CDF
Internal Format Description

Version 2.6, December 18, 1997

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

This document will present the physical file layout used by the Common Data Format (CDF) for CDF Version 2.6. No attempt will be made to teach the concepts of CDF. For that please refer to the CDF User's Guide, CDF C Reference Manual, and CDF Fortran Reference Manual. This document will assume that you are familiar with rVariables, zVariables, attributes, gEntries, rEntries, zEntries, and all of the other CDF concepts. Using the contents of this document you should be able to rewrite the CDF library in your spare time.

Chapter 1

Introduction

A CDF may have one of two formats: single-file or multi-file. A single-file CDF contains everything in one file having an extension of `.cdf`. A multi-file CDF stores everything except variable values in one file (with an extension of `.cdf`). The variable values are stored in separate files — one per variable. Variable files are described in Chapter 3. The `.cdf` file of a CDF will be referred to as the dotCDF file throughout this document.

The dotCDF file of a CDF contains magic numbers and numerous internal records used to organize information about the contents of the CDF (for both single-file and multi-file CDFs). Chapter 2 describes the magic numbers and the various internal records. The data encodings used by CDF are described in Chapter 5.

The file attributes of a dotCDF or variable file are not an issue on UNIX-based systems, the PC, or the Macintosh[1] because all files on those platforms are simply treated as a sequence of bytes. On OpenVMS-based systems, however, file attributes are very much an issue. The file attributes of a dotCDF or variable file created by the CDF library on an OpenVMS-based system are as follows:

File organization:	Sequential
Record format:	Fixed length 512 byte records
Record attributes:	None
RMS attributes:	None

These are also the file attributes for a file which has been FTPed to an OpenVMS-based system in binary mode. With these file attributes the CDF library is able to read the file as if it simply consisted of a sequence of bytes. Transferring a CDF file to an OpenVMS-based systems as a text file will result in a different set of file attributes as well as the insertion of additional bytes into the file (because the file system thinks there are suppose to be lines of text). CDF files transferred in this way will not be readable by the CDF library.

CDFs created while running the POSIX Shell on a DEC Alpha (running OpenVMS), however, will have a different set of file attributes when the POSIX Shell is not being used. These file attributes are:

File organization:	Sequential
Record format:	Stream_LF, maximum 32767 bytes
Record attributes:	Carriage return carriage control
RMS attributes:	None

[1]On a Macintosh only the data fork of a file is used in a dotCDF or variable file.

A CDF file with these attributes appears to be readable by the CDF library on current versions of OpenVMS for a DEC Alpha. Some older version of OpenVMS apparently treat these file attributes differently and may cause a problem for the CDF library.

Chapter 2

dotCDF File

This chapter will describe the contents of the dotCDF file. The dotCDF file contains a magic number and two or more internal records (IRs) that are used to organize the contents of a CDF. Different types of internal records are used to store information about various aspects and/or objects in the CDF. Each internal record contains two or more fields. The first field (at internal record offset[1] `0x0`), referred to as the `RecordSize` field, is a 4-byte unsigned integer containing the size of the internal record in bytes. The second field (at internal record offset `0x4`), referred to as the `RecordType` field, is a 4-byte signed integer containing the type of internal record. Fields from the third through the last depend on the type of internal record. Each field is stored contiguously, however, and some fields may not be present in a particular instance of a type of internal record. Note that internal record fields are also referred to as “internal values.”

Table 2.1 lists the types of internal records, the associated `RecordType` values, and brief descriptions. Detailed descriptions are found in the corresponding sections.

All dotCDF files contain a CDF Descriptor Record (CDR) and a Global Descriptor Record (GDR). Other internal records will be present depending on the contents of the CDF. The CDR is always at file offset[2] `0x00000008` which immediately follows the magic number(s) described in Section 2.1. The file offset of the GDR is stored in the CDR.

The only internal record at a fixed location in the dotCDF file is the CDR. All other internal records (including the GDR) may be present in any order (which generally depends on the order in which the contents of the CDF were created by an application). File offsets are used to “point” to other internal records. Linked lists of internal records are implemented by storing the file offset of the first internal record on the linked list, having that internal record store the file offset of the next internal record on the linked list, and so on. Figure 2.1 shows a possible arrangement of internal records in a ”uncompressed” dotCDF file. Note that the GDR “points” to the first zVDR that in turn “points” to the next zVDR. File offsets as described in the sections to follow are used to implement this linked list. Keep in mind that this is only an example of how a dotCDF file might be arranged. The internal records shown could be ordered in a number of different ways depending on how the CDF was written by the application. Figure 2.2 shows a possible arrangement of internal records in a dotCDF file which has a variable compressed. Figure 2.3 shows the file arrangement of internal records in a fully compressed dotCDF file.

[1] The offset in (hexadecimal) bytes from the beginning of the internal record.
[2] The offset in (hexadecimal) bytes from the beginning of the file.

Type of Internal Record	`RecordType` Field Internal Value	Purpose/Contents
CDR	1	CDF Descriptor Record. General information about the CDF. See Section 2.2.
GDR	2	Global Descriptor Record. Additional general information about the CDF. See Section 2.3.
rVDR	3	rVariable Descriptor Record. Information about an rVariable. See Section 2.6.
ADR	4	Attribute Descriptor Record. Information about an attribute. See Section 2.4.
AgrEDR	5	Attribute g/rEntry Descriptor Record. Information about a gEntry or rEntry of an attribute. See Section 2.5.
VXR	6	Variable Index Record. Indexing information for a variable. See Section 2.7.
VVR	7	Variable Values Record. One or more variable records. See Section 2.8.
zVDR	8	zVariable Descriptor Record. Information about a zVariable. See Section 2.6.
AzEDR	9	Attribute zEntry Descriptor Record. Information about a zEntry of an attribute. See Section 2.5.
CCR	10	Compressed CDF Record. Information about a compressed CDF/variable. See Section 2.9.
CPR	11	Compression Parameters Record. Information about the compression used for a CDF/variable. See Section 2.10.
SPR	12	Sparseness Parameters Record. Information about the specified sparseness array. See Section 2.11.
CVVR	13	Compressed Variable Values Record. Information for the compressed CDF/variable. See Section 2.12.
UIR	-1	Unused Internal Record. An internal record not currently being used. See Section 2.13.

