CDF
User's Guide

Version 2.6, October 1, 1998

National Space Science Data Center

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

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

Contents

List of Figures

List of Tables

Preface

About This Document

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

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

What is CDF?

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

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

How is CDF Used?

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

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

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

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

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

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

The programming layer (CDF library) provides the essential framework on which graphical and data analysis packages can be created. The CDF library allows developers of CDF-based systems to easily create applications that permit users to slice data across multidimensional subspaces, access entire structures of data, perform subsampling of data, and access one data element independently regardless of its relationship to any other data element. CDF data sets are portable across any platform supported by CDF. These currently consist of VAX (OpenVMS and POSIX shell), Sun (SunOS & SOLARIS), DECstation (ULTRIX), DEC Alpha (OSF/1 & OpenVMS), Silicon Graphics Iris and Power Series (IRIX), IBM RS6000 series (AIX), HP 9000 series (HP-UX), NeXT (Mach), PC (DOS, Windows 3.x, Windows NT/95, Linux, & QNX), and Macintosh (MacOS).

How is CDF Useful to Me?

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

Chapter 1

Primer

1.1 Introduction

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

1.2 Why Use CDF?

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

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

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

1.3 Conceptual Organization

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

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

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

1.4 Features of the CDF Library

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

1.4.1 File Format Options

The CDF library gives the user the option to choose from one of two file formats in which to store the data and metadata. The first option is the traditional CDF multi-file format, which is also the default. This file format is illustrated in Figure 1.4 (assuming a CDF containing four variables). The `example.cdf` file contains all of the control information and metadata for the CDF. In addition to the `.cdf` file,[4] a file exists for each variable in the CDF and contains only the data associated with that variable. This is illustrated by the files `example.v0` through `example.v3`. The second option is the single-file format, 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

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

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

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

[4] This file is referred to as the dotCDF file.

