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

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

4.15 Limits of Names and Other Character Strings

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

[2] The ASCII null character, 0 octal.

Chapter 5

Standard Interface

The following sections describe the Standard Interface routines callable from C applications. Most functions return a status code of type `CDFstatus` (see Chapter 7). The Internal Interface is described in Chapter 6. An application can use both interfaces when necessary. Note that zVariables and vAttribute zEntries are only accessible via the Internal Interface.

Each section begins with a function prototype for the routine being described. The include file `cdf.h` contains the same function prototypes (as well as function prototypes for the Internal Interface and EPOCH utility routines). Note that many of the Standard Interface functions are implemented as macros (which call the Internal Interface).

5.1 CDFcreate

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

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

The arguments to `CDFcreate` are defined as follows:

`CDFpath` The pathname of the CDF to create. (Do not specify an extension.) This may be at most `CDF_PATHNAME_LEN` characters (excluding the `NUL` terminator). A

CDF pathname may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on VMS systems and environment variables on UNIX systems).

UNIX: Pathnames are case-sensitive.

`numDims` Number of dimensions the rVariables in the CDF are to have. This may be as few as zero (0) and at most `CDF_MAX_DIMS`.

`dimSizes` The size of each dimension. `dimSizes` is a 1-dimensional array containing `numDims` elements. Each element of `dimSizes` specifies the corresponding dimension size. Each size must be greater then zero (0). If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

`encoding` The encoding for variable data and attribute entry data. Specify one of the encodings described in Section 4.6.

`majority` The majority for variable data. Specify one of the majorities described in Section 4.8.

`id` The identifier for the created CDF. This identifier must be used in all subsequent operations on the CDF.

When a CDF is created, both read and write access are allowed. The default format for a CDF created with `CDFcreate` is specified in the configuration file of your CDF distribution. Consult your system manager for this default. The `CDFlib` function (Internal Interface) may be used to change a CDF's format.

NOTE: `CDFclose` must be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk (see Section 5.5).

5.1.1 Example(s)

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

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

```
.
.
```

ROW_MAJOR and NETWORK_ENCODING are defined in cdf.h.

5.2 CDFopen

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

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

The arguments to CDFopen are defined as follows:

CDFpath — The pathname of the CDF to open. (Do not specify an extension.) This may be at most CDF_PATHNAME_LEN characters (excluding the NUL terminator). A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on VMS systems and environment variables on UNIX systems).

UNIX: Pathnames are case-sensitive.

id — The identifier for the opened CDF. This identifier must be used in all subsequent operations on the CDF.

NOTE: CDFclose must be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk (see Section 5.5).

5.2.1 Example(s)

The following example will open a CDF named NOAA1.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                          /* CDF identifier. */
CDFstatus   status;                      /* Returned status code. */
static char CDFpath[] = { "NOAA1" };     /* Pathname of CDF. */
.
```

```
.
status = CDFopen (CDFpath, &id);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

5.3 CDFdoc

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

`CDFdoc` is used to inquire general documentation about a CDF. The version/release of the CDF library that created the CDF is provided (e.g., CDF V2.4 is version 2, release 4) along with the CDF copyright notice.

The arguments to `CDFdoc` are defined as follows:

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

The copyright notice is formatted for printing without modification. The version and release are used together (e.g., CDF V2.4 is version 2, release 4).

