice.

```
    .
    .
    INCLUDE '<path>cdf.inc'
    .
    .
    INTEGER*4 id                              ! CDF identifier.
    INTEGER*4 status                          ! Returned status code.
    INTEGER*4 version                         ! CDF version number.
    INTEGER*4 release                         ! CDF release number.
    CHARACTER copyright*(CDF_COPYRIGHT_LEN)   ! Copyright notice.
    INTEGER*4 last_char                       ! Last character position
                                              ! actually used in the copyright.
    INTEGER*4 start_char                      ! Starting character position
                                              ! in a line of the copyright.
    CHARACTER lf*1                            ! Linefeed character.
    .
    .
    CALL CDF_doc (id, version, release, copyright, status)
    IF (status .LT. CDF_OK) THEN                ! INFO status codes ignored
      CALL UserStatusHandler (status)
    ELSE
      WRITE (6,101) version, release
101   FORMAT (' ','Version: ',I3,' Release:',I3)

      last_char = CDF_COPYRIGHT_LEN
      DO WHILE (copyright(last_char:last_char) .EQ. ' ')
        last_char = last_char - 1
      END DO

      lf = CHAR(10)

      start_char = 1
      DO i = 1, last_char
        IF (copyright(i:i) .EQ. lf) THEN
          WRITE (6,301) copyright(start_char:i-1)
301       FORMAT (' ',A)
          start_char = i + 1
        END IF
      END DO
```

```
    END IF
    .
    .
```

5.8.4 CDF_inquire (Standard Interface)

```
SUBROUTINE CDF_inquire (id, num_dims, dim_sizes, encoding, majority,
1                       max_rec, num_vars, num_attrs, status)

INTEGER*4 id                        ! in  -- CDF identifier.
INTEGER*4 num_dims                  ! out -- Number of dimensions, rVariables.
INTEGER*4 dim_sizes(CDF_MAX_DIMS)   ! out -- Dimension sizes, rVariables.
INTEGER*4 encoding                  ! out -- Data encoding.
INTEGER*4 majority                  ! out -- Variable majority.
INTEGER*4 max_rec                   ! out -- Maximum record number in the CDF,
                                             rVariables.
INTEGER*4 num_vars                  ! out -- Number of rVariables in
                                             the CDF.
INTEGER*4 num_attrs                 ! out -- Number of attributes in the CDF.
INTEGER*4 status                    ! out -- Completion status.
```

`CDF_inquire` inquires the basic characteristics of a CDF. An application needs to know the number of dimensions and their sizes before it can access variable data. Knowing the variable majority can be used to optimize performance and is necessary to properly access the buffer of data values used with the hyper functions — `CDF_var_hyper_put` and `CDF_var_hyper_get`.

The arguments to `CDF_inquire` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDF_create` or `CDF_open`.
`num_dims`	The number of dimensions in the CDF for the rVariables.
`dim_sizes`	The dimension sizes for the rVariables. `dim_sizes` is a 1-dimensional array containing one element per dimension. Each element of `dim_sizes` receives the corresponding dimension size. If there are zero (0) dimensions this argument is ignored (but a place holder is necessary).
`encoding`	The encoding of the variable data and attribute entry data. The encodings are defined in Section 5.7.4.
`majority`	The majority of the variable data. The majorities are defined in Section 5.7.5.
`max_rec`	The maximum record number written to an rVariable in the CDF. Note that the maximum record number written is also kept separately for each rVariable in the CDF. `max_rec` is the largest of these. Some rVariables may have fewer records actually written. `CDF_lib` (Internal Interface) may be used to inquire the maximum record written for an individual rVariable (see Section 5.8.24).
`num_vars`	The number of rVariables in the CDF.

num_attrs	The number of attributes in the CDF.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example will inquire the basic information about a CDF.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
INTEGER*4 num_dims                  ! Number of dimensions, rVariables.
INTEGER*4 dim_sizes(CDF_MAX_DIMS)   ! Dimension sizes, rVariables
                                    ! (allocate to allow the maximum
                                    ! number of dimensions).
INTEGER*4 encoding                  ! Data encoding.
INTEGER*4 majority                  ! Variable majority.
INTEGER*4 max_rec                   ! Maximum record number.
INTEGER*4 num_vars                  ! Number of rVariables in CDF.
INTEGER*4 num_attrs                 ! Number of attributes in CDF.
.
.
CALL CDF_inquire (id, num_dims, dim_sizes, encoding, majority,
.                 max_rec, num_vars, num_attrs, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.5 CDF_close (Standard Interface)

```
SUBROUTINE CDF_close (id, status)

INTEGER*4 id             ! in  -- CDF identifier.
INTEGER*4 status         ! out -- Completion status.
```

CDF_close closes the specified CDF. The CDF's data buffers are flushed, the CDF's open file is closed (or files in the case of a multi-file CDF), and the CDF identifier is made available for reuse.

You MUST close a CDF with CDF_close to guarantee that all modifications you have made will actually be written to the CDF's file(s). If your program exits, normally or otherwise, without a successful call to CDF_close, the CDF's data buffers are left unflushed. If you have only read from a CDF, and not written to it, failure to close it is of no consequence. Of course, it is always good practice to close your CDFs before exiting your application even if they are only read.

The arguments to CDF_close are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example will close an open CDF.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
.
.
CALL CDF_close (id, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.6 CDF_delete (Standard Interface)

```
SUBROUTINE CDFdelete (id, status)

INTEGER*4 id             ! in  -- CDF identifier.
INTEGER*4 status         ! out -- Completion status.
```

CDF_delete deletes the specified CDF. The CDF files deleted include the .cdf file and if the CDF consists of multiple files, the .v0,.v1,... files and .z0,.z1,... files.

You must open a CDF before you are allowed to deleted it. If you have no privilege to delete the CDF file(s), they will not be deleted. If the CDF is corrupted and cannot be opened, the CDF file(s) must be deleted at the command line.

The arguments to CDF_delete are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example will open and then delete an existing CDF.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
.
.
CALL CDF_open ('test2', id, status)
IF (status .LT. CDF_OK) THEN        ! INFO status codes ignored.
  CALL UserStatusHandler (status)
ELSE
  CALL CDF_delete (id, status)
  IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
END IF
.
.
```

5.8.7 CDF_error (Standard Interface)

```
SUBROUTINE CDF_error (status, message)

INTEGER*4 status                        ! in  -- Status code.
CHARACTER message*(CDF_STATUSTEXT_LEN)  ! out -- Explanation text for
                                                 the status code.
```

`CDF_error` is used to inquire the explanation of a given status code (not just error codes). Section 5.9 explains how to interpret status codes and Appendix C lists all of the possible status codes.

The arguments to `CDF_error` are defined as follows.

`status`	The status code to check.
`message`	The explanation of the status code. This character string must be large enough to hold `CDF_STATUSTEXT_LEN` characters and will be blank padded if necessary.

Example(s)

The following example displays the explanation text if an error code is returned from a call to `CDF_open`.

```
.
```

```
      .
      INCLUDE '<path>cdf.inc'
      .
      .
      INTEGER*4 id                          ! CDF identifier.
      INTEGER*4 status                      ! Returned status code.
      CHARACTER text*(CDF_STATUSTEXT_LEN)   ! Explanation text.
      INTEGER*4 last_char                   ! Last character position
                                            ! actually used in the copyright.
      .
      .
      CALL CDF_open ('giss_wetl', id, status)
      IF (status .LT. CDF_WARN) THEN         ! INFO and WARNING codes ignored.
        CALL CDF_error (status, text)
        last_char = CDF_STATUSTEXT_LEN
        DO WHILE (text(last_char:last_char) .EQ. ' ')
          last_char = last_char - 1
        END DO
        WRITE (6,101) text(1:last_char)
 101    FORMAT (' ','ERROR> ',A)
      END IF
      .
      .
```

5.8.8 CDF_attr_create (Standard Interface)

```
SUBROUTINE CDF_attr_create (id, attr_name, attr_scope, attr_num, status)
INTEGER*4 id                  ! in  -- CDF identifier.
CHARACTER attr_name*(*)       ! in  -- Attribute name.
INTEGER*4 attr_scope          ! in  -- Scope of attribute.
INTEGER*4 attr_num            ! out -- Attribute number.
INTEGER*4 status              ! out -- Completion status.
```

`CDF_attr_create` creates an attribute in the specified CDF. An attribute with the same name must not already exist in the CDF.

The arguments to `CDF_attr_create` are defined as follows.

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDF_create` or `CDF_open`.
`attr_name`	The name of the attribute to create. This may be at most `CDF_ATTR_NAME_LEN` characters. Attribute names are case-sensitive.
`attr_scope`	The scope of the new attribute. Specify one of the scopes described in Section 5.7.7.
`attr_num`	The number assigned to the new attribute. This number must be used in subsequent CDF function calls when referring to this attribute. An existing attribute's number may be determined with the `CDF_attr_num` function.

`status` The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example creates two attributes. The `TITLE` attribute is created with global scope — it applies to the entire CDF (most likely the title of the data set stored in the CDF). The `Units` attribute is created with variable scope — each entry describes some property of the corresponding variable (in this case the units for the data).

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
CHARACTER UNITS_attr_name*5         ! Name of "Units" attribute.
INTEGER*4 UNITS_attr_num            ! "Units" attribute number.
INTEGER*4 TITLE_attr_num            ! "TITLE" attribute number.
INTEGER*4 TITLE_attr_scope          ! "TITLE" attribute scope.

DATA UNITS_attr_name/'Units'/, TITLE_attr_scope/GLOBAL_SCOPE/
.
.
CALL CDF_attr_create (id, 'TITLE', TITLE_attr_scope, TITLE_attr_num, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL CDF_attr_create (id, UNITS_attr_name, VARIABLE_SCOPE, UNITS_attr_num,
1                     status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.9 CDF_attr_num (Standard Interface)

```
INTEGER*4 FUNCTION CDF_attr_num (id, attr_name)

INTEGER*4 id                ! in --  CDF id.
CHARACTER attr_name*(*)     ! in -- attribute name.
```

`CDF_attr_num` is used to determine the attribute number associated with a given attribute name. If the attribute is found, `CDF_attr_num` returns its number — which will be equal to or greater than one (1). If an error occurs (eg. the attribute name does not exist in the CDF), an error code is returned. Error codes are less than zero (0).

The arguments to `CDF_attr_num` are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
attr_name	The name of the attribute for which to search. This may be at most CDF_ATTR_NAME_LEN characters. Attribute names are case-sensitive.

CDF_attr_num may be used as an embedded function call when an attribute number is needed. CDF_attr_num is declared in cdf.inc (FORTRAN functions must be declared so that the returned value is interpreted correctly).

NOTE: Some FORTRAN compilers give a warning if CDF_attr_num is not used in a routine where cdf.inc has been included. This warning may be ignored. To eliminate the warning message, remove the declaration of CDF_attr_num from cdf.inc but be sure to declare CDF_attr_num in all routines that use it.

Example(s)

In the following example the attribute named pressure will be renamed PRESSURE with CDF_attr_num being used as an embedded function call.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
.
.
CALL CDF_attr_rename (id, CDF_attr_num(id,'pressure'), 'PRESSURE', status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

Note that if the attribute pressure did not exist in the CDF, the call to CDF_attr_num would have returned an error code. Passing that error code to CDF_attr_rename as an attribute number would have resulted in CDF_attr_rename also returning an error code. CDF_attr_rename is described in Section 5.8.10.

5.8.10 CDF_attr_rename (Standard Interface)

```
SUBROUTINE CDF_attr_rename (id, attr_num, attr_name, status)

INTEGER*4 id              ! in  -- CDF identifier.
INTEGER*4 attr_num        ! in  -- Attribute number.
CHARACTER attr_name*(*)   ! in  -- New attribute name.
INTEGER*4 status          ! out -- Completion status.
```

CDF_attr_rename is used to rename an existing attribute. An attribute with the new name must not already exist in the CDF.

The arguments to CDF_attr_rename are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
attr_num	The number of the attribute to rename. This number may be determined with a call to CDF_attr_num (see Section 5.8.9).
attr_name	The new attribute name. This may be at most CDF_ATTR_NAME_LEN characters. Attribute names are case-sensitive.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

In the following example the attribute named LAT is renamed LATITUDE.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
.
.
CALL CDF_attr_rename (id, CDF_attr_num(id,'LAT'), 'LATITUDE', status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.11 CDF_attr_inquire (Standard Interface)

```
SUBROUTINE CDF_attr_inquire (id, attr_num, attr_name, attr_scope, max_entry,
1                            status)
INTEGER*4 id                              ! in  -- CDF identifier.
INTEGER*4 attr_num                        ! in  -- Attribute number.
CHARACTER attr_name*(CDF_ATTR_NAME_LEN)   ! out -- Attribute name.
INTEGER*4 attr_scope                      ! out -- Attribute scope.
INTEGER*4 max_entry                       ! out -- Maximum entry
                                          !        number for attribute.
INTEGER*4 status                          ! out -- Completion status.
```

CDF_attr_inquire is used to inquire about the specified attribute. To inquire about a specific attribute entry, use CDF_attr_entry_inquire (Section 5.8.12).

The arguments to CDF_attr_inquire are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
attr_num	The number of the attribute to inquire. This number may be determined with a call to CDF_attr_num (see Section 5.8.9).
attr_name	The attribute's name. This character string must be large enough to hold CDF_ATTR_NAME_LEN characters and will be blank padded if necessary.
attr_scope	The scope of the attribute. Attribute scopes are defined in Section 5.7.7.
max_entry	For gAttributes this is the maximum entry number used. For vAttributes this is the maximum rEntry number used. In either case this may not correspond with the number of entries (if some entry numbers were not used). The number of entries actually used may be inquired with the CDF_lib function (Internal Interface, see Section 5.8.24).
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example displays the name of each attribute in a CDF. The number of attributes in the CDF is first determined using CDF_inquire. Note that attribute numbers start at one (1) and are consecutive.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                            ! CDF identifier.
INTEGER*4 status                        ! Returned status code.
INTEGER*4 num_dims                      ! Number of dimensions.
INTEGER*4 dim_sizes(CDF_MAX_DIMS)       ! Dimension sizes (allocate to
                                        ! allow the maximum number of
                                        ! dimensions).
INTEGER*4 encoding                      ! Data encoding.
INTEGER*4 majority                      ! Variable majority.
INTEGER*4 max_rec                       ! Maximum record number in CDF.
INTEGER*4 num_vars                      ! Number of variables in CDF.
INTEGER*4 num_attrs                     ! Number of attributes in CDF.
INTEGER*4 attr_n                        ! Attribute number.
CHARACTER attr_name*(CDF_ATTR_NAME_LEN) ! Attribute name.
INTEGER*4 attr_scope                    ! Attribute scope.
INTEGER*4 max_entry                     ! Maximum entry number.
.
.
CALL CDF_inquire (id, num_dims, dim_sizes, encoding, majority,
1                 max_rec, num_vars, num_attrs, status)
```

```
   IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

   DO attr_n = 1, num_attrs
     CALL CDF_attr_inquire (id, attr_n, attr_name, attr_scope, max_entry,
     1                      status)
     IF (status .LT. CDF_OK) THEN       ! INFO status codes ignored.
       CALL UserStatusHandler (status)
     ELSE
       WRITE (6,10) attr_name
10     FORMAT (' ',A)
     END IF
   END DO
   .
   .
```

5.8.12 CDF_attr_entry_inquire (Standard Interface)

```
SUBROUTINE CDF_attr_entry_inquire (id, attr_num, entry_num, data_type,
1                                  num_elements, status)
INTEGER*4 id               ! in  -- CDF identifier.
INTEGER*4 attr_num         ! in  -- Attribute number.
INTEGER*4 entry_num        ! in  -- Entry number.
INTEGER*4 data_type        ! out -- Data type.
INTEGER*4 num_elements     ! out -- Number of elements (of the data type).
INTEGER*4 status           ! out -- Completion status.
```

CDF_attr_entry_inquire is used to inquire about a specific attribute entry. To inquire about the attribute in general, use CDF_attr_inquire (Section 5.8.11). CDF_attr_entry_inquire would normally be called before calling CDF_attr_get in order to determine the data type and number of elements (of that data type) for an entry. This would be necessary to correctly allocate enough memory to receive the value read by CDF_attr_get.

The arguments to CDF_attr_entry_inquire are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
attr_num	The attribute number for which to inquire an entry. This number may be determined with a call to CDF_attr_num (see Section 5.8.9).
entry_num	The entry number to inquire. If the attribute is global in scope, this is simply the entry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the entry).
data_type	The data type of the specified entry. The data types are defined in Section 5.7.3.
num_elements	The number of elements of the data type. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. For

all other data types, this is the number of elements in an array of that data type.

`status` The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example inquires each entry for an attribute. Note that entry numbers need not be consecutive — not every entry number between one (1) and the maximum entry number must exist. For this reason, `NO_SUCH_ENTRY` is an expected error code. Note also that if the attribute is variable in scope, then the rEntry numbers are actually rVariable numbers.

```
   .
   .
   INCLUDE '<path>cdf.inc'
   .
   .
   INTEGER*4 id                              ! CDF identifier.
   INTEGER*4 status                          ! Returned status code.
   INTEGER*4 attr_n                          ! Attribute number.
   INTEGER*4 entryN                          ! Entry number.
   CHARACTER attr_name*(CDF_ATTR_NAME_LEN)   ! Attribute name.
   INTEGER*4 attr_scope                      ! Attribute scope.
   INTEGER*4 max_entry                       ! Maximum entry number used.
   INTEGER*4 data_type                       ! Data type.
   INTEGER*4 num_elems                       ! Number of elements (of the
                                             ! data type).
   .
   .
   attr_n = CDF_attr_num (id, 'TMP')
   IF (attr_n .LT. 1) CALL UserStatusHandler (attr_n)  ! If less than one (1),
                                                       ! then it must be a
                                                       ! warning/error code.

   CALL CDF_attr_inquire (id, attr_n, attr_name, attr_scope, max_entry, status)
   IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

   DO entryN = 1, max_entry
     CALL CDF_attr_entry_inquire (id, attr_n, entryN, data_type, num_elems,
     1                            status)
     IF (status .LT. CDF_OK) THEN
       IF (status .NE. NO_SUCH_ENTRY) CALL UserStatusHandler (status)
     ELSE
C      (process entries)
       .
       .
     END IF
   END DO
```

5.8.13 CDF_attr_put (Standard Interface)

```
SUBROUTINE CDF_attr_put (id, attr_num, entry_num, data_type, num_elements,
1                        value, status)

INTEGER*4 id,              ! in --  CDF identifier.
INTEGER*4 attr_num         ! in --  Attribute number.
INTEGER*4 entry_num        ! in --  Entry number.
INTEGER*4 data_type        ! in --  Data type of the entry.
INTEGER*4 num_elements     ! in --  Number of elements (of the data type).
<type>    value            ! in --  Value (<type> depends on the data type
                           !        of the entry).
INTEGER*4 status           ! out -- Completion status.
```

CDF_attr_put is used to write an attribute entry to a CDF. The entry may or may not already exist. If it does exist, it is overwritten. The data type and number of elements (of that data type) may be changed when overwriting an existing entry.

The arguments to CDF_attr_put are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
attr_num	The attribute number. This number may be determined with a call to CDF_attr_num (see Section 5.8.9).
entry_num	The entry number. If the attribute is global in scope, this is simply the entry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).
data_type	The data type of the entry. Specify one of the data types defined in Section 5.7.3.
num_elements	The number of elements of the data type. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. For all other data types, this is the number of elements in an array of that data type.
value	The value(s) to write. The value is written to the CDF from memory addres value.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example writes two attribute entries. The first is to gEntry number one (1) of the gAttribute TITLE. The second is to vAttribute VALIDs for the rEntry which corresponds to the rVariable TMP.

.
.

```
INCLUDE '<path>cdf.inc'
.
.
PARAMETER TITLE_LEN = 10           ! Length of CDF title.
.
.
INTEGER*4 id                       ! CDF identifier.
INTEGER*4 status                   ! Returned status code.
INTEGER*4 entry_num                ! Entry number.
INTEGER*4 num_elements             ! Number of elements (of data type).
CHARACTER title*(TITLE_LEN)        ! Value of TITLE attribute, rEntry
                                   ! number 1.
INTEGER*2 TMPvalids(2)             ! Value(s) of VALIDs attribute,
                                   ! rEntry for rVariable TMP

DATA title/'CDF title.'/, TMPvalids/15,30/
.
.
entry_num = 1
CALL CDF_attr_put (id, CDF_attr_num(id,'TITLE'), entry_num, CDF_CHAR,
1                  TITLE_LEN, title, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
num_elements = 2
CALL CDF_attr_put (id, CDF_attr_num(id,'VALIDs'), CDF_var_num(id,'TMP'),
1                  CDF_INT2, num_elements, TMPvalids, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.14 CDF_attr_get (Standard Interface)

```
SUBROUTINE CDF_attr_get (id, attr_num, entry_num, value, status)

INTEGER*4 id             ! in  -- CDF identifier.
INTEGER*4 attr_num       ! in  -- Attribute number.
INTEGER*4 entry_num      ! in  -- Entry number.
<type>    value          ! out -- Value (<type> is dependent on the data
                         !        type of the entry).
INTEGER*4 status         ! out -- Completion status.
```

`CDF_attr_get` is used to read an attribute entry from a CDF. In most cases it will be necessary to call `CDF_attr_entry_inquire` before calling `CDF_attr_get` in order to determine the data type and number of elements (of that data type) for the entry.

The arguments to `CDF_attr_get` are defined as follows.

`id` The identifier of the CDF. This identifier must have been initialized by a call

to CDF_create or CDF_open.

attr_num	The attribute number. This number may be determined with a call to CDF_attr_num (see Section 5.8.9).
entry_num	The entry number. If the attribute is global in scope, this is simply the entry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).
value	The value read. This buffer must be large enough to hold the value. CDF_attr_entry_inquire would be used to determine the entry data type and number of elements (of that data type). The value is read from the CDF and placed into memory at address value.

Example(s)

The following example displays the value of the UNITS attribute for the rEntry corresponding to the PRES_LVL rVariable (but only if the data type is CDF_CHAR).

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
INTEGER*4 attr_n                    ! Attribute number.
INTEGER*4 entryN                    ! Entry number.
INTEGER*4 data_type                 ! Data type.
INTEGER*4 num_elems                 ! Number of elements (of data type).
CHARACTER buffer*100                ! Buffer to receive value (in
                                    ! this case it is assumed that 100
                                    ! characters is enough).
.
.
attr_n = CDF_attr_Num (id, 'UNITS')
IF (attr_n .LT. 0) CALL UserStatusHandler (attr_n)  ! If less than one (1),
                                                    ! then it must be a
                                                    ! warning/error code.

entryN = CDF_var_num (id, 'PRES_LVL')              ! The rEntry number is
                                                   ! the rVariable number.
IF (entryN .LT. 0) CALL UserStatusHandler (entryN) ! If less than one (1),
                                                   ! then it must be a
                                                   ! warning/error code.

CALL CDF_attr_entry_inquire (id, attr_n, entryN, data_type, num_elems,
1                            status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
```

```
   IF (data_type .EQ. CDF_CHAR) THEN
     CALL CDF_attr_get (id, attr_n, entryN, buffer, status)
     IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
     WRITE (6,10) buffer(1:num_elems)
10   FORMAT (' ',A)
   END IF
   .
   .
```

5.8.15 CDF_attr_correct_scope (Standard Interface)

```
SUBROUTINE CDF_attr_correct_scope (id, attr_num, attr_scope, status)

INTEGER*4 id                  ! in  -- CDF identifier.
INTEGER*4 attr_num            ! in  -- Attribute number.
INTEGER*4 attr_scope          ! in  -- New scope of attribute.
INTEGER*4 status              ! out -- Completion status.
```

CDF_attr_correct_scope is used to correct the scope of an attribute. Section 2.8.3 describes attribute scopes and the need for this function.

The arguments to CDF_attr_correct_scope are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
attr_num	The attribute number. This number may be determined with a call to CDF_attr_num (see Section 5.8.9).
attr_scope	The new scope for the attribute. Specify one of the definite scopes defined in Section 5.7.7.
status	The completion status code. Section 5.9 explains how to interpret status codes.

IMPORTANT NOTE

This function is not a permanent one — it may be deleted from a future release of the CDF library. Version 1 CDFs should be converted to Version 2 CDFs as soon as possible. Any assumed attribute scopes should then be corrected by using this function or the CDFscope toolkit program.

