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

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

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

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

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

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

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

No variable values should be put in the skeleton table.

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

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

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

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

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

All variable values should be put in the skeleton table.

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

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

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

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

`/NONRV` (VMS)

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

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

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

Put NRV variable data values in the skeleton table.

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

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

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

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

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

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

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

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

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

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

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

Specifies the types of return status codes from the CDF library that should be reported/displayed. The `<types>` option is a comma-separated list of zero or more of the following symbols: `errors`, `warnings`, or `informationals`. Note that these symbols can be truncated (e.g., `e`, `w`, and `i`).

`/CACHE=(<sizes>)` (VMS)

`-cache "<sizes>"` (UNIX & MS-DOS)
`Cache sizes` edit field (Macintosh)

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

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

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

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

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

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

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

Example(s):

VMS:

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

UNIX:

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

MS-DOS:

```
> skeletontable -nolog -report "errors" fgge3b
```

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

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

3.7.4 Output from the SkeletonTable Program

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

3.8 SkeletonCDF

3.8.1 Introduction

The SkeletonCDF [10] program is used to make a fully structured CDF, called a skeleton CDF, by reading a text file called a skeleton table. The SkeletonCDF program allows a CDF to be created with the following:

1. The necessary header information — the number of dimensions and dimension sizes for the rVariables, format, data encoding, and variable majority.
2. The gAttribute definitions and any number of gEntries for each.
3. The rVariable and zVariable definitions.
4. The vAttribute definitions and the entries corresponding to each variable.
5. The data values for any or all of the variables.

The created CDF is referred to as a skeleton CDF.

3.8.2 Executing the SkeletonCDF Program

Usage:

VMS:

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

UNIX:

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

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

MS-DOS:[11]

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

Macintosh:

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

Parameter(s):

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

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

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

Qualifier(s):

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

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

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

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

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

`/[NO]NEG2POSFP0` (VMS)

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

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

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

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

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

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

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

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

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

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

Specifies the zMode that should be used with the skeleton table. If zMode is enabled, zVariables will be created from the definitions in the rVariables section. The zMode may be one of the following:

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

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

Specifies the types of return status codes from the CDF library that should be reported/displayed. The `<types>` option is a comma-separated list of zero or more of the following symbols: `errors`, `warnings`, or `informationals`. Note that these symbols can be truncated (e.g., `e`, `w`, and `i`).

Example(s):

VMS:

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

UNIX:

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

MS-DOS:

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

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

3.8.3 Creating the Skeleton Table

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

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

3.9 CDFinquire

3.9.1 Introduction

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

3.9.2 Executing the CDFinquire Program

Usage:

VMS:

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

UNIX:

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

MS-DOS:[12]

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

Macintosh:

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

Parameter(s):

`(None)`

Qualifier(s):

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

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

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

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

[12] On MS-DOS systems the executable is named `CDFINQ.EXE`.

Example(s):

VMS:

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

UNIX:

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

MS-DOS:

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

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

3.9.3 Output from the CDFinquire Program

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

3.10 CDFdir

3.10.1 Introduction

The CDFdir utility is used to display a directory listing of a CDF's files.[13] The dotCDF file is displayed first followed by the rVariable files and then the zVariable files (if either exist in a multi-file CDF) in numerical order.

3.10.2 Executing the CDFdir Program

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

Usage:

VMS:

[13] CDFdir is not available on a Macintosh.

```
$ 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.

3.10.3 Output from the CDFdir Program

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

3.11 CDFbrowse

CDFbrowse has been replaced by CDFedit (see Section 3.2).[14]

3.12 CDFlist

CDFlist has been replaced by CDFexport (see Section 3.3).

3.13 CDFwalk

The functionallity of CDFwalk has been added to CDFexport (see Section 3.3).

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

Appendix A

Skeleton Table Format

A.1 Introduction

Skeleton tables are both created by and read by CDF utility programs. `SkeletonTable` creates a skeleton table by reading a CDF. `SkeletonCDF` creates a CDF by reading a skeleton table. In almost all cases the format of the skeleton tables read and written will be the same. Any differences are minor and will be described where appropriate.

The skeleton table has a free format (except where noted) — you need not be concerned with any column alignments, spaces between fields, or spaces between successive lines. However, certain syntax rules do apply to skeleton tables.

1. Lines are limited to 132 characters.
2. Keywords for the header section, gAttributes section, vAttributes section, rVariables section, and end section must always be specified (in that order). The zVariables section is optional — its keyword may be omitted.
3. An exclamation point (`!`) at any point signifies a comment until the end of the line. Any characters encountered after the exclamation point will be ignored. An exclamation point may begin a line (making the entire line a comment). Exclamation points inside delimited character strings are part of the string and do not cause the start of a comment.
4. Attribute and variable names must be delimited. Any character not in the name may be used as the delimiter with the following exceptions:
   (a) Do not use an exclamation point (`!`) to delimit an attribute or variable name.
   (b) Do not use a period (`.`) to delimit an attribute name in the variables section.
   (c) Do not use a left square bracket (`[`) or a numeral to delimit a variable name.
5. When specifying a character string attribute entry value, do not use a hyphen (`-`) to delimit the string or strings (if the string is split across one or more lines).
6. All items are referenced from one (1). These include gAttribute gEntry numbers and NRV variable index values.

In the descriptions that follow, optional fields are shown in brackets ([...]).

A.2 Header Section

The header section contains general information about the CDF. The format of the header section is as follows:

```
#header

                    CDF NAME: <cdf-name>
               DATA ENCODING: <data-encoding>
                    MAJORITY: <variable-majority>
                      FORMAT: <cdf-format>