Table 2.1: Internal Records

2.1 Magic Numbers

CDF Version 2.6 uses two magic numbers.[3] The first one is `0xCDF26002`[4] at file offset `0x00000000` stored as a 4-byte, unsigned integer with big-endian byte ordering. It is followed by the second one, another 4-byte

[3]They don't seem like magic to me but looking at these values is how you would determine the identity of a file.

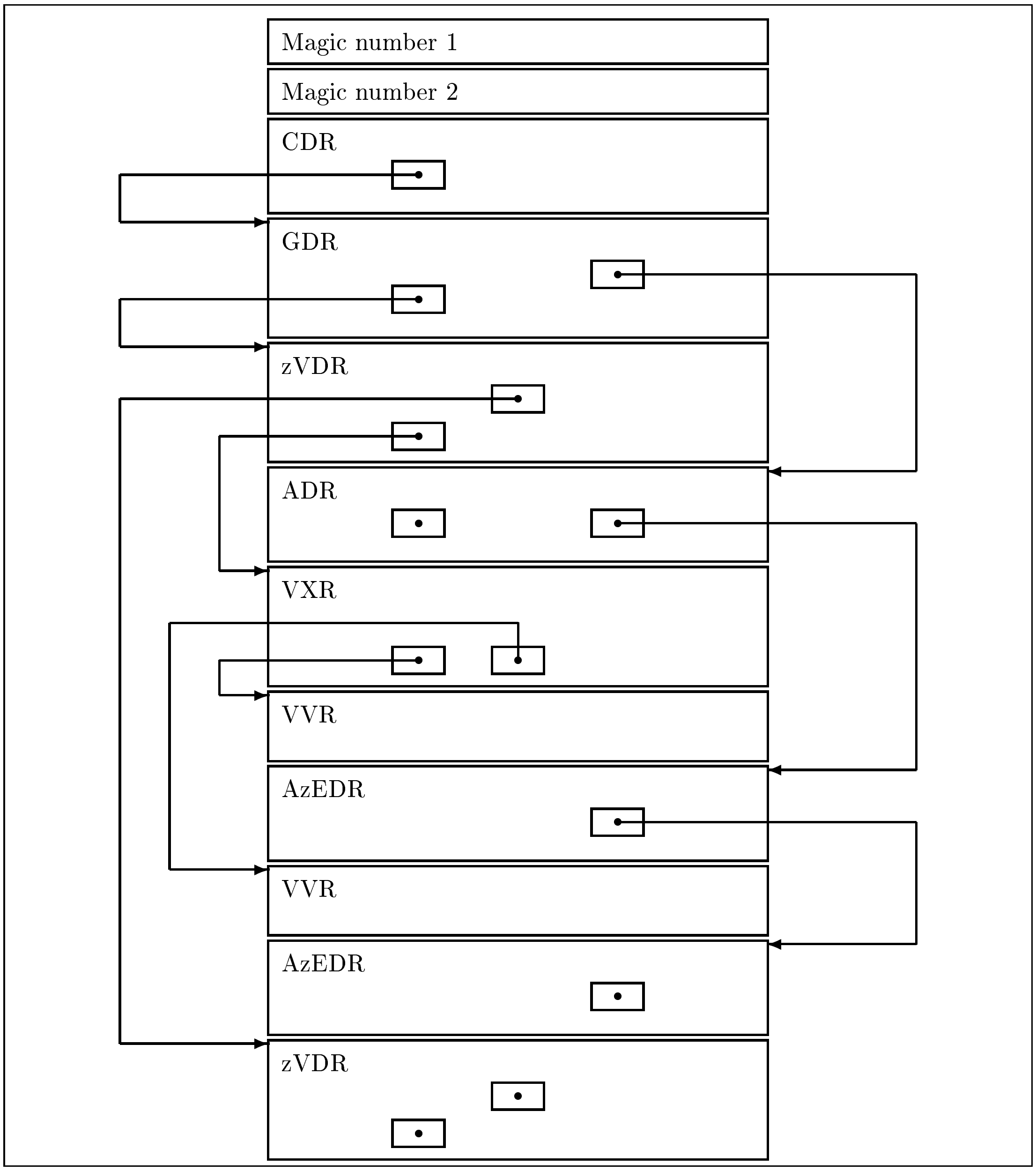

Figure 2.1: Example of an Uncompressed dotCDF File Arrangement

unsigned integer of `0x0000FFFF` for a regular CDF file[5] or `0xCCCC0001` for a compressed CDF file[6] at file offset `0x00000004`. The first internal record is stored at file offset `0x00000008`.

[4]Pre-V2.6, it is `0x0000FFFF`.
[5]That means an uncompressed CDF or a CDF with a selected variable(s) compressed
[6]Compression is not available for Pre-V2.6 CDFs. For Pre-V2.6, it is `0x0000FFFF`, repeated from the first number.