Example(s)

The following example will correct the scope of every attribute in a CDF. Assumed scopes will be corrected to the corresponding definite scope. Note that it may not always be the case that an assumed scope should be corrected to the corresponding definite scope (see Section 2.8.3). The toolkit program CDFscope could also be used to correct attribute scopes.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                            ! CDF identifier.
INTEGER*4 status                        ! Returned status code.
INTEGER*4 num_dims                      ! Number of dimensions.
INTEGER*4 dim_sizes(CDF_MAX_DIMS)       ! Dimension sizes (allocate to allow
                                        ! the maximum number of dimensions).
INTEGER*4 encoding                      ! Data encoding.
INTEGER*4 majority                      ! Variable majority.
INTEGER*4 max_rec                       ! Maximum record number in CDF.
INTEGER*4 num_vars                      ! Number of variables in CDF.
INTEGER*4 num_attrs                     ! Number of attributes in CDF.
INTEGER*4 attr_n                        ! Attribute number.
CHARACTER attr_name*(CDF_ATTR_NAME_LEN) ! Attribute name.
INTEGER*4 attr_scope                    ! Attribute scope.
INTEGER*4 max_entry                     ! Maximum entry number.
.
.
CALL CDF_inquire (id, num_dims, dim_sizes, encoding, majority,
1                 max_rec, num_vars, num_attrs, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

DO attr_n = 1, num_attrs
  CALL CDF_attr_inquire (id, attr_n, attr_name, attr_scope, max_entry,
  1                      status)
  IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

  IF (attr_scope .EQ. GLOBAL_SCOPE_ASSUMED) THEN
    CALL CDF_attr_correct_scope (id, attr_n, GLOBAL_SCOPE, status)
    IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
  ELSE
    IF (attr_scope .EQ. VARIABLE_SCOPE_ASSUMED) THEN
      CALL CDF_attr_correct_scope (id, attr_n, VARIABLE_SCOPE, status)
      IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
    END IF
  END IF
END DO
.
.
```

5.8.16 CDF_var_create (Standard Interface)

```
SUBROUTINE CDF_var_create (id, var_name, data_type, num_elements,
1                          rec_variance, dim_variances, var_num, status)

INTEGER*4 id                 ! in --  CDF identifier.
```

```
CHARACTER var_name*(*)       ! in --  rVariable name.
INTEGER*4 data_type          ! in --  Data type.
INTEGER*4 num_elements       ! in --  Number of elements (of the data type).
INTEGER*4 rec_variance       ! in --  Record variance.
INTEGER*4 dim_variances(*)   ! in --  Dimension variances.
INTEGER*4 var_num            ! out -- rVariable number.
INTEGER*4 status             ! out -- Completion status.
```

CDF_var_create is used to create a new rVariable in a CDF. A variable (rVariable or zVariable) with the same name must not already exist in the CDF.

The arguments to CDF_var_create are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_name	The name of the rVariable to create. This may be at most CDF_VAR_NAME_LEN characters. Variable names are case-sensitive.
data_type	The data type of the new rVariable. Specify one of the data types defined in Section 5.7.3.
num_elements	The number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (each value consists of the entire string). For all other data types, this must always be one (1) — multiple elements at each value are not allowed for non-character string data types.
rec_variance	The variable's record variance. Specify one of the variances defined in Section 5.7.6.
dim_variances	The variable's dimension variances. dim_variances is a 1-dimensional array containing one element per rVariable dimension. Each element of dim_variances specifies the corresponding dimension variance. For each dimension, specify one of the variances defined in Section 5.7.6. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
var_num	The number assigned to the new rVariable. This number must be used in subsequent CDF function calls when referring to this rVariable. An existing rVariable's number may be determined with the CDF_var_num function.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example will create several rVariables in a CDF. In this case, EPOCH, LAT, and LON are independent rVariables and TMP is a dependent rVariable.

```
.
.
INCLUDE '<path>cdf.inc'
```

```
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
INTEGER*4 EPOCH_rec_vary            ! EPOCH record variance.
INTEGER*4 LAT_rec_vary              ! LAT record variance.
INTEGER*4 LON_rec_vary              ! LON record variance.
INTEGER*4 TMP_rec_vary              ! TMP record variance.
INTEGER*4 EPOCH_dim_varys(2)        ! EPOCH dimension variances.
INTEGER*4 LAT_dim_varys(2)          ! LAT dimension variances.
INTEGER*4 LON_dim_varys(2)          ! LON dimension variances.
INTEGER*4 TMP_dim_varys(2)          ! TMP dimension variances.
INTEGER*4 EPOCH_var_num             ! EPOCH variable number.
INTEGER*4 LAT_var_num               ! LAT rVariable number.
INTEGER*4 LON_var_num               ! LON rVariable number.
INTEGER*4 TMP_var_num               ! TMP rVariable number.

DATA EPOCH_rec_vary/VARY/, LAT_rec_vary/NOVARY/,
1    LON_rec_vary/NOVARY/, TMP_rec_vary/VARY/
DATA EPOCH_dim_varys/NOVARY,NOVARY/, LAT_dim_varys/NOVARY,VARY/,
1    LON_dim_varys/VARY,NOVARY/, TMP_dim_varys/VARY,VARY/
.
.
CALL CDF_var_create (id, 'EPOCH', CDF_EPOCH, 1,
1                    EPOCH_rec_vary, EPOCH_dim_varys, EPOCH_var_num, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL CDF_var_create (id, 'LATITUDE', CDF_INT2, 1,
1                    LAT_rec_vary, LAT_dim_varys, LAT_var_num, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL CDF_var_create (id, 'LONGITUDE', CDF_INT2, 1,
1                    LON_rec_vary, LON_dim_varys, LON_var_num, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL CDF_var_create (id, 'TEMPERATURE', CDF_REAL4, 1,
1                    TMP_rec_vary, TMP_dim_varys, TMP_var_num, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.17 CDF_var_num (Standard Interface)

```
INTEGER*4 FUNCTION CDF_var_num (id, var_name)

INTEGER*4 id                ! in -- CDF identifier.
CHARACTER var_name*(*)      ! in -- rVariable name.
```

CDF_var_num is used to determine the rVariable number associated with a given rVariable name. If the rVariable is found, CDF_var_num returns its number — which will be equal to or greater than one (1). If an error occurs (eg. the rVariable does not exist in the CDF), an error code is returned. Error codes are less than zero (0).

The arguments to CDF_var_num are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_name	The name of the rVariable for which to search. This may be at most CDF_VAR_NAME_LEN characters. Variable names are case-sensitive.

CDF_var_num may be used as an embedded function call when an rVariable number is needed. CDF_var_num is declared in cdf.inc (FORTRAN functions must be declared so that the returned value is interpreted correctly).

NOTE: Some FORTRAN compilers give a warning if CDF_var_num is not used in a routine where cdf.inc has been included. This warning may be ignored. To eliminate the warning message, remove the declaration of CDF_var_num from cdf.inc but be sure to declare CDF_var_num in all routines that use it.

Example(s)

In the following example CDF_var_num is used as an embedded function call when inquiring about an rVariable.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                            ! CDF identifier.
INTEGER*4 status                        ! Returned status code.
CHARACTER var_name*(CDF_VAR_NAME_LEN)   ! rVariable name.
INTEGER*4 data_type                     ! Data type of the rVariable.
INTEGER*4 num_elements                  ! Number of elements (of the
                                        ! data type).
INTEGER*4 rec_variances                 ! Record variance.
INTEGER*4 dim_variances(CDF_MAX_DIMS)   ! Dimension variances.
.
.
CALL CDF_var_inquire (id, CDF_var_num(id,'LATITUDE'), var_name, data_type,
1                     num_elements, rec_variance, dim_variances, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

In this example the rVariable named LATITUDE was inquired. Note that if LATITUDE did not exist in the CDF, the call to CDF_var_num would have returned an error code. Passing that error code to CDF_var_inquire

as an rVariable number would have resulted in CDF_var_inquire also returning an error code. Also note that the name written into var_name is already known (LATITUDE). In some cases the rVariable names will be unknown — CDF_var_inquire would be used to determine them. CDF_var_inquire is described in Section 5.8.19.

5.8.18 CDF_var_rename (Standard Interface)

```
SUBROUTINE CDF_var_rename (id, var_num, var_name, status)

INTEGER*4 id               ! in  -- CDF identifier.
INTEGER*4 var_num          ! in  -- rVariable number.
CHARACTER var_name*(*)     ! in  -- New name.
INTEGER*4 status           ! out -- Completion status.
```

CDF_var_rename is used to rename an existing rVariable. A variable (rVariable or zVariable) with the same name must not already exist in the CDF.

The arguments to CDF_var_rename are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable to rename. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
var_name	The new rVariable name. This may be at most CDF_VAR_NAME_LEN characters. Variables names are case-sensitive.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

In the following example the rVariable named TEMPERATURE is renamed TMP (if it exists). Note that if CDF_var_num returns a value less than one (1), that value is not an rVariable number but rather a warning/error code.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
INTEGER*4 var_num                   ! rVariable number.
.
.
var_num = CDF_var_num (id, 'TEMPERATURE')
```

```
IF (var_num .LT. 1) THEN
  IF (var_num .NE. NO_SUCH_VAR) CALL UserStatusHandler (var_num)
ELSE
  CALL CDF_var_rename (id, var_num, 'TMP', status)
  IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
END IF
.
.
```

5.8.19 CDF_var_inquire (Standard Interface)

```
SUBROUTINE CDF_var_inquire (id, var_num, var_name, data_type, num_elements,
1                           rec_variance, dim_variances, status)

INTEGER*4 id                            ! in  -- CDF identifier.
INTEGER*4 var_num                       ! in  -- rVariable number.
CHARACTER var_name*(CDF_VAR_NAME_LEN)   ! out -- rVariable name.
INTEGER*4 data_type                     ! out -- Data type.
INTEGER*4 num_elements                  ! out -- Number of elements (of the
                                        !        data type).
INTEGER*4 rec_variance                  ! out -- Record variance.
INTEGER*4 dim_variances(CDF_MAX_DIMS)   ! out -- Dimension variances.
INTEGER*4 status                        ! out -- Completion status.
```

CDF_var_inquire is used to inquire about the specified rVariable. This function would normally be used before reading rVariable values (with CDF_var_get or CDF_var_hyper_get) to determine the data type and number of elements (of that data type).

The arguments to CDF_var_inquire are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable to inquire. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
var_name	The rVariable's name. This character string must be large enough to hold CDF_VAR_NAME_LEN characters and will be blank padded if necessary.
data_type	The data type of the rVariable. The data types are defined in Section 5.7.3.
num_elements	The number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (each value consists of the entire string). For all other data types, this will always be one (1) — multiple elements at each value are not allowed for non-character string data types.
rec_variance	The record variance. The record variances are defined in Section 5.7.6.
dim_variances	The dimension variances. dim_variances is a 1-dimensional array containing one element per rVariable dimension. Each element of dim_variances receives

	the corresponding dimension variance. The dimension variances are defined in Section 5.7.6. For 0-dimensional rVariable this argument is ignored (but a place holder is necessary).
`status`	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example inquires about an rVariable named `HEAT_FLUX` in a CDF. Note that the rVariable name returned by `CDF_var_inquire` will be the same as that passed in to `CDF_var_num`.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                           ! CDF identifier.
INTEGER*4 status                       ! Returned status code.
CHARACTER var_name*(CDF_VAR_NAME_LEN)  ! rVariable name.
INTEGER*4 data_type                    ! Data type.
INTEGER*4 num_elems                    ! Number of elements (of data type).
INTEGER*4 rec_vary                     ! Record variance.
INTEGER*4 dim_varys(CDF_MAX_DIMS)      ! Dimension variances (allocate
                                       ! to allow the maximum number of
                                       ! dimensions).
.
.
CALL CDF_var_inquire (id, CDF_var_num(id,'HEAT_FLUX'), var_name, data_type,
1                     num_elems, rec_vary, dim_varys, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.20 CDF_var_put (Standard Interface)

```
SUBROUTINE CDF_var_put (id, var_num, rec_num, indices, value, status)

INTEGER*4 id             ! in -- CDF identifier.
INTEGER*4 var_num        ! in -- rVariable number.
INTEGER*4 rec_num        ! in -- Record number.
INTEGER*4 indices(*)     ! in -- Dimension indices.
<type>    value          ! in -- Value. (<type> depends on the data type
                         !       of the rVariable).
INTEGER*4 status         ! out -- Completion status.
```

`CDF_var_put` is used to write a single value to an rVariable. `CDF_var_hyper_put` may be used to write more than one rVariable value with a single call.

The arguments to CDF_var_put are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable to which to write. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
rec_num	The record number at which to write.
indices	The array indices within the specified record at which to write. indices is a 1-dimensional array containing one element per rVariable dimension. Each element of indices specifies the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
value	The value to write. The value is written to the CDF from memory address value.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example writes values to the rVariable LATITUDE of a CDF whose rVariables are 2-dimensional and have dimension sizes [360,181]. For LATITUDE, the record variance is NOVARY, the dimension variances are [NOVARY,VARY], and the data type is CDF_INT2.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
INTEGER*2 lat                       ! Latitude value.
INTEGER*4 var_n                     ! rVariable number.
INTEGER*4 rec_num                   ! Record number.
INTEGER*4 indices(2)                ! Dimension indices.

DATA rec_num/1/, indices/1,1/
.
.
var_n = CDF_var_num (id, 'LATITUDE')
IF (var_n .LT. 1) CALL UserStatusHandler (var_n)  ! If less than one (1),
                                                  ! then not an rVariable
                                                  ! number but rather a
                                                  ! warning/error code.
DO lat = -90, 90
   indices(2) = 91 + lat
   CALL CDF_var_put (id, var_n, rec_num, indices, lat, status)
   IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
```

```
END DO
.
.
```

Since the record variance is NOVARY, the record number (rec_num) is set to one (1). Also note that because the dimension variances are [NOVARY,VARY], only the second dimension is varied as values are written (the values are "virtually" the same at each index of the first dimension).

5.8.21 CDF_var_get (Standard Interface)

```
SUBROUTINE CDF_var_get (id, var_num, rec_num, indices, value, status)

INTEGER*4 id             ! in  -- CDF identifier.
INTEGER*4 var_num        ! in  -- rVariable number.
INTEGER*4 rec_num        ! in  -- Record number.
INTEGER*4 indices(*)     ! in  -- Dimension indices.
<type>    value          ! out -- Value (<type> depends on the data type
                         !        of the rVariable).
INTEGER*4 status         ! out -- Completion status.
```

CDF_var_get is used to read a single value from an rVariable. CDF_var_hyper_get may be used to read more than one rVariable value with a single call.

The arguments to CDF_var_get are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable from which to read. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
rec_num	The record number at which to read.
indices	The array indices within the specified record at which to read. indices is a 1-dimensional array containing one element per rVarible dimension. Each element of indices specifies the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
value	The value read. This buffer must be large enough to hold the value. CDF_var_inquire would be used to determine the rVariable's data type and number of elements (of that data type) at each value. The value is read from the CDF and placed into memory at address value.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example will read and hold an entire record of data from an rVariable. The CDF's rVariables are 3-dimensional with sizes [180,91,10]. For the variable, the record variance is VARY, the dimension

variances are [VARY,VARY,VARY], and the data type is CDF_REAL4.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
REAL*4    tmp(180,91,10)            ! Temperature values.
INTEGER*4 indices(3)                ! Dimension indices.
INTEGER*4 var_n                     ! rVariable number.
INTEGER*4 rec_num                   ! Record number.
INTEGER*4 d1, d2, d3                ! Dimension index values.
.
.
var_n = CDF_var_num (id, 'Temperature')
IF (var_n .LT. 1) CALL UserStatusHandler (var_n)  ! If less than one (1),
                                                  ! then it is actually a
                                                  ! warning/error code.

rec_num = 13

DO d1 = 1, 180
  indices(1) = d1
  DO d2 = 1, 91
    indices(2) = d2
    DO d3 = 1, 10
      indices(3) = d3
      CALL CDF_var_get (id, var_n, rec_num, indices, tmp(d1,d2,d3), status)
      IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
    END DO
  END DO
END DO
.
.
```

5.8.22 CDF_var_hyper_put (Standard Interface)

```
SUBROUTINE CDF_var_hyper_put (id, var_num, rec_start, rec_count, rec_interval,
1                             indices, counts, intervals, buffer, status)

INTEGER*4 id               ! in -- CDF identifier.
INTEGER*4 var_num          ! in -- rVariable number.
INTEGER*4 rec_start        ! in -- Starting record number.
INTEGER*4 rec_count        ! in -- Number of records.
INTEGER*4 rec_interval     ! in -- Interval between records.
INTEGER*4 indices(*)       ! in -- Dimension indices of starting value.
INTEGER*4 counts(*)        ! in -- Number of values along each dimension.
```

```
INTEGER*4 intervals(*)     ! in -- Interval between values along each
                           !       dimension.
<type>    buffer           ! in -- Buffer of values (<type> depends on the
                           !       data type of the rVariable.)
INTEGER*4 status           ! out -- Completion status.
```

CDF_var_hyper_put is used to write a buffer of one or more values to an rVariable. It is important to know the variable majority of the CDF before using CDF_var_hyper_put because the values in the buffer to be written must be in the same majority. CDF_inquire is used to determine the variable majority of a CDF. Section 2.7.9 describes the variable majorities.

The arguments to CDF_var_hyper_put are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable to which to write. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
rec_start	The record number at which to start writing.
rec_count	The number of records to write.
rec_interval	The interval between records for subsampling[2] (eg. an interval of 2 means write to every other record).
indices	The indices (within each record) at which to start writing. indices is a 1-dimensional array containing one element per rVariable dimension. Each element of indices specifies the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
counts	The number of values along each dimension to write. counts is a 1-dimensional array containing one element per rVariable dimension. Each element of counts specifies the corresponding dimension count. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
intervals	For each dimension, the interval between values for subsampling[3] (eg. an interval of 2 means write to every other value). intervals is a 1-dimensional array containing one element per rVariable dimension. Each element of intervals specifies the corresponding dimension interval. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
buffer	The buffer of values to write. The majority of the values in this buffer must be the same as that of the CDF. The values starting at memory address buffer are written to the CDF.
status	The completion status code. Section 5.9 explains how to interpret status codes.

[2] "Subsampling" is not the best term to use when writing data but you should know what we mean.
[3] Again, not the best term.

Example(s)

The following example writes values to the rVariable `LATITUDE` in a CDF whose rVariables are 2-dimensional and have dimension sizes `[360,181]`. For `LATITUDE`, the record variance is `NOVARY`, the dimension variances are `[NOVARY,VARY]`, and the data type is `CDF_INT2`. This example is similar to the example in Section 5.8.20 except that it uses a single call to `CDF_var_hyper_put` rather than numerous calls to `CDF_var_put`.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
INTEGER*2 lat                       ! Latitude value.
INTEGER*2 lats(181)                 ! Buffer of latitude values.
INTEGER*4 var_n                     ! rVariable number.
INTEGER*4 rec_start                 ! Record number.
INTEGER*4 rec_count                 ! Record counts.
INTEGER*4 rec_interval              ! Record interval.
INTEGER*4 indices(2)                ! Dimension indices.
INTEGER*4 counts(2)                 ! Dimension counts.
INTEGER*4 intervals(2)              ! Dimension intervals.

DATA rec_start/1/, rec_count/1/, rec_interval/1/,
1    indices/1,1/, counts/1,181/, intervals/1,1/
.
.
var_n = CDF_var_num (id, 'LATITUDE')
IF (var_n .LT. 1) CALL UserStatusHandler (var_n)  ! If less than one (1),
                                                  ! then not an rVariable
                                                  ! number but rather a
                                                  ! warning/error code
DO lat = -90, 90
  lats(91+lat) = lat
END DO

CALL CDF_var_hyper_put (id, var_n, rec_start, rec_count, rec_interval,
1                       indices, counts, intervals, lats, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.23 CDF_var_hyper_get (Standard Interface)

```
SUBROUTINE CDF_var_hyper_get (id, var_num, rec_start, rec_count, rec_interval,
1                             indices, counts, intervals, buffer, status)
```

```
INTEGER*4 id                ! in -- CDF identifier.
INTEGER*4 var_num           ! in -- rVariable number.
INTEGER*4 rec_start         ! in -- Starting record number.
INTEGER*4 rec_count         ! in -- Number of records.
INTEGER*4 rec_interval      ! in -- Subsampling interval between records.
INTEGER*4 indices(*)        ! in -- Dimension indices of starting value.
INTEGER*4 counts(*)         ! in -- Number of values along each dimension.
INTEGER*4 intervals(*)      ! in -- Subsampling intervals along each dimension.
<type>    buffer            ! out -- Buffer of values (<type> depends on the
                            !        data type of the rVariable).
INTEGER*4 status            ! out -- Completion status.
```

CDF_var_hyper_get is used to read a buffer of one or more values from an rVariable. It is important to know the variable majority of the CDF before using CDF_var_hyper_get because the values placed into the buffer will be in that majority. CDF_var_inquire is used to determine the variable majority of a CDF. Section 2.7.9 describes the variable majorities.

The arguments to CDF_var_hyper_get are defined as follows.

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable from which to read. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
rec_start	The record number at which to start reading.
rec_count	The number of records to read.
rec_interval	The interval between records for subsampling (eg. an interval of 2 means read every other record).
indices	The indices (within each record) at which to start reading. indices is a 1-dimensional array containing one element per rVariable dimension. Each element of indices specifies the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
counts	The number of values along each dimension to read. counts is a 1-dimensional array containing one element per rVariable dimension. Each element of counts specifies the corresponding dimension count. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
intervals	For each dimension, the interval between values for subsampling (eg. an interval of 2 means read every other value). intervals is a 1-dimensional array containing one element per rVariable dimension. Each element of intervals specifies the corresponding dimension interval. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
buffer	The buffer of values read. The majority of the values in this buffer will be the same as that of the CDF. This buffer must be large enought to hold the values. The values are read from the CDF and placed into memory starting at address buffer. CDF_var_inquire would be used to determine the variable's data type and number of elements (of that data type) at each value.

`status` The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example will read an entire record of data from an rVariable. The CDF's rVariables are 3-dimensional with sizes `[180,91,10]` and the CDF's variable majority is `COL_MAJOR`. For the variable, the record variance is `VARY`, the dimension variances are `[VARY,VARY,VARY]`, and the data type is `CDF_REAL4`. This example is similar to the example in Section 5.8.21 except that it uses a single call to `CDF_var_hyper_get` rather than numerous calls to `CDF_var_get`.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                      ! CDF identifier.
INTEGER*4 status                  ! Returned status code.
REAL*4    tmp(180,91,10)          ! Temperature values.
INTEGER*4 var_n                   ! rVariable number.
INTEGER*4 rec_start               ! Record number.
INTEGER*4 rec_count               ! Record counts.
INTEGER*4 rec_interval            ! Record interval.
INTEGER*4 indices(3)              ! Dimension indices.
INTEGER*4 counts(3)               ! Dimension counts.
INTEGER*4 intervals(3)            ! Dimension intervals.

DATA rec_start/13/, rec_count/1/, rec_interval/1/,
1    indices/1,1,1/, counts/180,91,10/, intervals/1,1,1/
.
.
var_n = CDF_var_num (id, 'Temperature')
IF (var_n .LT. 1) CALL UserStatusHandler (var_n)  ! If less than one (1),
                                                  ! then it is actually a
                                                  ! warning/error code.

CALL CDF_var_hyper_get (id, var_n, rec_start, rec_count, rec_interval,
1                       indices, counts, intervals, tmp, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

Note that if the CDF's variable majority had been `ROW_MAJOR`, the `tmp` array would have been declared `REAL*4 tmp(10,91,180)` for proper indexing.

5.8.24 CDF_lib (Internal Interface)

```
SUBROUTINE CDF_lib (..., status)
```

```
INTEGER*4 status                ! out -- Completion status.
```

The Internal Interface consists of only one function, CDF_lib. CDF_lib can be used to perform all possible operations on a CDF. (In fact, all of the Standard Interface functions are implemented using the Internal Interface). CDF_lib must be used to perform operations not possible with the Standard Interface functions. These operations would involve CDF features added after the Standard Interface functions had been defined (eg. specifying a single-file format for a CDF, accessing zVariables, or specifying a pad value for an rVariable or zVariable). Note that CDF_lib can also be used to perform certain operations more efficiently than with the Standard Interface functions.

CDF_lib takes a variable number of arguments which specify one or more operations to be performed. The operations are performed according to the order of the arguments. Each operation is performed on an item. An item may be either an object (eg. a CDF, variable, or attribute) or a state (eg. a CDF's format or a variable's data type/number of elements). The possible operations and corresponding items (on which to perform those operations) are described in detail later in this section.

Example(s)

The easiest way to explain how to use CDF_lib would be to start with a few examples. The following example will show how a CDF would be created with the single-file format.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
CHARACTER CDF_name*5                ! Name of the CDF.
INTEGER*4 num_dims                  ! Number of dimensions.
INTEGER*4 dim_sizes(2)              ! Dimension sizes.
INTEGER*4 encoding                  ! Data encoding.
INTEGER*4 majority                  ! Variable data majority.
INTEGER*4 format                    ! Format of CDF.

DATA CDF_name/'test1'/, num_dims/2/, dim_sizes/100,200/,
1    encoding/HOST_ENCODING/, majority/ROW_MAJOR/, format/SINGLE_FILE/
.
.
CALL CDF_create (CDF_name, num_dims, dim_sizes, encoding, majority,
1                id, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL CDF_lib (SELECT_, CDF_, id,
1             PUT_, CDF_FORMAT_, format,
2             NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
```

```
.
.
```

The call to `CDF_create` created the CDF as expected but the default format is `MULTI_FILE` (a separate file for each variable in the CDF) The call to `CDF_lib` is then used to change the format to `SINGLE_FILE` (which must be done before any variables are created in the CDF).

The arguments to `CDF_lib` in this example are explained as follows.