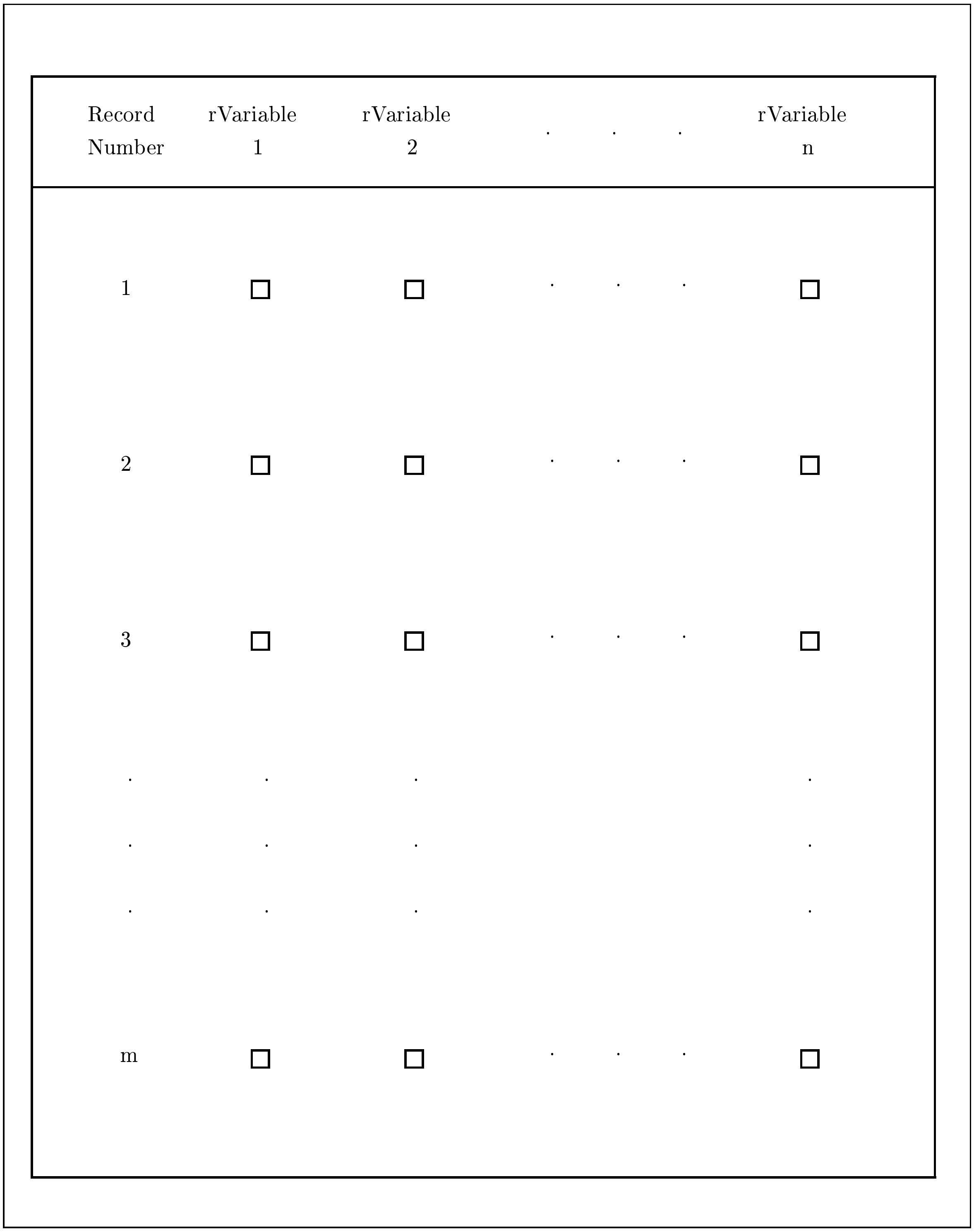

Figure 1.1: Conceptual View of a CDF, 0-Dimensional rVariables