!     Variables   G.Attributes V.Attributes   Records     Dims       Sizes
!     ---------   ------------ ------------   -------     ----    -----------
   <rVars>/<zVars>   <gAttrs>    <vAttrs>   <n-recs>/z  <n-dims>  <dim-sizes>
```

The fields are defined as follows:

`<cdf-name>`	The name of the CDF. When `SkeletonTable` creates a skeleton table, this will be the name of the corresponding CDF (not the full pathname specified). When `SkeletonCDF` reads a skeleton table, this will be the name of the CDF created unless a CDF pathname is specified on the command line. If the CDF name in the skeleton table is to be used, a full pathname must be specified (if desired) or else the CDF will be created in the default/current directory.
`<data-encoding>`	The data encoding of the CDF. When specifying a data encoding to the `SkeletonCDF` program, the following encodings are valid: `HOST`, `NETWORK`, `VAX`, `ALPHAVMSd`, `ALPHAVMSg`, `ALPHAVMSi`, `SUN`, `SGi`, `DECSTATION`, `ALPHAOSF1`, `IBMRS`, `HP`, `PC`, `MAC`, and `NeXT`. When a skeleton table is created by `SkeletonTable`, all of the above encodings with the exception of `HOST` are possible. Data encoding is described in Section 2.2.8.
`<variable-majority>`	The variable majority of the CDF. This may be either `ROW` or `COLUMN`. Variable majority is described in Section 2.3.15.
`<cdf-format>`	The format of the CDF. This may be either `SINGLE` or `MULTI`. CDF formats are described in Section 2.2.7. Note that this line is optional. Skeleton tables created by `SkeletonTable` in CDF V2.0 did not have this line because the single-file option did not exist. To allow `SkeletonCDF` to read skeleton tables created with `SkeletonTable` in CDF V2.0, this line was made optional. If omitted, `SkeletonCDF` will create a CDF with the default format for your CDF distribution. Consult your system manager to determine this default. `SkeletonTable` (in CDF V2.1 and beyond) always generates this line regardless of the version of the CDF being read.

`<rVars>` The number of rVariables in the CDF. `SkeletonTable` always places the correct number here. However, when `SkeletonCDF` reads a skeleton table, this value is ignored (but a place holder is necessary). The number of rVariables created is determined by the number of rVariable definitions in the rVariable definitions section.

`<zVars>` The number of zVariables in the CDF. `SkeletonTable` always places the correct number here. However, when `SkeletonCDF` reads a skeleton table, this value is ignored (but a place holder is necessary). The number of zVariables created is determined by the number of zVariable definitions in the zVariable definitions section.

`<gAttrs>` The number of gAttributes in the CDF. `SkeletonTable` always places the correct number here. However, when `SkeletonCDF` reads a skeleton table, this value is ignored (but a place holder is necessary). The number of gAttributes created is determined by the number of definitions in the gAttributes section.

`<vAttrs>` The number of vAttributes in the CDF. `SkeletonTable` always places the correct number here. However, when `SkeletonCDF` reads a skeleton table, this value is ignored (but a place holder is necessary). The number of vAttributes created is determined by the number of definitions in the vAttributes section.

`<n-recs>` The (maximum) number of rVariable records in the CDF. `SkeletonTable` always places the correct number here. However, when `SkeletonCDF` reads a skeleton table, this value is ignored (but a place holder is necessary). The number of records written to the CDF depends on whether or not any values are specified for variables. NRV variables are described in Section 2.3.10.

`<n-dims>` The number of dimensions for the rVariables in the CDF.

`<dim-sizes>` The dimension sizes for the rVariables in the CDF — one value per dimension. If the rVariables have zero (0) dimensions, this field would be left blank.

An example header section for a CDF with 2-dimensional rVariables follows:

```
#header

                  CDF NAME: sample2
             DATA ENCODING: NETWORK
                  MAJORITY: ROW
                    FORMAT: SINGLE

! Variables G.Attributes V.Attributes Records  Dims   Sizes
! --------- ------------ ------------ -------  ----  -------
    14/0         18             4        1/z      2   180 360
```

If the rVariables had zero dimensions, the header section would be as follows:

```
#header
```

```
                CDF NAME: sample0
           DATA ENCODING: NETWORK
                MAJORITY: ROW
                  FORMAT: SINGLE

! Variables G.Attributes V.Attributes Records  Dims  Sizes
! --------- ------------ ------------ -------  ----  -----
    14/0         18            4        1/z      0
```

A.3 gAttributes Section

The gAttributes section contains the definition of each gAttribute as well as any gEntries for those gAttributes. The format of the gAttributes section is as follows:

```
#GLOBALattributes

[<global-scope-attribute-definition>
 <global-scope-attribute-definition>
 <global-scope-attribute-definition>
                 .
                 .
                 .
 <global-scope-attribute-definition>]
```

Where `<global-scope-attribute-definition>`, needless to say, is a gAttribute definition.

Zero or more gAttribute definitions are allowed. (There is no limit on the number of attributes that a CDF may have.) The format of each gAttribute definition is as follows:

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

  <attr-name>   [<entry-n>:    <data-type>    <value>
                 <entry-n>:   [<data-type>]   <value>
                 <entry-n>:   [<data-type>]   <value>
                     .              .            .
                     .              .            .
                     .              .            .
                 <entry-n>:   [<data-type>]   <value>] .          ! note the "."
```

The fields are defined as follows:

`<attr-name>` The name of the gAttribute. The name must be delimited with a character not appearing in the name itself (e.g., `"TITLE"` or `'History'`). The delimiting characters are not part of the gAttribute name in the CDF.

`<entry-n>` The gEntry number. Zero or more gEntries may be specified for a gAttribute, and there are no restrictions on the gEntry numbers that may be used (except that they must be greater than zero).

`<data-type>` The data type for the gEntry. The data type must be one of the following: CDF_BYTE, CDF_INT1, CDF_UINT1, CDF_INT2, CDF_UINT2, CDF_INT4, CDF_UINT4, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_CHAR, or CDF_UCHAR. The `<data-type>` field is optional for all but the first gEntry specified. If omitted, the data type of the previous gEntry is assumed.

`<value>` The value(s) for the gEntry. A period (`.`) follows the value(s) of the last gEntry for a gAttribute.

Attribute Entry Values

An attribute entry can have more than one element of the specified data type. For character data types (CDF_CHAR and CDF_UCHAR), each character is the element of a string. The character string must be delimited with a character not appearing in the string itself, and the entire delimited string must be enclosed in braces (e.g., `{ "The CDF title." }`). If the string will not fit on one line, it may be continued on additional lines. The substrings are each delimited with a unique character, and a dash (`-`) is placed at the end (after the terminating delimiter) of each line except the last one. For example,

```
{ "This is a longer " -
  "CDF title that will" -
  " not fit on one line." }
```

For non-character data types, the elements are enclosed in braces and separated by commas (e.g., `{ 1, 2, 3 }`). If the elements will not all fit on one line, they may be continued on additional lines. For example,

```
{ 1.0, 2.0, 3.0, 4.0, 5.0,
  6.0, 7.0, 8.0, 9.0, 10.0 }
```

Note that an individual element value may not be split across lines.