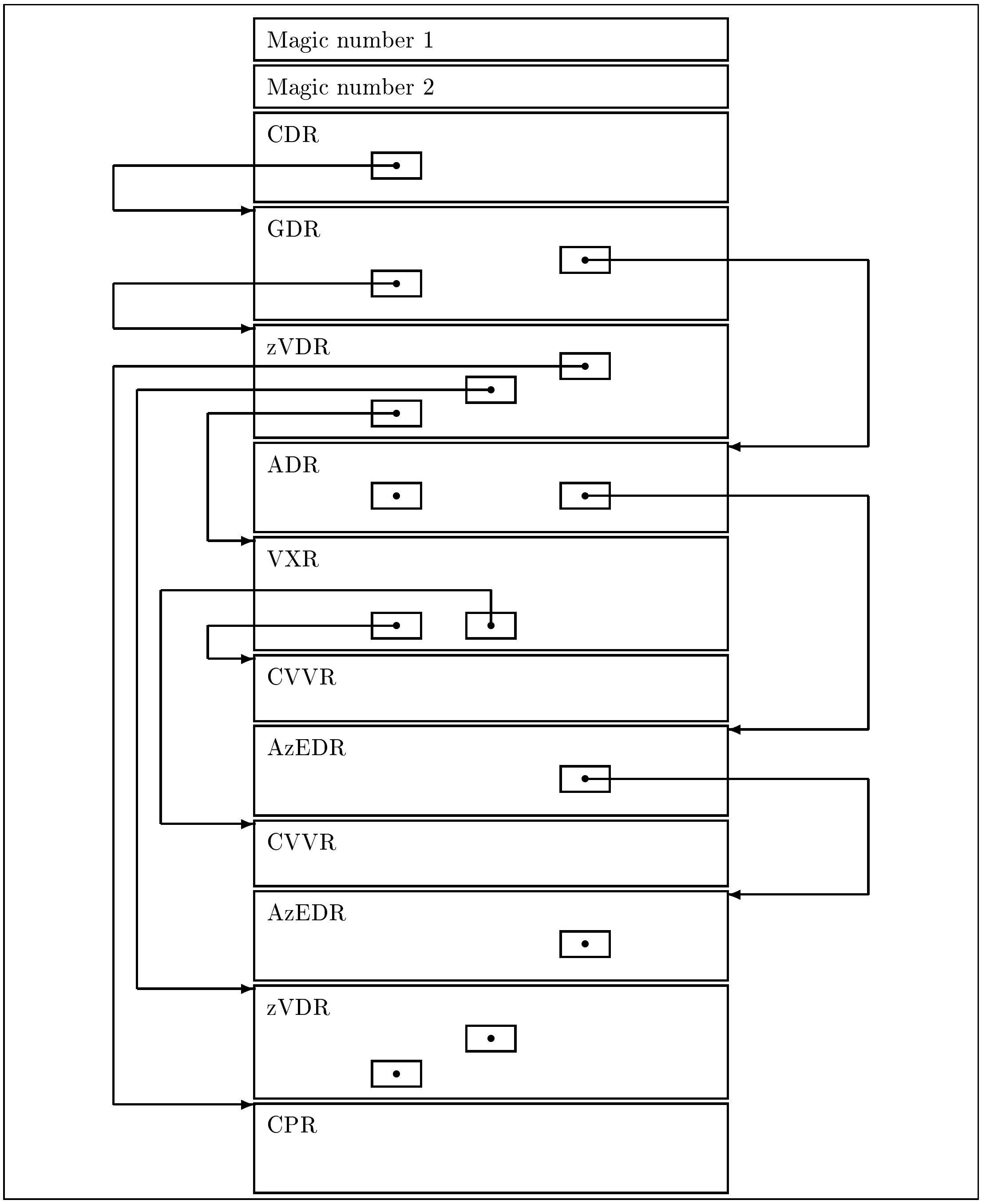

Figure 2.2: Example of a File Arrangement of a dotCDF File with a Compressed Variable

2.2 CDF Descriptor Record

All dotCDF files contain a single CDF Descriptor Record (CDR) at file offset `0x00000008`. The CDR contains general information about the CDF (as does the GDR described in Section 2.3).

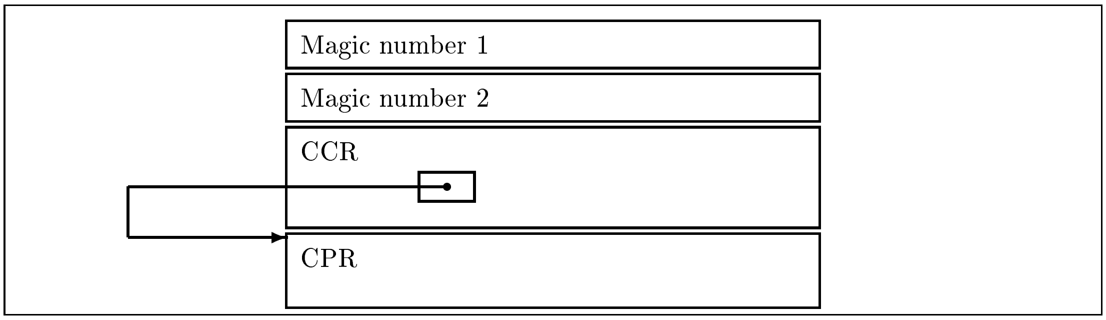

Figure 2.3: Example of a File Arrangement of a Fully Compressed dotCDF File

The CDR, as shown in Figure 2.4, contains the following contiguous fields...

`RecordSize` — Signed 4-byte integer, big-endian byte ordering.
The size in bytes of this CDR (including this field).

`RecordType` — Signed 4-byte integer, big-endian byte ordering.
The value `1` which identifies this as the CDR.

`GDRoffset` — Signed 4-byte integer, big-endian byte ordering.
The file offset of the GDR. The GDR is described in Section 2.3.

`Version` — Signed 4-byte integer, big-endian byte ordering.
The version of the CDF distribution (library) that created this CDF. CDF distributions are identified with four values: version, release, increment, and sub-increment. For example, CDF V2.5.8a is CDF version 2, release 5, increment 8, sub-increment ‘a’. Note that the sub-increment is not stored in a CDF.

`Release` — Signed 4-byte integer, big-endian byte ordering.
The release of the CDF distribution that created this CDF. See the `Version` field above.

`Encoding` — Signed 4-byte integer, big-endian byte ordering.
The data encoding for attribute entry and variable values. Section 5.3 describes the supported data encodings and their corresponding internal values.

`Flags` — Signed 4-byte integer, big-endian byte ordering.
Boolean flags, one per bit, describing some aspect of the CDF. Bit numbering is described in Chapter 5. The meaning of each bit is as follows...