5.3.1 Example(s)

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

```
.
.
#include "cdf.h"
.
.
CDFid     id;                          /* CDF identifier. */
CDFstatus status;                      /* Returned status code. */
long      version;                     /* CDF version number. */
```

```
long      release;                          /* CDF release number. */
char      copyRight[CDF_COPYRIGHT_LEN+1];  /* Copyright notice -- +1 for
                                               NUL terminator. */
.
.
status = CDFdoc (id, &version, &release, copyRight);

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

5.4 CDFinquire

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

`CDFinquire` inquires the basic characteristics of a CDF. An application needs to know the number of rVariable dimensions and their sizes before it can access rVariable data. Knowing the variable majority can be used to optimize performance and is necessary to properly use the variable hyper functions (for both rVariables and zVariables).

The arguments to `CDFinquire` are defined as follows:

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

`numDims` The number of dimensions for the rVariables in the CDF.

`dimSizes` The dimension sizes of the rVariables in the CDF. `dimSizes` is a 1-dimensional array containing one element per dimension. Each element of `dimSizes` receives the corresponding dimension size. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

`encoding` The encoding of the variable data and attribute entry data. The encodings are defined in Section 4.6.

`majority` The majority of the variable data. The majorities are defined in Section 4.8.

`maxRec` The maximum record number written to an rVariable in the CDF. Note that the maximum record number written is also kept separately for each rVariable in the CDF. The value of `maxRec` is the largest of these. Some rVariables may have fewer records actually written. `CDFlib` (Internal Interface) may be used to inquire the maximum record written for an individual rVariable (see Section 6).

`numVars` The number of rVariables in the CDF.

`numAttrs` The number of attributes in the CDF.

5.4.1 Example(s)

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

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

5.5 CDFclose

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

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

NOTE: You must close a CDF with `CDFclose` to guarantee that all modifications you have made will actually be written to the CDF's file(s). If your program exits, normally or otherwise, without a successful call to `CDFclose`, the CDF's cache buffers are left unflushed.

The arguments to `CDFclose` are defined as follows:

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

5.5.1 Example(s)

The following example will close an open CDF.

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

5.6 CDFdelete

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

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

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

The arguments to `CDFdelete` are defined as follows:

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

5.6.1 Example(s)

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

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

5.7 CDFerror

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

`CDFerror` is used to inquire the explanation of a given status code (not just error codes). Chapter 7 explains how to interpret status codes and Appendix A lists all of the possible status codes.

The arguments to `CDFerror` are defined as follows:

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

5.7.1 Example(s)

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

```
.
```

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

5.8 CDFattrCreate

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

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

The arguments to `CDFattrCreate` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrName`	The name of the attribute to create. This may be at most `CDF_ATTR_NAME_LEN` characters (excluding the `NUL` terminator). Attribute names are case-sensitive.
`attrScope`	The scope of the new attribute. Specify one of the scopes described in Section 4.10.
`attrNum`	The number assigned to the new attribute. This number must be used in subsequent CDF function calls when referring to this attribute. An existing attribute's number may be determined with the `CDFattrNum` function.

5.8.1 Example(s)

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

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

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

5.9 CDFattrNum

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

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

The arguments to `CDFattrNum` are defined as follows:

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

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

5.9.1 Example(s)

In the following example the attribute named `pressure` will be renamed `PRESSURE` with `CDFattrNum` being used as an embedded function call. Note that if the attribute `pressure` did not exist in the CDF, the call to `CDFattrNum` would have returned an error code. Passing that error code to `CDFattrRename` as an attribute number would have resulted in `CDFattrRename` also returning an error code. `CDFattrRename` is described in Section 5.10.

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

5.10 CDFattrRename

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

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

The arguments to `CDFattrRename` are defined as follows:

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

`attrNum` The number of the attribute to rename. This number may be determined with a call to `CDFattrNum` (see Section 5.9).

`attrName` The new attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters (excluding the `NUL` terminator). Attribute names are case-sensitive.

5.10.1 Example(s)

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

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

5.11 CDFattrInquire

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

`CDFattrInquire` is used to inquire about the specified attribute. To inquire about a specific attribute entry, use `CDFattrEntryInquire` (Section 5.12).

The arguments to `CDFattrInquire` are defined as follows:

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

`attrNum` The number of the attribute to inquire. This number may be determined with a call to `CDFattrNum` (see Section 5.9).

`attrName` The attribute's name. This character string must be large enough to hold `CDF_ATTR_NAME_LEN + 1` characters (including the `NUL` terminator).

`attrScope` The scope of the attribute. Attribute scopes are defined in Section 4.10.

`maxEntry` For gAttributes this is the gEntry number used. For vAttributes this is the maximum rEntry number used. In either case this may not correspond with the number of entries (if some entry numbers were not used). The number of entries actually used may be inquired with the `CDFlib` function (see Section 6). If no entries exist for the attribute, then a value of `-1` will be passed back.

5.11.1 Example(s)

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

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

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

5.12 CDFattrEntryInquire