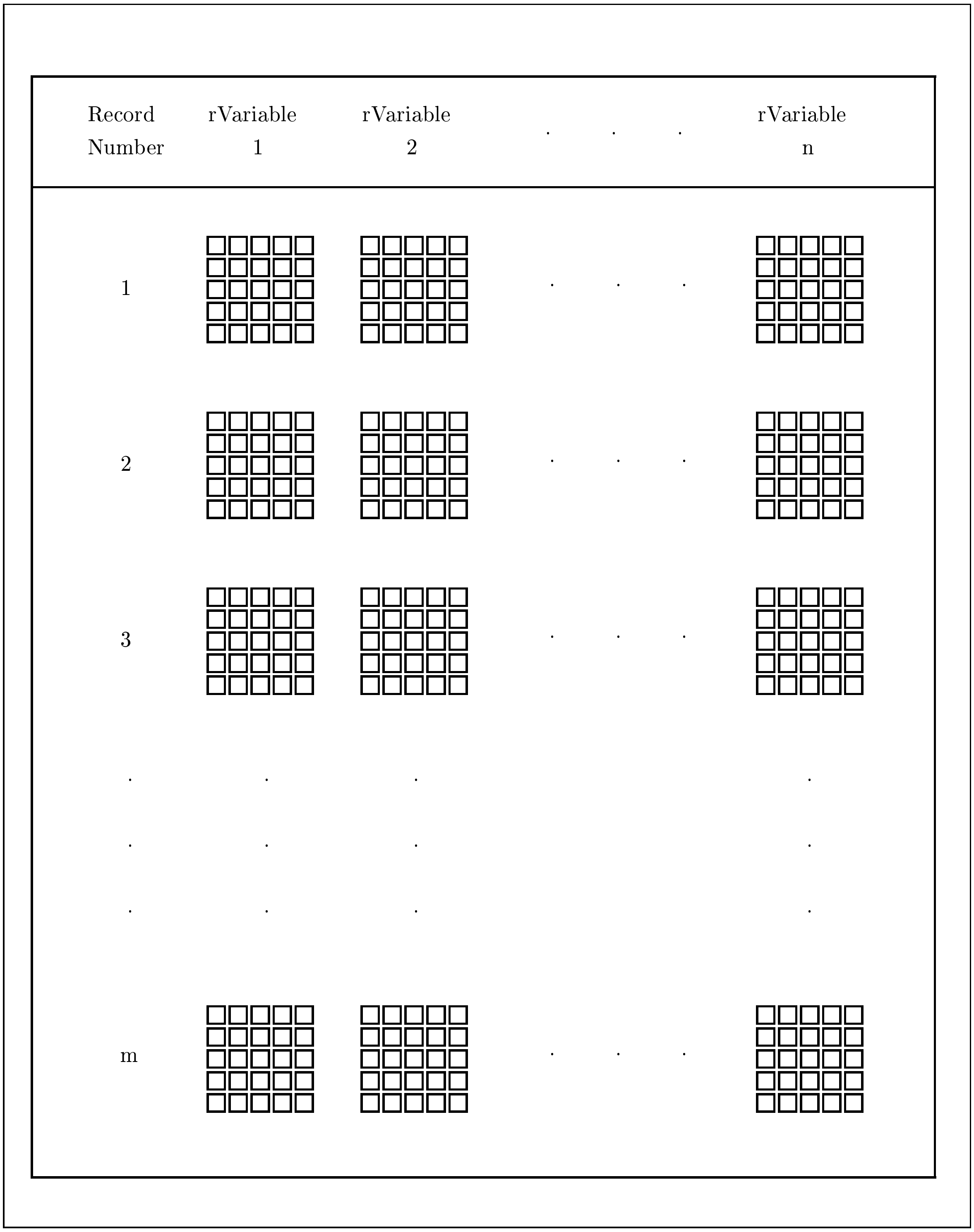

Figure 1.2: Conceptual View of a CDF, 2-Dimensional rVariables

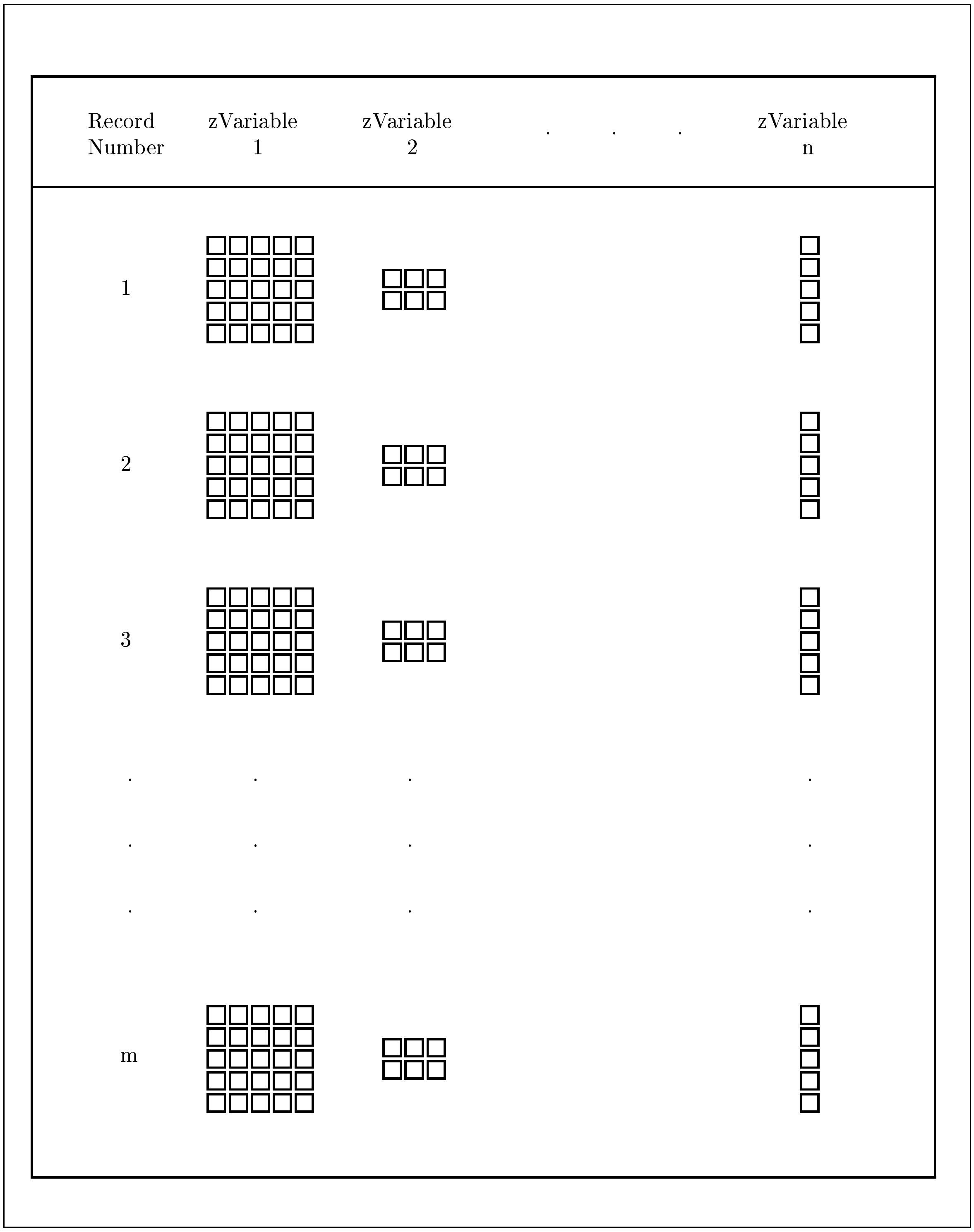

Figure 1.3: Conceptual View of a CDF, zVariables

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.