The format of a value for the CDF_EPOCH data type (which is also considered a non-character data type) is defined in Section 2.5.4. A CDF_EPOCH value may not be split across two lines.

Several example gAttribute definitions follow:

```
#GLOBALattributes

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

```
  "TITLEa"              1:    CDF_CHAR     { "CDAW-9A; SABRE" }.

  ^TITLEb^              1:    CDF_CHAR     { "CDAW-9A; SABRE " -
                                             "Backscatter Radar, 20s." }.

  "History"             1:    CDF_CHAR     { "CDF created 02-Jan-1961" }
                        2:                 { "CDF modified 23-Oct-1964" }.

  "TIMES"               1:    CDF_EPOCH    { 04-Jul-1976 12:00:00.000,
                                             31-Oct-1976 00:00:00.000 }
                        2:                 { 25-Dec-1976 01:10:00.000,
                                             01-Jan-1977 01:10:30.000 }.

  &Factors&             1:    CDF_REAL4    { 12.5 }
                        2:                 { 17.4 }
                        3:                 {  8.5 }
                        4:    CDF_INT2     { 7 }
                        5:                 { 12 }.
```

A.4 vAttributes Section

The vAttributes section contains the names of the vAttributes in the CDF. Any rEntries or zEntries for these vAttributes are defined in the rVariables/zVariables sections (following the definition of the corresponding variable). The format of the vAttributes section is as follows:

```
#VARIABLEattributes

[<attribute-name>
 <attribute-name>
 <attribute-name>
        .
        .
        .
 <attribute-name>]
```

Where `<attribute-name>` is a vAttribute name delimited with a character not appearing in the name itself (e.g., `"VALIDMIN"` or `'Units'`). The delimiting characters are not part of the vAttribute name in the CDF. There may be zero or more vAttribute names. (There is no limit on the number of attributes that a CDF may have.)

An example vAttributes section follows:

```
#VARIABLEattributes

  "FIELDNAM"
  "VALIDMIN"
  "Units"
```

A.5 rVariables Section

The rVariables section contains the definition of each rVariable in the CDF, the values for any vAttribute rEntries associated with each rVariable, and (optionally) data values for those rVariables. The format of the rVariables section is as follows:

```
#variables

[<variable-definition>
 <variable-definition>
 <variable-definition>
           .
           .
           .
 <variable-definition>]
```

Where `<variable-definition>` is an rVariable definition. The format of each rVariable definition is as follows:

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

  <var-name>     <var-data-type>  <n-elems> <rec-vary>  <dim-varys>

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

     [<attr-name>  <entry-data-type>    <entry-value>
      <attr-name>  <entry-data-type>    <entry-value>
      <attr-name>  <entry-data-type>    <entry-value>
           .               .                   .
           .               .                   .
           .               .                   .
      <attr-name>  <entry-data-type>    <entry-value>].        ! Note the "."

   [[<rec-num>:]<indices> = <value>
    [<rec-num>:]<indices> = <value>
    [<rec-num>:]<indices> = <value>
        .            .
        .            .
        .            .
    [<rec-num>:]<indices> = <value>]
```

Each field is defined as follows:

`<var-name>` The name of the rVariable. The name must be delimited with a character

	not appearing in the name itself (e.g., `"EPOCH"` or `'Temperature'`). The delimiting characters are not part of the rVariable name in the CDF.
`<var-data-type>`	The data type for the rVariable. The data type must be one of the following: CDF_BYTE, CDF_INT1, CDF_UINT1, CDF_INT2, CDF_UINT2, CDF_INT4, CDF_UINT4, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_CHAR, or CDF_UCHAR.
`<n-elems>`	The number of elements of the data type. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in each string. For non-character data types, this value must be one (1).
`<rec-vary>`	The record variance of the rVariable. This must be either `T` (the values vary from record to record) or `F` (the values do not vary from record to record).
`<dim-varys>`	The dimension variances of the rVariable. For each dimension there must be either a `T` (the values vary along that dimension) or `F` (the values do not vary along that dimension). Each dimension variance must be separated by at least one space. If the rVariables have zero dimensions, this field would be left blank.
`<attr-name>`	The name of the vAttribute for which to specify an rEntry for this rVariable. The vAttribute must have been specified in the vAttributes section. The name must be delimited with a character not appearing in the name itself (e.g., `"SCALEMAX"` or `'range'`). The delimiting characters are not part of the vAttribute name in the CDF.
`<entry-data-type>`	The data type for the vAttribute rEntry. The data type must be one of the following: CDF_BYTE, CDF_INT1, CDF_UINT1, CDF_INT2, CDF_UINT2, CDF_INT4, CDF_UINT4, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_CHAR, or CDF_UCHAR.
`<entry-value>`	The value(s) for the vAttribute rEntry. The format of attribute entry values is described in Section A.3. **NOTE:** The last rEntry MUST be followed by a period (`.`). If no rEntries are specified for an rVariable, the period must still be present.
`<rec-num>`	The record number of an rVariable value. This will be present only for record-variant (RV) rVariables.
`<indices>`	The indices of an rVariable value. The indices are enclosed in brackets and separated by commas (e.g., `[23,1]` or `[1,80]`). If the rVariables have zero dimensions, `[]` would be specified (the brackets are still required).
`<value>`	The value at the given record/indices. For character data types (CDF_CHAR or CDF_UCHAR) the string must be delimited with a unique character and enclosed in braces (`{...}`) in the same manner as for an attribute entry for a character data type. For non-character data types the value is not enclosed in braces (the braces are not necessary because there can only be one element). The format for CDF_EPOCH values is described in Section 2.5.4.

The vAttribute rEntries are optional. If omitted, the terminating period is still required. The rVariable values are also optional.

Several sample rVariable definitions for a CDF with 2-dimensional rVariables follow:

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

  "Latitude"       CDF_REAL4       1          F          F T

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

    "VALIDMIN"     CDF_REAL4    { -90.0 }
    "VALIDMAX"     CDF_REAL4    { 90.0 }
    "scale"        CDF_REAL4    { -60.0, 60.0 }.