```
CDFstatus CDFattrEntryInquire(
CDFid id,            /* in  -- CDF identifier. */
long  attrNum,       /* in  -- Attribute number. */
long  entryNum,      /* in  -- Entry number. */
```

```
long  *dataType,      /* out -- Data type. */
long  *numElements);  /* out -- Number of elements (of the data type). */
```

`CDFattrEntryInquire` is used to inquire about a specific attribute entry. To inquire about the attribute in general, use `CDFattrInquire` (see Section 5.11). `CDFattrEntryInquire` would normally be called before calling `CDFattrGet` in order to determine the data type and number of elements (of that data type) for an entry. This would be necessary to correctly allocate enough memory to receive the value read by `CDFattrGet`.

The arguments to `CDFattrEntryInquire` are defined as follows:

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

`attrNum` The attribute number for which to inquire an entry. This number may be determined with a call to `CDFattrNum` (see Section 5.9).

`entryNum` The entry number to inquire. If the attribute is global in scope, this is simply the entry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).

`dataType` The data type of the specified entry. The data types are defined in Section 4.5.

`numElements` The number of elements of the data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string (an array of characters). For all other data types this is the number of elements in an array of that data type.

5.12.1 Example(s)

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

```
.
.
#include "cdf.h"
.
.
CDFid     id;                             /* CDF identifier. */
CDFstatus status;                         /* Returned status code. */
long      attrN;                          /* Attribute number. */
long      entryN;                         /* Entry number. */
char      attrName[CDF_ATTR_NAME_LEN+1];  /* Attribute name, +1 for NUL
                                             terminator. */
long      attrScope;                      /* Attribute scope. */
long      maxEntry;                       /* Maximum entry number used. */
long      dataType;                       /* Data type. */
```

```
long      numElems;                        /* Number of elements (of the
                                              data type). */
.
.
attrN = CDFattrNum (id, "TMP");
if (attrN < 0) UserStatusHandler (attrN);  /* If less than zero (0), then
                                              it must be a warning/error
                                              code. */

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

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

5.13 CDFattrPut

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

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

The arguments to `CDFattrPut` are defined as follows:

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

attrNum The attribute number. This number may be determined with a call to `CDFattrNum` (see Section 5.9).

entryNum The entry number. If the attribute is global in scope, this is simply the entry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).

`dataType` — The data type of the specified entry. Specify one of the data types defined in Section 4.5.

`numElements` — The number of elements of the data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string (an array of characters). For all other data types this is the number of elements in an array of that data type.

`value` — The value(s) to write. The entry value is written to the CDF from memory address `value`.

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

5.13.1 Example(s)

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

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

```
numElements = 2;
status = CDFattrPut (id, CDFattrNum(id,"VALIDs"), CDFvarNum(id,"TMP"),
                     CDF_INT2, numElements, TMPvalids);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

5.14 CDFattrGet

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

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

The arguments to `CDFattrGet` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrNum`	The attribute number. This number may be determined with a call to `CDFattrNum` (see Section 5.9).
`entryNum`	The entry number. If the attribute is global in scope, this is simply the entry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).
`value`	The value read. This buffer must be large enough to hold the value. The function `CDFattrEntryInquire` would be used to determine the entry data type and number of elements (of that data type). The value is read from the CDF and placed into memory at address `value`.

5.14.1 Example(s)

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

```
.
.
```

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

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

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

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

  status = CDFattrGet (id, attrN, entryN, buffer);
  if (status != CDF_OK) UserStatusHandler (status);

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

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

  free (buffer);
}
.
.
```

5.15 CDFvarCreate

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

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

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

The arguments to `CDFvarCreate` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`varName`	The name of the rVariable to create. This may be at most `CDF_VAR_NAME_LEN` characters (excluding the `NUL` terminator). Variable names are case-sensitive.
`dataType`	The data type of the new rVariable. Specify one of the data types defined in Section 4.5.
`numElements`	The number of elements of the data type at each value. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string (each value consists of the entire string). For all other data types this must always be one (1) — multiple elements at each value are not allowed for non-character string data types.
`recVariance`	The rVariable's record variance. Specify one of the variances defined in Section 4.9.
`dimVariances`	The rVariable's dimension variances. `dimVariances` is a 1-dimensional array containing one element per rVariable dimension. Each element of `dimVariances` specifies the corresponding dimension variance. For each dimension specify one of the variances defined in Section 4.9. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
`varNum`	The number assigned to the new rVariable. This number must be used in subsequent CDF function calls when referring to this rVariable. An existing rVariables's number may be determined with the `CDFvarNum` function.