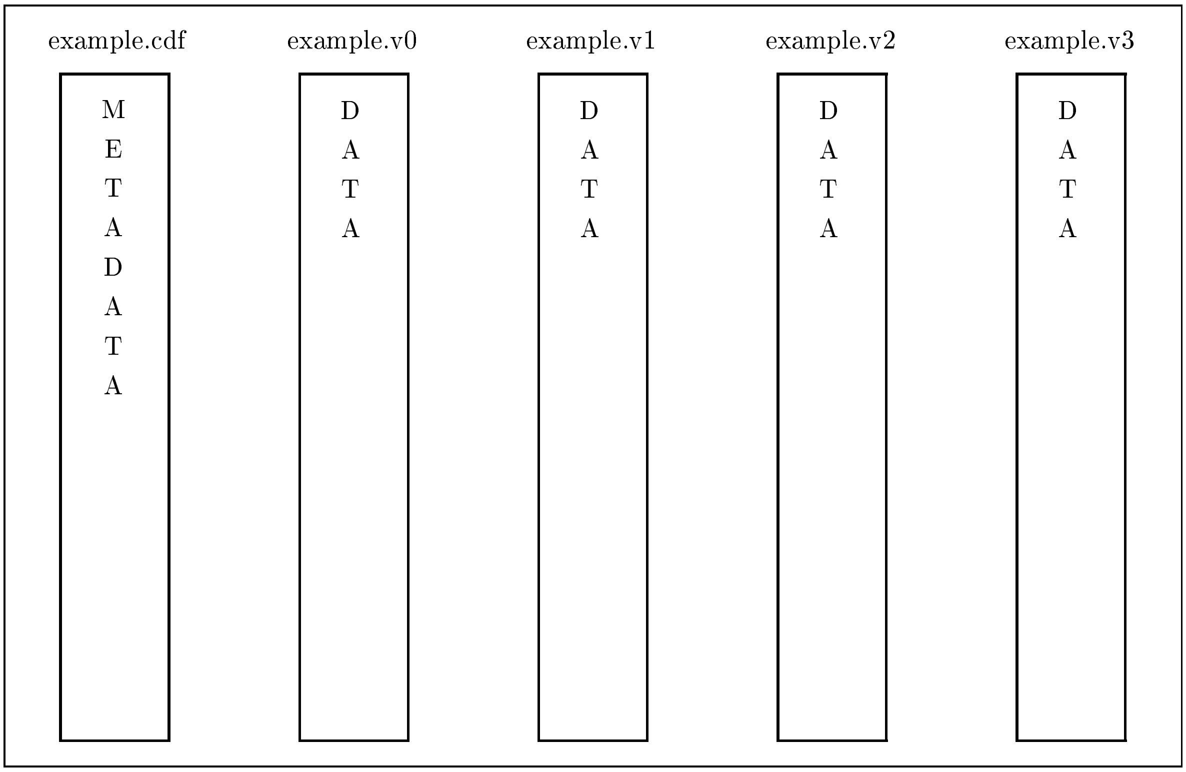

Figure 1.4: Multi-File Format

1.4.2 Data Encoding Options

When creating a CDF, a user has the option of using any of the supported encodings: VAX, Sun, SGi Personal Iris and Power Series, DECstation, DEC Alpha/OSF1, DEC Alpha/OpenVMS (`D_FLOAT`, `G_FLOAT` or `IEEE_FLOAT` double-precision floating-point), IBM RS6000 series, HP 9000 series, NeXT, PC, Macintosh, or network (XDR - eXternal Data Representation). The created CDF may then be copied to any of the supported computers and read by the CDF library. When a value is read from the CDF, the CDF library may be requested to decode the value into the encoding of the computer being used or any of the other encodings (which may be desirable for various reasons). A CDF with any of the supported encodings may be read from and written to on any supported computer.

1.4.3 Compression

A compression may be specified for a single-file CDF and the CDF library can be instructed to compress a CDF as it is written to disk. This compression occurs transparently to the user. When a compressed CDF is opened, it is automatically decompressed by the CDF library. An application does not have to even know