- 0 — The majority of variable values within a variable record. Variable records are described in Chapter 4. Set indicates row-majority. Clear indicates column-majority.
- 1 — The file format of the CDF. Set indicates single-file. Clear indicates multi-file.
- 2-31 — Reserved for future use. These bits are always clear.

`rfuA` — Signed 4-byte integer, big-endian byte ordering.

Reserved for future use. Always set to zero (`0`).

`rfuB` — Signed 4-byte integer, big-endian byte ordering.
Reserved for future use. Always set to zero (`0`).

`Increment` — Signed 4-byte integer, big-endian byte ordering.
The increment of the CDF distribution that created this CDF. See the `Version` field above. Prior to CDF V2.1 this field was always set to zero (`0`).

`rfuD` — Signed 4-byte integer, big-endian byte ordering.
Reserved for future use. Always set to negative one (`-1`).

`rfuE` — Signed 4-byte integer, big-endian byte ordering.
Reserved for future use. Always set to negative one (`-1`).

`Copyright` — Character string, ASCII character set.
The CDF copyright notice.[7] This consists of a string of characters containing one or more lines of text with each line of text separated by a newline character (`0x0A`). If the total number of characters in the copyright is less than the size of this field, a NUL character (`0x00`) will be used to terminate the string. In that case, the characters beyond the NUL-terminator (up to the size of this field) are undefined.
This field may be one of two sizes. Prior to CDF V2.5, this field consisted of `1945` characters (bytes).[8] Since the release of CDF V2.5 this field has been reduced to `256` characters (bytes).

Field	**Size**	**Comments**
`RecordSize`	4 bytes	
`RecordType`	4 bytes	
`GDRoffset`	4 bytes	
`Version`	4 bytes	
`Release`	4 bytes	
`Encoding`	4 bytes	
`Flags`	4 bytes	
`rfuA`	4 bytes	
`rfuB`	4 bytes	
`Increment`	4 bytes	
`rfuD`	4 bytes	
`rfuE`	4 bytes	
`Copyright`	variable	1945 or 256 bytes in length depending on the CDF distribution that created/modified the CDF.

Figure 2.4: CDF Descriptor Record (CDR)

[7] Well, sort of a copyright.
[8] Much of which was space reserved for future use. That future use never occurred.

2.3 Global Descriptor Record

All dotCDF files contain a single Global Descriptor Record (GDR) at the file offset contained in the `GDRoffset` field of the CDR (described in Section 2.2). The GDR contains general information about the CDF (as does the CDR).

The GDR, shown in Figure 2.5, contains the following contiguous fields...

`RecordSize`	Signed 4-byte integer, big-endian byte ordering. The size in bytes of this GDR (including this field).
`RecordType`	Signed 4-byte integer, big-endian byte ordering. The value 2 which identifies this as the GDR.
`rVDRhead`	Signed 4-byte integer, big-endian byte ordering. The file offset of the first rVariable Descriptor Record (rVDR). The first rVDR contains a file offset to the next rVDR and so on. An rVDR will exist for each rVariable in the CDF. This field will contain `0x00000000` if the CDF contains no rVariables. Beginning with CDF V2.1 the last rVDR will contain a file offset of `0x00000000` for the file offset of the next rVDR (to indicate the end of the rVDRs). Prior to CDF V2.1 the "next VDR" file offset in the last rVDR is undefined. rVDRs are described in Section 2.6.
`zVDRhead`	Signed 4-byte integer, big-endian byte ordering. The file offset of the first zVariable Descriptor Record (zVDR). The first zVDR contains a file offset to the next zVDR and so on. A zVDR will exist for each zVariable in the CDF. Because zVariables were not supported by CDF until CDF V2.2, prior to CDF V2.2 this field is undefined. Beginning with CDF V2.2 this field will contain either a file offset to the first zVDR or `0x00000000` if the CDF contains no zVariables. The last zVDR will always contain `0x00000000` for the file offset of the next zVDR (to indicate the end of the zVDRs). zVDRs are described in Section 2.6.
`ADRhead`	Signed 4-byte integer, big-endian byte ordering. The file offset of the first Attribute Descriptor Record (ADR). The first ADR contains a file offset to the next ADR and so on. An ADR will exist for each attribute in the CDF. This field will contain `0x00000000` if the CDF contains no attributes. Beginning with CDF V2.1 the last ADR will contain a file offset of `0x00000000` for the file offset of the next ADR (to indicate the end of the ADRs). Prior to CDF V2.1 the "next ADR" file offset in the last ADR is undefined. ADRs are described in Section 2.4.
`eof`	Signed 4-byte integer, big-endian byte ordering. The end-of-file (EOF) position in the dotCDF file. This is the file offset of the byte that is one beyond the last byte of the last internal record. (This value is also the total number of bytes used in the dotCDF file.) Prior to CDF V2.1, this field is undefined.
`NrVars`	Signed 4-byte integer, big-endian byte ordering. The number of rVariables in the CDF. This will correspond to the number of rVDRs in the dotCDF file.
`NumAttr`	Signed 4-byte integer, big-endian byte ordering.

	The number of attributes in the CDF. This will correspond to the number of ADRs in the dotCDF file.
`rMaxRec`	Signed 4-byte integer, big-endian byte ordering. The maximum rVariable record number in the CDF. Note that variable record numbers are numbered beginning with zero (`0`). If no rVariable records exist, this value will be negative one (`-1`).
`rNumDims`	Signed 4-byte integer, big-endian byte ordering. The number of dimensions for rVariables.
`NzVars`	Signed 4-byte integer, big-endian byte ordering. The number of zVariables in the CDF. This will correspond to the number of zVDRs in the dotCDF file. Prior to CDF V2.2 this value will always be zero (`0`).
`UIRhead`	Signed 4-byte integer, big-endian byte ordering. The file offset of the first Unused Internal Record (UIR). The first UIR contains the file offset of the next UIR and so on. The last UIR contains a file offset of `0x00000000` for the file offset of the next UIR (indicating the end of the UIRs). Prior to CDF V2.5 this field will always contain a file offset of `0x00000000` (indicating no UIRs). Internal records that are unused may exist, however, prior to CDF V2.5. They have slightly different contents than UIRs and will be discussed in Section 2.13 along with actual UIRs.
`rfuC`	Signed 4-byte integer, big-endian byte ordering. Reserved for future use. Always set to zero (`0`).
`rfuD`	Signed 4-byte integer, big-endian byte ordering. Reserved for future use. Always set to negative one (`-1`).
`rfuE`	Signed 4-byte integer, big-endian byte ordering. Reserved for future use. Always set to negative one (`-1`).
`rDimSizes`	Signed 4-byte integers, big-endian byte ordering within each. Zero or more contiguous rVariable dimension sizes depending on the value of the `rNumDims` field described above.