    [1,1] = -60.0
    [1,2] = -30.0
    [1,3] = 0.0
    [1,4] = 30.0
    [1,5] = 60.0


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

  "EPOCH"          CDF_EPOCH       1          T          F F

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

    "scale"        CDF_EPOCH    { 10-Oct-1991 00:00:00.000,
                                  20-Oct-1991 23:59:59.999 }.


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

  'Tmp'            CDF_INT2        1          T          T T

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

    'Fieldname'    CDF_CHAR     { "Temperature (C)" }.
```

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

  "Pres_lvl"       CDF_REAL4      1          T         F F

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

    .   ! no attribute entries

    1:[1,1] = 1013.1
    2:[1,1] = 1015.0
    3:[1,1] = 1012.3
```

A sample variable definition for a CDF with 0-dimensional rVariables follows:

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

  "Latitude"       CDF_REAL4      1          F

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

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

    [] = -12.3
```

A.6 zVariables Section

The optional zVariables section contains the definition of each zVariable in the CDF, the values for any vAttribute zEntries associated with each zVariable, and (optionally) data values for those zVariables. The format of the zVariables section is as follows:

```
#zVariables

[<variable-definition>
 <variable-definition>
 <variable-definition>
           .
```

```
          .
          .
 <variable-definition>]
```

Where <variable-definition> is a zVariable definition. The format of each zVariable definition is as follows:

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

  <var-name> <var-data-type> <n-elems> <dims> <sizes> <rec-vary> <dim-varys>

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

     [<attr-name>  <entry-data-type>    <entry-value>
      <attr-name>  <entry-data-type>    <entry-value>
      <attr-name>  <entry-data-type>    <entry-value>
           .               .                  .
           .               .                  .
           .               .                  .
      <attr-name>  <entry-data-type>    <entry-value>].         ! Note the "."

   [[<rec-num>:]<indices> = <value>
    [<rec-num>:]<indices> = <value>
    [<rec-num>:]<indices> = <value>
        .            .
        .            .
        .            .
    [<rec-num>:]<indices> = <value>]
```

Each field is defined as follows:

`<var-name>`	The name of the zVariable. The name must be delimited with a character not appearing in the name itself (e.g., `"EPOCH"` or `'Temperature'`). The delimiting characters are not part of the zVariable name in the CDF.
`<var-data-type>`	The data type for the zVariable. The data type must be one of the following: `CDF_BYTE`, `CDF_INT1`, `CDF_UINT1`, `CDF_INT2`, `CDF_UINT2`, `CDF_INT4`, `CDF_UINT4`, `CDF_REAL4`, `CDF_FLOAT`, `CDF_REAL8`, `CDF_DOUBLE`, `CDF_EPOCH`, `CDF_CHAR`, or `CDF_UCHAR`.
`<n-elems>`	The number of elements of the data type. For character data types (`CDF_CHAR` and `CDF_UCHAR`) this is the number of characters in each string. For non-character data types this value must be one (1).
`<dims>`	The number of dimensions for the zVariable.

<sizes> The dimension sizes — one value per dimension. If the zVariable has zero (0) dimensions, this field would be left blank.

<rec-vary> The record variance of the zVariable. This must be either T (the values vary from record to record) or F (the values do not vary from record to record).

<dim-varys> The dimension variances of the zVariable. For each dimension there must be either a T (the values vary along that dimension) or F (the values do not vary along that dimension). Each dimension variance must be separated by at least one space. If the zVariable has zero dimensions, this field would be left blank.

<attr-name> The name of the vAttribute for which to specify a zEntry for this zVariable. The vAttribute must have been specified in the vAttributes section. The name must be delimited with a character not appearing in the name itself (e.g., "SCALEMAX" or 'range'). The delimiting characters are not part of the vAttribute name in the CDF.

<entry-data-type> The data type for the vAttribute zEntry. The data type must be one of the following: CDF_BYTE, CDF_INT1, CDF_UINT1, CDF_INT2, CDF_UINT2, CDF_INT4, CDF_UINT4, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, CDF_CHAR, or CDF_UCHAR.

<entry-value> The value(s) for the vAttribute zEntry. The format of attribute entry values is described in Section A.3.

NOTE: The last zEntry MUST be followed by a period (.). If no zEntries are specified for a zVariable, the period must still be present.

<rec-num> The record number of an zVariable value. This will be present only for record-variant (RV) zVariables.

<indices> The indices of an zVariable value. The indices are enclosed in brackets and separated by commas (e.g., [23,1] or [1,80]). If the zVariable has zero dimensions, [] would be specified (the brackets are still required).

<value> The value at the given record/indices. For character data types (CDF_CHAR or CDF_UCHAR) the string must be delimited with a unique character and enclosed in braces ({...}) in the same manner as for an attribute entry for a character data type. For non-character data types the value is not enclosed in braces (the braces are not necessary because there can only be one element). The format for CDF_EPOCH values is described in Section 2.5.4.

The vAttribute zEntries are optional. If omitted, the terminating period is still required. The zVariables values are also optional.

Several sample zVariable definitions follow:

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

  "Instrument"    CDF_CHAR      10       0              F
```

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

    "FIELDNAM"    CDF_CHAR     { "Measuring instrument" }.

  [] = { "Gonkulator" }


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

  "Ticks"         CDF_BYTE       1       1     3        T         T

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

  .    ! no attribute entries

  1:[1] = 1
  1:[2] = 2
  1:[3] = 3
  2:[1] = 3
  2:[2] = 2
  2:[3] = 1


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

  "WIND VELOCITY"  CDF_REAL4      1       3   360 180 10      T       T T T

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

    "FIELDNAM"    CDF_CHAR     { "Wind velocity." }
    "VALIDMIN"    CDF_REAL4    { 0.0 }
    "VALIDMAX"    CDF_REAL4    { 300.0 }
    "UNITS   "    CDF_CHAR     { "Knots" }
    "FORMAT  "    CDF_CHAR     { "F9.1    " }.
```

A.7 End Section

This section simply consists of the keyword `#end`. This section is required.