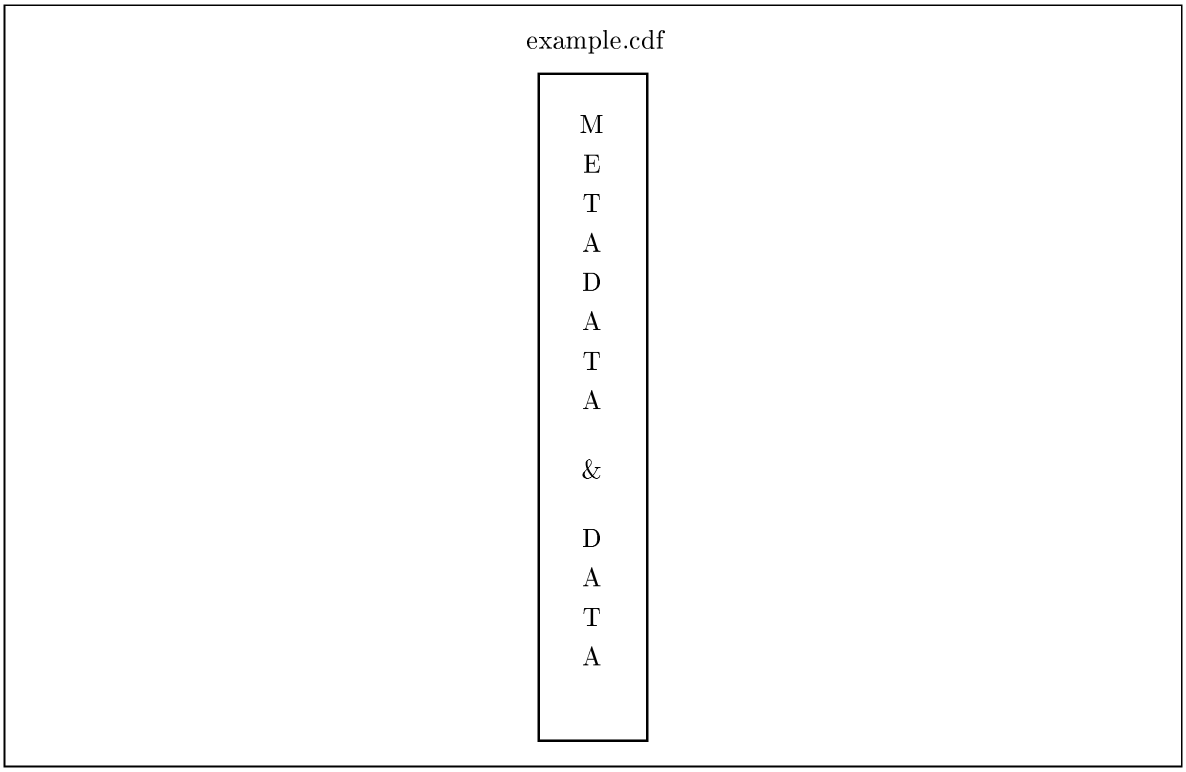

Figure 1.5: Single-File Format

that a CDF is compressed. Any type of access is allowed on a compressed CDF. When a compressed CDF is closed by an application, it is automatically recompressed as it is written back to disk.

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

Several different compression algorithms are supported by the CDF library. When compression is specified for a CDF or one of its variables, the compression algorithm to be used must be selected. There will be trade-offs between the different compression algorithms regarding execution performance and disk space savings. The nature of the data in a CDF (or variable) will affect the selection of the best compression algorithm to be used.

1.4.4 Sparseness

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

1.4.5 Variable Data Access Options

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

1.5 Organizing Your Data in a CDF

1.5.1 Variables

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

rVariables[5]

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

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

[5]Although rVariables are described here first, the trend amoung CDF users is toward CDFs containing only zVariables. zVariables are described in the next section.

Record Number	Time	Longitude	Latitude	Temperature
		rVariables		
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)

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

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

values contained in each of those rVariables. Additional dependent rVariables would be stored in the same way as `Temperature`.

Although conceptually there is a 2-dimensional array structure for each rVariable in each record of the CDF, this would not be an efficient way to store the data. For instance, the time for each record need only be stored once as opposed to being stored four times as shown in each 2-dimensional array (Table 1.2). This problem is circumvented by specifying "variances." For each rVariable there are variances associated with the array dimensions as well as the records. "Record variance" indicates whether or not an rVariable has unique values from record to record in the CDF. `Time` changes for each record so the record variance for `Time` is `[TRUE]`. One could also say that `Time` is record-variant. `Latitude` and `Longitude` repeat their values from record to record so the record variance for each is `[false]`. `Latitude` and `Longitude` are non-record-variant (NRV). The `Temperature` values change from record to record so they are record-variant. The record variances for this example are shown in Table 1.3.

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

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

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

When the record and dimension variances have been defined correctly, the amount of physical storage needed for the CDF is drastically reduced. In the above example, 2-dimensional arrays are not physically stored for each rVariable in a CDF record. Instead, the physical storage for each rVariable consists of just one value for `Time` in each CDF record, a single 1-dimensional array of values for the `Longitude` and `Latitude` rVariables (in only the first CDF record), and a full 2-dimensional array of values for `Temperature` in each CDF record. The actual physical storage (physical view) is shown in Table 1.4. The conceptual view of the CDF, however, is still that of one 2-dimensional array per rVariable in each CDF record as shown in Table 1.2 (the physically stored values are shown in boldface type).

zVariables

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

Consider a data set that consists of some number of images, each containing 1024 by 1024 pixels. The data set also contains a palette that is used to map pixel values to the actual color/shade to be displayed. Palettes are also referred to as lookup tables or color lookup tables. For this example 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.