`SELECT_`	The first operation to be performed. In this case an item is going to be selected. `SELECT_` is defined in `cdf.inc` (as are all CDF constants).
`CDF_`	The item to be selected — the current CDF (the CDF on which subsequent operations are to be performed).
`id`	The identifier of the CDF being selected.
`PUT_`	The next operation to be performed. In this case an item is going to be put to the CDF (a new format). `PUT_` is defined in `cdf.inc` (as are all CDF constants).
`CDF_FORMAT_`	The item to be put. In this case it is the CDF's format. One required argument for this operation follows.
`format`	The actual format for the CDF. Depending on the item being put, one or more arguments would have been necessary. In this case only one argument is necessary. `format` contains the value `SINGLE_FILE`.
`NULL_`	This argument could have been one of two things. It could have been another item to put (followed by the arguments required for that item) or it could have been a new operation to perform. In this case it is a new operation to perform — the `NULL_` operation. `NULL_` indicates the end of the call to `CDF_lib`. Specifying `NULL_` at the end of the argument list is required because not all FORTRAN compilers/operating systems provide the ability for a called function to determine how many arguments were passed in by the calling function.
`status`	The completion status code. Section 5.9 explains how to interpret status codes.

The next example shows how the same CDF could have been created using only one call to `CDF_lib` (the declarations would be the same).

```
.
.
CALL CDF_lib (CREATE_, CDF_, CDF_name, num_dims, dim_sizes, id,
1             PUT_, CDF_ENCODING_, encoding,
2                   CDF_MAJORITY_, majority,
3                   CDF_FORMAT_, format,
4             NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

The purpose of each argument is as follows.

CREATE_	The first operation to be performed. In this case something will be created.
CDF_	The item to be created — a CDF in this case. There are four required argument which must follow. When a CDF is created (with CDF_lib), the format defaults to multi-file (MULTI_FILE), the data encoding defaults to host (HOST_ENCODING), and the variable majority defaults to row major (ROW_MAJOR).
CDF_name	The name of the CDF.
num_dims	The number of dimensions in the CDF.
dim_sizes	The dimension sizes.
id	The identifier to be used when referencing the created CDF in subsequent operations.
PUT_	This argument could have been one of two things. Another item to create or a new operation to perform. In this case it is another operation to perform — something will be put.
CDF_ENCODING_	The item to be put — in this case the CDF's encoding. Note that the CDF did not have to be selected. It was automatically selected as the current CDF when it was created. One required argument follows.
encoding	The encoding to be put to the CDF.
CDF_MAJORITY_	This argument could have been one of two things. Another item to put or a new operation to perform. In this case it is another item to put — the CDF's majority. One required argument follows.
majority	The majority to be put to the CDF.
CDF_FORMAT_	Once again this argument could have been either another item to put or a new operation to perform. It is another item to put — the CDF's format. One required argument follows.
format	The format to be put to the CDF.
NULL_	This argument could have been either another item to put or a new operation to perform. Here it is another operation to perform — the NULL_ operation which ends the call to CDF_lib.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Note that the operations are performed in the order that they appear in the argument list. The CDF had to be created before the encoding, majority, and format could be specified (put).

Current Objects/States (Items)

The use of CDF_lib requires that an application be aware of the current objects/states maintained by the CDF library. The following current objects/states are used by the CDF library when performing operations...

CDF	A CDF operation is always performed on the current CDF. The current CDF is implicitly selected whenever a CDF is opened

or created. The current CDF may be explicitly selected using the <SELECT_,CDF_>[4] operation. There is no current CDF until one is opened or created (which implicitly selects it) or until one is explicitly selected.

NOTE: The current CDF is not maintained by the CDF library from one call to the next of CDFlib. The current CDF must be selected in each call to CDFlib (if an operation is to be performed on a CDF).

rVariable

An rVariable operation is always performed on the current rVariable in the current CDF. For each open CDF a current rVariable is maintained. This current rVariable is implicitly selected when an rVariable is created (in the current CDF) or it may be explicitly selected with the <SELECT_,rVAR_> or <SELECT_,rVAR_NAME_> operations. There is no current rVariable in a CDF until one is created (which implicitly selects it) or until one is explicitly selected.

zVariable

A zVariable operation is always performed on the current zVariable in the current CDF. For each open CDF a current zVariable is maintained. This current zVariable is implicitly selected when a zVariable is created (in the current CDF) or it may be explicitly selected with the <SELECT_,zVAR_> or <SELECT_,zVAR_NAME_> operations. There is no current zVariable in a CDF until one is created (which implicitly selects it) or until one is explicitly selected.

attribute

An attribute operation is always performed on the current attribute in the current CDF. For each open CDF a current attribute is maintained. This current attribute is implicitly selected when an attribute is created (in the current CDF) or it may be explicitly selected with the <SELECT_,ATTR_> or <SELECT_,ATTR_NAME_> operations. There is no current attribute in a CDF until one is created (which implicitly selects it) or until one is explicitly selected.

gEntry/rEntry number

An attribute gEntry/rEntry operation is always performed on the current gEntry/rEntry number in the current CDF for the current attribute in that CDF. The current gEntry/rEntry number has a dual purpose. For gAttributes it is the gEntry number at which to perform an operation. For vAttributes it is the rEntry number at which to perform an operation. For each open CDF a current gEntry/rEntry number is maintained. This current gEntry/rEntry number must be explicitly selected with the <SELECT_,gENTRY_> operation (for gAttributes) or <SELECT_,rENTRY_ operation (for vAttributes) — there is no implicit or default selection of the current gEntry/rEntry number for a CDF. Note that the current gEntry/rEntry number is maintained for the CDF (not each attribute) — it applies to all of the attributes in that CDF.

[4] This notation is used to specify an operation to be performed on an item. The syntax is: <operation,item>.

zEntry number

A vAttribute zEntry operation is always performed on the current zEntry number in the current CDF for the current attribute in that CDF. For each open CDF a current zEntry number is maintained. This current zEntry number must be explicitly selected with the <SELECT_,zENTRY_> operation (there is no implicit or default selection of the current zEntry number for a CDF). Note that the current zEntry number is maintained for the CDF (not each attribute) — it applies to all of the attributes in that CDF.

record number, rVariables

An rVariable read or write operation is always performed at (for single reads and writes) or starting at (for hyper reads and writes) the current record number for the rVariables in the current CDF. When a CDF is opened or created the current record number for its rVariables is initialized to zero (0). It may then be explicitly selected using the <SELECT_,rVARs_RECNUMBER_> operation. Note that the current record number for rVariables is maintained for a CDF (not each rVariable) — it applies to all of the rVariables in that CDF.

record count, rVariables

An rVariable hyper read or write operation is always performed using the current record count for the rVariables in the current CDF. When a CDF is opened or created the current record count for its rVariables is initialized to one (1). It may then be explicitly selected using the <SELECT_,rVARs_RECCOUNT_> operation. Note that the current record count for rVariables is maintained for a CDF (not each rVariable) — it applies to all of the rVariables in that CDF.

record interval, rVariables

An rVariable hyper read or write operation is always performed using the current record interval for the rVariables in the current CDF. When a CDF is opened or created the current record interval for its rVariables is initialized to one (1). It may then be explicitly selected using the <SELECT_,rVARs_RECINTERVAL_> operation. Note that the current record interval for rVariables is maintained for a CDF (not each rVariable) — it applies to all of the rVariables in that CDF.

dimension indices, rVariables

An rVariable read or write operation is always performed at (for single reads and writes) or starting at (for hyper reads and writes) the current dimension indices for the rVariables in the current CDF. When a CDF is opened or created the current dimension indices for its rVariables are initialized to zeroes (0,0,...). They may then be explicitly selected using the <SELECT_,rVARs_DIMINDICES_> operation. Note that the current dimension indices for rVariables are maintained for a CDF (not each rVariable) — they apply to all of the rVariables in that CDF. For 0-dimensional rVariables the current dimension indices are not applicable.

dimension counts, rVariables

An rVariable hyper read or write operation is always performed using the current dimension counts for the rVariables

in the current CDF. When a CDF is opened or created the current dimension counts for its rVariables are initialized to the dimension sizes of the rVariables (which specifies the entire array). They may then be explicitly selected using the <SELECT_,rVARs_DIMCOUNTS_> operation. Note that the current dimension counts for rVariables are maintained for a CDF (not each rVariable) — they apply to all of the rVariables in that CDF. For 0-dimensional rVariables the current dimension counts are not applicable.

dimension intervals, rVariables

An rVariable hyper read or write operation is always performed using the current dimension intervals for the rVariables in the current CDF. When a CDF is opened or created the current dimension intervals for its rVariables are initialized to ones (1,1,...). They may then be explicitly selected using the <SELECT_,rVARs_DIMINTERVALS_> operation. Note that the current dimension intervals for rVariables are maintained for a CDF (not each rVariable) — they apply to all of the rVariables in that CDF. For 0-dimensional rVariables the current dimension intervals are not applicable.

record number, zVariable

A zVariable read or write operation is always performed at (for single reads and writes) or starting at (for hyper reads and writes) the current record number for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened) the current record number for that zVariable is initialized to zero (0). It may then be explicitly selected using the <SELECT_,zVAR_RECNUMBER_> operation (which only affects the current zVariable in the current CDF). Note that a current record number is maintained for each zVariable in a CDF.

record count, zVariable

A zVariable hyper read or write operation is always performed using the current record count for the current zVariable in the current CDF. When a zVariable created (or for each zVariable in a CDF being opened) the current record count for that zVariable is initialized to one (1). It may then be explicitly selected using the <SELECT_,zVAR_RECCOUNT_> operation (which only affects the current zVariable in the current CDF). Note that a current record count is maintained for each zVariable in a CDF.

record interval, zVariable

A zVariable hyper read or write operation is always performed using the current record interval for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened) the current record interval for that zVariable is initialized to one (1). It may then be explicitly selected using the <SELECT_,zVAR_RECINTERVAL_> operation (which only affects the current zVariable in the current CDF). Note that a current record interval is maintained for each zVariable in a CDF.

dimension indices, zVariable

A zVariable read or write operation is always performed at (for single reads and writes) or starting at (for hyper reads and writes) the current dimension indices for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened) the current dimension indices for that zVariable are initialized to zeroes (0,0,...). They may then be explicitly selected using the <SELECT_,zVAR_DIMINDICES_> operation (which only affects the current zVariable in the current CDF). Note that current dimension indices are maintained for each zVariable in a CDF. For 0-dimensional zVariables the current dimension indices are not applicable.

dimension counts, zVariable

A zVariable hyper read or write operation is always performed using the current dimension counts for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened) the current dimension counts for that zVariable are initialized to the dimension sizes of that zVariable (which specifies the entire array). They may then be explicitly selected using the <SELECT_,zVAR_DIMCOUNTS_> operation (which only affects the current zVariable in the current CDF). Note that current dimension counts are maintained for each zVariable in a CDF. For 0-dimensional zVariables the current dimension counts are not applicable.

dimension intervals, zVariable

A zVariable hyper read or write operation is always performed using the current dimension intervals for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened) the current dimension intervals for that zVariable are initialized to ones (1,1,...). They may then be explicitly selected using the <SELECT_,zVAR_DIMINTERVALS_> operation (which only affects the current zVariable in the current CDF). Note that current dimension intervals are maintained for each zVariable in a CDF. For 0-dimensional zVariables the current dimension intervals are not applicable.

status code

When inquiring the explanation of a CDF status code, the text returned is always for the current status code. One current status code is maintained for the entire CDF library (regardless of the number of open CDFs). The current status code may be selected using the <SELECT_,CDF_STATUS_> operation. There is no default current status code. Note that the current status code is NOT the status code from the last operation performed).

NOTE: The current status code is not maintained by the CDF library from one call to the next of `CDFlib`.

Returned Status

CDF_lib returns a status code of type INTEGER*4. Since more than one operation may be performed with a single call to CDF_lib, the following rules apply.

1. The first error detected aborts the call to CDF_lib and the corresponding status code is returned.
2. In the absence of any errors, the status code for the last warning detected is returned.
3. In the absence of any errors or warnings, the status code for the last informational condition is returned.
4. In the absence of any errors, warnings, or informational conditions, CDF_OK is returned.

Section 5.9 explains how to interpret status codes and Appendix C lists the possible status codes.

Indentation/Style

Indentation should be used to make calls to CDF_lib readable. The following example shows a call to CDF_lib using proper indentation.

```
CALL CDF_lib (CREATE_, CDF_, CDF_name, num_dims, dim_sizes, id,
1             PUT_, CDF_FORMAT_, format,
2                   CDF_MAJORITY_, majority,
3             CREATE_, ATTR_, attr_name, scope, attr_num,
4                      rVAR_, var_name, data_type, num_elements,
5                             rec_vary, dim_varys, var_num,
6             NULL_, status)
```

Note that the operations (CREATE_, PUT_, and NULL_) are indented the same and that the items (CDF_, CDF_FORMAT_, CDF_MAJORITY_, ATTR_, and rVAR_) are indented the same under their corresponding operations.

The following example shows the same call to CDF_lib without the proper indentation.

```
CALL CDF_lib (CREATE_, CDF_, CDF_name, num_dims, dim_sizes, id, PUT_,
1             CDF_FORMAT_, format, CDF_MAJORITY_, majority, CREATE_,
2             ATTR_, attr_name, scope, attr_num, rVAR_, var_name, data_type,
3             num_elements, rec_vary, dim_varys, var_num, NULL_, status)
```

The need for proper indentation to ensure the readability of your applications should be obvious.[5]

Syntax

CDF_lib takes a variable number of arguments. There must always be at least one argument (the first operation to perform). The maximum number of arguments is not limited by CDF but rather the FORTRAN

[5] If it isn't, call us, we can help you.

compiler and operating system being used. Under normal circumstances that limit would never be reached (or even approached). Note that a call to `CDF_lib` with a large number of arguments can always be broken up into two or more calls to `CDF_lib` with fewer arguments.

Operations/items/arguments...

The syntax for CDF_lib is as follows...