A.8 Example Skeleton Table

An example skeleton table containing rVariables and zVariables follows:

```
! Skeleton table for the "example2" CDF.
! Generated: Thursday, 17-Nov-1994 14:07:58
! CDF created/modified by CDF V2.4.10
! Skeleton table created by CDF V2.5.0


#header

                  CDF NAME: example2
             DATA ENCODING: NETWORK
                  MAJORITY: ROW
                    FORMAT: SINGLE

! Variables G.Attributes V.Attributes Records  Dims  Sizes
! --------- ------------ ------------ -------  ----  -----
      4           1            7         1       2   11 7


#GLOBALattributes

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

  "TITLE"               1:    CDF_CHAR     { "Title for example2 CDF." } .


#VARIABLEattributes

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


#variables

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

  "EPOCH"           CDF_EPOCH      1          T          F F
```

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

    "FIELDNAM"      CDF_CHAR     { "Time since 0 A.D.   " }
    "VALIDMIN"      CDF_EPOCH    { 01-Jan-0000 00:00:00.000 }
    "VALIDMAX"      CDF_EPOCH    { 01-Jan-2089 00:00:00.000 }
    "SCALEMIN"      CDF_EPOCH    { 01-Apr-1986 07:00:00.000 }
    "SCALEMAX"      CDF_EPOCH    { 01-Apr-1986 23:00:00.000 }
    "UNITS"         CDF_CHAR     { "milliseconds (UT)   " }
    "FORMAT"        CDF_CHAR     { "E14.0    " } .

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

  "LONGITUD"        CDF_REAL4      1          F         T F

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

    "FIELDNAM"      CDF_CHAR     { "Longitude variable  " }
    "VALIDMIN"      CDF_REAL4    { 0.0 }
    "VALIDMAX"      CDF_REAL4    { 180.0 }
    "SCALEMIN"      CDF_REAL4    { -50.0 }
    "SCALEMAX"      CDF_REAL4    { 50.0 }
    "UNITS"         CDF_CHAR     { "Degrees             " }
    "FORMAT"        CDF_CHAR     { "F8.3    " } .

    [1,1] = -50.0
    [2,1] = -40.0
    [3,1] = -30.0
    [4,1] = -20.0
    [5,1] = -10.0
    [6,1] = 0.0
    [7,1] = 10.0
    [8,1] = 20.0
    [9,1] = 30.0
    [10,1] = 40.0
    [11,1] = 50.0

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

  "LATITUDE"        CDF_REAL4      1          F         F T

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

```
    "FIELDNAM"      CDF_CHAR     { "Latitude variable   " }
    "VALIDMIN"      CDF_REAL4    { 0.0 }
    "VALIDMAX"      CDF_REAL4    { 90.0 }
    "SCALEMIN"      CDF_REAL4    { -30.0 }
    "SCALEMAX"      CDF_REAL4    { 30.0 }
    "UNITS"         CDF_CHAR     { "Degrees             " }
    "FORMAT"        CDF_CHAR     { "F8.3    " } .

    [1,1] = -30.0
    [1,2] = -20.0
    [1,3] = -10.0
    [1,4] = 0.0
    [1,5] = 10.0
    [1,6] = 20.0
    [1,7] = 30.0

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

  "TEMPERATURE"     CDF_INT4        1          T         T T

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

    "FIELDNAM"      CDF_CHAR     { "Temperature         " }
    "VALIDMIN"      CDF_INT4     { 0 }
    "VALIDMAX"      CDF_INT4     { 50 }
    "SCALEMIN"      CDF_INT4     { 0 }
    "SCALEMAX"      CDF_INT4     { 10 }
    "UNITS"         CDF_CHAR     { "Deg C               " }
    "FORMAT"        CDF_CHAR     { "I2      " } .


#zVariables

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

  "BIAS"          CDF_INT4      1       0              T

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

    "FIELDNAM"    CDF_CHAR     { "Correction bias for temperature" }
    "VALIDMIN"    CDF_INT4     { -5 }
    "VALIDMAX"    CDF_INT4     { 5 }
```

```
    "UNITS   "    CDF_CHAR     { "deg C               " }
    "FORMAT  "    CDF_CHAR     { "I2      " } .

    1:[] = 34
    2:[] = 28
    3:[] = 17


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

  "Coefficients"  CDF_REAL4      1       1     3        F         T

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

    "FIELDNAM"    CDF_CHAR     { "Temperature model coefficients." }
    "FORMAT  "    CDF_CHAR     { "F9.1    " } .

  [1] = -0.0254
  [2] = 14.2338
  [3] = -9.9444


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

  "TMP-model"     CDF_REAL4      1       2  360 180     T        T T

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

    "FIELDNAM"    CDF_CHAR     { "Temperature model." }
    "VALIDMIN"    CDF_REAL4    { -20.0 }
    "VALIDMAX"    CDF_REAL4    { 50.0 }
    "SCALEMIN"    CDF_REAL4    { 0.0 }
    "SCALEMAX"    CDF_REAL4    { 30.0 }
    "UNITS   "    CDF_CHAR     { "deg C               " }
    "FORMAT  "    CDF_CHAR     { "F9.6    " } .


#end
```

Appendix B

IDL Support

B.1 Introduction

The CDF distribution includes a set of IDL functions/procedures that allow access to the CDF library (hereafter referred to as the CDF/IDL interface). The functions/procedures are used in the same manner as the Standard and Internal Interfaces are used in programs written in C.[1] CDFs can be both read and written/modified.

Research Systems, Inc. (the developers of IDL) have also implemented an interface to CDF that is part of the IDL product. It differs from the interface provided with the CDF distribution in that it is intended more for the non-programmer (and is functionally similar to other interfaces they provide). Because IDL's CDF interface relies on a version of the CDF library being linked into the IDL executable, IDL's interface may lag behind the most recent CDF distribution. This can cause problems as new features are added to the CDF library (which IDL's interface will not know about).

CDF's interface relies on IDL's support of dynamic linking. IDL in turn relies on the operating system being used to support dynamic linking in a reasonable way. Currently, only the VAX (OpenVMS), DEC Alpha (OpenVMS & OSF/1), Sun (SunOS & SOLARIS), HP (HP-UX), SGi (IRIX 5.x & 6.x), and IBM RS6000 (AIX) support dynamic linking in a way that makes it possible to use CDF's interface in 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 (assuming CDF is also supported on that machine).