2.4 Attribute Descriptor Record

An Attribute Descriptor Record (ADR) contains a description of an attribute in a CDF. There will be one ADR per attribute. The `ADRhead` field of the GDR contains the file offset of the first ADR.

Each ADR, as shown in Figure 2.6, contains the following contiguous fields...

`RecordSize`	Signed 4-byte integer, big-endian byte ordering. The size in bytes of this ADR (including this field).
`RecordType`	Signed 4-byte integer, big-endian byte ordering. The value `4` which identifies this as an ADR.

Field	Size	Comments
`RecordSize`	4 bytes	
`RecordType`	4 bytes	
`rVDRhead`	4 bytes	
`zVDRhead`	4 bytes	
`ADRhead`	4 bytes	
`eof`	4 bytes	
`NrVars`	4 bytes	
`NumAttr`	4 bytes	
`rMaxRec`	4 bytes	
`rNumDims`	4 bytes	
`NzVars`	4 bytes	
`UIRhead`	4 bytes	
`rfuC`	4 bytes	
`rfuD`	4 bytes	
`rfuE`	4 bytes	
`rDimSizes`	variable	Size depends on `rNumDims` field. If zero rVariable dimensions, this field will not be present.

Figure 2.5: Global Descriptor Record (CDR)

`ADRnext` Signed 4-byte integer, big-endian byte ordering.
The file offset of the next ADR. Beginning with CDF V2.1 the last ADR will contain a file offset of `0x00000000` in this field (to indicate the end of the ADRs). Prior to CDF V2.1 this file offset is undefined in the last ADR.

`AgrEDRhead` Signed 4-byte integer, big-endian byte ordering.
The file offset of the first Attribute g/rEntry Descriptor Record (AgrEDR) for this attribute. The first AgrEDR contains a file offset to the next AgrEDR and so on. An AgrEDR will exist for each g/rEntry for this attribute. This field will contain `0x00000000` if the attribute has no g/rEntries. Beginning with CDF V2.1 the last AgrEDR will contain a file offset of `0x00000000` for the file offset of the next AgrEDR (to indicate the end of the AgrEDRs). Prior to CDF V2.1 the "next AgrEDR" file offset in the last AgrEDR is undefined.

Note that the term g/rEntry is used to refer to an entry that may be either a gEntry or an rEntry. The type of entry described by an AgrEDR depends on the scope of the corresponding attribute. AgrEDRs of a global-scoped attribute describe gEntries. AgrEDRs of a variable-scoped attribute describe rEntries.

`Scope` Signed 4-byte integer, big-endian byte ordering.
The intended scope of this attribute. The following internal values are possible...

1 Global scope.

2 Variable scope.

3 Global scope assumed.

	4 Variable scope assumed.
	Note that assumed scopes only exist prior to CDF V2.5.
`Num`	Signed 4-byte integer, big-endian byte ordering. This attribute's number. Attributes are numbered beginning with zero (`0`).
`NgrEntries`	Signed 4-byte integer, big-endian byte ordering. The number of g/rEntries for this attribute.
`MAXgrEntry`	Signed 4-byte integer, big-endian byte ordering. The maximum numbered g/rEntry for this attribute. g/rEntries are numbered beginning with zero (`0`). If there are no g/rEntries, this field will contain negative one (`-1`).
`rfuA`	Signed 4-byte integer, big-endian byte ordering. Reserved for future used. Always set to zero (`0`).
`AzEDRhead`	Signed 4-byte integer, big-endian byte ordering. The file offset of the first Attribute zEntry Descriptor Record (AzEDR) for this attribute. The first AzEDR contains a file offset to the next AzEDR and so on. An AzEDR will exist for each zEntry for this attribute. This field will contain `0x00000000` if this attribute has no zEntries. The last AzEDR will contain a file offset of `0x00000000` for the file offset of the next AzEDR (to indicate the end of the AzEDRs). Because zEntries were not supported by CDF until CDF V2.2, prior to CDF V2.2 this field will always contain a file offset of `0x00000000`.
`NzEntries`	Signed 4-byte integer, big-endian byte ordering. The number of zEntries for this attribute. Prior to CDF V2.2 this field will always contain a value of zero (`0`).
`MAXzEntry`	Signed 4-byte integer, big-endian byte ordering. The maximum numbered zEntry for this attribute. zEntries are numbered beginning with zero (`0`). Prior to CDF V2.2 this field will always contain a value of negative one (`-1`).
`rfuE`	Signed 4-byte integer, big-endian byte ordering. Reserved for future use. Always set to negative one (`-1`).
`Name`	Character string, ASCII character set. The name of this attribute. This field is always `64` bytes in length. If the number of characters in the name is less than `64`, a NUL character (`0x00`) will be used to terminate the string. In that case, the characters beyond the NUL-terminator (up to the size of this field) are undefined.

2.5 Attribute Entry Descriptor Record

An Attribute Entry Descriptor Record (AEDR) contains a description of an attribute entry. There are two types of AEDRs: AgrEDRs describing g/rEntries and AzEDRs describing zEntries.[9] The `AgrEDRhead`

[9] Because the only difference between AgrEDRs and AzEDRs is the value of the `RecordType` field, they will be referred to as AEDRs throughout this document.