```
   CALL CDF_lib (op1, item1, arg1, arg2, ...argN,
  +                   item2, arg1, arg2, ...argN,
                         .
                         .
  +                   itemN, arg1, arg2, ...argN,
  +               op2, item1, arg1, arg2, ...argN,
  +                   item2, arg1, arg2, ...argN,
                         .
                         .
  +                   itemN, arg1, arg2, ...argN,
                   .
                   .
  +               opN, item1, arg1, arg2, ...argN,
  +                   item2, arg1, arg2, ...argN,
                         .
                         .
  +                   itemN, arg1, arg2, ...argN,
  +               NULL_, status)
```

where `op`x is an operation to perform, `item`x is the item on which to perform the operation, and `arg`x is a required argument for the operation. The `NULL_` operation must be used to end the call to `CDF_lib`. The completion status from `CDF_lib` is placed into `status`.

The supported operations are as follows.

`CREATE_`	Used to create an item.
`OPEN_`	Used to open an item.
`DELETE_`	Used to delete an item.
`CLOSE_`	Used to close an item.
`SELECT_`	Used to select the value of an item.
`CONFIRM_`	Used to confirm the value of an item.
`PUT_`	Used to put (write) something to an item.
`GET_`	Used to get (read) something from an item.

For each operation, the supported items and required arguments are as follows.

`CREATE_` The following items may be created...

`CDF_` A new CDF will be created. It is illegal to create a CDF that already exists. The created CDF implicitly becomes the current CDF. Required arguments are...

`in: CHARACTER CDF_name*(*)`

Pathname of the CDF to be created (do not specify an extension). This can be at most `CDF_PATHNAME_LEN` characters. A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on VMS systems and environment variables on UNIX systems).

NOTE: On UNIX systems pathnames are case sensitive.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `CDF_name` MUST be of type `CHARACTER`.

`in: INTEGER*4 num_dims`

Number of dimensions for the rVariables. This can be as few as zero (0) and at most `CDF_MAX_DIMS`.

`in: INTEGER*4 dim_sizes(*)`

Dimension sizes for the rVariables. `dim_sizes` is a 1-dimensional array containing `num_dims` elements. Each element of `dim_sizes` specifies the corresponding dimension size. Each size must be greater than zero (0). For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).

`out: INTEGER*4 id`

CDF identifier to be used in subsequent operations on the CDF.

A CDF is created with a default format of multi-file (`MULTI_FILE`), a default data encoding of host (`HOST_ENCODING`), and a default variable majority of row major (`ROW_MAJOR`). These defaults can be changed with the corresponding `PUT_` operations (of `CDF_FORMAT_`, `CDF_ENCODING_` and `CDF_MAJORITY_` respectively).

`<CLOSE_,CDF_>` MUST be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk.

`rVAR_` A new rVariable will be created in the current CDF. A variable (rVariable or zVariable) with the same name must not already exist in the CDF. The created rVariable implicitly becomes the current rVariable (in the current CDF). Required arguments are...

`in: CHARACTER var_name*(*)`

Name of the rVariable to be created. This can be at most `CDF_VAR_NAME_LEN` characters. Variable names are case-sensitive.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

`in: INTEGER*4 data_type`

Data type of the new rVariable. Specify one of the data types described in Section 5.7.3.

in: INTEGER*4 num_elements

Number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in each string. A string exists at each value. For the non-character string data types this must be one (1) — multiple elements are not allowed for non-character string data types.

in: INTEGER*4 rec_vary

Record variance. Specify one of the variances described in Section 5.7.6.

in: INTEGER*4 dim_varys(*)

Dimension variances. dim_varys is a 1-dimensional array containing one element per rVariable dimension. Each element of dim_varys specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 5.7.6. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).

out: INTEGER*4 var_num Number assigned to the new rVariable. This number must be used in subsequent CDF function calls when referring to this rVariable. An existing rVariable's number may be determined with the <GET_,rVAR_NUMBER_> operation.

zVAR_

A new zVariable will be created in the current CDF. A variable (rVariable or zVariable) with the same name must not already exist in the CDF. The created zVariable implicitly becomes the current zVariable (in the current CDF). Required arguments are...

in: CHARACTER var_name*(*)

Name of the zVariable to be created. This can be at most CDF_VAR_NAME_LEN characters. Variable names are case-sensitive.

NOTE: On UNIX systems, for the proper operation of CDF_lib, var_name MUST be of type CHARACTER.

in: INTEGER*4 data_type

Data type of the new zVariable. Specify one of the data types described in Section 5.7.3.

in: INTEGER*4 num_elements

Number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in each string. A string exists at each value. For the non-character string data types this must be one (1) — multiple elements are not allowed for non-character string data types.

in: INTEGER*4 num_dims

Number of dimensions for the zVariable. This may be as few as zero and at most CDF_MAX_DIMS.

in: INTEGER*4 dim_sizes(*)

The dimension sizes. `dim_sizes` is a 1-dimensional array containing `num_dims` elements. Each element of `dim_sizes` specifies the corresponding dimension size. Each dimension size must be greater than zero (0). For a 0-dimensional zVariable this argument is ignored (but a place holder is necessary).

`in: INTEGER*4 rec_vary`

Record variance. Specify one of the variances described in Section 5.7.6.

`in: INTEGER*4 dim_varys(*)`

Dimension variances. `dim_varys` is a 1-dimensional array containing one element per dimension. Each element of `dim_varys` specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 5.7.6. For a 0-dimensional zVariable this argument is ignored (but a place holder is necessary).

`out: INTEGER*4 var_num`

Number assigned to the new zVariable. This number must be used in subsequent CDF function calls when referring to this zVariable. An existing zVariable's number may be determined with the `<GET_,zVAR_NUMBER_>` operation.

`ATTR_` A new attribute will be created in the current CDF. An attribute with the same name must not already exist in the CDF. The created attribute implicitly becomes the current attribute (in the current CDF). Required arguments are...

`in: CHARACTER attr_name*(*)`

Name of the attribute to be created. This can be at most `CDF_ATTR_NAME_LEN` characters. Attribute names are case-sensitive.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `attr_name` MUST be of type `CHARACTER`.

`in: INTEGER*4 scope`

Scope of the new attribute. Specify one of the scopes described in Section 5.7.7.

`out: INTEGER*4 attr_num`

Number assigned to the new attribute. This number must be used in subsequent CDF function calls when referring to this attribute. An existing attribute's number may be determined with the `<GET_,ATTR_NUMBER_>` operation.

`OPEN_` The follow items may be opened...

`CDF_` Opens the named CDF. The opened CDF implicitly becomes the current CDF. Required arguments are...

`in: CHARACTER CDF_name*(*)`

Pathname of the CDF to be opened. This can be at most

CDF_PATHNAME_LEN characters. A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on VMS systems and environment variables on UNIX systems).

NOTE: On UNIX systems pathnames are case sensitive.

NOTE: On UNIX systems, for the proper operation of CDF_lib, CDF_name MUST be of type CHARACTER.

out: INTEGER*4 id

CDF identifier to be used in subsequent operations of the CDF.

DELETE_ The following items may be deleted...

CDF_ Deletes the current CDF. A CDF must be opened before it can be deleted. When a CDF is deleted, there is no longer a current CDF. There are no arguments.

CLOSE_ The following items may be closed...

CDF_ Closes the current CDF. When a CDF is closed, there is no longer a current CDF. A CDF must be closed to ensure that it will be properly written to disk. There are no arguments.

rVAR_ Closes the current rVariable (in the current CDF). There are no arguments.

zVAR_ Closes the current zVariable (in the current CDF). There are no arguments.

SELECT_ The following items may be selected...

CDF_ Explicitly selects the current CDF. Required arguments are...

in: INTEGER*4 id

Identifier of the CDF. This identifier must have been initialized by a successful <CREATE_,CDF_> or <OPEN_,CDF_> operation.

CDF_STATUS_ Selects the current status code. Required arguments are...

in: INTEGER*4 status

CDF status code.

rVAR_ Explicitly selects the current rVariable (in the current CDF) by number. Required arguments are...

in: INTEGER*4 var_num

rVariable number.

`rVAR_NAME_` Explicitly selects the current rVariable (in the current CDF) by name. Required arguments are...

`in: CHARACTER var_name*(*)`

rVariable name. This may be at most `CDF_VAR_NAME_LEN` characters.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

`rVARs_RECNUMBER_` Selects the current record number for all rVariables in the current CDF. Required arguments are...

`in: INTEGER*4 rec_num`

Record number.

`rVARs_RECCOUNT_` Selects the current record count for all rVariables in the current CDF. Required arguments are...

`in: INTEGER*4 rec_count`

Record count.

`rVARs_RECINTERVAL_` Selects the current record interval for all rVariables in the current CDF. Required arguments are...

`in: INTEGER*4 rec_interval`

Record interval.

`rVARs_DIMINDICES_` Selects the current dimension indices for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

`in: INTEGER*4 indices(*)`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` specifies the corresponding dimension index.

`rVARs_DIMCOUNTS_` Selects the current dimension counts for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

`in: INTEGER*4 counts(*)`

Dimension counts. `counts` is a 1-dimensional array containing one element per dimension. Each element of `counts` specifies the corresponding dimension count.

`rVARs_DIMINTERVALS_` Selects the current dimension intervals for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

`in: INTEGER*4 intervals(*)`

Dimension intervals. `intervals` is a 1-dimensional array containing one element per dimension. Each element of `intervals` specifies the corresponding dimension interval.

`rVAR_SEQPOS_` Selects the current position for sequential access for the current rVariable (in the current CDF). Note that the current position is maintained for each rVariable individually. Required arguments are...

`in: INTEGER*4 rec_num`

Record number.

`in: INTEGER*4 indices(*)`

Dimension indices. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` specifies the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).

`zVAR_` Explicitly selects the current zVariable (in the current CDF) by number. Required arguments are...

`in: INTEGER*4 var_num`

zVariable number.

`zVAR_NAME_` Explicitly selects the current zVariable (in the current CDF) by name. Required arguments are...

`in: CHARACTER var_name*(*)`

zVariable name. This may be at most `CDF_VAR_NAME_LEN` characters.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

`zVAR_RECNUMBER_` Selects the current record number for the current zVariable in the current CDF. Required arguments are...

`in: INTEGER*4 rec_num`

Record number.

`zVAR_RECCOUNT_` Selects the current record count for the current zVariable in the current CDF. Required arguments are...

`in: INTEGER*4 rec_count`

Record count.

`zVAR_RECINTERVAL_` Selects the current record interval for the current zVariable in the current CDF. Required arguments are...

`in: INTEGER*4 rec_interval`

Record interval.

`zVAR_DIMINDICES_` Selects the current dimension indices for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`in: INTEGER*4 indices(*)`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` specifies the corresponding dimension index.

`zVAR_DIMCOUNTS_` Selects the current dimension counts for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`in: INTEGER*4 counts(*)`

Dimension counts. `counts` is a 1-dimensional array containing one element per dimension. Each element of `counts` specifies the corresponding dimension count.

`zVAR_DIMINTERVALS_` Selects the current dimension intervals for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`in: INTEGER*4 intervals(*)`

Dimension intervals. `intervals` is a 1-dimensional array containing one element per dimension. Each element of `intervals` specifies the corresponding dimension interval.

`zVAR_SEQPOS_` Selects the current position for sequential access for the current zVariable (in the current CDF). Note that the current position is maintained for each zVariable individually. Required arguments are...

`in: INTEGER*4 rec_num`

Record number.

`in: INTEGER*4 indices(*)`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` specifies the corresponding dimension index. For 0-dimensional zVariables this argument is ignored (but a place holder is necessary).

`ATTR_` Explicitly selects the current attribute (in the current CDF). Required arguments are...

`in: INTEGER*4 attr_num`

Attribute number.

`ATTR_NAME_` Explicitly selects the current attribute (in the current CDF) by name. Required arguments are...

`in: CHARACTER attr_name*(*)`

Attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `attr_name` MUST be of type `CHARACTER`.

`gENTRY_` Selects the current gEntry/rEntry number for all attributes in the

current CDF. **NOTE:** Only use this operation when dealing with gAttributes. Required arguments are...

in: INTEGER*4 entry_num
gEntry/rEntry number.

rENTRY_ Selects the current gEntry/rEntry number for all attributes in the current CDF. **NOTE:** Only use this operation when dealing with vAttributes. Required arguments are...

in: INTEGER*4 entry_num
gEntry/rEntry number.

zENTRY_ Selects the current zEntry number for all attributes in the current CDF. **NOTE:** Only use this operation when dealing with vAttributes. Required arguments are...

in: INTEGER*4 entry_num
zEntry number.

CONFIRM_ The following items may be confirmed...

CDF_NAME_ Confirms the pathname of the current CDF. Required arguments are...

out: CHARACTER CDF_name*(CDF_PATHNAME_LEN)
Pathname of the CDF. This character string must be large enough to hold CDF_PATHNAME_LEN characters and will be blank padded if necessary.
NOTE: On UNIX systems, for the proper operation of CDF_lib, CDF_name MUST be of type CHARACTER.

rVAR_ Confirms the current rVariable (in the current CDF). Required arguments are...

out: INTEGER*4 var_num
rVariable number.

rVARs_RECNUMBER_ Confirms the current record number for all rVariables in the current CDF. Required arguments are...

out: INTEGER*4 rec_num
Record number.

rVARs_RECCOUNT_ Confirms the current record count for all rVariables in the current CDF. Required arguments are...

out: INTEGER*4 rec_count
Record count.

rVARs_RECINTERVAL_ Confirms the current record interval for all rVariables in the current CDF. Required arguments are...

`out: INTEGER*4 rec_interval`

Record interval.

`rVARs_DIMINDICES_` Confirms the current dimension indices for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

`out: INTEGER*4 indices(*)`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` receives the corresponding dimension index.

`rVARs_DIMCOUNTS_` Confirms the current dimension counts for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

`out: INTEGER*4 counts(*)`

Dimension counts. `counts` is a 1-dimensional array containing one element per dimension. Each element of `counts` receives the corresponding dimension count.

`rVARs_DIMINTERVALS_` Confirms the current dimension intervals for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

`out: INTEGER*4 intervals(*)`

Dimension intervals. `intervals` is a 1-dimensional array containing one element per dimension. Each element of `intervals` receives the corresponding dimension interval.

`rVAR_SEQPOS_` Confirms the current position for sequential access for the current rVariable (in the current CDF). Note that a current position is maintained for each rVariable individually. Required arguments are...

`out: INTEGER*4 rec_num`

Record number.

`out: INTEGER*4 indices(*)`

Dimension indices. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` receives the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).

`zVAR_` Confirms the current zVariable (in the current CDF). Required arguments are...

`out: INTEGER*4 var_num`

zVariable number.

`zVAR_RECNUMBER_` Confirms the current record number for the current zVariable in the current CDF. Required arguments are...

`out: INTEGER*4 rec_num`

Record number.

`zVAR_RECCOUNT_` Confirms the current record count for the current zVariable in the current CDF. Required arguments are...

`out: INTEGER*4 rec_count`

Record count.

`zVAR_RECINTERVAL_` Confirms the current record interval for the current zVariable in the current CDF. Required arguments are...

`out: INTEGER*4 rec_interval`

Record interval.

`zVAR_DIMINDICES_` Confirms the current dimension indices for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`out: INTEGER*4 indices(*)`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` receives the corresponding dimension index.

`zVAR_DIMCOUNTS_` Confirms the current dimension counts for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`out: INTEGER*4 counts(*)`

Dimension counts. `counts` is a 1-dimensional array containing one element per dimension. Each element of `counts` receives the corresponding dimension count.

`zVAR_DIMINTERVALS_` Confirms the current dimension intervals for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`out: INTEGER*4 intervals(*)`

Dimension intervals. `intervals` is a 1-dimensional array containing one element per dimension. Each element of `intervals` receives the corresponding dimension interval.

`VAR_SEQPOS_` Confirms the current position for sequential access for the current zVariable (in the current CDF). Note that a current position is maintained for each zVariable individually. Required arguments are...

`out: INTEGER*4 rec_num`

Record number.

`out: INTEGER*4 indices(*)`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` receives the corresponding dimension index. For 0-dimensional

zVariables this argument is ignored (but a place holder is necessary).

`ATTR_` Confirms the current attribute (in the current CDF). Required arguments are...

`out: INTEGER*4 attr_num`

Attribute number.

`gENTRY_` Confirms the current gEntry/rEntry number for all attributes in the current CDF. **NOTE:** Only use this operation when dealing with gAttributes. Required arguments are...

`out: INTEGER*4 entry_num`

gEntry/rEntry number.

`rENTRY_` Confirms the current gEntry/rEntry number for all attributes in the current CDF. **NOTE:** Only use this operation when dealing with vAttributes. Required arguments are...

`out: INTEGER*4 entry_num`

gEntry/rEntry number.

`zENTRY_` Confirms the current zEntry number for all attributes in the current CDF. **NOTE:** Only use this operation when dealing with vAttributes. Required arguments are...

`out: INTEGER*4 entry_num`

zEntry number.

`PUT_` The following items may be written (put)...

`CDF_ENCODING_` Respecifies the data encoding of the current CDF. A CDF's data encoding may not be changed after any variable values (including the pad value) or attribute entries have been written. Required arguments are...

`in: INTEGER*4 encoding`

New data encoding. Specify one of the encodings described in Section 5.7.4.

`CDF_MAJORITY_` Respecifies the variable majority of the current CDF. A CDF's variable majority may not be changed after any variable values have been written. Required arguments are...

`in: INTEGER*4 majority`

New variable majority. Specify one of the majorities described in Section 5.7.5.

`CDF_FORMAT_` Respecifies the format of the current CDF. A CDF's format may not be changed after any variables have been created. Required arguments are...

in: INTEGER*4 format

New CDF format. Specify one of the formats described in Section 5.7.2.

rVAR_NAME_ Renames the current rVariable (in the current CDF). A variable (rVariable or zVariable) with the same name must not already exist in the CDF. Required arguments are...

in: CHARACTER var_name*(*)

New name of the rVariable. This may consist of at most CDF_VAR_NAME_LEN characters.

NOTE: On UNIX systems, for the proper operation of CDF_lib, var_name MUST be of type CHARACTER.

rVAR_DATASPEC_ Respecifies the data specification (data type and number of elements) of the current rVariable (in the current CDF). An rVariable's data specification may not be changed if the new data specification is not equivalent to the old data specification and any values (including the pad value) have been written. Data specifications are considered equivalent if the data types are equivalent (see Section 2.5.5) and the number of elements are the same. Required arguments are...

in: INTEGER*4 data_type

New data type. Specify one of the data types described in Section 5.7.3.

in: INTEGER*4 num_elements

Number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in each string (a string exists at each value). For non-character string data types this must be one (1) — multiple elements are not allowed for non-character string data types.

rVAR_RECVARY_ Respecifies the record variance of the current rVariable (in the current CDF). An rVariable's record variance may not be changed if any values have been written (except for an explicit pad value — it may have been written). Required arguments are...

in: INTEGER*4 rec_vary

New record variance. Specify one of the variances described in Section 5.7.6.

rVAR_DIMVARYS_ Respecifies the dimension variances of the current rVariable (in the current CDF). An rVariable's dimension variances may not be changed if any values have been written (except for an explicit pad value — it may have been written). For 0-dimensional rVariables this operation is not applicable. Required arguments are...

in: INTEGER*4 dim_varys(*)

New dimension variances. dim_varys is a 1-dimensional array containing one element per dimension. Each element

of `dim_varys` specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 5.7.6.

`rVAR_PADVALUE_` Specifies the pad value for the current rVariable (in the current CDF). An rVariable's pad value may be specified (or respecified) at any time without affecting already written values (including where pad values were used). Section 2.7.12 describes variable pad values. Required arguments are...

`in: <type> value`

Pad value. `<type>` depends on the data type of the rVariable. The pad value is written to the CDF from memory address `value`.

NOTE: On UNIX systems, if the rVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`rVAR_INITIALRECS_` Specifies the number of records to initially write to the current rVariable (in the current CDF). This may be specified only once per rVariable and before any other records have been written to that rVariable. If a pad value has not yet been specified, the default is used (see Section 2.7.12). If a pad value has been explicitly specified, that value is written to the records. Section 2.7.5 describes initial records. Required arguments are...

`in: INTEGER*4 n_initial_recs`

Number of records to write.

`rVAR_EXTENDRECS_` Specifies the number of physical records to extend the current rVariable (in the current CDF) whenever that rVariable needs to be extended. Section 2.7.5 describes extend records. Required arguments are...

`in: INTEGER*4 n_extend_recs`

Number of records to extend.

`rVAR_DATA_` Writes one value to the current rVariable (in the current CDF). The value is written at the current record number and current dimension indices for the rVariables (in the current CDF). Required arguments are...

`in: <type> value`

Value. `<type>` depends on the data type of the rVariable. The value is written to the CDF from memory address `value`.

NOTE: On UNIX systems, if the rVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`rVAR_HYPERDATA_` Writes one or more values to the current rVariable (in the current CDF). The values are written based on the current record number,

current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals for the rVariables (in the current CDF). Required arguments are...

in: <type> buffer

Values. <type> depends on the data type of the rVariable. The values starting at memory address buffer are written to the CDF.

NOTE: On UNIX systems, if the rVariable has a character string data type, then <type> MUST be CHARACTER to ensure the proper operation of CDF_lib.

rVAR_SEQDATA_ Writes one value to the current rVariable (in the current CDF) at the current sequential position for that rVariable. After the write, the sequential position is automatically incremented to the next value (crossing a record boundary if necessary). If the sequential position is past the last record for the rVariable, the rVariable is extended as necessary. Required arguments are...

in: <type> value

Value. <type> depends on the data type of the rVariable. The value is written to the CDF from memory address value.

NOTE: On UNIX systems, if the rVariable has a character string data type, then <type> MUST be CHARACTER to ensure the proper operation of CDF_lib.

zVAR_NAME_ Renames the current zVariable (in the current CDF). A variable (rVariable or zVariable) with the same name must not already exist in the CDF. Required arguments are...

in: CHARACTER var_name*(*)

New name of the zVariable. This may consist of at most CDF_VAR_NAME_LEN characters.

NOTE: On UNIX systems, for the proper operation of CDF_lib, var_name MUST be of type CHARACTER.

zVAR_DATASPEC_ Respecifies the data specification (data type and number of elements) of the current zVariable (in the current CDF). A zVariable's data specification may not be changed if the new data specification is not equivalent to the old data specification and any values (including the pad value) have been written. Data specifications are considered equivalent if the data types are equivalent (see Section 2.5.5) and the number of elements are the same. Required arguments are...

in: INTEGER*4 data_type

New data type. Specify one of the data types described in Section 5.7.3.

in: INTEGER*4 num_elements

Number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the

number of characters in each string (a string exists at each value). For non-character string data types this must be one (1) — multiple elements are not allowed for non-character string data types.

`zVAR_RECVARY_` Respecifies the record variance of the current zVariable (in the current CDF). A zVariable's record variance may not be changed if any values have been written (except for an explicit pad value — it may have been written). Required arguments are...

`in: INTEGER*4 rec_vary`

New record variance. Specify one of the variances described in Section 5.7.6.

`zVAR_DIMVARYS_` Respecifies the dimension variances of the current zVariable (in the current CDF). A zVariable's dimension variances may not be changed if any values have been written (except for an explicit pad value — it may have been written). For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`in: INTEGER*4 dim_varys(*)`

New dimension variances. `dim_varys` is a 1-dimensional array containing one element per dimension. Each element of `dim_varys` specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 5.7.6.

`zVAR_PADVALUE_` Specifies the pad value for the current zVariable (in the current CDF). A zVariable's pad value may be specified (or respecified) at any time without affecting already written values (including where pad values were used). Section 2.7.12 describes variable pad values. Required arguments are...

`in: <type> value`

Pad value. `<type>` depends on the data type of the zVariable. The pad value is written to the CDF from memory address `value`.

NOTE: On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`zVAR_INITIALRECS_` Specifies the number of records to initially write to the current zVariable (in the current CDF). This may be specified only once per zVariable and before any other records have been written to that zVariable. If a pad value has not yet been specified, the default is used (see Section 2.7.12). If a pad value has been explicitly specified, that value is written to the records. Section 2.7.5 describes initial records. Required arguments are...

`in: INTEGER*4 n_initial_recs`

Number of records to write.

`zVAR_EXTENDRECS_` Specifies the number of physical records to extend the current zVariable (in the current CDF) whenever that zVariable needs to be extended. Section 2.7.5 describes extend records. Required arguments are...

`in: INTEGER*4 n_extend_recs`

Number of records to extend.

`zVAR_DATA_` Writes one value to the current zVariable (in the current CDF). The value is written at the current record number and current dimension indices for that zVariable (in the current CDF). Required arguments are...

`in: <type> value`

Value. `<type>` depends on the data type of the zVariable. The value is written to the CDF from memory address `value`.

NOTE: On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`zVAR_HYPERDATA_` Writes one or more values to the current zVariable (in the current CDF). The values are written based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals for that zVariable (in the current CDF). Required arguments are...

`in: <type> buffer`

Values. `<type>` depends on the data type of the zVariable. The values starting at memory address `buffer` are written to the CDF.

NOTE: On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`zVAR_SEQDATA_` Writes one value to the current zVariable (in the current CDF) at the current sequential position for that zVariable. After the write, the sequential position is automatically incremented to the next value (crossing a record boundary if necessary). If the sequential position is past the last record for the zVariable, the zVariable is extended as necessary. Required arguments are...

`in: <type> value`

Value. `<type>` depends on the data type of the zVariable. The value is written to the CDF from memory address `value`.

NOTE: On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`ATTR_SCOPE_` Respecifies the scope for the current attribute (in the current CDF). Required arguments are...

in: INTEGER*4 scope

New attribute scope. Specify one of the scopes described in Section 5.7.7.

ATTR_NAME_ Renames the current attribute (in the current CDF). An attribute with the same name must not already exist in the CDF. Required arguments are...

in: CHARACTER attr_name*(*)

New attribute name. This may be at most CDF_ATTR_NAME_LEN characters.

NOTE: On UNIX systems, for the proper operation of CDF_lib, attr_name MUST be of type CHARACTER.

gENTRY_DATA_ Writes a gEntry to the current attribute at the current gEntry/rEntry number (in the current CDF). An existing gEntry may be overwritten with a new gEntry having the same data specification (data type and number of elements) or a different data specification. **NOTE:** This operation should only be performed on gAttributes. Required arguments are...

in: INTEGER*4 data_type

Data type of the gEntry. Specify one of the data types described in Section 5.7.3.

in: INTEGER*4 num_elements

Number of elements of the data type. This may be greater than one (1) for any of the supported data types. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. For all other data types, this is the number of elements in an array of that data type.

in: <type> value

Value(s). <type> depends on the data type of the gEntry. The value is written to the CDF from memory address value.

NOTE: On UNIX systems, if the gEntry has a character string data type, then <type> MUST be CHARACTER to ensure the proper operation of CDF_lib.

rENTRY_DATA_ Writes a rEntry to the current attribute at the current gEntry/rEntry number (in the current CDF). An existing rEntry may be overwritten with a new rEntry having the same data specification (data type and number of elements) or a different data specification. **NOTE:** This operation should only be performed on vAttributes. Required arguments are...

in: INTEGER*4 data_type

Data type of the rEntry. Specify one of the data types described in Section 5.7.3.

in: INTEGER*4 num_elements

Number of elements of the data type. This may be greater

than one (1) for any of the supported data types. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. For all other data types, this is the number of elements in an array of that data type.

in: <type> value

Value(s). <type> depends on the data type of the rEntry. The value is written to the CDF from memory address value.

NOTE: On UNIX systems, if the rEntry has a character string data type, then <type> MUST be CHARACTER to ensure the proper operation of CDF_lib.

zENTRY_DATA_ Writes a zEntry to the current attribute at the current zEntry number (in the current CDF). An existing zEntry may be overwritten with a new zEntry having the same data specification (data type and number of elements) or a different data specification. **NOTE:** This operation should only be performed on vAttributes. Required arguments are...

in: INTEGER*4 data_type

Data type of the zEntry. Specify one of the data types described in Section 5.7.3.

in: INTEGER*4 num_elements

Number of elements of the data type. This may be greater than one (1) for any of the supported data types. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. For all other data types, this is the number of elements in an array of that data type.

in: <type> value

Value(s). <type> depends on the data type of the zEntry. The value is written to the CDF from memory address value.

NOTE: On UNIX systems, if the zEntry has a character string data type, then <type> MUST be CHARACTER to ensure the proper operation of CDF_lib.

GET_ The following items may be read (gotten)...

rVARs_NUMDIMS_ Inquires the number of dimensions for the rVariables in the current CDF. Required arguments are...

out: INTEGER*4 num_dims

Number of dimensions.

rVARs_DIMSIZES_ Inquires the size of each dimension for the rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are...

out: INTEGER*4 dim_sizes(*)

Dimension sizes. dim_sizes is a 1-dimensional array containing one element per rVariable dimension. Each element of dim_sizes receives the corresponding dimension size.

`zVAR_NUMDIMS_` Inquires the number of dimensions for the current zVariable in the current CDF. Required arguments are...

`out: INTEGER*4 num_dims`

Number of dimensions.

`zVAR_DIMSIZES_` Inquires the size of each dimension for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are...

`out: INTEGER*4 dim_sizes(*)`

Dimension sizes. `dim_sizes` is a 1-dimensional array containing one element per dimension. Each element of `dim_sizes` receives the corresponding dimension size.

`CDF_ENCODING_` Inquires the data encoding of the current CDF. Required arguments are...

`out: INTEGER*4 encoding`

Data encoding. The encodings are described in Section 5.7.4.

`CDF_MAJORITY_` Inquires the variable majority of the current CDF. Required arguments are...

`out: INTEGER*4 majority`

Variable majority. The majorities are described in Section 5.7.5.

`CDF_FORMAT_` Inquires the format of the current CDF. Required arguments are...

`out: INTEGER*4 format`

CDF format. The formats are described in Section 5.7.2.

`CDF_COPYRIGHT_` Reads the copyright notice for the CDF library which created the current CDF. Required arguments are...

`out: CHARACTER copy_right*(CDF_COPYRIGHT_LEN)`

CDF copyright text. This character string must be large enough hold `CDF_COPYRIGHT_LEN` characters and will be blank padded if necessary.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `copy_right` MUST be of type `CHARACTER`.

`LIB_COPYRIGHT_` Reads the copyright notice for the CDF library being used. Required arguments are...

`out: CHARACTER copy_right*(CDF_COPYRIGHT_LEN)`

CDF library copyright text. This character string must be large enough hold `CDF_COPYRIGHT_LEN` characters and will be blank padded if necessary.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `copy_right` MUST be of type `CHARACTER`.

`CDF_NUMrVARS_` Inquires the number of rVariables in the current CDF. Required arguments are...

`out: INTEGER*4 num_vars`
Number of rVariables.

`CDF_NUMzVARS_` Inquires the number of zVariables in the current CDF. Required arguments are...

`out: INTEGER*4 num_vars`
Number of zVariables.

`CDF_NUMATTRS_` Inquires the number of attributes in the current CDF. Required arguments are...

`out: INTEGER*4 num_attrs`
Number of attributes.

`CDF_NUMgATTRS_` Inquires the number of gAttributes in the current CDF. Required arguments are...

`out: INTEGER*4 num_attrs`
Number of gAttributes.

`CDF_NUMvATTRS_` Inquires the number of vAttributes in the current CDF. Required arguments are...

`out: INTEGER*4 num_attrs`
Number of vAttributes.

`rVARs_MAXREC_` Inquires the maximum record number of the rVariables in the current CDF. A value of zero (0) indicates that no records have been written to the rVariables in the CDF. The maximum record number for an individual rVariable may be inquired using `<GET_,rVAR_MAXREC_>`. Required arguments are...

`out: INTEGER*4 max_rec`
Maximum record number.

`CDF_VERSION_` Inquires the version number of the CDF library which created the current CDF. Required arguments are...

`out: INTEGER*4 version`
Version number.

`CDF_RELEASE_` Inquires the release number of the CDF library which created the current CDF. Required arguments are...

`out: INTEGER*4 release`
Release number.

`CDF_INCREMENT_` Inquires the incremental number of the CDF library which created the current CDF. Required arguments are...

`out: INTEGER*4 increment`

Incremental number.

`LIB_VERSION_` Inquires the version number of the CDF library being used. Required arguments are...

`out: INTEGER*4 version`

Version number.

`LIB_RELEASE_` Inquires the release number of the CDF library being used. Required arguments are...

`out: INTEGER*4 release`

Release number.

`LIB_INCREMENT_` Inquires the incremental number of the CDF library being used. Required arguments are...

`out: INTEGER*4 increment`

Incremental number.

`LIB_subINCREMENT_` Inquires the subincremental character of the CDF library being used. Required arguments are...

`out: CHARACTER*1 subincrement`

Subincremental character.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `subincrement` MUST be of type `CHARACTER`.

`rVAR_NAME_` Inquires the name of the current rVariable (in the current CDF). Required arguments are...

`out: CHARACTER var_name*(CDF_VAR_NAME_LEN)`

Name of the rVariable. This character string must be large enough to hold `CDF_VAR_NAME_LEN` characters and will be blank padded if necessary.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

`rVAR_DATATYPE_` Inquires the data type of the current rVariable (in the current CDF). Required arguments are...

`out: INTEGER*4 data_type`

Data type. The data types are described in Section 5.7.3.

`rVAR_NUMELEMS_` Inquires the number of elements (of the data type) for the current rVariable (in the current CDF). Required arguments are...

`out: INTEGER*4 num_elements`

Number of elements of the data type at each value. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the

number of characters in the string (each value consists of the entire string). For all other data types, this will always be one (1) — multiple elements at each value are not allowed for non-character string data types.

`rVAR_RECVARY_` Inquires the record variance of the current rVariable (in the current CDF). Required arguments are...

`out: INTEGER*4 rec_vary`

Record variance. The variances are described in Section 5.7.6.

`rVAR_DIMVARYS_` Inquires the dimension variances of the current rVariable (in the current CDF). For 0-dimensional rVariables this operation is not applicable. Required arguments are...

`out: INTEGER*4 dim_varys(*)`

Dimension variances. `dim_varys` is a 1-dimensional array containing one element per rVariable dimension. Each element of `dim_varys` receives the corresponding dimension variance. The variances are described in Section 5.7.6.

`rVAR_NUMBER_` Gets the number of the named rVariable (in the current CDF). Required arguments are...

`in: CHARACTER var_name*(*)`

The rVariable name. This may be at most `CDF_VAR_NAME_LEN` characters.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

`out: INTEGER*4 var_num`

The rVariable number.

`rVAR_PADVALUE_` Inquires the pad value of the current rVariable (in the current CDF). Required arguments are...

`out: <type> value`

Pad value. `<type>` depends on the data type of the rVariable. The pad value is read from the CDF and placed into memory at address `value`. This buffer must be large enough to hold the pad value.

NOTE: On UNIX systems, if the rVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`rVAR_EXTENDRECS_` Inquires the number of extend records for the current rVariable (in the current CDF). Extend records are described in Section 2.7.5. Required arguments are...

`out: INTEGER*4 n_extend_recs`

Number of records to extend.

`rVAR_MAXREC_` Inquires the maximum record number for the current rVariable (in the current CDF). For rVariables with a record variance of `NOVARY`, this will be at most one (1). A value of zero (0) indicates that no records have been written. Required arguments are...

`out: INTEGER*4 var_max_rec`

Maximum record number.

`rVAR_DATA_` Reads a value from the current rVariable (in the current CDF). The value is read at the current record number and current dimension indices for the rVariables (in the current CDF). Required arguments are...

`out: <type> value`

Value. `<type>` depends on the data type of the rVariable. This buffer must be large enough to hold the value. The value is read from the CDF and placed into memory at address `value`.

NOTE: On UNIX systems, if the rVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`rVAR_HYPERDATA_` Reads one or more values from the current rVariable (in the current CDF). The values are read based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals for the rVariables (in the current CDF). Required arguments are...

`out: <type> buffer`

Value(s). `<type>` depends on the data type of the rVariable. The values are read from the CDF and placed into memory starting at address `buffer`. This buffer must be large enough to hold the values.

NOTE: On UNIX systems, if the rVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`rVAR_SEQDATA_` Reads one value from the current rVariable (in the current CDF) at the current sequential position for that rVariable. After the read, the sequential position is automatically incremented to the next value (crossing a record boundary if necessary). An error is returned if the sequential position is past the last record for the rVariable. Required arguments are...

`out: <type> value`

Value. `<type>` depends on the data type of the rVariable. The value is read from the CDF and placed into memory at address `value`. This buffer must be large enough to hold the value.

NOTE: On UNIX systems, if the rVariable has a character

string data type, then <type> MUST be CHARACTER to ensure the proper operation of CDF_lib.

zVAR_NAME_ Inquires the name of the current zVariable (in the current CDF). Required arguments are...

out: CHARACTER var_name*(CDF_VAR_NAME_LEN)

Name of the zVariable. This character string must be large enough to hold CDF_VAR_NAME_LEN characters and will be blank padded if necessary.

NOTE: On UNIX systems, for the proper operation of CDF_lib, var_name MUST be of type CHARACTER.

zVAR_DATATYPE_ Inquires the data type of the current zVariable (in the current CDF). Required arguments are...

out: INTEGER*4 data_type

Data type. The data types are described in Section 5.7.3.

zVAR_NUMELEMS_ Inquires the number of elements (of the data type) for the current zVariable (in the current CDF). Required arguments are...

out: INTEGER*4 num_elements

Number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (each value consists of the entire string). For all other data types, this will always be one (1) — multiple elements at each value are not allowed for non-character string data types.

zVAR_RECVARY_ Inquires the record variance of the current zVariable (in the current CDF). Required arguments are...

out: INTEGER*4 rec_vary

Record variance. The variances are described in Section 5.7.6.

zVAR_DIMVARYS_ Inquires the dimension variances of the current zVariable (in the current CDF). For 0-dimensional zVariables this operation is not applicable. Required arguments are...

out: INTEGER*4 dim_varys(*)

Dimension variances. dim_varys is a 1-dimensional array containing one element per dimension. Each element of dim_varys receives the corresponding dimension variance. The variances are described in Section 5.7.6.

zVAR_NUMBER_ Gets the number of the named zVariable (in the current CDF). Required arguments are...

in: CHARACTER var_name*(*)

The zVariable name. This may be at most CDF_VAR_NAME_LEN characters.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

`out: INTEGER*4 var_num` The rVariable number.

`zVAR_PADVALUE_` Inquires the pad value of the current zVariable (in the current CDF). Required arguments are...

`out: <type> value`

Pad value. `<type>` depends on the data type of the zVariable. The pad value is read from the CDF and placed into memory at address `value`. This buffer must be large enough to hold the pad value.

NOTE: On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`zVAR_EXTENDRECS_` Inquires the number of extend records for the current zVariable (in the current CDF). Extend records are described in Section 2.7.5. Required arguments are...

`out: INTEGER*4 n_extend_recs`

Number of records to extend.

`zVAR_MAXREC_` Inquires the maximum record number for the current zVariable (in the current CDF). For zVariables with a record variance of `NOVARY`, this will be at most one (1). A value of zero (0) indicates that no records have been written. Required arguments are...

`out: INTEGER*4 var_max_rec`

Maximum record number.

`zVAR_DATA_` Reads a value from the current zVariable (in the current CDF). The value is read at the current record number and current dimension indices for that zVariable (in the current CDF). Required arguments are...

`out: <type> value`

Value. `<type>` depends on the data type of the zVariable. This buffer must be large enough to hold the value. The value is read from the CDF and placed into memory at address `value`.

NOTE: On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`zVAR_HYPERDATA_` Reads one or more values from the current zVariable (in the current CDF). The values are read based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals for that zVariable (in the current CDF). Required arguments are...

`out: <type> buffer`

Value(s). `<type>` depends on the data type of the zVariable. The values are read from the CDF and placed into memory starting at address `buffer`. This buffer must be large enough to hold the values.

NOTE: On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`zVAR_SEQDATA_` Reads one value from the current zVariable (in the current CDF) at the current sequential position for that zVariable. After the read, the sequential position is automatically incremented to the next value (crossing a record boundary if necessary). An error is returned if the sequential position is past the last record for the zVariable. Required arguments are...

`out: <type> value`

Value. `<type>` depends on the data type of the zVariable. The value is read from the CDF and placed into memory at address `value`. This buffer must be large enough to hold the value.

NOTE: On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`ATTR_SCOPE_` Inquires the scope of the current attribute (in the current CDF). Required arguments are...

`out: INTEGER*4 scope`

Attribute scope. The scopes are described in Section 5.7.7.

`ATTR_NAME_` Inquires the name of the current attribute (in the current CDF). Required arguments are...

`out: CHARACTER attr_name*(CDF_ATTR_NAME_LEN)`

Attribute name. This character string must be large enough to hold `CDF_ATTR_NAME_LEN` characters and will be blank padded if necessary.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `attr_name` MUST be of type `CHARACTER`.

`ATTR_NUMBER_` Gets the number of the named attribute (in the current CDF). Required arguments are...

`in: CHARACTER attr_name*(*)`

Attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `attr_name` MUST be of type `CHARACTER`.

`out: INTEGER*4 attr_num`

The attribute number.

`ATTR_MAXgENTRY_` Inquires the maximum gEntry number used for the current attribute (in the current CDF). This does not necessarily correspond with the number of gEntries for the attribute. **NOTE:** Use this operation with gAttributes only. Required arguments are...

`out: INTEGER*4 max_entry`

The maximum gEntry number for the attribute.

`ATTR_NUMgENTRIES_` Inquires the number of gEntries for the current attribute (in the current CDF). This does not necessarily correspond with the maximum gEntry number used. **NOTE:** Use this operation with gAttributes only. Required arguments are...

`out: INTEGER*4 num_entries`

The number of gEntries for the attribute.

`ATTR_MAXrENTRY_` Inquires the maximum rEntry number used for the current attribute (in the current CDF). This does not necessarily correspond with the number of rEntries for the attribute. **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: INTEGER*4 max_entry`

The maximum rEntry number for the attribute.

`ATTR_NUMrENTRIES_` Inquires the number of rEntries for the current attribute (in the current CDF). This does not necessarily correspond with the maximum rEntry number used. **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: INTEGER*4 num_entries`

The number of rEntries for the attribute.

`ATTR_MAXzENTRY_` Inquires the maximum zEntry number used for the current attribute (in the current CDF). This does not necessarily correspond with the number of zEntries for the attribute. **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: INTEGER*4 max_entry`

The maximum zEntry number for the attribute.

`ATTR_NUMzENTRIES_` Inquires the number of zEntries for the current attribute (in the current CDF). This does not necessarily correspond with the maximum zEntry number used. **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: INTEGER*4 num_entries`

The number of zEntries for the attribute.

`gENTRY_DATATYPE_` Inquires the data type of the gEntry at the current gEntry/rEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with gAttributes only. Required arguments are...

`out: INTEGER*4 data_type`

Data type. The data types are described in Section 5.7.3.

`gENTRY_NUMELEMS_` Inquires the number of elements (of the data type) of the gEntry at the current gEntry/rEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with gAttributes only. Required arguments are...

`out: INTEGER*4 num_elements`

Number of elements of the data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string. For all other data types, this is the number of elements in an array of that data type.

`gENTRY_DATA_` Reads the gEntry data value from the current attribute at the current gEntry/rEntry number (in the current CDF). **NOTE:** Use this operation with gAttributes only. Required arguments are...

`out: <type> value`

Value. `<type>` depends on the data type of the gEntry. The value is read from the CDF and placed into memory at address `value`. This buffer must be large enought to hold the value.

NOTE: On UNIX systems, if the gEntry has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`rENTRY_DATATYPE_` Inquires the data type of the rEntry at the current gEntry/rEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: INTEGER*4 data_type`

Data type. The data types are described in Section 5.7.3.

`rENTRY_NUMELEMS_` Inquires the number of elements (of the data type) of the rEntry at the current gEntry/rEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: INTEGER*4 num_elements`

Number of elements of the data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string. For all other data types, this is the number of elements in an array of that data type.

`rENTRY_DATA_` Reads the rEntry data value from the current attribute at the current gEntry/rEntry number (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: <type> value`

Value. `<type>` depends on the data type of the rEntry. The value is read from the CDF and placed into memory at address `value`. This buffer must be large enought to hold the value.

NOTE: On UNIX systems, if the rEntry has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`zENTRY_DATATYPE_` Inquires the data type of the zEntry at the current zEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: INTEGER*4 data_type`

Data type. The data types are described in Section 5.7.3.

`zENTRY_NUMELEMS_` Inquires the number of elements (of the data type) of the zEntry at the current zEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: INTEGER*4 num_elements`

Number of elements of the data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string. For all other data types, this is the number of elements in an array of that data type.

`zENTRY_DATA_` Reads the zEntry data value from the current attribute at the current zEntry number (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are...

`out: <type> value`

Value. `<type>` depends on the data type of the zEntry. The value is read from the CDF and placed into memory at address `value`. This buffer must be large enought to hold the value.

NOTE: On UNIX systems, if the zEntry has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

`STATUS_TEXT_` Inquires the explanation text for the current status code. Note that the current status code is NOT the status code from the last operation performed. Required arguments are...

`out: CHARACTER text*(CDF_STATUSTEXT_LEN)`

Text explaining the status code. This string must be large enough to hold `CDF_STATUSTEXT_LEN` characters and will be blank padded if necessary.

NOTE: On UNIX systems, for the proper operation of `CDF_lib`, `text` MUST be of type `CHARACTER`.

More Examples

Several more examples of the use of `CDF_lib` follow.

rVariable creation In this example an rVariable will be created with a pad value being specified, initial records will be written, and the number of records to extend the rVariable when necessary being specified. Note that the pad value was specified before the initial records. This results in the specified pad value being written. Had the pad value not been specified first, the initial records would have been written with the default pad value.