5.15.1 Example(s)

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

```
.
.
#include "cdf.h"
.
.
CDFid     id;                                /* CDF identifier. */
CDFstatus status;                            /* Returned status code. */
```

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

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

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

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

5.16 CDFvarNum

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

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

The arguments to `CDFvarNum` are defined as follows:

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

`varName`	The name of the rVariable for which to search. This may be at most `CDF_VAR_NAME_LEN` characters (excluding the `NUL` terminator). Variable names are case-sensitive.

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

5.16.1 Example(s)

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

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

In this example the rVariable named `LATITUDE` was inquired. Note that if `LATITUDE` did not exist in the CDF, the call to `CDFvarNum` would have returned an error code. Passing that error code to `CDFvarInquire` as an rVariable number would have resulted in `CDFvarInquire` also returning an error code. Also note that the name written into `varName` is already known (`LATITUDE`). In some cases the rVariable names will be unknown — `CDFvarInquire` would be used to determine them. `CDFvarInquire` is described in Section 5.18.

5.17 CDFvarRename

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

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

The arguments to CDFvarRename are defined as follows:

- **id** — The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate or CDFopen.
- **varNum** — The number of the rVariable to rename. This number may be determined with a call to CDFvarNum (see Section 5.16).
- **varName** — The new rVariable name. This may be at most CDF_VAR_NAME_LEN characters (excluding the NUL terminator). Variable names are case-sensitive.

5.17.1 Example(s)

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

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

5.18 CDFvarInquire

```
CDFstatus CDFvarInquire(
CDFid id,                            /* in  -- CDF identifier. */
long  varNum,                        /* in  -- rVariable number. */
char  varName,                       /* out -- rVariable name. */
long  *dataType,                     /* out -- Data type. */
long  *numElements,                  /* out -- Number of elements (of the
                                             data type). */
```

```
long  *recVariance,                   /* out -- Record variance. */
long  dimVariances[CDF_MAX_DIMS]);    /* out -- Dimension variances. */
```

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

The arguments to CDFvarInquire are defined as follows:

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

varNum
: The number of the rVariable to inquire. This number may be determined with a call to CDFvarNum (see Section 5.16).

varName
: The rVariable's name. This character string must be large enough to hold CDF_VAR_NAME_LEN + 1 characters (including the NUL terminator).

dataType
: The data type of the rVariable. The data types are defined in Section 4.5.

numElements
: The number of elements of the data type at each rVariable value. For character string data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. (Each value consists of the entire string.) For all other data types, this will always be one (1) — multiple elements at each value are not allowed for non-character string data types.

recVariance
: The record variance. The record variances are defined in Section 4.9.

dimVariances
: The dimension variances. dimVariances is a 1-dimensional array containing one element per rVariable dimension in the CDF. Each element of dimVariances receives the corresponding dimension variance. The dimension variances are defined in Section 4.9. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

5.18.1 Example(s)

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

```
.
.
#include "cdf.h"
.
.
CDFid     id;                          /* CDF identifier. */
CDFstatus status;                      /* Returned status code. */
char      varName[CDF_VAR_NAME_LEN+1]; /* rVariable name, +1
                                          for NUL terminator. */
long      dataType;                    /* Data type of the rVariable. */
long      numElems;                    /* Number of elements (of data
```

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

5.19 CDFvarPut

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

`CDFvarPut` is used to write a single value to an rVariable. `CDFvarHyperPut` may be used to write more than one rVariable value with a single call (see Section 5.21).

The arguments to `CDFvarPut` are defined as follows:

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

`varNum` The number of the rVariable to which to write. This number may be determined with a call to `CDFvarNum` (see Section 5.16).

`recNum` The record number at which to write.

`indices` The array indices within the specified record at which to write. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` specifies the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).

`value` The value to write. The value is written to the CDF from memory address `value`.