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

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

Attempting to store the images and the palette using only rVariables would result in one of two undesirable situations. If the CDF's rVariables had a dimensionality of `2:[1024,1024]`[6] (to store the images), the palette would have to be stored in a 1024 by 1024 array that does not make sense logically and would waste disk space regardless of how the dimension variances are set. If the CDF's rVariables had a dimensionality of `3:[1024,1024,256]`, the images could be stored in an rVariable having dimension variances `T/TTF`[7] and the palette could be stored in an rVariable having dimension variances `F/FFT`. This would not waste any disk space but is not the intuitive way to store the data — nothing in the data set is 3-dimensional.

Using zVariables to store the images and palette would solve both problems. The images would be stored in a zVariable with dimensionality `2:[1024,1024]` (and variances of `T/TT`) and the palette would be stored in a zVariable with a dimensionality of `1:[256]` (and variances of `F/T`). This would waste no disk space and logically makes sense.

The use of zVariables is recommended because of this added flexibility. Note that zVariables can always be used instead of rVariables. In the rVariable example where temperature values were being stored, zVariables could also have been used. Each zVariable would have the same dimensionality and their dimension variances would be used in the same way as they were used for the rVariables.

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

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

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

with dimensionalities of `1:[2]`, `1:[3]`, `1:[5]`, `1:[7]`, and `1:[25]` would be straight forward and efficient.

1.6 Attributes

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

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

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

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

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

Table 1.5: vAttribute rEntries for the `Temperature` rVariable

1.7 CDF Toolkit

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

The available toolkit programs are as follows:

CDFedit[8]	Allows the display, creation, and modification of attribute and variable data in a CDF.
CDFexport[9]	Allows the contents of a CDF to be exported to the terminal screen, a text file, or another CDF. The CDF may be filtered in order to export a subset of its contents.
CDFconvert	Allows the format, encoding, majority, compression, and sparseness of a CDF to be changed. In all cases a new CDF is created. The original CDF is not modified.
SkeletonCDF[10]	Reads a specially formatted text file (called a skeleton table) and creates a skeleton CDF. A skeleton CDF is complete except for record-variant data.
SkeletonTable	Reads a CDF and produces a specially formatted text file called a skeleton table. The skeleton table may be modified and then input to SkeletonCDF to create a skeleton CDF.
CDFinquire	Displays the version of your CDF distribution, many of the configurable parameters, and the default CDF toolkit qualifiers.
CDFstats	Produces a report containing various statistics about the variables in a CDF.
CDFcompare	Reports the differences between two CDFs.
CDFdir	Produces a directory listing of a CDF's files. For a multi-file CDF the variable files are listed in ascending numerical order.

1.8 Library Interface Routines

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

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

[9] CDFexport has replaced CDFlist and CDFwalk.

[10] SkeletonCDF was previously named CDFskeleton.

1.8.1 Standard Interface

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

The general CDF functions are as follows:

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

The rVariable functions are as follows:

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

The attribute functions are as follows:

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

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

1.8.2 Internal Interface

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

1.9 Examples

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

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

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

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

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

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

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

#include "cdf.h"

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

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

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

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

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

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

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

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

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

  long Time;            /* 'Time' rVariable value. */
  float Lat;            /* 'Latitude' rVariable value. */
```

```
float Lon;          /* 'Longitude' rVariable value. */
float Tmp;          /* 'Temperature' rVariable value. */

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

status = CDFattrPut (id, CDFattrNum(id,"VALIDMIN"),
                     CDFvarNum(id,"Latitude"), CDF_REAL4, 1L, &LatValidMin);
```

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

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

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

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

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

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

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

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

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

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

         if (recNum == 0 && indices[0] == 0) {
```

```
            status = CDFvarPut (id, LatVarNum, recNum, indices, &Lat);
            if (status != CDF_OK) StatusHandler (status);
          }

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

  fclose (fp);

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

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

  return;
}
```

1.9.2 Creating a CDF, an Easier Way

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

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

#header

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

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


#GLOBALattributes

! Attribute         Entry       Data
```