```
.
.
INCLUDE '<path>cdf.h'
.
.
INTEGER*4 id                          ! CDF identifier (handle).
INTEGER*4 status                      ! Status returned from CDF library.
INTEGER*4 dim_varys(2)                ! Dimension variances.
INTEGER*4 var_num                     ! rVariable number.
REAL*4    pad_value                   ! Pad value.

DATA pad_value/-999.9/
.
.
dim_varys(1) = VARY
dim_varys(2) = VARY
CALL CDF_lib (SELECT_, CDF_, id,
0             CREATE_, rVAR_, 'HUMIDITY', CDF_REAL4, 1, VARY,
1                             dim_varys, var_num,
2             PUT_, rVAR_PADVALUE_, pad_value,
3                   rVAR_INITIALRECS_, 500,
4                   rVAR_EXTENDRECS_, 50,
5             NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

zVariable creation (character string data type) In this example a zVariable with a character string data type will be created with a pad value being specified.

```
.
.
INCLUDE '<path>CDF.INC'
.
.
INTEGER*4 id                          ! CDF identifier (handle).
INTEGER*4 status                      ! Status returned from CDF library.
```

```
INTEGER*4 dim_varys(1)              ! Dimension variances.
INTEGER*4 var_num                   ! zVariable number.
INTEGER*4 num_dims                  ! Number of dimension.
INTEGER*4 dim_sizes(1)              ! Dimension sizes.
INTEGER*4 num_elems                 ! Number of elements (of the data
                                    ! type).
CHARACTER pad_value                 ! Pad value.

DATA pad_value/'**********'/,
0    num_dims/1/,
1    dim_sizes/20/,
2    num_elems/10/
.
.
dim_varys(0) = VARY
CALL CDF_lib (SELECT_, CDF_, id,
0             CREATE_, zVAR_, 'Station', CDF_CHAR, num_elems, num_dims,
1                      dim_sizes, NOVARY, dim_varys, var_num,
2             PUT_, zVAR_PADVALUE_, pad_value,
3             NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

Hyper read with subsampling In this example an rVariable will be subsampled in a CDF whose rVariables are 2-dimensional and have dimension sizes `[100,200]`. The CDF is column major and the data type of the rVariable is `CDF_INT2`.

```
.
.
INCLUDE '<path>CDF.INC'
.
.
INTEGER*4 id                    ! CDF identifier (handle).
INTEGER*4 status                ! Status returned from CDF library.
INTEGER*2 values(50,100)        ! Buffer to receive values.
INTEGER*4 rec_count             ! Record count, one record per hyper get.
INTEGER*4 rec_interval          ! Record interval, set to one to indicate
                                ! contiguous records (really meaningless
                                ! since record count is one).
INTEGER*4 indices(2)            ! Dimension indices, start each read
                                ! at 1,1 of the array.
INTEGER*4 counts(2)             ! Dimension counts, half of the values along
                                ! each dimension will be read.
INTEGER*4 intervals(2)          ! Dimension intervals, every other value
                                ! along each dimension will be read.
INTEGER*4 rec_num               ! Record number.
INTEGER*4 max_rec               ! Maximum rVariable record in the
                                ! CDF - this was determined with a call
```

```
                                ! to CDF_inquire.

DATA rec_count/1/, rec_interval/1/, indices/1,1/, counts/50,100/,
1    intervals/2,2/
.
.
CALL CDF_lib (SELECT_, CDF_, id,
1                      rVAR_NAME_, 'BRIGHTNESS',
2                      rVARs_RECCOUNT_, rec_count,
3                      rVARs_RECINTERVAL_, rec_interval,
4                      rVARs_DIMINDICES_, indices,
5                      rVARs_DIMCOUNTS_, counts,
6                      rVARs_DIMINTERVALS_, intervals,
7              NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

DO rec_num = 1, max_rec
  CALL CDF_lib (SELECT_, CDF_, id,
  0                      rVARs_RECNUMBER_, rec_num,
  1              GET_, rVAR_HYPERDATA_, values,
  2              NULL_, status)
  IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
  .
  .
  ! process values
  .
  .
END DO
.
.
```

Attribute renaming In this example the attribute named `Tmp` will be renamed `TMP`.

```
.
.
INCLUDE '<path>CDF.INC'
.
.
INTEGER*4 id                    ! CDF identifier (handle).
INTEGER*4 status                ! Status returned from CDF library.
.
.
CALL CDF_lib (SELECT_, CDF_, id,
0              ATTR_NAME_, 'Tmp',
1              PUT_, ATTR_NAME, 'TMP',
2              NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

Sequential Access In this example the values for a zVariable will be averaged. The values will be read using the sequential access method (see Section 2.7.11). Each value in each record will be read and averaged. It is assumed that the data type of the zVariable has been determined to be `CDF_REAL4`.

```
.
.
INCLUDE '<path>CDF.INC'
.
.
INTEGER*4 id                    ! CDF identifier (handle).
INTEGER*4 status                ! Status returned from CDF library.
INTEGER*4 var_num               ! zVariable number.
INTEGER*4 rec_num               ! Record number, start at first record.
INTEGER*4 indices(2)            ! Dimension indices.
REAL*4    value                 ! Value read.
REAL*8    sum                   ! Sum of all values.
INTEGER*4 count                 ! Number of values.
REAL*4    ave                   ! Average value.

DATA indices/1,1/, sum/0.0/, count/0/, rec_num/1/
.
.
CALL CDF_lib (SELECT_, CDF_, id,
0             GET_, zVAR_NUMBER_, 'FLUX', var_num,
1             NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL CDF_lib (SELECT_, CDF_, id,
0                     zVAR_, var_num,
1                     zVAR_SEQPOS_, rec_num, indices,
2             GET_, zVAR_SEQDATA_, value,
3             NULL_, status)

DO WHILE (status .GE. CDF_OK)
  sum = sum + value
  count = count + 1
  CALL CDF_lib (SELECT_, CDF_, id,
  0             GET_, zVAR_SEQDATA_, value,
  1             NULL_, status)
END DO

IF (status .NE. END_OF_VAR) CALL UserStatusHandler (status)

ave = sum / count
.
.
```

Attribute rEntry writes In this example a set of attribute regualr entries for a particular rVariable will be written.

```
.
.
INCLUDE '<path>CDF.INC'
.
.
INTEGER*4 id                        ! CDF identifier (handle)
INTEGER*4 status                    ! Status returned from CDF library.
REAL*4    scale(2)                  ! Scale, minimum/maximum.

DATA scale/-90.0,90.0/
.
.
CALL CDF_lib (SELECT_, CDF_, id,
0                      VAR_NAME_, 'LATITUDE',
1                      ATTR_NAME_, 'FIELDNAM',
2             PUT_, rENTRY_DATA_, CDF_CHAR, 20, 'Latitude           ',
3             SELECT_, ATTR_NAME_, 'SCALE',
4             PUT_, rENTRY_DATA_, CDF_REAL4, 2, scale,
5             SELECT_, ATTR_NAME_, 'UNITS',
6             PUT_, rENTRY_DATA_, CDF_CHAR, 20, 'Degrees north      ',
7             NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.9 Interpreting CDF Status Codes

Most CDF functions return a status code of type `INTEGER*4`. The symbolic names for these codes are defined in `cdf.inc` and should be used in your applications rather than using the true numeric values. Appendix C explains each status code. When testing the status code returned from a CDF function, the following rules apply.

`status > CDF_OK`	Indicates successful completion but some additional information is provided. These are informational codes.
`status = CDF_OK`	Indicates successful completion.
`CDF_WARN < status < CDF_OK`	Indicates that the function completed but probably not as expected. These are warning codes.
`status < CDF_WARN`	Indicates that the function did not complete. These are error codes.

The following example shows how you could check the status code returned from CDF functions.

```
INTEGER*4 status
.
.
```

```
CALL CDF_function (..., status)   ! any CDF function returning status
IF (status .NE. CDF_OK) THEN
  CALL UserStatusHandler (status, ...)
  .
  .
END IF
```

In your own status handler you can take whatever action is appropriate to the application. An example status handler follows. Note that no action is taken in the status handler if the status is `CDF_OK`.

```
INCLUDE '<path>cdf.inc'

SUBROUTINE UserStatusHandler (status)
INTEGER*4 status

CHARACTER message*(CDF_ERRTEXT_LEN)

IF (status .LT. CDF_WARN) THEN
  WRITE (6,10)
10   FORMAT (' ','An error has occurred, halting...')
  CALL CDF_error (status, message)
  WRITE (6,11) message
11   FORMAT (' ',A)
  STOP
ELSE
  IF (status .LT. CDF_OK) THEN
    WRITE (6,12)
12     FORMAT (' ','Warning, function may not have completed as expected...')
    CALL CDF_error (status, message)
    WRITE (6,13) message
13     FORMAT (' ',A)
  ELSE
    IF (status .GT. CDF_OK) THEN
      WRITE (6,14)
14       FORMAT (' ','Function completed successfully, but be advised that...')
      CALL CDF_error (status, message)
      WRITE (6,15) message
15       FORMAT (' ',A)
    END IF
  END IF
END IF

RETURN
END
```

Explanations for all CDF status codes are available to your applications through the function `CDF_error`. `CDF_error` writes to a text string an explanation of a given status code.

5.10 CDF_EPOCH Values

Two subroutines exist which compute and breakdown CDF_EPOCH values. These subroutines should be used by applications using the CDF_EPOCH data type. Section 2.5.4 describes EPOCH values.

5.10.1 compute_EPOCH

compute_EPOCH calculates a CDF_EPOCH value given the individual components.

```
SUBROUTINE compute_EPOCH (year, month, day, hour, minute, second, msec, EPOCH)

INTEGER*4 year              ! in  -- Year (AD, eg. 1991).
INTEGER*4 month             ! in  -- Month (1-12).
INTEGER*4 day               ! in  -- Day (1-31).
INTEGER*4 hour              ! in  -- Hour (0-23).
INTEGER*4 minute            ! in  -- Minute (0-59).
INTEGER*4 second            ! in  -- Second (0-59).
INTEGER*4 msec              ! in  -- Millisecond (0-999).
REAL*8    EPOCH             ! out -- CDF_EPOCH value.
```

compute_EPOCH is a subroutine in the CDF library.

5.10.2 EPOCH_breakdown

EPOCH_breakdown breaks down a CDF_EPOCH value into the individual components.

```
SUBROUTINE EPOCH_breakdown (EPOCH, year, month, day, hour, minute, second,
1                           msec)

REAL*8    EPOCH           ! in  -- The CDF_EPOCH value.
INTEGER*4 year            ! out -- Year (AD, eg. 1991).
INTEGER*4 month           ! out -- Month (1-12).
INTEGER*4 day             ! out -- Day (1-31).
INTEGER*4 hour            ! out -- Hour (0-23).
INTEGER*4 minute          ! out -- Minute (0-59).
INTEGER*4 second          ! out -- Second (0-59).
INTEGER*4 msec            ! out -- Millisecond (0-999).
```

EPOCH_breakdown is a subroutine in the CDF library.

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. `CDFskeleton` 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 (`[`) to delimit a variable name.
5. When specifying a character string attribute entry value, do not use a hypen (`-`) to delimit the string or strings (if the string is split across one or more lines).
6. All items are referenced from one (1). These include gAttribute gEntry numbers and NRV variable index values.

In the descriptions that follow, optional fields are shown in brackets ([...]).

A.2 Header Section

The header section contains general information about the CDF. The format of the header section is as follows...

```
#header

                    CDF NAME: <cdf-name>
                DATA ENCODING: <data-encoding>
                    MAJORITY: <variable-majority>
                      FORMAT: <cdf-format>

!     Variables   G.Attributes V.Attributes   Records     Dims       Sizes
!     ---------   ------------ ------------   -------     ----    -----------
   <rVars>/<zVars>   <gAttrs>    <vAttrs>   <n-recs>/z  <n-dims>  <dim-sizes>
```

The fields are defined as follows...

`<cdf-name>` The name of the CDF. When `SkeletonTable` creates a skeleton table, this will be the name of the corresponding CDF (not the full pathname specified). When `CDFskeleton` reads a skeleton table, this will be the name of the CDF created unless a CDF pathname is specified on the command line. If the CDF name in the skeleton table is to be used, a full pathname must be specified (if desired) or else the CDF will be created in the default/current directory.

`<data-encoding>` The data encoding of the CDF. When specifying a data encoding to `CDFskeleton`, the following are valid: `HOST`, `NETWORK`, `VAX`, `SUN`, `MIPSEB`, `MIPSEL`, `IBMRS`, `HP`, `IBMPC`, 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 2.4.

`<variable-majority>` The variable majority of the CDF. This may be either `ROW` or `COLUMN`. Variable majority is described in Section 2.7.9.

`<cdf-format>` The format of the CDF. This may be either `SINGLE` or `MULTI`. CDF formats are described in Section 2.3. 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 `CDFskeleton` to read skeleton tables created with `SkeletonTable` in CDF V2.0, this line was made optional. If omitted, `CDFskeleton` will create a multi-file CDF by 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 `CDFskeleton` 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 `CDFskeleton` 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 `CDFskeleton` 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 `CDFskeleton` 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 `CDFskeleton` 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 NRV (non-record-variant) variables. NRV variables are described in Section 2.7.7.

`<n-dims>` The number of dimensions for the rVariables in the CDF.

`<dim-sizes>` The dimension sizes for the rVariables in the CDF — one value per dimension. If the rVariables have zero (0) dimensions, this field would be left blank.

An example header section for a CDF with 2-dimensional rVariables follows...

```
#header

                CDF NAME: sample2
           DATA ENCODING: NETWORK
                MAJORITY: ROW
                  FORMAT: SINGLE

! Variables G.Attributes V.Attributes Records  Dims   Sizes
! --------- ------------ ------------ -------  ----  -------
    14/0          18             4        1/z      2   180 360
```

If the rVariables had zero dimensions, the header section would be as follows...

```
#header
```

```
                CDF NAME: sample0
           DATA ENCODING: NETWORK
                MAJORITY: ROW
                  FORMAT: SINGLE

! Variables G.Attributes V.Attributes Records  Dims  Sizes
! --------- ------------ ------------ -------  ----  -----
    14/0          18             4        1/z      0
```

A.3 gAttributes Section

The gAttributes section contains the definition of each gAttribute as well as any gEntries for those gAttributes. The format of the gAttributes section is as follows...

```
#GLOBALattributes

[<global-scope-attribute-definition>
 <global-scope-attribute-definition>
 <global-scope-attribute-definition>
                  .
                  .
                  .
 <global-scope-attribute-definition>]
```

Where `<global-scope-attribute-definition>`, needless to say, is a gAttribute definition.

Zero or more gAttribute definitions are allowed (there is no limit on the number of attributes that a CDF may have). The format of each gAttribute definition is as follows...

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

  <attr-name>   [<entry-n>:    <data-type>    <value>
                 <entry-n>:   [<data-type>]   <value>
                 <entry-n>:   [<data-type>]   <value>
                     .              .            .
                     .              .            .
                     .              .            .
                 <entry-n>:   [<data-type>]   <value>] .         ! note the "."
```

The fields are defined as follows...

`<attr-name>` — The name of the gAttribute. The name must be delimited with a character not appearing in the name itself (eg. `"TITLE"` or `'History'`). The delimiting characters are not part of the gAttribute name in the CDF.

`<entry-n>` The gEntry number. More than one gEntry may be specified for a gAttribute and there are no restrictions on the gEntry numbers which may be used (except that they must be greater than zero).

`<data-type>` The data type for the gEntry. The data type must be one of the following: `CDF_BYTE`, `CDF_INT1`, `CDF_UINT1`, `CDF_INT2`, `CDF_UINT2`, `CDF_INT4`, `CDF_UINT4`, `CDF_REAL4`, `CDF_FLOAT`, `CDF_REAL8`, `CDF_DOUBLE`, `CDF_EPOCH`, `CDF_CHAR`, or `CDF_UCHAR`. The `<data-type>` field is optional for all but the first gEntry specified. If omitted, the data type of the previous gEntry is assumed.