Field	Size	Comments
RecordSize	4 bytes	
RecordType	4 bytes	
ADRnext	4 bytes	
AgrEDRhead	4 bytes	
Scope	4 bytes	
Num	4 bytes	
NgrEntries	4 bytes	
MAXgrEntry	4 bytes	
rfuA	4 bytes	
AzEDRhead	4 bytes	
NzEntries	4 bytes	
MAXzEntry	4 bytes	
rfuE	4 bytes	
Name	64 bytes	

Figure 2.6: Attribute Descriptor Record (ADR)

field of an ADR contains the file offset of the first AgrEDR for the corresponding attribute. Likewise, the `AzEDRhead` field of an ADR contains the file offset of the first AzEDR. The linked lists of AEDRs starting at `AgrEDRhead` and `AzEDRhead` will contain only AEDRs of that type — AgrEDRs or AzEDRs, respectively.

Note that the term g/rEntry is used to refer to an entry that may be either a gEntry or an rEntry. The type of entry described by an AgrEDR depends on the scope of the corresponding attribute. AgrEDRs of a global-scoped attribute describe gEntries. AgrEDRs of a variable-scoped attribute describe rEntries. The scope of an attribute is stored in the `Scope` field of the corresponding ADR.

Each AEDR, as shown in Figure 2.7, contains the following contiguous fields...

`RecordSize` — Signed 4-byte integer, big-endian byte ordering.
The size in bytes of this AEDR (including this field).

`RecordType` — Signed 4-byte integer, big-endian byte ordering.
Either the value `5` which identifies this as an AgrEDR or the value `9` if an AzEDR. Because zEntries were not supported until CDF V2.2, prior to CDF V2.2 AzEDRs will not occur in a dotCDF file.

`AEDRnext` — Signed 4-byte integer, big-endian byte ordering.
The file offset of the next AEDR. Beginning with CDF V2.1 the last AEDR will contain a file offset of `0x00000000` in this field (to indicate the end of the AEDRs). Prior to CDF V2.1 this file offset is undefined in the last AEDR.[10]

`Num` — Signed 4-byte integer, big-endian byte ordering.
The attribute number to which this entry corresponds. Attributes are numbered beginning with zero (`0`).

`DataType` — Signed 4-byte integer, big-endian byte ordering.
The data type of this entry. The possible data type internal values are

[10] Note that prior to CDF V2.2 this only applies to AgrEDRs because zEntries were not yet supported.

described in Section 5.3.

`EntryNum` Signed 4-byte integer, big-endian byte ordering.
This entry's number. Entries are numbered beginning with zero (`0`).

`NumElems` Signed 4-byte integer, big-endian byte ordering.
The number of elements of the data type (specified by the `DataType` field) for this entry.

`rfuA` Signed 4-byte integer, big-endian byte ordering.
Reserved for future used. Always set to zero (`0`).

`rfuB` Signed 4-byte integer, big-endian byte ordering.
Reserved for future used. Always set to zero (`0`).

`rfuC` Signed 4-byte integer, big-endian byte ordering.
Reserved for future used. Always set to zero (`0`).

`rfuD` Signed 4-byte integer, big-endian byte ordering.
Reserved for future used. Always set to negative one (`-1`).

`rfuE` Signed 4-byte integer, big-endian byte ordering.
Reserved for future used. Always set to negative one (`-1`).

`Value` This entry's value. This consists of the number of elements (specified by the `NumElems` field) of the data type (specified by the `DataType` field). This can be thought of as a 1-dimensional array of values (stored contiguously). The size of this field is the product of the number of elements and the size in bytes of each element. The encoding of the elements depends on the data encoding of the CDF (which is contained in the `Encoding` field of the CDR). The possible encodings are described in Section 5.3.

Field	**Size**	**Comments**
`RecordSize`	4 bytes	
`RecordType`	4 bytes	
`AEDRnext`	4 bytes	
`AttrNum`	4 bytes	
`DataType`	4 bytes	
`Num`	4 bytes	
`NumElems`	4 bytes	
`rfuA`	4 bytes	
`rfuB`	4 bytes	
`rfuC`	4 bytes	
`rfuD`	4 bytes	
`rfuE`	4 bytes	
`Value`	variable	Size depends on the `DataType` and `NumElems` fields.

Figure 2.7: Attribute Entry Descriptor Record (AEDR)

2.6 Variable Descriptor Record

A Variable Descriptor Record (VDR) contains a description of a variable in a CDF. There are two types of VDRs: rVDRs describing rVariables and zVDRs describing zVariables.[11] The `rVDRhead` field of the GDR contains the file offset of the first rVDR. Likewise, the `zVDRhead` field of the GDR contains the file offset of the first zVDR. The linked lists of VDRs starting at `rVDRhead` and `zVDRhead` will contain only VDRs of that type — rVDRs or zVDRs, respectively. If this variable is compressed, a pointer to a Compressed Parameters Record (CPR) is set in the CPRorSPRoffset field.

Each VDR, as shown in Figure 2.8, contains the following contiguous fields...[12]

`RecordSize` Signed 4-byte integer, big-endian byte ordering.
The size in bytes of this VDR (including this field).

`RecordType` Signed 4-byte integer, big-endian byte ordering.
Either the value `3` which identifies this as an rVDR or the value `8` if a zVDR. Because zVariables were not supported until CDF V2.2, prior to CDF V2.2 zVDRs will not occur in a dotCDF file.

`VDRnext` Signed 4-byte integer, big-endian byte ordering.
The file offset of the next VDR. Beginning with CDF V2.1 the last VDR will contain a file offset of `0x00000000` in this field (to indicate the end of the VDRs). Prior to CDF V2.1 this file offset is undefined in the last VDR.[13]

`DataType` Signed 4-byte integer, big-endian byte ordering.
The data type of this entry. The possible data type internal values are described in Section 5.3.