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-type>` on UNIX systems where `<shell-type>` is the type of shell being used: `C` for the C-shell (`csh` and `tcsh`), `K` for the

[1] In previous CDF releases, when using the Internal Interface it was required that the current CDF be selected in each call to `CDFlib`. This is no longer necessary. The current CDF is now maintained from one call to the next of `CDFlib` (as is the current status code).

Korn (`ksh`), BASH, and POSIX shells, and `B` for the Bourne shell (`sh`). The "definitions" file defines the logical name `IDL_STARTUP` on VMS systems and the environment variable `IDL_STARTUP` on UNIX systems to the full directory path of an IDL startup file provided with the CDF distribution. When IDL is started, this startup file is automatically executed (because `IDL_STARTUP` is defined). The startup file performs the following functions...

1. Makes known to IDL the functions/procedures that are used to call the CDF library (and perform other utility operations).
2. Sets IDL's internal paths. These are used when searching for a batch file to be executed or when locating online help files.
3. Executes one or more batch files that creates a number of local variables necessary for calling CDF functions/procedures and then interpreting the results. Check with your system administrator to see which batch files are being executed.

A list of messages should be displayed indicating the CDF functions/procedures that have been compiled.

If you are already using your own startup file (pointed to by your definition of `IDL_STARTUP`), you should remove the definition of `IDL_STARTUP` from the CDF "definitions" file being used and do one of the following...

1. Add the contents of the CDF/IDL startup file to your own startup file.
2. Have your startup file execute the CDF/IDL startup file.

The definition of `IDL_STARTUP` in the CDF "definitions" file being used points to the location of the CDF/IDL startup file.

B.3 CDF "Include" Files

A number of parameters (macro constants) are necessary when calling functions in the CDF library from a C program. These parameters are defined in the `cdf.h` header file. In a Fortran application the `cdf.inc` include file defines these parameters. There are three available methods for making these parameters known when calling CDF functions/procedures in IDL: constant structures, mapping functions, and individual local variables.

B.3.1 Constant Structures

A batch file,[2] `cdf0x.pro`, is available that creates a set of IDL structure variables that contain the CDF parameter values. Each structure variable contains one or more tags (fields) which are initialized to the proper value. The following structure variables are created by `cdf0x.pro`.[3]

[2] Batch files are also known as include files.

[3] The batch file `cdf0.pro` is also available that serves the same purpose. The only difference is that its structure variable names are longer.

Structure Variable	Contents
CDFx	General CDF constants
CDFdt	Data type constants
CDFen	Encoding constants
CDFde	Decoding constants
CDFic	Informational status code constants
CDFwc	Warning status code constants
CDFec	Error status code constants
CDFiif	Internal Interface function constants
CDFiix	General Internal Interface item constants
CDFiia	Internal Interface attribute item constants
CDFiie	Internal Interface entry item constants
CDFiir	Internal Interface rVariable item constants
CDFiiz	Internal Interface zVariable item constants

To display the contents of a particular structure variable, use the IDL `help` command with the `/structures` keyword. For example...

```
IDL> @cdf0x
IDL> help, /structures, cdfx
** Structure <40023808>, 52 tags, length=208, refs=1:
   CDF_MIN_DIMS    LONG                0
   CDF_MAX_DIMS    LONG               10
         .
         .
         .
IDL>
```

The following example will show how the structure variables created by `cdf0x.pro` would be used.

```
IDL> @cdf0x
IDL> dimSizes = lonarr(CDFx.CDF_MAX_DIMS)
IDL> status = CDFcreate ('test1', 0L, dimSizes, CDFen.NETWORK_ENCODING, $
IDL>                     CDFx.ROW_MAJOR, id)
IDL> if (status lt CDFic.CDF_OK) ...
IDL> status = CDFattrCreate (id, 'TITLE', CDFx.GLOBAL_SCOPE, attrNum)
IDL> if (status lt CDFic.CDF_OK) ...
IDL> status = CDFattrPut (id, attrNum, 0L, CDFdt.CDF_CHAR, 6L, 'Test 1')
IDL> if (status lt CDFic.CDF_OK) ...
IDL> status = CDFclose (id)
IDL> if (status lt CDFic.CDF_OK) ...
```

B.3.2 Mapping Functions

Three mapping functions are available that take as an argument a string containing the name of a parameter and return the numerical value for that parameter as an IDL longword. These mapping function are made known to IDL in the CDF/IDL startup file.

Mapping Function	Purpose
MCP	Map general CDF parameters
MSC	Map status code constants
MII	Map Internal Interface constants

For example, the status code value for `NO_SUCH_CDF` would be retrieved as follows...

```
IDL> status = CDFopen ('test1', id)
IDL> if (status eq MSC('NO_SUCH_CDF')) print, "CDF not found..."
```

The use of these mapping functions will add execution overhead because of the number of string comparisons that must be performed by the functions to locate the proper parameter value.

B.3.3 Individual Local Variables

Three batch files are available that define a local variable for each CDF parameter that might be needed. Unfortunately, IDL limits the number of local variables that may be defined in any one function/procedure. This limit will almost certainly be reached if you have to include one or more of these batch files. For this reason one of the two previously described methods should be used to define the necessary CDF parameters.

NOTE: These batch files are maintained in the CDF distribution only in the event that IDL might raise the limit on local variables.

B.4 On-Line Help

IDL provides an on-line help facility to which help for user written procedures/functions can be added.[4] A specially formatted file is provided with the CDF distribution that contains on-line help for the procedures/functions in the CDF/IDL interface. This file is named `cdf26cdfif.help` and is located in the "help" directory of the CDF distribution. If you are using the appropriate "definitions" file, IDL will be able to locate this file when online help is requested. Choose the topic called `CDF26CDFIF` rather than `CDF`. (`CDF` provides online help for IDL's CDF interface — not the interface provided with the CDF distribution.)

B.5 Available Functions

The CDF functions/procedures available at the IDL command line exactly mirror the Standard and Internal Interface functions available for a program written in C.[5] The CDF C Reference Manual describes the functionality of each in detail. The syntax required when calling each function is as follows...

[4] At least this used to be the case. IDL seems to be using a new online help facility that does not recognize user supplied help files.