`<value>` The value(s) for the gEntry. A period (`.`) follows the value(s) of the last gEntry for a gAttribute.

Attribute Entry Values

An attribute entry can have more than one element of the specified data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), each character is an element. The character string must be delimited with a character not appearing in the string itself and the entire delimited string must be enclosed in braces (eg. `{ "The CDF title." }`). If the string will not fit on one line, it may be continued on additional lines. The substrings are each delimited with a unique character and a dash (`-`) is placed at the end (after the terminating delimiter) of each line except the last one. For example,

```
{ "This is a longer " -
  "CDF title that will" -
  " not fit on one line." }
```

For non-character string data types, the elements are enclosed in braces and separated by commas (eg. `{ 1, 2, 3 }`). If the elements will not all fit on one line, they may be continued on additional lines. For example,

```
{ 1.0, 2.0, 3.0, 4.0, 5.0,
  6.0, 7.0, 8.0, 9.0, 10.0 }
```

Note that an individual element value may not be split across lines.

The format of a value for the `CDF_EPOCH` data type (which is also considered a non-character string data type) is defined in Section 2.5.4. 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:    CDF_INT2     { 7 }
                        5:                 { 12 }.
```

A.4 vAttributes Section

The vAttributes section contains the names of the vAttributes in the CDF. Any rEntries or zEntries for these vAttributes are defined in the rVariables/zVariables sections (following the definition of the corresponding variable). The format of the vAttributes section is as follows...

```
#VARIABLEattributes

[<attribute-name>
 <attribute-name>
 <attribute-name>
        .
        .
        .
 <attribute-name>]
```

Where `<attribute-name>` is a vAttribute name delimited with a character not appearing in the name itself (eg. `"VALIDMIN"` or `'Units'`). The delimiting characters are not part of the vAttribute name in the CDF. There may be zero or more vAttribute names (there is no limit on the number of attributes that a CDF may have).

An example vAttributes section follows...

```
#VARIABLEattributes

  "FIELDNAM"
  "VALIDMIN"
  "Units"
```

A.5 rVariables Section

The rVariables section contains the definition of each rVariable in the CDF, the values for any vAttribute rEntries associated with each rVariable, and (optionally) data values for those rVariables that are non-record-variant (NRV). 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 "."

   [<indices> = <NRV-value>
    <indices> = <NRV-value>
    <indices> = <NRV-value>
        .             .
        .             .
        .             .
    <indices> = <NRV-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 (eg. "EPOCH" or 'Temperature'). The delimiting characters are not part of the rVariable name in the CDF.

<var-data-type> The data type for the rVariable. The data type must be one of the following: CDF_BYTE, CDF_INT1, CDF_UINT1, CDF_INT2, CDF_UINT2, CDF_INT4, CDF_UINT4, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_CHAR, or CDF_UCHAR.

<n-elems> The number of elements of the data type. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in each string. For non-character string data types, this value must be one (1).

<rec-vary> The record variance of the rVariable. This must be either T (the values vary from record to record) or F (the values do not vary from record to record).

<dim-varys> The dimension variances of the rVariable. For each dimension, there must be either a T (the values vary along that dimension) or F (the values do not vary along that dimension). Each dimension variance must be separated by at least one space. If the rVariables have zero dimensions, this field would be left blank.

<attr-name> The name of the vAttribute for which to specify a 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 (eg. "SCALEMAX" or 'range'). The delimiting characters are not part of the vAttribute name in the CDF.

<entry-data-type> The data type for the vAttribute rEntry. The data type must be one of the following: CDF_BYTE, CDF_INT1, CDF_UINT1, CDF_INT2, CDF_UINT2, CDF_INT4, CDF_UINT4, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_CHAR, or CDF_UCHAR.

<entry-value> The value(s) for the vAttribute rEntry. The format of attribute entry values is described in Section A.3.

NOTE: The last rEntry MUST be followed by a period (.). If no rEntries are specified for an rVariable, the period must still be present.

<indices> The indices of an NRV value. The indices are enclosed in brackets and separated by commas (eg. [23, 1] or [1, 80]). If the rVariables have zero dimensions, [ ] would be specified (the brackets are still required).

<NRV-value> The value at the given indices. For character string data types (CDF_CHAR or CDF_UCHAR), the string must be delimited with a unique character and enclosed in braces ({...}) in the same manner as for an attribute entry for a character string data type. For non-character string data types, the value is not enclosed in braces (the braces are not necessary because there can only be one element). The format for CDF_EPOCH values is described in Section 2.5.4.

NRV values may only be specified for NRV variables (described in Section 2.7.7).

The vAttribute rEntries are optional. If omitted, the terminating period is still required. The NRV values are also optional (NRV variables are described in Section 2.7.7).

Several sample rVariable definitions for a CDF with 2-dimensional rVariables follow...

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

  "Latitude"        CDF_REAL4       1          F         F T

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

    "VALIDMIN"      CDF_REAL4    { -90.0 }
    "VALIDMAX"      CDF_REAL4    { 90.0 }
    "scale"         CDF_REAL4    { -60.0, 60.0 }.

    [ 1, 1 ] = -60.0
    [ 1, 2 ] = -30.0
    [ 1, 3 ] = 0.0
    [ 1, 4 ] = 30.0
    [ 1, 5 ] = 60.0



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

  "EPOCH"           CDF_EPOCH       1          T         F F

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

    "scale"         CDF_EPOCH    { 10-Oct-1991 00:00:00.000,
                                   20-Oct-1991 23:59:59.999 }.



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

  'Tmp'             CDF_INT2        1          T         T T

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

    'Fieldname'     CDF_CHAR     { "Temperature (C)" }.
```

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

  "Pres_lvl"        CDF_REAL4       1          F         F F

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

    .   ! no attribute entries

    [ 1, 1 ] = 1013.0
```

Several sample variable definitions for a CDF with 0-dimensional rVariables follow...

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

  "Latitude"        CDF_REAL4       1          F

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

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

    [ ] = -12.3
```

A.6 zVariables Section

The optional zVariables section contains the definition of each zVariable in the CDF, the values for any vAttribute zEntries associated with each zVariable, and (optionally) data values for those zVariables that are non-record-variant (NRV). 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 "."

    [<indices> = <NRV-value>
     <indices> = <NRV-value>
     <indices> = <NRV-value>
         .            .
         .            .
         .            .
     <indices> = <NRV-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 (eg. "EPOCH" or 'Temperature'). The delimiting characters are not part of the zVariable name in the CDF.
<var-data-type>	The data type for the zVariable. The data type must be one of the following: CDF_BYTE, CDF_INT1, CDF_UINT1, CDF_INT2, CDF_UINT2, CDF_INT4, CDF_UINT4, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_CHAR, or CDF_UCHAR.
<n-elems>	The number of elements of the data type. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in each string. For non-character string data types, this value must be one (1).
<dims>	The number of dimensions for the zVariable.
<sizes>	The dimension sizes — one value per dimension. If the zVariable has zero (0) dimensions, this field would be left blank.

`<rec-vary>` The record variance of the zVariable. This must be either `T` (the values vary from record to record) or `F` (the values do not vary from record to record).

`<dim-varys>` The dimension variances of the zVariable. For each dimension, there must be either a `T` (the values vary along that dimension) or `F` (the values do not vary along that dimension). Each dimension variance must be separated by at least one space. If the zVariable has zero dimensions, this field would be left blank.

`<attr-name>` The name of the vAttribute for which to specify a zEntry for this zVariable. The vAttribute must have been specified in the vAttributes section. The name must be delimited with a character not appearing in the name itself (eg. `"SCALEMAX"` or `'range'`). The delimiting characters are not part of the vAttribute name in the CDF.

`<entry-data-type>` The data type for the vAttribute zEntry. The data type must be one of the following: `CDF_BYTE`, `CDF_INT1`, `CDF_UINT1`, `CDF_INT2`, `CDF_UINT2`, `CDF_INT4`, `CDF_UINT4`, `CDF_REAL4`, `CDF_FLOAT`, `CDF_REAL8`, `CDF_DOUBLE`, `CDF_EPOCH`, `CDF_CHAR`, or `CDF_UCHAR`.

`<entry-value>` The value(s) for the vAttribute zEntry. The format of attribute entry values is described in Section A.3.

NOTE: The last zEntry MUST be followed by a period (`.`). If no zEntries are specified for a zVariable, the period must still be present.

`<indices>` The indices of an NRV value. The indices are enclosed in brackets and separated by commas (eg. `[ 23, 1 ]` or `[ 1, 80 ]`). If the zVariable has zero dimensions, `[ ]` would be specified (the brackets are still required).

`<NRV-value>` The value at the given indices. For character string data types (`CDF_CHAR` or `CDF_UCHAR`), the string must be delimited with a unique character and enclosed in braces (`{...}`) in the same manner as for an attribute entry for a character string data type. For non-character string data types, the value is not enclosed in braces (the braces are not necessary because there can only be one element). The format for `CDF_EPOCH` values is described in Section 2.5.4.

NRV values may only be specified for NRV variables (described in Section 2.7.7).

The vAttribute zEntries are optional. If omitted, the terminating period is still required. The NRV values are also optional (NRV variables are described in Section 2.7.7).

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
! --------           ----     --------  ----  ----------  --------  ---------

  "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: Fri 31 Jan 1992 14:44:16

#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      " } .


! 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

CDF's IDL Support

B.1 Introduction

The CDF distribution includes a set of IDL functions/procedures which allow access to the CDF library. The functions/procedures are used in the same manner as the Standard Interface is used in programs written in C.[1] Every Standard Interface function has been implemented. CDFs can be both read and written/modified.

Research Systems, Inc. (the developers of IDL) are also going to implement an interface to CDF which will become part of the IDL product. It differs from the interface provided with the CDF distribution in that it is intended more for the non-programmer. Which interface you choose to use will depend on your personal preferences.

CDF's interface relies on IDL's support of dynamic linking. IDL in turn relies on the operating system being used to support dynaminc linking in a reasonable way. Currently, only VAX/VMS, Sun/SunOS, HP9000/HP-UX, and IBM-RS6000/AIX support dynamic linking in a way which makes it possible to use CDF's interface to IDL. Because IDL's CDF interface is built into the IDL executable, it does not rely on dynamic linking and is therefore available on every machine on which IDL is supported.

B.2 Using CDF's IDL Interface

This section assumes that you are using the 'definitions' file provided with the CDF distribution (and that it has been properly configured and named by your system manager). The 'definitions' file is a command file named `DEFINITIONS.COM` on VMS systems and a script file named `definitions.<shell>` on UNIX systems (where `<shell>` is the shell being used). The 'definitions' file defines the logical name `IDL_STARTUP` on VMS systems and the environment variable `IDL_STARTUP` on UNIX systems to the full directory path of an IDL startup procedure provided with the CDF distribution. When IDL is started, this startup procedure is automatically executed (because `IDL_STARTUP` is defined). The startup procedure performs the following functions...

[1] The Internal Interface has not yet been implemented.

1. Makes known to IDL the functions/procedures which are used to call the CDF library (and perform other utility functions).
2. Sets IDL's internal path which is used when searching for a function/procedure being executed.
3. Executes a procedure (`cdf.pro`) which creates a number of local variables necessary for calling CDF functions/procedures and then interpreting the results.

A list of messages should be displayed indicating the CDF functions/procedures which have been compiled.

B.3 CDF "Include" Files

A number of parameters (macro constants) are necessary when calling functions in the CDF library from a C program. These parameters are defined in `cdf.h`. When calling the CDF functions in IDL, local variables are used to accomplish the same thing. An IDL procedure, `cdf.pro`, is provided which creates/initializes the necessary local variables. `cdf.pro` is executed automatically by the startup procedure when you run IDL. All necessary CDF parameters (local variables) will be available except for the status codes. Most users will not need the status codes since the `CDFerror` function can be used to determine the explanation of a status code. If you do require the status codes, execute the procedure `cdf1.pro` (the IDL internal path will know where to find it).

B.4 Available Functions/Procedures

The CDF functions available at the IDL command line exactly mirror the Standard Interface functions available for a program written in C. Chapter 4 describes the functionality of each in detail. The syntax required when calling each function is as follows...

```
IDL> status = CDFcreate (CDFname, numDims, dimSizes, encoding, majority, $
IDL>                     id)
IDL> status = CDFopen (CDFname, id)
IDL> status = CDFdoc (id, version, release, copyright)
IDL> status = CDFinquire (id, numDims, dimSizes, encoding, majority, $
IDL>                      maxRec, numVars, numAttrs)
IDL> status = CDFclose (id)
IDL> status = CDFdelete (id)
IDL> status = CDFerror (status, text)
IDL> status = CDFattrCreate (id, attrName, attrScope, attrNum)
IDL> status = CDFattrRename (id, attrNum, attrName)
IDL> status = CDFattrInquire (id, attrNum, attrName, attrScope, maxEntry)
IDL> status = CDFattrEntryInquire (id, attrNum, entryNum, dataType, $
IDL>                               numElems)
IDL> status = CDFattrPut (id, attrNum, entryNum, dataType, numElems, value)
IDL> status = CDFattrGet (id, attrNum, entryNum, value)
IDL> status = CDFattrCorrectScope (id, attrNum, attrScope)
IDL> status = CDFvarCreate (id, varName, dataType, numElems, recVary, $
IDL>                        dimVarys, varNum)
```

```
IDL> status = CDFvarRename (id, varNum, varName)
IDL> status = CDFvarInquire (id, varNum, varName, dataType, numElems, $
IDL>                         recVary, dimVarys)
IDL> status = CDFvarPut (id, varNum, recNum, indices, value)
IDL> status = CDFvarGet (id, varNum, recNum, indices, value)
IDL> status = CDFvarHyperPut (id, varNum, recStart, recCount, recInterval, $
IDL>                          indices, counts, intervals, buffer)
IDL> status = CDFvarHyperGet (id, varNum, recStart, recCount, recInterval, $
IDL>                          indices, counts, intervals, buffer)
IDL> status = CDFvarClose (id, varNum)
IDL> attrNum = CDFattrNum (id, attrName)
IDL> varNum = CDFvarNum (id, varName)
```

The IDL data types to use for each argument are always longwords or arrays of longwords (except for attribute entry and variable values). `CDFid` and `CDFstatus` are also longwords. Chapter 4 may be used as a guide. Arguments passed to the CDF functions must obviously be created and initialized before the call. Arrays are necessary for dimension sizes, variances, etc. Arguments returned by the CDF functions will be created to be of the proper data type and size.

A procedure, `row_to_col`, is also provided which may be used to change the majority of an array (or arrays) of values from row-major to column-major. This procedure is necessary because CDF can store variable values in either row or column major order. The syntax of `row_to_col` is as follows...

```
IDL> row_to_col, inArray, outArray, numDims, dimSizes, numBytes, $
                 arrayCount
```

where...

`inArray`	The array (or arrays) of values to convert — row major ordering. Any valid IDL data type is allowed.
`outArray`	The converted array (or arrays) of values — column major ordering. This array will be created by `row_to_col`.
`numDims`	The number of dimensions in the array (or arrays). This value must be a longword.
`dimSizes`	The size of each dimension. This must be an array of longword — one longword per dimension.
`numBytes`	The number of bytes in each value. This value must be a longword.
`arrayCount`	The number of arrays to convert. The order of the arrays will not be affected. This value must be a longword.

Multiple arrays (records) of CDF variable values should always be stored with the last dimension of the IDL array corresponding to the record number. For example, assume a CDF variable with two dimensions whose sizes are `[180,360]`. If `50` records are to be stored, an IDL array with three dimensions whose sizes are `[180,360,50]` should be used. If the CDF variable's have row major ordering, `row_to_col` would be called with `numDims` set to `2`, `dimSizes` set to `[180,360]`, and `arrayCount` set to `50`.

B.5 Example IDL Session

The following IDL session will open a CDF (unsuccessfully at first — typo), inquire about a variable named 'LATITUDE', and then close the CDF.

```
.
.
IDL> status=CDFopen('yrdy3',id)
IDL> status=CDFerror(status,text)
IDL> print,text
NO_SUCH_CDF: The specified CDF does not exist.
IDL> status=CDFopen('test2',id)
IDL> status=CDFerror(status,text)
IDL> print,text
CDF_OK: Function completed successfully.
IDL> status=CDFvarInquire(id,CDFvarNum(id,'LATITUDE'),varName,dataType,$
IDL> numElems,recVary,dimVarys)
IDL> status=CDFerror(status,text)
IDL> print,text
CDF_OK: Function completed successfully.
IDL> print,dataType,numElems
          21           1
IDL> status=CDFclose(id)
IDL> status=CDFerror(status,text)
IDL> print,text
CDF_OK: Function completed successfully.
.
.
```

As you can see, checking the return status from each call to the CDF library can be fairly tedious. A procedure such as the following could be used to ease status code checking...

```
pro checkstatus, status
@cdf
@cdf1
if (status lt CDF_OK) then begin
  statusT = CDFerror (status, text)
  print, text
endif
return
end
```

The sample IDL session could then be as follows...

```
.
.
IDL> status=CDFopen('yrdy3',id)
```

```
IDL> checkstatus,status
NO_SUCH_CDF: The specified CDF does not exist.
IDL> checkstatus,CDFopen('test2',id)
CDF_OK: Function completed successfully.
IDL> status=CDFvarInquire(id,CDFvarNum(id,'LATITUDE'),varName,dataType,$
IDL> numElems,recVary,dimVarys)
IDL> checkstatus,status
CDF_OK: Function completed successfully.
IDL> print,dataType,numElems
          21           1
IDL> checkstatus,CDFclose(id)
CDF_OK: Function completed successfully.
.
.
```

As you can see, a call to a CDF function can be embedded in a call to `checkstatus`.

B.6 Example IDL Function

The following example function is also available in the 'samples' subdirectory of the CDF distribution in a file named `idlload1.pro`.

```
;
; This function will load all data for a specified variable from a CDF into
; IDL.  All of the IDL/CDF interface routines may entered at the IDL command
; line, but for demonstration purposes they will be used in an IDL function.
;
; Argument descriptions:                          data type
; CDF_name      input CDF name                    string
; var_name      variable name in CDF              string
; data          array to be returned to IDL       (see creation below)
;
; Executing
; status = get_var('SAMPLE_CDF','VARNAME',data)
;
function get_var, CDF_name, var_name, data
;
; include all CDF definitions and error codes
;
@cdf
@cdf1
;
; Open the CDF, note that CDF_id will be returned as the proper data type
;
status = CDFopen(CDF_name,CDF_id)
;
; check the status of the open, if a error is detected print a error and
```

```
; return
;
if(status lt CDF_OK)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; Inquire about the structure of the CDF
;
status = CDFinquire(CDF_id,num_dims,dim_sizes,encoding,majority,$
                    max_record_num,num_vars,num_attrs)
if(status lt CDF_OK)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; Get the number in the CDF of the variable
;
var_num = CDFvarNum(CDF_id, var_name)
if(var_num lt CDF_OK)then begin
   status = CDFerror(var_num, error_mes)
   print,error_mes
   return,status
endif
;
; Inquire about the variable
;
status = CDFvarInquire(CDF_id, var_num, out_var_name, data_type, $
                       num_bytes, record_variance, dim_variances)
if(status lt CDF_OK)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; We will retrieve some of the metadata from this variable,
; since CDF attributes can possibly be arrays of attributes we
; will need to test for this and create a output array if necessary
; scalars attributes will be handled by the software.  FIELDNAM is
; a CDF standard attribute and is a description of the variable
;
attr_num = CDFattrNum(CDF_id, "FIELDNAM")
if(attr_num < CDF_OK)then begin
   status = CDFerror(attr_num, error_mes)
   print,error_mes
   return,status
endif
status = CDFattrInquire(CDF_id, attr_num, out_attr_name, $
                        attr_scope, max_entry)
```

```
if(status lt CDF_OK)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; Need the data type and size of this attribute
;
status = CDFattrEntryInquire(CDF_id, attr_num, var_num, $
                             attr_data_type, num_elements)
if(status lt CDF_OK)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
status = CDFattrGet(CDF_id, attr_num, var_num, attr_value)
if(status lt CDF_OK)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
print,'Loading variable ',var_name,': ',attr_value
;
; We will now get ready to load all data for this variable, use the CDF
; structure to initialize all parameters to perform this operation
;
recStart = 0l
recCount = max_record_num+1
recInterval=1l
indices = lonarr(CDF_MAX_DIMS)
counts = lonarr(CDF_MAX_DIMS)
intervals = lonarr(CDF_MAX_DIMS)
for dim_num=0, CDF_MAX_DIMS-1 do begin
    indices(dim_num) = 0;
    counts(dim_num) = dim_sizes(dim_num)
    intervals(dim_num) = 1
endfor
;
; Retrieve the data
;
status = CDFvarHyperGet(CDF_id,var_num, recStart,recCount,recInterval, $
                        indices,counts,intervals,inbuf)
if(status lt CDF_OK)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; IDL needs to have the CDF in column majority in order to plot properly,
; we will supply a routine that will convert from ROW to COLUMN major.
; Converting character data is not supported in this release
```

```
;
if((majority eq ROW_MAJOR) and $
   (data_type ne CDF_CHAR and data_type ne CDF_UCHAR))then begin
   print,'Converting from ROW to COLUMN major'
;
; Need to know the number of bytes of this data type (e.g. 1,2,4,8)
;
   nbytes,data_type, num_bytes_dt
;
; Now convert
;
   row_to_col,inbuf, data, num_dims, dim_sizes, num_bytes_dt, recCount
endif else begin
;
; No conversion necessary, load input buffer into return data buffer
;
   data = inbuf
endelse
status=CDFclose(CDF_id)
return,CDF_OK
end
pro nbytes,data_type,num_bytes
@cdf
num_bytes = 0l
Case 1 of
 (data_type eq CDF_INT1)   : num_bytes = 1l
 (data_type eq CDF_INT2)   : num_bytes = 2l
 (data_type eq CDF_INT4)   : num_bytes = 4l
 (data_type eq CDF_UINT1)  : num_bytes = 1l
 (data_type eq CDF_UINT2)  : num_bytes = 2l
 (data_type eq CDF_UINT4)  : num_bytes = 4l
 (data_type eq CDF_REAL4)  : num_bytes = 4l
 (data_type eq CDF_REAL8)  : num_bytes = 8l
 (data_type eq CDF_EPOCH)  : num_bytes = 8l
 (data_type eq CDF_BYTE)   : num_bytes = 1l
 (data_type eq CDF_FLOAT)  : num_bytes = 4l
 (data_type eq CDF_DOUBLE) : num_bytes = 8l
 (data_type eq CDF_CHAR)   : num_bytes = 1l
 (data_type eq CDF_UCHAR)  : num_bytes = 1l
endcase
return
end
```

Appendix C

CDF 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 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.

C.2 Success Code and Message

`CDF_OK`	Function completed successfully.

C.3 Informational Codes and Messages

Informational codes indicate success but provide some additional information.

`VERSION_1_CDF`	This is a Version 1 CDF, some restrictions may apply. The restrictions are described in Section 2.2.2.
`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 Section 2.7.5.

C.4 Warning Codes and Messages

Warning codes are considered non-normal and indicate a potential problem that may lead to errors in program operation or errors of data interpretation.

ATTR_NAME_TRUNC	Attribute name truncated to CDF_ATTR_NAME_LEN characters. The attribute was created but with a truncated name.
CDF_INCOMPATIBLE	Certain operations on Version 2 CDFs are incompatible with the Obsolete Interface (VMS only). These deal mainly with attributes having different data types and/or number of elements per entry. The incompatibilities are described in Section 2.1.3.
CDF_NAME_TRUNC	CDF path name truncated to CDF_PATHNAME_LEN characters. The CDF was created but with a truncated name.
NO_PADVALUE_SPECIFIED	A pad value has not yet been specified. The default pad value is currently being used for the variable.
NO_VARS_IN_CDF	This CDF contains no rVariables. The operation performed is not applicable to a CDF with no rVariables.
SINGLE_FILE_FORMAT	The specified operation is not applicable to CDFs with the single-file format. CDF formats are described in Section 2.3.
VAR_ALREADY_CLOSED	The specified variable is already closed.
VAR_NAME_TRUNC	Variable name truncated to CDF_VAR_NAME_LEN characters. The variable was created but with a truncated name.

C.5 Error Codes and Messages

These codes are errors. The application should take corrective action or halt. Note that some errors may not be fatal to an application.

ATTR_EXISTS	Named attribute already exists - cannot create it. Each attribute in a CDF must have a unique name. Note that trailing blanks are ignored by the CDF library when comparing attribute names.
BAD_ARGUMENT	An illegal/undefined argument was passed. Check that all arguments are properly declared and initialized.
BAD_ATTR_NAME	Illegal attribute name specified. Attribute names must contain at least one character and each character must be printable.
BAD_ATTR_NUM	Illegal attribute number specified. Attribute numbers must be zero or greater for C applications and one or greater for FORTRAN applications.
BAD_CDFSTATUS	Unknown CDF status value received. The status code specified is not used by the CDF library.