`MaxRec` Signed 4-byte integer, big-endian byte ordering.
The maximum record number written to this variable. Variable records are numbered beginning at zero (`0`). If no records have been written to this variable, this field will contain negative one (`-1`).

`VXRhead` Signed 4-byte integer, big-endian byte ordering.
The file offset of the first Variable Index Record (VXR). VXRs are used in single-file CDFs to store the locations of Variable Value Records (VVRs). VVRs are used to store variable records in single-file CDFs. VXRs are described in Section 2.7 and VVRs are described in Section 2.8. The first VXR contains the file offset of the next VXR and so on. The last VXR contains a file offset of `0x00000000` for the file offset of the next VXR (to indicate the end of the VXRs). In single-file CDFs, if no records have been written to this variable, this field will contain a file offset of `0x00000000`.

For multi-file CDFs variable records are stored in separate files and this field will always contain a file offset of `0x00000000`. The variable files of a multi-file CDF are described in Chapter 3.

`VXRtail` Signed 4-byte integer, big-endian byte ordering.

[11] The term VDR is used when something applies to both rVDRs and zVDRs. The terms rVDR and zVDR will be used when a distinction must be made.
[12] With the exceptions for rVariables being noted.
[13] Note that prior to CDF V2.2 this only applies to rVDRs because zVariables were not yet supported.

	The file offset of the last VXR. See the `VXRhead` field above for a description of VXRs.
`Flags`	Signed 4-byte integer, big-endian byte ordering. Boolean flags, one per bit, describing some aspect of this variable. Bit numbering is described in Chapter 5. The meaning of each bit is as follows...

0	The record variance of this variable. Set indicates a `TRUE` record variance. Clear indicates a `FALSE` record variance.
1	Whether or not a pad value is specified for this variable. Set indicates that a pad value has been specified. Clear indicates that a pad value has not been specified. The `PadValue` field described below is only present if a pad value has been specified.
2	Whether or not a compression method is applied to this variable. Set indicates that a compression has been used. Clear indicates that a compression has not been used. The `CPRorSPRoffset` field described below provides the offset of the Compressed Parameters Record if this compression bit is set.
3-31	Reserved for future use. These bits are always clear.

`sRecords`	Signed 4-byte integer, big-endian byte ordering. Type of sparse records: no sparserecords, padded sparserecords, or previous sparserecords.
`rfuB`	Signed 4-byte integer, big-endian byte ordering. Reserved for future use. Always set to zero (`0`).
`rfuC`	Signed 4-byte integer, big-endian byte ordering. Reserved for future use. Always set to negative one (`-1`).
`rfuF`	Signed 4-byte integer, big-endian byte ordering. Reserved for future use. Always set to negative one (`-1`).
`NumElems`	Signed 4-byte integer, big-endian byte ordering. The number of elements of the data type (specified by the `DataType` field) for this variable at each value.
`Num`	Signed 4-byte integer, big-endian byte ordering. This variable's number. Variables are numbered beginning with zero (`0`). Note that rVariables and zVariables are each numbered beginning with zero (`0`) and are considered two separate groups of variables.
`CPRorSPRoffset`	Signed 4-byte integer, big-endian byte ordering. CPR/SPR offset depending on bits set in 'Flags'. If neither compression nor sparse arrays, set to `0xFFFFFFFF`.
`BlockingFactor`	Signed 4-byte integer, big-endian byte ordering. Blocking factor for this variable.

`Name`	Character string, ASCII character set. The name of this variable. This field is always `64` bytes in length. If the number of characters in the name is less than `64`, a NUL character (`0x00`) will be used to terminate the string. In that case, the characters beyond the NUL-terminator (up to the size of this field) are undefined.
`zNumDims`	Signed 4-byte integer, big-endian byte ordering. The number of dimensions for this zVariable. This field will not be present if this is an rVDR (rVariable).
`zDimSizes`	Signed 4-byte integers, big-endian byte ordering within each. Zero or more contiguous dimension sizes for this zVariable depending on the value of the `zNumDims` field. This field will not be present if this is an rVDR (rVariable).
`DimVarys`	Signed 4-byte integers, big-endian byte ordering within each. Zero or more contiguous dimension variances. If this is an rVDR, the number of dimension variances will correspond to the value of the `rNumDims` field of the GDR. If this is a zVDR, the number of dimension variances will correspond to the value of the `zNumDims` field in this zVDR. A value of negative one (`-1`) indicates a `TRUE` dimension variance and a value of zero (`0`) indicates a `FALSE` dimension variance.
`PadValue`	The variable's pad value. If bit 1 of the `Flags` field of this VDR is clear, then a pad value has not been specified for this variable and this field will not be present. If a pad value has been specified, the size of this field depends on the number of elements and the size of the data type. The encoding of the elements depends on the encoding of the CDF (which is contained in the `Encoding` field of the CDR). The possible encodings are described in Section 5.3.

2.7 Variable Index Record

Variable Index Records (VXRs) are used in single-file CDFs to store the file offsets of Variable Values Records (VVRs). VVRs contain a group of records written to a variable and are described in Section 2.8. VXRs (and VVRs) will not exist in the dotCDF file of a multi-file CDF (because the variable records are stored in separate files as described in Chapter 3).

The `VXRhead` field of a VDR in a single-file CDF contains the file offset of the first VXR for the corresponding variable. The first VXR contains the file offset of the next VXR and so on. As many VXRs as are necessary will exist (depending on the number of VVRs for the variable). The `VXRtail` field of a VDR contains the file offset of the last VXR for the corresponding variable.

Each VXR, as shown in Figure 2.9, contains the following contiguous fields…

`RecordSize`	Signed 4-byte integer, big-endian byte ordering. The size in bytes of this VXR (including this field).
`RecordType`	Signed 4-byte integer, big-endian byte ordering. The value `6` which identifies this as a VXR.