[5] With the exception of the multiple variable access operations available via the Internal Interface (`<GET_,rVARs_RECDATA_>`, `<PUT_,rVARs_RECDATA_>`, `<GET_,zVARs_RECDATA_>`, and `<PUT_,zVARs_RECDATA_>`). Because these operations deal with variables having different data types, the use of structures to store the values would be the logical method to use. Unfortunately, IDL does not currently support the manipulation of structures from within a C application (which is how CDF's IDL interface is implemented). Support of these operations will hopefully be added in the future.

```
IDL> status = CDFlib (fnc1, ...)
IDL> status = CDFcreate (CDFpath, numDims, dimSizes, encoding, majority, $
IDL>                     id)
IDL> status = CDFopen (CDFpath, id)
IDL> status = CDFdoc (id, version, release, copyright)
IDL> status = CDFinquire (id, numDims, dimSizes, encoding, majority, $
IDL>                      maxRec, numVars, numAttrs)
IDL> status = CDFclose (id)
IDL> status = CDFdelete (id)
IDL> status = CDFerror (status, text)
IDL> status = CDFattrCreate (id, attrName, attrScope, attrNum)
IDL> status = CDFattrRename (id, attrNum, attrName)
IDL> status = CDFattrInquire (id, attrNum, attrName, attrScope, maxEntry)
IDL> status = CDFattrEntryInquire (id, attrNum, entryNum, dataType, $
IDL>                               numElems)
IDL> status = CDFattrPut (id, attrNum, entryNum, dataType, numElems, value)
IDL> status = CDFattrGet (id, attrNum, entryNum, value)
IDL> status = CDFvarCreate (id, varName, dataType, numElems, recVary, $
IDL>                        dimVarys, varNum)
IDL> status = CDFvarRename (id, varNum, varName)
IDL> status = CDFvarInquire (id, varNum, varName, dataType, numElems, $
IDL>                         recVary, dimVarys)
IDL> status = CDFvarPut (id, varNum, recNum, indices, value)
IDL> status = CDFvarGet (id, varNum, recNum, indices, value)
IDL> status = CDFvarHyperPut (id, varNum, recStart, recCount, recInterval, $
IDL>                          indices, counts, intervals, buffer)
IDL> status = CDFvarHyperGet (id, varNum, recStart, recCount, recInterval, $
IDL>                          indices, counts, intervals, buffer)
IDL> status = CDFvarClose (id, varNum)
IDL> attrNum = CDFattrNum (id, attrName)
IDL> varNum = CDFvarNum (id, varName)
IDL> epoch = computeEPOCH (year, month, day, hour, minute, second, $
IDL>                       millisecond)
IDL> EPOCHbreakdown, epoch, year, month, day, hour, minute, second, $
IDL>                 millisecond
IDL> epoch = parseEPOCH (epochString)
IDL> epoch = parseEPOCH1 (epochString)
IDL> epoch = parseEPOCH2 (epochString)
IDL> epoch = parseEPOCH3 (epochString)
IDL> encodeEPOCH, epoch, epochString
IDL> encodeEPOCH1, epoch, epochString
IDL> encodeEPOCH2, epoch, epochString
IDL> encodeEPOCH3, epoch, epochString
IDL> encodeEPOCHx, epoch, format, epochString
```

The IDL data types to use for integer arguments are always longwords or arrays of longwords using IDL's data type of `TYP_LONG`.[6] Attribute entries and variable values should have the data types corresponding their CDF data types. CDF pathnames and variable/attribute names are IDL character strings. `CDFid`

[6]On DEC Alpha's running OSF/1, don't confuse the C data type of `long` with IDL's data type of `TYP_LONG`. The C `long` is eight bytes while IDL's `TYP_LONG` is four bytes.

and `CDFstatus` are also longwords. The CDF C Reference Manual may be used as a guide (especially for `CDFlib`). Arguments passed to the CDF functions/procedures must obviously be created and initialized before the call. Arrays of longwords are necessary for dimension sizes, variances, etc.[7] Arguments returned by the CDF functions will be created to be of the proper data type and size (they do not have to be created before calling the CDF function).

Also provided are two functions, `row_to_col` and `col_to_row`, that may be used to change the majority of an array (or arrays) of values. This function is necessary because CDF can store variable values in either row or column major order. The syntax of `row_to_col` is as follows...

```
IDL> status = row_to_col (inArray, outArray, numDims, dimSizes, numBytes, $
                          arrayCount
```

where...

`inArray`	The array (or arrays) of values to convert — row major ordering. Any valid IDL data type is allowed.
`outArray`	The converted array (or arrays) of values — column major ordering. This array will be created by `row_to_col`.
`numDims`	The number of dimensions in the array (or arrays). This value must be a longword.
`dimSizes`	The size of each dimension. This must be an array of longwords — one longword per dimension.
`numBytes`	The number of bytes in each value. This value must be a longword.
`arrayCount`	The number of arrays to convert. The order of the arrays will not be affected. This value must be a longword.
`status`	Completion status code. This value will be a longword. The possible values are defined in `cdf1.pro`.

Multiple arrays (records) of CDF variable values should always be stored with the last dimension of the IDL array corresponding to the record number. For example, assume a CDF variable with two dimensions whose sizes are `[180,360]`. If `50` records are to be stored, an IDL array with three dimensions whose sizes are `[180,360,50]` should be used. If the CDF variables have row major ordering, `row_to_col` would be called with `numDims` set to 2, `dimSizes` set to `[180,360]`, and `arrayCount` set to `50`.

The syntax of `col_to_row` is as follows...

```
IDL> status = col_to_row (inArray, outArray, numDims, dimSizes, numBytes, $
                          arrayCount
```

where...

[7] If an array in a call to a CDF function won't be used because a variable is 0-dimensional, a dummy array must still be passed to avoid an error from the CDF library. Note that embedding a call to `lonarr` [such as `lonarr(1)`] in the call to the CDF function will work but will result in a memory leak because the allocated array will not be freed.

inArray	The array (or arrays) of values to convert — column major ordering. Any valid IDL data type is allowed.
outArray	The converted array (or arrays) of values — row major ordering. This array will be created by col_to_row.
numDims	The number of dimensions in the array (or arrays). This value must be a longword.
dimSizes	The size of each dimension. This must be an array of longwords — one longword per dimension.
numBytes	The number of bytes in each value. This value must be a longword.
arrayCount	The number of arrays to convert. The order of the arrays will not be affected. This value must be a longword.
status	Completion status code. This value will be a longword. The possible values are defined in cdf1.pro.

col_to_row would be used in those cases where an array of values created by IDL (in column-major ordering) are to be written to a row-major CDF.