BAD_CDF_ID	CDF identifier is unknown or invalid. The CDF identifier specified is not for a currently open CDF.
BAD_CDF_NAME	Illegal CDF name specified. CDF names must contain at least one character and each character must be printable.
BAD_DATA_TYPE	Unknown data type specified.
BAD_DIM_COUNT	Illegal dimension count specified.
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.
BAD_DIM_INTERVAL	Illegal dimension interval specified.
BAD_DIM_SIZE	One or more dimension size specified as zero or less.
BAD_ENCODING	Unknown or unsupported data encoding.
BAD_ENTRY_NUM	Illegal attribute entry number specified.
BAD_EXTEND_RECS	An illegal number of extend records has been specified. The number of extend records must be at least one (1).
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.
BAD_FORMAT	Unknown CDF format specified.
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).
BAD_MAJORITY	Unknown majority specified.
BAD_MALLOC	Unable to allocate dynamic memory - system limit reached. Contact CDF User Support.
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.
BAD_NUM_ELEMS	The number of elements of the data type is illegal. The number of elements must be at least one (1). For non-character string variables, the number of elements must always be one (1).
BAD_REC_COUNT	Illegal record count specified.
BAD_REC_INTERVAL	Illegal record interval specified.
BAD_REC_NUM	Record number is out of range. Note that a valid value must be specified regardless of the record variance.
BAD_SCOPE	Unrecognized scope for attribute creation.

BAD_VAR_NAME	Illegal variable name specified. Variable names must contain at least one character and each character must be printable.
BAD_VAR_NUM	Illegal variable number specified.
CANNOT_CHANGE	Because of dependencies on the value, it cannot be changed.
CDF_CLOSE_ERROR	Error detected while trying to close CDF.
CDF_CREATE_ERROR	Cannot create CDF specified — error from file system. Make sure sure that sufficient privilege exists to create the CDF file(s) in the disk/directory location specified.
CDF_EXISTS	The CDF named already exists - cannot create it. The CDF library will not overwrite an existing CDF.
CDF_OPEN_ERROR	Cannot open CDF specified — error from file system. Check that the CDF file(s) are not corrupted and that sufficient privilege exists to open the CDF file(s).
CDH_CLOSE_ERROR	An error occurred while closing the `.cdh` file (V1 CDF).
CDH_OPEN_ERROR	An error occurred while opening the `.cdh` file (V1 CDF). Check that all of the CDF file(s) exist in the disk/directory specified and that sufficient privilege exists to access them.
CORRUPTED_V1_CDF	This Version 1 CDF is corrupted. An internal error has been detected in the CDF's control information. If the CDF file(s) are known to be uncorrupted, please contact CDF User Support.
CORRUPTED_V2_CDF	This Version 2 CDF is corrupted. An internal error has been detected in the CDF's control information. If the CDF file(s) are known to be uncorrupted, please contact CDF User Support.
END_OF_VAR	The value offset (for sequential access) is at the end of the variable. Reading beyond the end of the last physical record for a variable is not allowed (when performing sequential access).
HP_ENCODING_UNSUPPORTED	HP encoding is not supported on the host machine. The CDF may only be read on an HP 9000 series computer.
IBMPC_ENCODING_UNSUPPORTED	IBMPC encoding is not supported on the host machine. The CDF may only be read on an IBM PC.
IBMRS_ENCODING_UNSUPPORTED	IBMRS encoding is not supported on the host machine. The CDF may only be read on an IBM RS/6000 series computer.
IBM_PC_OVERFLOW	An operation involving a buffer greater than 64k bytes in size has been specified.
ILLEGAL_ON_V1_CDF	The specified operation is not allowed on Version 1 CDFs. Section 2.2.2 describes the restrictions when accessing Version 1 CDFs.
MIPSEB_ENCODING_UNSUPPORTED	MIPSEB encoding is not supported on the host machine. The CDF may only be read on a Silicon Graphics Iris or Power

	Series computer.
MIPSEL_ENCODING_UNSUPPORTED	MIPSEL encoding is not supported on the host machine. The CDF may only be read on a DECstation computer.
NeXT_ENCODING_UNSUPPORTED	NeXT encoding is not supported on the host machine. The CDF may only be read on a NeXT computer.
NO_ATTR_SELECTED	An attribute has not yet been selected. First specify the attribute on which to perform the operation.
NO_CDF_SELECTED	A CDF has not yet been selected. First specify the CDF on which to perform the operation.
NO_DELETE_ACCESS	Deleting is not allowed (read-only access). Make sure that delete access is allowed on the CDF file(s).
NO_ENTRY_SELECTED	An attribute entry has not yet been selected. First specify the entry number on which to perform the operation.
NO_STATUS_SELECTED	A CDF status code has not yet been selected. First specify the status code on which to perform the operation.
NO_SUCH_ATTR	Attribute not found in this CDF.
NO_SUCH_CDF	The specified CDF does not exist.
NO_SUCH_ENTRY	No such entry for specified attribute.
NO_SUCH_VAR	Variable not found in this CDF.
NO_VAR_SELECTED	A variable has not yet been selected. First specify the variable on which to perform the operation.
NO_WRITE_ACCESS	Writing is not allowed (read-only access). Make sure that write access is allowed on the CDF file(s).
READ_ONLY_DISTRIBUTION	Your CDF distribution has been build to allow only read access to CDFs. Check with your system manager if you require write access.
SUN_ENCODING_UNSUPPORTED	Sun encoding is not supported on the host machine. The CDF may only be read on a Sun computer.
TOO_MANY_VARS	A multi-file CDF on an IBM PC may contain only a limited number of variables due to the 8.3 file naming convention of MS-DOS. This consists of 100 rVariables and 100 zVariables.
V1_CDF_UNSUPPORTED	Version 1 CDFs are not supported on the host machine.
VAR_CLOSE_ERROR	Error detected while trying to close variable file.
VAR_CREATE_ERROR	An error occurred while creating a variable file in a multi-file CDF.
VAR_EXISTS	Named variable already exists - cannot create it. 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.

VAR_OPEN_ERROR	An error occurred while opening variable file. Check that sufficient privilege exists to open or create the variable file. Also make sure that the associated variable files exist.
VAR_READ_ERROR	Failed to read variable as requested — file read error.
VAR_WRITE_ERROR	Failed to write variable as requested — file write error.
VAX_ENCODING_UNSUPPORTED	VAX encoding is not supported on the host machine. The CDF may only be read on a VAX computer.

Appendix D

C Programming Summary

D.1 Introduction

Below is a summary of the CDF library functions callable from C applications. Each is written as an ANSI-style function prototype.

D.2 Standard Interface

```
CDFstatus CDFcreate(
char  *CDFname,                                          /* in  */
long  numDims,                                           /* in  */
long  dimSizes[],                                        /* in  */
long  encoding,                                          /* in  */
long  majority,                                          /* in  */
CDFid *id);                                              /* out */

CDFstatus CDFopen(
char  *CDFname,                                          /* in  */
CDFid *id);                                              /* out */

CDFstatus CDFdoc(
CDFid id,                                                /* in  */
long  *version,                                          /* out */
long  *release,                                          /* out */
char  text[CDF_DOCUMENT_LEN+1]);                         /* out */

CDFstatus CDFinquire(
CDFid id,                                                /* in  */
long  *numDims,                                          /* out */
long  dimSizes[CDF_MAX_DIMS],                            /* out */
long  *encoding,                                         /* out */
```

```
long  *majority,                                    /* out */
long  *maxRec,                                      /* out */
long  *numVars,                                     /* out */
long  *numAttrs);                                   /* out */

CDFstatus CDFclose(
CDFid id);                                          /* in  */

CDFstatus CDFdelete(
CDFid id);                                          /* in  */

CDFstatus CDFerror(
CDFstatus status,                                   /* in  */
char message[CDF_STATUSTEXT_LEN+1]);                /* out */

CDFstatus CDFattrCreate(
CDFid id,                                           /* in  */
char  *attrName,                                    /* in  */
long  attrScope,                                    /* in  */
long  *attrNum);                                    /* out */

long CDFattrNum(
CDFid id,                                           /* in  */
char  *attrName);                                   /* in  */

CDFstatus CDFattrRename(
CDFid id,                                           /* in  */
long  attrNum,                                      /* in  */
char  *attrName);                                   /* in  */

CDFstatus CDFattrInquire(
CDFid id,                                           /* in  */
long  attrNum,                                      /* in  */
char  *attrName,                                    /* out */
long  *attrScope,                                   /* out */
long  *maxEntry);                                   /* out */

CDFstatus CDFattrEntryInquire(
CDFid id,                                           /* in  */
long  attrNum,                                      /* in  */
long  entryNum,                                     /* in  */
long  *dataType,                                    /* out */
long  *numElements);                                /* out */

CDFstatus CDFattrPut(
CDFid id,                                           /* in  */
long  attrNum,                                      /* in  */
long  entryNum,                                     /* in  */
long  dataType,                                     /* in  */
long  numElements,                                  /* in  */
void  *value);                                      /* in  */
```

```
CDFstatus CDFattrGet(
CDFid id,                                                   /* in  */
long  attrNum,                                              /* in  */
long  entryNum,                                             /* in  */
void  *value);                                              /* out */

CDFstatus CDFattrCorrectScope(
CDFid id,                                                   /* in  */
long  attrNum,                                              /* in  */
long  attrScope)                                            /* in  */

CDFstatus CDFvarCreate(
CDFid id,                                                   /* in  */
char  *varName,                                             /* in  */
long  dataType,                                             /* in  */
long  numElements,                                          /* in  */
long  recVariance,                                          /* in  */
long  dimVariances[],                                       /* in  */
long  *varNum);                                             /* out */

long CDFvarNum(
CDFid id,                                                   /* in  */
char  *varName);                                            /* in  */

CDFstatus CDFvarRename(
CDFid id,                                                   /* in  */
long  varNum,                                               /* in  */
char  *varName);                                            /* in  */

CDFstatus CDFvarInquire(
CDFid id,                                                   /* in  */
long  varNum,                                               /* in  */
char  *varName,                                             /* out */
long  *dataType,                                            /* out */
long  *numElements,                                         /* out */
long  *recVariance,                                         /* out */
long  dimVariances[CDF_MAX_DIMS]);                          /* out */

CDFstatus CDFvarPut(
CDFid id,                                                   /* in  */
long  varNum,                                               /* in  */
long  recNum,                                               /* in  */
long  indices[],                                            /* in  */
void  *value);                                              /* in  */

CDFstatus CDFvarGet(
CDFid id,                                                   /* in  */
long  varNum,                                               /* in  */
long  recNum,                                               /* in  */
long  indices[],                                            /* in  */
```

```
void  *value);                                          /* out */

CDFstatus CDFvarHyperPut(
CDFid id,                                               /* in  */
long  varNum,                                           /* in  */
long  recStart,                                         /* in  */
long  recCount,                                         /* in  */
long  recInterval,                                      /* in  */
long  indices[],                                        /* in  */
long  counts[],                                         /* in  */
long  intervals[],                                      /* in  */
void  *buffer);                                         /* in  */

CDFstatus CDFvarHyperGet(
CDFid id,                                               /* in  */
long  varNum,                                           /* in  */
long  recStart,                                         /* in  */
long  recCount,                                         /* in  */
long  recInterval,                                      /* in  */
long  indices[],                                        /* in  */
long  counts[],                                         /* in  */
long  intervals[],                                      /* in  */
void  *buffer);                                         /* out */

CDFstatus CDFvarClose(
CDFid id,                                               /* in  */
long  varNum);                                          /* in  */
```

D.3 Internal Interface

```
CDFstatus CDFlib(...);

   CREATE_    CDF_              char *CDFname                 /* in */
                                long numDims                  /* in  */
                                long dimSizes[]               /* in  */
                                CDFid *id                     /* out */
              rVAR_             char *varName                 /* in  */
                                long dataType                 /* in  */
                                long numElements              /* in  */
                                long recVary                  /* in  */
                                long dimVarys                 /* in  */
                                long *varNum                  /* out */
              zVAR_             char *varName                 /* in  */
                                long dataType                 /* in  */
                                long numElements              /* in  */
                                long numDims                  /* in  */
                                long dimSizes[]               /* in  */
                                long recVary                  /* in  */
                                long dimVarys                 /* in  */
```

```
                          long *varNum                    /* out */
          ATTR_           char *attrName                  /* in  */
                          long scope                      /* in  */
                          long *attrNum                   /* out */

OPEN_     CDF_            char *CDFname                   /* in  */
                          CDFid *id                       /* out */

DELETE_   CDF_

CLOSE_    CDF_
          VAR_

SELECT_   CDF_            CDFid id                        /* in  */
          CDF_STATUS_     CDFstatus status                /* in  */
          rVAR_           long varNum                     /* in  */
          rVAR_NAME_      char *varName                   /* in  */
          rVARs_RECNUMBER_ long recNum                    /* in  */
          rVARs_RECCOUNT_ long recCount                   /* in  */
          rVARs_RECINTERVAL_ long recInterval             /* in  */
          rVARs_DIMINDICES_ long indices[]                /* in  */
          rVARs_RECCOUNTS_ long counts[]                  /* in  */
          rVARs_RECINTERVALS_ long intervals[]            /* in  */
          rVAR_SEQPOS_    long recNum                     /* in  */
                          long indices[]                  /* in  */
          zVAR_           long varNum                     /* in  */
          zVAR_NAME_      char *varName                   /* in  */
          zVAR_RECNUMBER_ long recNum                     /* in  */
          zVAR_RECCOUNT_  long recCount                   /* in  */
          zVAR_RECINTERVAL_ long recInterval              /* in  */
          zVAR_DIMINDICES_ long indices[]                 /* in  */
          zVAR_RECCOUNTS_ long counts[]                   /* in  */
          zVAR_RECINTERVALS_ long intervals[]             /* in  */
          zVAR_SEQPOS_    long recNum                     /* in  */
                          long indices[]                  /* in  */
          ATTR_           long attrNum                    /* in  */
          ATTR_NAME_      char *attrName                  /* in  */
          ENTRY_          long entryNum                   /* in  */
          rENTRY_         long entryNum                   /* in  */
          zENTRY_         long entryNum                   /* in  */

CONFIRM_  CDF_NAME_       char *CDFname                   /* out */
          rVAR_           long *varNum                    /* out */
          rVARs_RECNUMBER_ long *recNum                   /* out */
          rVARs_RECCOUNT_ long *recCount                  /* out */
          rVARs_RECINTERVAL_ long *recInterval            /* out */
          rVARs_DIMINDICES_ long indices[]                /* out */
          rVARs_DIMCOUNTS_ long counts[]                  /* out */
          rVARs_DIMINTERVALS_ long intervals[]            /* out */
          rVAR_SEQPOS_    long *recNum                    /* out */
                          long indices[]                  /* out */
```

```
            zVAR_               long *varNum                    /* out */
            zVAR_RECNUMBER_     long *recNum                    /* out */
            zVAR_RECCOUNT_      long *recCount                  /* out */
            zVAR_RECINTERVAL_   long *recInterval               /* out */
            zVAR_DIMINDICES_    long indices[]                  /* out */
            zVAR_DIMCOUNTS_     long counts[]                   /* out */
            zVAR_DIMINTERVALS_  long intervals[]                /* out */
            zVAR_SEQPOS_        long *recNum                    /* out */
                                long indices[]                  /* out */
            ATTR_               long *attrNum                   /* out */
            ENTRY_              long *entryNum                  /* out */
            rENTRY_             long *entryNum                  /* out */
            zENTRY_             long *entryNum                  /* out */

PUT_        CDF_ENCODING_       long encoding                   /* in  */
            CDF_MAJORITY_       long majority                   /* in  */
            CDF_FORMAT_         long format                     /* in  */
            rVAR_NAME_          char *varName                   /* in  */
            rVAR_DATASPEC_      long dataType                   /* in  */
                                long numElements                /* in  */
            rVAR_RECVARY_       long recVary                    /* in  */
            rVAR_DIMVARYS_      long dimVarys[]                 /* in  */
            rVAR_PADVALUE_      void *value                     /* in  */
            rVAR_INITIALRECS_   long NinitialRecs               /* in  */
            rVAR_EXTENDRECS_    long NextendRecs                /* in  */
            rVAR_DATA_          void *value                     /* in  */
            rVAR_HYPERDATA_     void *buffer                    /* in  */
            rVAR_SEQDATA_       void *value                     /* in  */
            zVAR_NAME_          char *varName                   /* in  */
            zVAR_DATASPEC_      long dataType                   /* in  */
                                long numElements                /* in  */
            zVAR_RECVARY_       long recVary                    /* in  */
            zVAR_DIMVARYS_      long dimVarys[]                 /* in  */
            zVAR_PADVALUE_      void *value                     /* in  */
            zVAR_INITIALRECS_   long NinitialRecs               /* in  */
            zVAR_EXTENDRECS_    long NextendRecs                /* in  */
            zVAR_DATA_          void *value                     /* in  */
            zVAR_HYPERDATA_     void *buffer                    /* in  */
            zVAR_SEQDATA_       void *value                     /* in  */
            ATTR_SCOPE_         long scope                      /* in  */
            ATTR_NAME_          char *attrName                  /* in  */
            ENTRY_DATA_         long dataType                   /* in  */
                                long numElements                /* in  */
                                void *value                     /* in  */
            rENTRY_DATA_        long dataType                   /* in  */
                                long numElements                /* in  */
                                void *value                     /* in  */
            zENTRY_DATA_        long dataType                   /* in  */
                                long numElements                /* in  */
                                void *value                     /* in  */
```

```
GET_        rVARs_NUMDIMS_      long *numDims                   /* out */
            rVARs_DIMSIZES_     long dimSizes[]                 /* out */
            zVAR_NUMDIMS_       long *numDims                   /* out */
            zVAR_DIMSIZES_      long dimSizes[]                 /* out */
            CDF_ENCODING_       long *encoding                  /* out */
            CDF_MAJORITY_       long *majority                  /* out */
            CDF_FORMAT_         long *format                    /* out */
            CDF_COPYRIGHT_      char *copyRight                 /* out */
            LIB_COPYRIGHT_      char *copyRight                 /* out */
            CDF_NUMrVARS_       long *numVars                   /* out */
            CDF_NUMzVARS_       long *numVars                   /* out */
            CDF_NUMATTRS_       long *numAttrs                  /* out */
            CDF_NUMgATTRS_      long *numAttrs                  /* out */
            CDF_NUMvATTRS_      long *numAttrs                  /* out */
            rVARs_MAXREC_       long *maxRec                    /* out */
            CDF_VERSION_        long *version                   /* out */
            CDF_RELEASE_        long *release                   /* out */
            CDF_INCREMENT_      long *increment                 /* out */
            LIB_VERSION_        long *version                   /* out */
            LIB_RELEASE_        long *release                   /* out */
            LIB_INCREMENT_      long *increment                 /* out */
            LIB_subINCREMENT_   char *subincrement              /* out */
            rVAR_NAME_          char *varName                   /* out */
            rVAR_DATATYPE_      long *dataType                  /* out */
            rVAR_NUMELEMS_      long *numElements               /* out */
            rVAR_RECVARY_       long *recVary                   /* out */
            rVAR_DIMVARYS_      long dimVarys[]                 /* out */
            rVAR_NUMBER_        char *varName                   /* in  */
                                long *varNum                    /* out */
            rVAR_PADVALUE_      void *value                     /* out */
            rVAR_EXTENDRECS_    long *NextendRecs               /* out */
            rVAR_MAXREC_        long *maxRec                    /* out */
            rVAR_DATA_          void *value                     /* out */
            rVAR_HYPERDATA_     void *buffer                    /* out */
            rVAR_SEQDATA_       void *value                     /* out */
            zVAR_NAME_          char *varName                   /* out */
            zVAR_DATATYPE_      long *dataType                  /* out */
            zVAR_NUMELEMS_      long *numElements               /* out */
            zVAR_RECVARY_       long *recVary                   /* out */
            zVAR_DIMVARYS_      long dimVarys[]                 /* out */
            zVAR_NUMBER_        char *varName                   /* in  */
                                long *varNum                    /* out */
            zVAR_PADVALUE_      void *value                     /* out */
            zVAR_EXTENDRECS_    long *NextendRecs               /* out */
            zVAR_MAXREC_        long *maxRec                    /* out */
            zVAR_DATA_          void *value                     /* out */
            zVAR_HYPERDATA_     void *buffer                    /* out */
            zVAR_SEQDATA_       void *value                     /* out */
            ATTR_SCOPE_         long *scope                     /* out */
            ATTR_NAME_          char *attrName                  /* out */
            ATTR_NUMBER_        char *attrName                  /* in  */
```

```
                    long *attrNum                   /* out */
ATTR_MAXgENTRY_     long *maxEntry                  /* out */
ATTR_NUMgENTRIES_   long *numEntries                /* out */
ATTR_MAXrENTRY_     long *maxEntry                  /* out */
ATTR_NUMrENTRIES_   long *numEntries                /* out */
ATTR_MAXzENTRY_     long *maxEntry                  /* out */
ATTR_NUMzENTRIES_   long *numEntries                /* out */
gENTRY_DATATYPE_    long *dataType                  /* out */
gENTRY_NUMELEMS_    long *numElements               /* out */
gENTRY_DATA_        void *value                     /* out */
zENTRY_DATATYPE_    long *dataType                  /* out */
zENTRY_NUMELEMS_    long *numElements               /* out */
zENTRY_DATA_        void *value                     /* out */
rENTRY_DATATYPE_    long *dataType                  /* out */
rENTRY_NUMELEMS_    long *numElements               /* out */
rENTRY_DATA_        void *value                     /* out */
STATUS_TEXT_        char *text                      /* out */
```

Appendix E

FORTRAN Programming Summary

E.1 Introduction

Below is a summary of the CDF library routines callable from FORTRAN applications. Each is written as a function or subroutine declaration.