5.19.1 Example(s)

The following example writes values to the rVariable named `LATITUDE` in a CDF whose rVariables are 2-dimensional with dimension sizes `[360,181]`. For `LATITUDE` the record variance is `NOVARY`, the dimension variances are `[NOVARY,VARY]`, and the data type is `CDF_INT2`.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                        /* CDF identifier. */
CDFstatus   status;                    /* Returned status code. */
short       lat;                       /* Latitude value. */
long        varN;                      /* rVariable number. */
static long recNum = 0;                /* Record number. */
static long indices[2] = {0,0};        /* Dimension indices. */
.
.
varN = CDFvarNum (id, "LATITUDE");
if (varN < 0) UserStatusHandler (varN);  /* If less than zero (0), not a
                                            rVariable number but
                                            rather a warning/error code. */

for (lat = -90; lat <= 90; lat += 1) {
   indices[1] = 90 + lat;
   status = CDFvarPut (id, varN, recNum, indices, &lat);
   if (status != CDF_OK) UserStatusHandler (status);
}
.
.
```

Since the record variance is `NOVARY`, the record number (`recNum`) is set to zero (0). Also note that because the dimension variances are `[NOVARY,VARY]`, only the second dimension is varied as values are written. (The values are "virtually" the same at each index of the first dimension.)

5.20 CDFvarGet

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

`CDFvarGet` is used to read a single value from an rVariable. `CDFvarHyperGet` may be used to read more than one rVariable value with a single call (see Section 5.22).

The arguments to `CDFvarGet` are defined as follows:

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

varNum The number of the rVariable from which to read. This number may be determined with a call to `CDFvarNum` (see Section 5.16).

recNum The record number at which to read.

indices The array indices within the specified record at which to read. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` specifies the corresponding dimension index. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

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

5.20.1 Example(s)

The following example will read and hold an entire record of data from an rVariable. The CDF's rVariables are 3-dimensional with sizes `[180,91,10]`. For this rVariable the record variance is `VARY`, the dimension variances are `[VARY,VARY,VARY]`, and the data type is `CDF_REAL4`.

```
.
.
#include "cdf.h"
.
.
CDFid     id;                /* CDF identifier. */
CDFstatus status;            /* Returned status code. */
float     tmp[180][91][10];  /* Temperature values. */
long      indices[3];        /* Dimension indices. */
long      varN;              /* rVariable number. */
long      recNum;            /* Record number. */
long      d0, d1, d2;        /* Dimension index values. */
.
.
varN = CDFvarNum (id, "Temperature");
if (varN < 0) UserStatusHandler (varN);  /* If less than zero (0), then
                                            it is actually a warning/error
                                            code. */

recNum = 13;

for (d0 = 0; d0 < 180; d0++) {
   indices[0] = d0;
   for (d1 = 0; d1 < 91; d1++) {
      indices[1] = d1;
      for (d2 = 0; d2 < 10; d2++) {
         indices[2] = d2;
         status = CDFvarGet (id, varN, recNum, indices, &tmp[d0][d1][d2]);
         if (status != CDF_OK) UserStatusHandler (status);
```