Field	Size	Comments
RecordSize	4 bytes	
RecordType	4 bytes	
VDRnext	4 bytes	
DataType	4 bytes	
MaxRec	4 bytes	
VXRhead	4 bytes	
VXRtail	4 bytes	
Flags	4 bytes	
sRecords	4 bytes	
rfuB	4 bytes	
rfuC	4 bytes	
rfuF	4 bytes	
NumElems	4 bytes	
Num	4 bytes	
CPRorSPRoffset	4 bytes	
BlockingFactor	4 bytes	
Name	64 bytes	
zNumDims	variable	4 bytes if a zVDR. Not present if an rVDR.
zDimSizes	variable	Size depends on the zNumDims field if a zVDR (but not present if zero dimensions). Not present if an rVDR.
DimVarys	variable	Size depends on the zNumDims field if a zVDR (but not present if zero dimensions). Size depends on the rNumDims field of the GDR if an rVDR (but not present if zero dimensions).
PadValue	variable	Size depends on DataType and NumElems fields. Not present if bit 1 of Flags field is not set.

Figure 2.8: Variable Descriptor Record (VDR)

VXRnext — Signed 4-byte integer, big-endian byte ordering.
The file offset of the next VXR. The last VXR will contain a file offset of 0x00000000 in this field (to indicate the end of the VXRs).

Nentries — Signed 4-byte integer, big-endian byte ordering.
The number of index entries in this VXR. This is the maximum number of VVRs that may be indexed using this VXR.

NusedEntries — Signed 4-byte integer, big-endian byte ordering.
The number of index entries actually used in this VXR.

First — Signed 4-byte integers, big-endian byte ordering within each.
This is a contiguous array of variable record numbers with each record number being the first variable record in the corresponding VVR. The size of this array depends on the value of the Nentries field. The n^{th} entry in this array corresponds to the n^{th} entry in the Last and Offset fields. Unused entries in this array contain 0xFFFFFFFF. Note that variable records are numbered beginning with zero (0).

Last — Signed 4-byte integers, big-endian byte ordering within each.
This is a contiguous array of variable record numbers with each record

number being the last variable record in the corresponding VVR. The size of this array depends on the value of the `Nentries` field. The `n`th entry in this array corresponds to the `n`th entry in the `First` and `Offset` fields. Unused entries in this array contain `0xFFFFFFFF`. Note that variable records are numbered beginning with zero (`0`).

`Offset` Signed 4-byte integers, big-endian byte ordering within each. This is a contiguous array of file offsets with each being the file offset of the corresponding VVR. The size of this array depends on the value of the `Nentries` field. The `n`th entry in this array corresponds to the `n`th entry in the `First` and `Last` fields. Unused entries in this array contain `0xFFFFFFFF`.

Field	**Size**	**Comments**
`RecordSize`	4 bytes	
`RecordType`	4 bytes	
`VXRnext`	4 bytes	
`Nentries`	4 bytes	
`NusedEntries`	4 bytes	
`First`	variable	Size depends on the `Nentries` field.
`Last`	variable	Size depends on the `Nentries` field.
`Offset`	variable	Size depends on the `Nentries` field.

Figure 2.9: Variable Index Record (VXR)

Consider the following example VXR contents (for a variable having only one VXR)...

```
RecordSize:        140
RecordType:        6
VXRnext:       0x00000000
Nentries:      10
NusedEntries:  2
First:     0, 100, 0xFFFFFFFF, 0xFFFFFFFF, ...
Last:      99, 149, 0xFFFFFFFF, 0xFFFFFFFF, ...
Offset:    0x0000A400, 0x0000B554, 0xFFFFFFFF, 0xFFFFFFFF, ...
```

There are two index entries being used. The first indicates that variable records `0` through `99` are stored in the VVR at file offset `0x0000A400` and the second indicates that variable records `100` through `149` are stored in the VVR at file offset `0x0000B554`.

2.8 Variable Values Record

Variable Value Records (VVRs) are used to store one or more variable records in a single-file CDF. VVRs will not exist in multi-file CDFs (where variable records are stored in separate files). The contents of a variable record is described in Chapter 4.

Each VVR, as shown in Figure 2.10, contains the following contiguous fields...

`RecordSize` Signed 4-byte integer, big-endian byte ordering.
The size in bytes of this VVR (including this field).

`RecordType` Signed 4-byte integer, big-endian byte ordering.
The value `7` which identifies this as a VVR.

`Records` A group of one or more variable records. The record numbers in this group will be contiguous. The size of this field depends on the number of variable records in the group and the size of each record. The size of each record will be the same and depends on the dimensionality, dimension variances, data type, and number of elements per value of the corresponding variable. These properties are discussed in Chapter 4. The encoding of the values in each variable record depends on the encoding of the CDF (which is stored in the `Encoding` field of the CDR). The possible encodings are described in Chapter 5.

Field	**Size**	**Comments**
`RecordSize`	4 bytes	
`RecordType`	4 bytes	
`Records`	variable	Size depends on the number of variable records in this VVR and the variable's data type, number of elements per value, dimensionality, and dimension variances.

Figure 2.10: Variable Values Record (VVR)

2.9 Compressed CDF Record

A Compressed CDF Record (CCR) is used to store the data from a compressed single-file CDF. A CCR is created when the whole CDF is compressed. It will not be created if only variables (some or even all) are compressed. Only two internal records exist in a fully compressed CDF. Other than a CCR, another record is a Compression Parameters Record (`CPR`) which is pointed to by the CCR. The CPR provides the compression information, e.g., compression method and level, etc., used to compress the CDF file. A CCR will not exist in multi-file CDFs.

Each CCR, as shown in Figure 2.11, contains the following contiguous fields…

`RecordSize` Signed 4-byte integer, big-endian byte ordering.
The size in bytes of this CCR (including this field).

`RecordType` Signed 4-byte integer, big-endian byte ordering.
The value `10` which identifies this as a CCR.

`CPRoffset` Signed 4-byte integer, big-endian byte ordering.
File offset to the Compressed Parameters Record (CPR) (bytes).

`uSize` Signed 4-byte integer, big-endian byte ordering.
Size of the CDF in its uncompressed IRs form. This byte count does `NOT` include the magic numbers.