E.2 Standard Interface

```
SUBROUTINE CDF_create (CDF_name, num_dims, dim_sizes, encoding, majority,
1                      id, status)
CHARACTER CDF_name                                                  ! in
INTEGER*4 num_dims                                                  ! in
INTEGER*4 dim_sizes(*)                                              ! in
INTEGER*4 encoding                                                  ! in
INTEGER*4 majority                                                  ! in
INTEGER*4 id                                                        ! out
INTEGER*4 status                                                    ! out

SUBROUTINE CDF_open (CDF_name, id, status)
CHARACTER CDF_name*(*)                                              ! in
INTEGER*4 id                                                        ! out
INTEGER*4 status                                                    ! out

SUBROUTINE CDF_doc (id, version, release, text, status)
INTEGER*4 id                                                        ! in
INTEGER*4 version                                                   ! out
INTEGER*4 release                                                   ! out
CHARACTER text*(CDF_DOCUMENT_LEN)                                   ! out
INTEGER*4 status                                                    ! out

SUBROUTINE CDF_inquire (id, num_dims, dim_sizes, encoding, majority,
```

```
1                       max_rec, num_vars, num_attrs, status)
INTEGER*4 id                                                          ! in
INTEGER*4 num_dims                                                    ! out
INTEGER*4 dim_sizes(CDF_MAX_DIMS)                                     ! out
INTEGER*4 encoding                                                    ! out
INTEGER*4 majority                                                    ! out
INTEGER*4 max_rec                                                     ! out
INTEGER*4 num_vars                                                    ! out
INTEGER*4 num_attrs                                                   ! out
INTEGER*4 status                                                      ! out

SUBROUTINE CDF_close (id, status)
INTEGER*4 id                                                          ! in
INTEGER*4 status                                                      ! out

SUBROUTINE CDF_delete (id, status)
INTEGER*4 id                                                          ! in
INTEGER*4 status                                                      ! out

SUBROUTINE CDF_error (status, text)
INTEGER*4 status                                                      ! in
CHARACTER text*(CDF_STATUSTEXT_LEN)                                   ! out

SUBROUTINE CDF_attr_create (id, attr_name, attr_scope, attr_num, status)
INTEGER*4 id                                                          ! in
CHARACTER attr_name*(*)                                               ! in
INTEGER*4 attr_scope                                                  ! in
INTEGER*4 attr_num                                                    ! out
INTEGER*4 status                                                      ! out

INTEGER*4 CDF_attr_num (id, attr_name)
INTEGER*4 id                                                          ! in
CHARACTER attr_name*(*)                                               ! in

SUBROUTINE CDF_attr_rename (id, attr_num, attr_name, status)
INTEGER*4 id                                                          ! in
INTEGER*4 attr_num                                                    ! in
CHARACTER attr_name*(*)                                               ! in
INTEGER*4 status                                                      ! out

SUBROUTINE CDF_attr_inquire (id, attr_num, attr_name, attr_scope, max_entry,
1                            status)
INTEGER*4 id                                                          ! in
INTEGER*4 attr_num                                                    ! in
CHARACTER attr_name*(CDF_ATTR_NAME_LEN)                               ! out
INTEGER*4 attr_scope                                                  ! out
INTEGER*4 max_entry                                                   ! out
INTEGER*4 status                                                      ! out

SUBROUTINE CDF_attr_entry_inquire (id, attr_num, entry_num, data_type,
1                                  num_elements, status)
```

```
INTEGER*4 id                                                    ! in
INTEGER*4 attr_num                                              ! in
INTEGER*4 entry_num                                             ! in
INTEGER*4 data_type                                             ! out
INTEGER*4 num_elements                                          ! out
INTEGER*4 status                                                ! out

SUBROUTINE CDF_attr_put (id, attr_num, entry_num, data_type, num_elements,
1                        value, status)
INTEGER*4 id                                                    ! in
INTEGER*4 attr_num                                              ! in
INTEGER*4 entry_num                                             ! in
INTEGER*4 data_type                                             ! in
INTEGER*4 num_elements                                          ! in
<type>    value                                                 ! in
INTEGER*4 status                                                ! out

SUBROUTINE CDF_attr_get (id, attr_num, entry_num, value, status)
INTEGER*4 id                                                    ! in
INTEGER*4 attr_num                                              ! in
INTEGER*4 entry_num                                             ! in
<type>    value                                                 ! out
INTEGER*4 status                                                ! out

SUBROUTINE CDF_attr_correct_scope (id, attr_num, attr_scope, status)
INTEGER*4 id                                                    ! in
INTEGER*4 attr_num                                              ! in
INTEGER*4 attr_scope                                            ! in
INTEGER*4 status                                                ! out

SUBROUTINE CDF_var_create (id, var_name, data_type, num_elements,
1                          rec_variance, dim_variances, var_num, status)
INTEGER*4 id                                                    ! in
CHARACTER var_name                                              ! in
INTEGER*4 data_type                                             ! in
INTEGER*4 num_elements                                          ! in
INTEGER*4 rec_variance                                          ! in
INTEGER*4 dim_variances(*)                                      ! in
INTEGER*4 var_num                                               ! out
INTEGER*4 status                                                ! out

INTEGER*4 FUNCTION CDF_var_num (id, var_name)
INTEGER*4 id                                                    ! in
CHARACTER var_name*(*)                                          ! in

SUBROUTINE CDF_var_rename (id, var_num, var_name, status)
INTEGER*4 id                                                    ! in
INTEGER*4 var_num                                               ! in
CHARACTER var_name*(*)                                          ! in
INTEGER*4 status                                                ! out
```

```
SUBROUTINE CDF_var_inquire (id, var_num, var_name, data_type, num_elements,
1                           rec_variance, dim_variances, status)
INTEGER*4 id                                                          ! in
INTEGER*4 var_num                                                     ! in
CHARACTER var_name*(CDF_VAR_NAME_LEN)                                 ! out
INTEGER*4 data_type                                                   ! out
INTEGER*4 num_elements                                                ! out
INTEGER*4 rec_variance                                                ! out
INTEGER*4 dim_variances(CDF_MAX_DIMS)                                 ! out
INTEGER*4 status                                                      ! out

SUBROUTINE CDF_var_put (id, var_num, rec_num, indices, value, status)
INTEGER*4 id                                                          ! in
INTEGER*4 var_num                                                     ! in
INTEGER*4 rec_num                                                     ! in
INTEGER*4 indices(*)                                                  ! in
<type>    value                                                       ! in
INTEGER*4 status                                                      ! out

SUBROUTINE CDF_var_get (id, var_num, rec_num, indices, value, status)
INTEGER*4 id                                                          ! in
INTEGER*4 var_num                                                     ! in
INTEGER*4 rec_num                                                     ! in
INTEGER*4 indices(*)                                                  ! in
<type>    value                                                       ! out
INTEGER*4 status                                                      ! out

SUBROUTINE CDF_var_hyper_put (id, var_num, rec_start, rec_count, rec_interval,
1                             indices, counts, intervals, buffer, status)
INTEGER*4 id                                                          ! in
INTEGER*4 var_num                                                     ! in
INTEGER*4 rec_start                                                   ! in
INTEGER*4 rec_count                                                   ! in
INTEGER*4 rec_interval                                                ! in
INTEGER*4 indices(*)                                                  ! in
INTEGER*4 counts(*)                                                   ! in
INTEGER*4 intervals(*)                                                ! in
<type>    buffer                                                      ! in
INTEGER*4 status                                                      ! out

SUBROUTINE CDF_var_hyper_get (id, var_num, rec_start, rec_count, rec_interval,
1                             indices, counts, intervals, buffer, status)
INTEGER*4 id                                                          ! in
INTEGER*4 var_num                                                     ! in
INTEGER*4 rec_start                                                   ! in
INTEGER*4 rec_count                                                   ! in
INTEGER*4 rec_interval                                                ! in
INTEGER*4 indices(*)                                                  ! in
INTEGER*4 counts(*)                                                   ! in
INTEGER*4 intervals(*)                                                ! in
<type>    buffer                                                      ! out
```

```
INTEGER*4 status                                                    ! out

SUBROUTINE CDF_var_close (id, var_num)
INTEGER*4 id                                                        ! in
INTEGER*4 var_num                                                   ! in
```

E.3 Internal Interface

```
INTEGER*4 FUNCTION CDF_lib (..., status)
INTEGER*4 status                                                    ! out

   CREATE_   CDF_                CHARACTER CDF_name*(*)             ! in
                                 INTEGER*4 num_dims                 ! in
                                 INTEGER*4 dim_sizes(*)             ! in
                                 INTEGER*4 id                       ! out
             rVAR_               CHARACTER var_name*(*)             ! in
                                 INTEGER*4 data_type                ! in
                                 INTEGER*4 num_elements             ! in
                                 INTEGER*4 rec_vary                 ! in
                                 INTEGER*4 dim_varys                ! in
                                 INTEGER*4 var_num                  ! out
             zVAR_               CHARACTER var_name*(*)             ! in
                                 INTEGER*4 data_type                ! in
                                 INTEGER*4 num_elements             ! in
                                 INTEGER*4 num_dims                 ! in
                                 INTEGER*4 dim_sizes(*)             ! in
                                 INTEGER*4 rec_vary                 ! in
                                 INTEGER*4 dim_varys                ! in
                                 INTEGER*4 var_num                  ! out
             ATTR_               CHARACTER attr_name*(*)            ! in
                                 INTEGER*4 scope                    ! in
                                 INTEGER*4 attr_num                 ! out

   OPEN_     CDF_                CHARACTER CDF_name*(*)             ! in
                                 INTEGER*4 id                       ! out

   DELETE_   CDF_

   CLOSE_    CDF_
             VAR_

   SELECT_   CDF_                INTEGER*4 id                       ! in
             CDF_STATUS_         INTEGER*4 status                   ! in
             rVAR_               INTEGER*4 var_num                  ! in
             rVAR_NAME_          CHARACTER var_name*(*)             ! in
             rVARs_RECNUMBER_    INTEGER*4 rec_num                  ! in
             rVARs_RECCOUNT_     INTEGER*4 rec_count                ! in
             rVARs_RECINTERVAL_  INTEGER*4 rec_interval             ! in
             rVARs_DIMINDICES_   INTEGER*4 indices(*)               ! in
```

```
            rVARs_RECCOUNTS_     INTEGER*4 counts(*)                  ! in
            rVARs_RECINTERVALS_  INTEGER*4 intervals(*)               ! in
            rVAR_SEQPOS_         INTEGER*4 rec_num                    ! in
                                 INTEGER*4 indices(*)                 ! in
            zVAR_                INTEGER*4 var_num                    ! in
            zVAR_NAME_           CHARACTER var_name*(*)               ! in
            zVAR_RECNUMBER_      INTEGER*4 rec_num                    ! in
            zVAR_RECCOUNT_       INTEGER*4 rec_count                  ! in
            zVAR_RECINTERVAL_    INTEGER*4 rec_interval               ! in
            zVAR_DIMINDICES_     INTEGER*4 indices(*)                 ! in
            zVAR_RECCOUNTS_      INTEGER*4 counts(*)                  ! in
            zVAR_RECINTERVALS_   INTEGER*4 intervals(*)               ! in
            zVAR_SEQPOS_         INTEGER*4 rec_num                    ! in
                                 INTEGER*4 indices(*)                 ! in
            ATTR_                INTEGER*4 attr_num                   ! in
            ATTR_NAME_           CHARACTER attr_name*(*)              ! in
            ENTRY_               INTEGER*4 entry_num                  ! in
            rENTRY_              INTEGER*4 entry_num                  ! in
            zENTRY_              INTEGER*4 entry_num                  ! in

CONFIRM_    CDF_NAME_            CHARACTER CDF_name*(*)               ! out
            rVAR_                INTEGER*4 var_num                    ! out
            rVARs_RECNUMBER_     INTEGER*4 rec_num                    ! out
            rVARs_RECCOUNT_      INTEGER*4 rec_count                  ! out
            rVARs_RECINTERVAL_   INTEGER*4 rec_interval               ! out
            rVARs_DIMINDICES_    INTEGER*4 indices(*)                 ! out
            rVARs_DIMCOUNTS_     INTEGER*4 counts(*)                  ! out
            rVARs_DIMINTERVALS_  INTEGER*4 intervals(*)               ! out
            rVAR_SEQPOS_         INTEGER*4 rec_num                    ! out
                                 INTEGER*4 indices(*)                 ! out
            zVAR_                INTEGER*4 var_num                    ! out
            zVAR_RECNUMBER_      INTEGER*4 rec_num                    ! out
            zVAR_RECCOUNT_       INTEGER*4 rec_count                  ! out
            zVAR_RECINTERVAL_    INTEGER*4 rec_interval               ! out
            zVAR_DIMINDICES_     INTEGER*4 indices(*)                 ! out
            zVAR_DIMCOUNTS_      INTEGER*4 counts(*)                  ! out
            zVAR_DIMINTERVALS_   INTEGER*4 intervals(*)               ! out
            zVAR_SEQPOS_         INTEGER*4 rec_num                    ! out
                                 INTEGER*4 indices(*)                 ! out
            ATTR_                INTEGER*4 attr_num                   ! out
            ENTRY_               INTEGER*4 entry_num                  ! out
            rENTRY_              INTEGER*4 entry_num                  ! out
            zENTRY_              INTEGER*4 entry_num                  ! out

PUT_        CDF_ENCODING_        INTEGER*4 encoding                   ! in
            CDF_MAJORITY_        INTEGER*4 majority                   ! in
            CDF_FORMAT_          INTEGER*4 format                     ! in
            rVAR_NAME_           CHARACTER var_name*(*)               ! in
            rVAR_DATASPEC_       INTEGER*4 data_type                  ! in
                                 INTEGER*4 num_elements               ! in
            rVAR_RECVARY_        INTEGER*4 rec_vary                   ! in
```

```
            rVAR_DIMVARYS_      INTEGER*4 dim_varys(*)              ! in
            rVAR_PADVALUE_      <type> value                        ! in
            rVAR_INITIALRECS_   INTEGER*4 n_initial_recs            ! in
            rVAR_EXTENDRECS_    INTEGER*4 _nextend_recs             ! in
            rVAR_DATA_          <type> value                        ! in
            rVAR_HYPERDATA_     <type> buffer                       ! in
            rVAR_SEQDATA_       <type> value                        ! in
            zVAR_NAME_          CHARACTER var_name*(*)              ! in
            zVAR_DATASPEC_      INTEGER*4 data_type                 ! in
                                INTEGER*4 num_elements              ! in
            zVAR_RECVARY_       INTEGER*4 rec_vary                  ! in
            zVAR_DIMVARYS_      INTEGER*4 dim_varys(*)              ! in
            zVAR_PADVALUE_      <type> value                        ! in
            zVAR_INITIALRECS_   INTEGER*4 n_initial_recs            ! in
            zVAR_EXTENDRECS_    INTEGER*4 _nextend_recs             ! in
            zVAR_DATA_          <type> value                        ! in
            zVAR_HYPERDATA_     <type> buffer                       ! in
            zVAR_SEQDATA_       <type> value                        ! in
            ATTR_SCOPE_         INTEGER*4 scope                     ! in
            ATTR_NAME_          CHARACTER attr_name*(*)             ! in
            ENTRY_DATA_         INTEGER*4 data_type                 ! in
                                INTEGER*4 num_elements              ! in
                                <type> value                        ! in
            rENTRY_DATA_        INTEGER*4 data_type                 ! in
                                INTEGER*4 num_elements              ! in
                                <type> value                        ! in
            zENTRY_DATA_        INTEGER*4 data_type                 ! in
                                INTEGER*4 num_elements              ! in
                                <type> value                        ! in

GET_        rVARs_NUMDIMS_      INTEGER*4 num_dims                  ! out
            rVARs_DIMSIZES_     INTEGER*4 dim_sizes(*)              ! out
            zVAR_NUMDIMS_       INTEGER*4 num_dims                  ! out
            zVAR_DIMSIZES_      INTEGER*4 dim_sizes(*)              ! out
            CDF_ENCODING_       INTEGER*4 encoding                  ! out
            CDF_MAJORITY_       INTEGER*4 majority                  ! out
            CDF_FORMAT_         INTEGER*4 format                    ! out
            CDF_COPYRIGHT_      CHARACTER copy_right*(*)            ! out
            LIB_COPYRIGHT_      CHARACTER copy_right*(*)            ! out
            CDF_NUMrVARS_       INTEGER*4 num_vars                  ! out
            CDF_NUMzVARS_       INTEGER*4 num_vars                  ! out
            CDF_NUMATTRS_       INTEGER*4 num_attrs                 ! out
            CDF_NUMgATTRS_      INTEGER*4 num_attrs                 ! out
            CDF_NUMvATTRS_      INTEGER*4 num_attrs                 ! out
            rVARs_MAXREC_       INTEGER*4 max_rec                   ! out
            CDF_VERSION_        INTEGER*4 version                   ! out
            CDF_RELEASE_        INTEGER*4 release                   ! out
            CDF_INCREMENT_      INTEGER*4 increment                 ! out
            LIB_VERSION_        INTEGER*4 version                   ! out
            LIB_RELEASE_        INTEGER*4 release                   ! out
            LIB_INCREMENT_      INTEGER*4 increment                 ! out
```

```
LIB_subINCREMENT_   CHARACTER subincrement*1                ! out
rVAR_NAME_          CHARACTER var_name*(*)                  ! out
rVAR_DATATYPE_      INTEGER*4 data_type                     ! out
rVAR_NUMELEMS_      INTEGER*4 num_elements                  ! out
rVAR_RECVARY_       INTEGER*4 rec_vary                      ! out
rVAR_DIMVARYS_      INTEGER*4 dim_varys(*)                  ! out
rVAR_NUMBER_        CHARACTER var_name*(*)                  ! in
                    INTEGER*4 var_num                       ! out
rVAR_PADVALUE_      <type> value                            ! out
rVAR_EXTENDRECS_    INTEGER*4 n_extend_recs                 ! out
rVAR_MAXREC_        INTEGER*4 max_rec                       ! out
rVAR_DATA_          <type> value                            ! out
rVAR_HYPERDATA_     <type> buffer                           ! out
rVAR_SEQDATA_       <type> value                            ! out
zVAR_NAME_          CHARACTER var_name*(*)                  ! out
zVAR_DATATYPE_      INTEGER*4 data_type                     ! out
zVAR_NUMELEMS_      INTEGER*4 num_elements                  ! out
zVAR_RECVARY_       INTEGER*4 rec_vary                      ! out
zVAR_DIMVARYS_      INTEGER*4 dim_varys(*)                  ! out
zVAR_NUMBER_        CHARACTER var_name*(*)                  ! in
                    INTEGER*4 var_num                       ! out
zVAR_PADVALUE_      <type> value                            ! out
zVAR_EXTENDRECS_    INTEGER*4 n_extend_recs                 ! out
zVAR_MAXREC_        INTEGER*4 max_rec                       ! out
zVAR_DATA_          <type> value                            ! out
zVAR_HYPERDATA_     <type> buffer                           ! out
zVAR_SEQDATA_       <type> value                            ! out
ATTR_SCOPE_         INTEGER*4 scope                         ! out
ATTR_NAME_          CHARACTER attr_name*(*)                 ! out
ATTR_NUMBER_        CHARACTER attr_name*(*)                 ! in
                    INTEGER*4 attr_num                      ! out
ATTR_MAXgENTRY_     INTEGER*4 max_entry                     ! out
ATTR_NUMgENTRIES_   INTEGER*4 num_entries                   ! out
ATTR_MAXrENTRY_     INTEGER*4 max_entry                     ! out
ATTR_NUMrENTRIES_   INTEGER*4 num_entries                   ! out
ATTR_MAXzENTRY_     INTEGER*4 max_entry                     ! out
ATTR_NUMzENTRIES_   INTEGER*4 num_entries                   ! out
gENTRY_DATATYPE_    INTEGER*4 data_type                     ! out
gENTRY_NUMELEMS_    INTEGER*4 num_elements                  ! out
gENTRY_DATA_        <type> value                            ! out
rENTRY_DATATYPE_    INTEGER*4 data_type                     ! out
rENTRY_NUMELEMS_    INTEGER*4 num_elements                  ! out
rENTRY_DATA_        <type> value                            ! out
zENTRY_DATATYPE_    INTEGER*4 data_type                     ! out
zENTRY_NUMELEMS_    INTEGER*4 num_elements                  ! out
zENTRY_DATA_        <type> value                            ! out
STATUS_TEXT_        CHARACTER text*(*)                      ! out
```

Appendix F

Glossary

dimensionality	The number of dimensions and the dimension sizes for the rVariables or a zVariable.
gAttribute	A global scoped attribute.
gEntry	An entry for a gAttribute.
NRV variable	Non-record variant variable. A variable whose values do not change from record to record (a record variance of FALSE [`NOVARY`]).
rEntry	An entry for a vAttribute corresponding to an rVariable.
rVariable	"R" variable. All rVariables have the same dimensionality.
vAttribute	A variable scoped attribute.
zEntry	An entry for a vAttribute corresponding to a zVariable.
zVariable	"Z" variable. zVariables can have dimensionalities which are different than that of the rVariables (and each other).

Appendix G

CDF Release Notes

G.1 Introduction

This chapter should provide you with all of the information you need to know to begin using CDF V2.3 assuming that you are already familar with CDF V2.2. For additional information contact CDF User Support.

G.2 Supported Systems

CDF V2.3 is currently supported on the following machines/operating systems.

1. VAX (VMS)
2. Sun (SunOS)
3. DECstation (ULTRIX)
4. Silicon Graphics Iris & Power Series (IRIX)
5. IBM RS/6000 series (AIX)
6. HP 9000 series (HP-UX)
7. IBM PC (MS-DOS)
8. NeXT (Mach)

Additional ports are planned for the future. They include: Cray (UNICOS) and Macintosh (MacOS ?).

G.3 Changes for CDF V2.3

Several changes have been made to the CDF distribution for V2.3. CDF V2.3 differs from CDF V2.2 in the following ways.

1. The distribution has been ported to the NeXT computer (Mach operating system) with the exception of the `CDFbrowse`, `CDFlist`, and `CDFwalk` toolkit programs. Those programs use features of System V Curses not supported on the NeXT. A way around this problem is being sought and will hopefully be included in a future release.

2. Support for zVariables has been added to the distribution. The variables supported in previous release of CDF are now referred to as rVariables. zVariables differ from rVariables in that their dimensionality (number of dimensions and dimension sizes) can be different from that of the rVariables (and the other zVariables). zVariables are only accessable via the Internal Interface (the Standard Interface deals exclusively with rVariables).

3. An interface allowing access to the CDF library at the IDL command line has been added to the distribution. This interface consists of a set of IDL functions which call the CDF library. The IDL functions mirror the Standard Interface functions as called from a program written in C. Appendix B describes how to use this interface.

4. A shareable version of the CDF library is now available on VMS systems and certain flavors of UNIX (currently Sun/SunOS, HP9000/HP-UX, and IBM RS6000/AIX).

5. A CDF with any encoding can now be read on any computer (eg. a VAX encoded CDF can be read on an SGi Iris). In addition, Version 1 CDFs are now readable on any computer.

Index

— A —

access, 4, 33
  variable data, 8
  version1, 33
  version2, 33
attributes, 13, 48
  entries, 49
    data type, 49
    elements, 49
    numbering, 49
  global scope, 13, 260
  limits, 93
  metadata, 13
  naming, 48
  nssdc standard, 50
  numbering, 48
  scopes, 48, 93, 176
  variable scope, 13, 262

— C —

CDF Data Set
  attribute
    correcting scope, 111, 196
    creating, 102, 186
    inquiring about, 105, 189
    inquiring about an entry, 106, 191
    number, 103, 187
    reading, 109, 194
    renaming, 104, 189
    writing, 108, 193
  CDFlib, 124, 211, 292, 301
  closing, 99, 136, 183, 223
  confirming, 140, 227
  creating, 92, 94, 133, 175, 178, 219
  deleting, 100, 136, 184, 223
  identifier, 90
  inquiring about, 98, 182
  inquiring about status codes, 101
  opening, 96, 136, 179, 222
  reading, 150, 237
  selecting, 136, 223
  status codes
    error, 284
    informational, 283
    inquiring explanation, 185
    interpreting, 164, 253
    returned, 131, 218
    success, 283
    warning, 284
  variable
    creating, 113, 198
    inquiring about, 117, 202
    number, 114, 200
    reading, 119, 123, 205, 209
    renaming, 116, 201
    writing, 118, 121, 203, 207
  variances, 92, 176
  writing, 143, 230
CDF organization, 3
CDF record, 4
CDF Toolkit, 13
cdf.h, 85, 87, 90, 164, 167
CDFattrCorrectScope, 111
CDFattrCreate, 102
CDFattrEntryInquire, 106
CDFattrGet, 109
CDFattrInquire, 104
CDFattrNum, 103
CDFattrPut, 107
CDFattrRename, 104
CDFbrowse, 13, 76
CDFcompare, 14, 55
CDFconvert, 14, 57
CDFcreate, 94
CDFdelete, 100
CDFdir, 14, 54
CDFdoc, 96
CDFerror, 101
cdfinc, 169, 174
CDFinquire, 14, 66, 98
CDFlist, 14, 78

CDFopen, 95
CDFscope, 14, 60
CDFskeleton, 14, 24, 73, 257
CDFstats, 14, 62
CDFvarCreate, 112
CDFvarGet, 119
CDFvarHyperGet, 123
CDFvarHyperPut, 121
CDFvarInquire, 116
CDFvarNum, 114
CDFvarPut, 118
CDFvarRename, 115
CDFwalk, 14, 81
CDF_attr_correct_scope, 196
CDF_attr_create, 186
CDF_attr_entry_inquire, 191
CDF_attr_get, 194
CDF_attr_inquire, 189
CDF_attr_num, 187
CDF_attr_put, 193
CDF_attr_rename, 188
CDF_close, 183
CDF_create, 177
CDF_delete, 184
CDF_doc, 180
CDF_error, 185
CDF_inquire, 182
CDF_open, 179
CDF_var_create, 197
CDF_var_get, 205
CDF_var_hyper_get, 208
CDF_var_hyper_put, 206
CDF_var_inquire, 202
CDF_var_num, 199
CDF_var_put, 203
CDF_var_rename, 201
compiling, 85, 169
Creating a CDF, 16, 24, 125, 211, 257

— D —

datatypes, 35, 90, 174
    character, 36
    epoch, 36
    equivalent, 37
    floating, 36
    integer, 36

— E —

encoding, 5, 34, 91, 175
    host, 35, 175
    network, 35, 175
    performance, 35
Epoch, 36, 51, 165, 255
examples
    attribute entries, 13, 163, 252
    attribute renaming, 162, 251
    CDFlib, 125, 160, 249
    creating a CDF, 16, 24, 125, 211
        skeletontable, 24
    hyper access, 45
    hyper read, 161, 250
    majority, 43
    organizing data, 9
    pad value, 160, 249
    sequential access, 46, 162, 252
    skeletontable, 269
        global scope attribute definitions, 261
        variable definition, 265, 268
    variable creation, 160, 249
    variable records, 39

— F —

features of a CDF, 4
formats, 4, 33
    multi, 4, 33, 90, 174
    performance, 34
    single, 4, 34, 90, 174
functions
    attribute, 15, 31
    custom, 166
    epoch, 165, 255
    general, 15, 31
    obsolete interface, 32
    variable, 15, 31

— I —

include files, 85, 87, 169
interface, 31
    internal, 31, 124, 210, 292, 301
    obsolete, 31
    standard, 31, 94, 177, 289, 297

— L —

library, 3
limits, 37
    attribute, 93, 177
    names, 93, 177
    operational, 93, 177
linking, 87, 171

— M —

macros, 166
majority, 92, 175
monoton, 62

— N —

NSSDC standard, 36, 50
    epoch, 51

— P —

pad values, 47, 145, 147, 232, 234

— S —

SkeletonTable, 14, 24, 71, 257
    attributes
        global scope, 260
        variable scope, 262
    variables, 263, 266
Standard Interface, 31, 94, 177
status codes
    error, 284
    informational, 283
    inquiring explanation, 101
    interpreting, 164, 253
    returned, 131
    success, 283
    warning, 284

— V —

variable record, 4
variables, 38
    access, 14
    arrays, 4
        dimensions, 4
    data-specification, 41
        elements, 42
        type, 42
    dimension-variance, 11, 42, 92, 176
    dimensionality, 38
    hyper-access, 44
    majority, 43
    naming, 38
    non-record-variant (NRV), 11, 42, 263, 266
    nssdc standard, 51
    numbering, 38
    organizing, 8
    pad-values, 41, 47
    record-variance, 11, 24, 42, 92, 176
    records, 39
        extend, 41
        initial, 41
    sequential-access, 46

— X —

XDR, 7, 35, 91, 175