```
        }
    }
}
.
.
```

5.21 CDFvarHyperPut

```
CDFstatus CDFvarHyperPut(
CDFid id,            /* in -- CDF identifier. */
long  varNum,        /* in -- rVariable number. */
long  recStart,      /* in -- Starting record number. */
long  recCount,      /* in -- Number of records. */
long  recInterval,   /* in -- Interval between records. */
long  indices[],     /* in -- Dimension indices of starting value. */
long  counts[],      /* in -- Number of values along each dimension. */
long  intervals[],   /* in -- Interval between values along each dimension. */
void  *buffer);      /* in -- Buffer of values. */
```

`CDFvarHyperPut` is used to write a buffer of one or more values to an rVariable. It is important to know the variable majority of the CDF before using `CDFvarHyperPut` because the values in the buffer to be written must be in the same majority. `CDFinquire` is used to determine the variable majority of a CDF. The Concepts chapter in the CDF User's Guide describes the variable majorities.

The arguments to `CDFvarHyperPut` are defined as follows:

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

`varNum` — The number of the rVariable to which to write. This number may be determined with a call to `CDFvarNum` (see Section 5.16).

`recStart` — The record number at which to start writing.

`recCount` — The number of records to write.

`recInterval` — The interval between records for subsampling[1] (e.g., an interval of 2 means write to every other record).

`indices` — The indices (within each record) at which to start writing. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` specifies the corresponding dimension index. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

`counts` — The number of values along each dimension to write. `counts` is a 1-dimensional array containing one element per rVariable dimension. Each element of `count` specifies the corresponding dimension count. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

[1] "Subsampling" is not the best term to use when writing data, but you should know what we mean.

intervals For each dimension the interval between values for subsampling[2] (e.g., an interval of 2 means write to every other value). `intervals` is a 1-dimensional array containing one element per rVariable dimension. Each element of `intervals` specifies the corresponding dimension interval. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

buffer The buffer of values to write. The majority of the values in this buffer must be the same as that of the CDF. The values starting at memory address `buffer` are written to the CDF.

5.21.1 Example(s)

The following example writes values to the rVariable `LATITUDE` of a CDF whose rVariables are 2-dimensional with dimension sizes `[360,181]`. For `LATITUDE` the record variance is `NOVARY`, the dimension variances are `[NOVARY,VARY]`, and the data type is `CDF_INT2`. This example is similar to the example in Section 5.19 except that it uses a single call to `CDFvarHyperPut` rather than numerous calls to `CDFvarPut`.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                        /* CDF identifier. */
CDFstatus   status;                    /* Returned status code. */
short       lat;                       /* Latitude value. */
short       lats[181];                 /* Buffer of latitude values. */
long        varN;                      /* rVariable number. */
long recStart = 0;                     /* Record number. */
long recCount = 1;                     /* Record counts. */
long recInterval = 1;                  /* Record interval. */
static long indices[2] = {0,0};        /* Dimension indices. */
static long counts[2] = {1,181};       /* Dimension counts. */
static long intervals[2] = {1,1};      /* Dimension intervals. */
.
.
varN = CDFvarNum (id, "LATITUDE");
if (varN < 0) UserStatusHandler (varN);  /* If less than zero (0), not an
                                            rVariable number but rather a
                                            warning/error code. */

for (lat = -90; lat <= 90; lat += 1) lats[90+lat] = lat;

status = CDFvarHyperPut (id, varN, recStart, recCount, recInterval,
                         indices, counts, intervals, lats);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

[2] Again, not the best term.

5.22 CDFvarHyperGet

```
CDFstatus CDFvarHyperGet(
CDFid id,            /* in --  CDF identifier. */
long  varNum,        /* in --  rVariable number. */
long  recStart,      /* in --  Starting record number. */
long  recCount,      /* in --  Number of records. */
long  recInterval,   /* in --  Subsampling interval between records. */
long  indices[],     /* in --  Dimension indices of starting value. */
long  counts[],      /* in --  Number of values along each dimension. */
long  intervals[],   /* in --  Subsampling intervals along each dimension. */
void  *buffer);      /* out -- Buffer of values. */
```

`CDFvarHyperGet` is used to read a buffer of one or more values from an rVariable. It is important to know the variable majority of the CDF before using `CDFvarHyperGet` because the values placed into the buffer will be in that majority. `CDFinquire` is used to determine the variable majority of a CDF. The Concepts chapter in the CDF User's Guide describes the variable majorities.

The arguments to `CDFvarHyperGet` are defined as follows:

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

`varNum` The number of the rVariable from which to read. This number may be determined with a call to `CDFvarNum` (see Section 5.16).

`recStart` The record number at which to start reading.

`recCount` The number of records to read.

`recInterval` The interval between records for subsampling (e.g., an interval of 2 means read every other record).

`indices` The indices (within each record) at which to start reading. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` specifies the corresponding dimension index. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

`counts` The number of values along each dimension to read. `counts` is a 1-dimensional array containing one element per rVariable dimension. Each element of `counts` specifies the corresponding dimension count. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

`intervals` For each dimension, the interval between values for subsampling (e.g., an interval of 2 means read every other value). `intervals` is a 1-dimensional array containing one element per rVariable dimension. Each element of `intervals` specifies the corresponding dimension interval. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

`buffer` The buffer of values read. The majority of the values in this buffer will be the same as that of the CDF. This buffer must be large to hold the values. `CDFvarInquire` would be used to determine the rVariable's data type and number of elements (of that data type) at each value. The values are read from the CDF and placed into memory starting at address `buffer`.

5.22.1 Example(s)

The following example will read an entire record of data from an rVariable. The CDF's rVariables are 3-dimensional with sizes `[180,91,10]` and CDF's variable majority is `ROW_MAJOR`. For the rVariable the record variance is `VARY`, the dimension variances are `[VARY,VARY,VARY]`, and the data type is `CDF_REAL4`. This example is similar to the example in Section 5.20 except that it uses a single call to `CDFvarHyperGet` rather than numerous calls to `CDFvarGet`.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                         /* CDF identifier. */
CDFstatus   status;                     /* Returned status code. */
float       tmp[180][91][10];           /* Temperature values. */
long        varN;                       /* rVariable number. */
long recStart = 13;                     /* Record number. */
long recCount = 1;                      /* Record counts. */
long recInterval = 1;                   /* Record interval. */
static long indices[3] = {0,0,0};       /* Dimension indices. */
static long counts[3] = {180,91,10};    /* Dimension counts. */
static long intervals[3] = {1,1,1};     /* Dimension intervals. */
.
.
varN = CDFvarNum (id, "Temperature");
if (varN < 0) UserStatusHandler (varN);  /* If less than zero (0), then
                                            it is actually a warning/error
                                            code. */

status = CDFvarHyperGet (id, varN, recStart, recCount, recInterval,
                         indices, counts, intervals, tmp);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

Note that if the CDF's variable majority had been `COL_MAJOR`, the `tmp` array would have been declared `float tmp[10][91][180]` for proper indexing.

5.23 CDFvarClose