B.6 Example IDL Session

The following IDL session will open a CDF (unsuccessfully at first — typo), add an entry to the MODS gAttribute, and then close the CDF.

```
IDL> @cdf0x
IDL> status = CDFlib (CDFiif.OPEN_, CDFiix.CDF_, 'yrdy3', id, $
IDL>                  CDFiif.NULL_)
IDL> status = CDFerror (status, text)
IDL> print, text
NO_SUCH_CDF: The specified CDF does not exist.
IDL> status = CDFlib (CDFiif.OPEN_, CDFiix.CDF_, 'test2', id, $
IDL>                  CDFiif.NULL_)
IDL> status = CDFerror (status, text)
IDL> print, text
CDF_OK: Function completed successfully.
IDL> status = CDFlib (CDFiif.SELECT_, CDFiia.ATTR_NAME_, 'MODS', $
IDL>                  CDFiif.GET_, CDFiia.ATTR_MAXgENTRY_, maxEntry, $
IDL>                  CDFiif.NULL_)
IDL> status = CDFerror (status, text)
IDL> print, text
CDF_OK: Function completed successfully.
IDL> status = CDFlib (CDFiif.SELECT_, CDFiie.gENTRY_, maxEntry + 1, $
IDL>                  CDFiif.PUT_, CDFiie.gENTRY_DATA_, CDFdt.CDF_CHAR, 14L, $
IDL>                                                   'Useless update', $
IDL>                  CDFiif.NULL_)
IDL> status = CDFerror (status, text)
```

```
IDL> print, text
CDF_OK: Function completed successfully.
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
@cdf0x
if (status lt CDFic.CDF_OK) then begin
  statusT = CDFerror (status, text)
  print, text
endif
return
end
```

The first part of the sample IDL session could then be as follows...

```
IDL> .run checkstatus
IDL> @cdf0x
IDL> status = CDFlib (CDFiif.OPEN_, CDFiix.CDF_, 'yrdy3', id, $
IDL>                 CDFiif.NULL_)
IDL> checkstatus, status
NO_SUCH_CDF: The specified CDF does not exist.
IDL> checkstatus, CDFlib (CDFiif.OPEN_, CDFiix.CDF_, 'test2', id, $
IDL>                     CDFiif.NULL_)
.
.
.
```

As you can see, a call to a CDF function can be embedded in a call to `checkstatus`.

Appendix C

Status Codes

C.1 Introduction

A status code is returned from most CDF functions. The `cdf.h` (for C) and `CDF.INC` (for Fortran) include files contain the numerical values (constants) for each of the status codes (and for any other constants referred to in the explanations). The CDF library Standard Interface functions `CDFerror` (for C) and `CDF_error` (for Fortran) can be used within a program to inquire the explanation text for a given status code. The Internal Interface can also be used to inquire explanation text.

There are three classes of status codes: informational, warning, and error. The purpose of each is as follows:

Informational	Indicates success but provides some additional information that may be of interest to an application.
Warning	Indicates that the function completed but possibly not as expected.
Error	Indicates that a fatal error occurred and the function aborted.

Status codes fall into classes as follows:

Error codes < `CDF_WARN` < Warning codes < `CDF_OK` < Informational codes

`CDF_OK` indicates an unqualified success (it should be the most commonly returned status code). `CDF_WARN` is simply used to distinguish between warning and error status codes.

C.2 Status Codes and Messages

The following list contains an explanation for each possible status code. Whether a particular status code is considered informational, a warning, or an error is also indicated.

`ATTR_EXISTS`	Named attribute already exists — cannot create or rename.

	Each attribute in a CDF must have a unique name. Note that trailing blanks are ignored by the CDF library when comparing attribute names. [Error]
ATTR_NAME_TRUNC	Attribute name truncated to CDF_ATTR_NAME_LEN characters. The attribute was created but with a truncated name. [Warning]
BAD_ALLOCATE_RECS	An illegal number of records to allocate for a variable was specified. For RV variables the number must be one or greater. For NRV variables the number must be exactly one. [Error]
BAD_ARGUMENT	An illegal/undefined argument was passed. Check that all arguments are properly declared and initialized. [Error]
BAD_ATTR_NAME	Illegal attribute name specified. Attribute names must contain at least one character, and each character must be printable. [Error]
BAD_ATTR_NUM	Illegal attribute number specified. Attribute numbers must be zero (0) or greater for C applications and one (1) or greater for Fortran applications. [Error]
BAD_BLOCKING_FACTOR[1]	An illegal blocking factor was specified. Blocking factors must be at least zero (0). [Error]
BAD_CACHESIZE	An illegal number of cache buffers was specified. The value must be at least zero (0). [Error]
BAD_CDF_EXTENSION	An illegal file extension was specified for a CDF. In general, do not specify an extension except possibly for a single-file CDF which has been renamed with a different file extension or no file extension. [Error]
BAD_CDF_ID	CDF identifier is unknown or invalid. The CDF identifier specified is not for a currently open CDF. [Error]
BAD_CDF_NAME	Illegal CDF name specified. CDF names must contain at least one character, and each character must be printable. Trailing blanks are allowed but will be ignored. [Error]
BAD_CDFSTATUS	Unknown CDF status code received. The status code specified is not used by the CDF library. [Error]
BAD_COMPRESSION_PARM	An illegal compression parameter was specified. [Error]
BAD_DATA_TYPE	An unknown data type was specified or encountered. The CDF data types are defined in cdf.h for C applications and in cdf.inc for Fortran applications. [Error]
BAD_DECODING	An unknown decoding was specified. The CDF decodings are defined in cdf.h for C applications and in cdf.inc for Fortran applications. [Error]
BAD_DIM_COUNT	Illegal dimension count specified. A dimension count must be at least one (1) and not greater than the size of the dimension.

[1] The status code BAD_BLOCKING_FACTOR was previously named BAD_EXTEND_RECS.