```
CDFstatus CDFvarClose(
CDFid id,             /* in --  CDF identifier. */
```

```
long  varNum);        /* in --  rVariable number. */
```

`CDFvarClose` is used to close an rVariable in a multi-file CDF. This function is not applicable to single-file CDFs. The use of `CDFvarClose` is not required since the CDF library automatically closes the rVariable files when a multi-file CDF is closed or when there are insufficient file pointers available (because of an open file quota) to keep all of the rVariable files open. `CDFvarClose` would be used by an application since it knows best how its rVariables are going to be accessed. Closing an rVariable would also free the cache buffers that are associated with the rVariable's file. This could be important in those situations where memory is limited (e.g., the IBM PC). The caching scheme used by the CDF library is described in the Concepts chapter in the CDF User's Guide. Note that there is not a function that opens an rVariable. The CDF library automatically opens an rVariable when it is accessed by an application (unless it is already open).

The arguments to `CDFvarClose` are defined as follows:

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

`varNum` The number of the rVariable to close. This number may be determined with a call to `CDFvarNum` (see Section 5.16).

5.23.1 Example(s)

The following example will close an rVariable in a multi-file CDF.

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

Chapter 6

Internal Interface — CDFlib

The Internal Interface consists of only one function, `CDFlib`. `CDFlib` can be used to perform all possible operations on a CDF. In fact, all of the Standard Interface functions are implemented using the Internal Interface. `CDFlib` must be used to perform operations not possible with the Standard Interface functions. These operations would involve CDF features added after the Standard Interface functions had been defined (e.g., specifying a single-file format for a CDF, accessing zVariables, or specifying a pad value for an rVariable or zVariable). Note that `CDFlib` can also be used to perform certain operations more efficiently than with the Standard Interface functions.

`CDFlib` takes a variable number of arguments that specify one or more operations to be performed (e.g., opening a CDF, creating an attribute, or writing a variable value). The operations are performed according to the order of the arguments. Each operation is performed on an item. An item may be either an object (e.g., a CDF, variable, or attribute) or a state (e.g., a CDF's format, a variable's data specification, or a CDF's current attribute). The possible operations and corresponding items (on which to perform those operations) are described in detail later in this section. The function prototype for `CDFlib` is as follows:

```
CDFstatus CDFlib (...);
```

This function prototype may also be found in the include file `cdf.h`.

6.1 Example(s)

The easiest way to explain how to use `CDFlib` would be to start with a few examples. The following example shows how a CDF would be created with the single-file format (assuming multi-file is the default).

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