Example(s)

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

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

4.7.7 CDFerror (Standard Interface)

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

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

The arguments to `CDFerror` are defined as follows:

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

Example(s)

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

```
.
```

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

4.7.8 CDFattrCreate (Standard Interface)

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

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

The arguments to `CDFattrCreate` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrName`	The name of the attribute to create. This may be at most `CDF_ATTR_NAME_LEN` characters (excluding the `NUL` terminator). Attribute names are case-sensitive.
`attrScope`	The scope of the new attribute. Specify one of the scopes described in Section 4.6.9.
`attrNum`	The number assigned to the new attribute. This number must be used in subsequent CDF function calls when referring to this attribute. An existing attribute's number may be determined with the `CDFattrNum` function.

Example(s)

The following example creates two attributes. The `TITLE` attribute is created with global scope — it applies to the entire CDF (most likely the title of the data set stored in the CDF). The `Units` attribute is created

with variable scope — each entry describes some property of the corresponding variable (in this case the units for the data).

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

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

4.7.9 CDFattrNum (Standard Interface)

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

`CDFattrNum` is used to determine the attribute number associated with a given attribute name. If the attribute is found, `CDFattrNum` returns its number — which will be equal to or greater than zero (0). If an error occurs (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.

Example(s)

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

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

4.7.10 CDFattrRename (Standard Interface)

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

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

The arguments to `CDFattrRename` are defined as follows:

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

Example(s)

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

```
.
```

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

4.7.11 CDFattrInquire (Standard Interface)

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

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

The arguments to `CDFattrInquire` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrNum`	The number of the attribute to inquire. This number may be determined with a call to `CDFattrNum` (see Section 4.7.9).
`attrName`	The attribute's name. This character string must be large enough to hold `CDF_ATTR_NAME_LEN + 1` characters (including the `NUL` terminator).
`attrScope`	The scope of the attribute. Attribute scopes are defined in Section 4.6.9.
`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 4.7.25).

Example(s)

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

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

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

4.7.12 CDFattrEntryInquire (Standard Interface)

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

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

The arguments to `CDFattrEntryInquire` are defined as follows:

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

Example(s)

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

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

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

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

4.7.13 CDFattrPut (Standard Interface)

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

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

The arguments to `CDFattrPut` are defined as follows:

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

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

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

Example(s)

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

4.7.14 CDFattrGet (Standard Interface)

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

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

The arguments to `CDFattrGet` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`attrNum`	The attribute number. This number may be determined with a call to `CDFattrNum` (see Section 4.7.9).
`entryNum`	The entry number. If the attribute is global in scope, this is simply the entry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).
`value`	The value read. This buffer must be large enough to hold the value. 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`.

Example(s)

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

```
.
.
#include "cdf.h"
.
.
CDFid     id;             /* CDF identifier. */
CDFstatus status;         /* Returned status code. */
long      attrN;          /* Attribute number. */
long      entryN;         /* Entry number. */
long      dataType;       /* Data type. */
long      numElems;       /* Number of elements (of data type). */
void      *buffer;        /* Buffer to receive value. */
```

```
.
.
attrN = CDFattrNum (id, "UNITS");
if (attrN < 0) UserStatusHandler (attrN);  /* If less than zero (0), then
                                              it must be a warning/error
                                              code. */

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

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

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

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

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

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

  free (buffer);
}
.
.
```

4.7.15 CDFattrCorrectScope (Standard Interface)

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

`CDFattrCorrectScope` is used to correct the scope of an attribute. Section 2.4.3 describes attribute scopes and the need for this function.

The arguments to `CDFattrCorrectScope` are defined as follows:

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

`attrNum`	The attribute number. This number may be determined with a call to `CDFattrNum` (see Section 4.7.9).
`attrScope`	The new scope for the attribute. Specify one of the definite scopes defined in Section 4.6.9.

IMPORTANT NOTE

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

Example(s)

The following example will correct the scope of every attribute in a CDF. Assumed scopes will be corrected to the corresponding definite scope. Note that it may not always be the case that an assumed scope should be corrected to the corresponding definite scope (see Section 2.4.3). The toolkit program `CDFscope` could also be used to correct attribute scopes.

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

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

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

4.7.16 CDFvarCreate (Standard Interface)

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

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

The arguments to `CDFvarCreate` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`varName`	The name of the rVariable to create. This may be at most `CDF_VAR_NAME_LEN` characters (excluding the `NUL` terminator). Variable names are case-sensitive.
`dataType`	The data type of the new rVariable. Specify one of the data types defined in Section 4.6.4.
`numElements`	The number of elements of the data type at each value. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string (each value consists of the entire string). For all other data types this must always be one (1) — multiple elements at each value are not allowed for non-character string data types.
`recVariance`	The rVariable's record variance. Specify one of the variances defined in Section 4.6.8.

`dimVariances`	The rVariable's dimension variances. `dimVariances` is a 1-dimensional array containing one element per rVariable dimension. Each element of `dimVariances` specifies the corresponding dimension variance. For each dimension specify one of the variances defined in Section 4.6.8. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
`varNum`	The number assigned to the new rVariable. This number must be used in subsequent CDF function calls when referring to this rVariable. An existing rVariables's number may be determined with the `CDFvarNum` function.

Example(s)

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

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

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

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

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

4.7.17 CDFvarNum (Standard Interface)

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

`CDFvarNum` is used to determine the 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.

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 4.7.19.

4.7.18 CDFvarRename (Standard Interface)

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

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

The arguments to `CDFvarRename` are defined as follows:

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

Example(s)

In the following example the rVariable named `TEMPERATURE` is renamed `TMP` (if it exists). Note that if `CDFvarNum` returns a value less than zero (0) 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);
}
.
.
```

4.7.19 CDFvarInquire (Standard Interface)

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

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

The arguments to `CDFvarInquire` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`varNum`	The number of the rVariable to inquire. This number may be determined with a call to `CDFvarNum` (see Section 4.7.17).
`varName`	The rVariable's name. This character string must be large enough to hold `CDF_VAR_NAME_LEN + 1` characters (including the `NUL` terminator).
`dataType`	The data type of the rVariable. The data types are defined in Section 4.6.4.
`numElements`	The number of elements of the data type at each rVariable value. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string. (Each value consists of the entire string.) For all other data types, this will always be one (1) — multiple elements at each value are not allowed for non-character string data types.
`recVariance`	The record variance. The record variances are defined in Section 4.6.8.

`dimVariances` The dimension variances. `dimVariances` is a 1-dimensional array containing one element per rVariable dimension in the CDF. Each element of `dimVariances` receives the corresponding dimension variance. The dimension variances are defined in Section 4.6.8. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).

Example(s)

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

```
.
.
#include "cdf.h"
.
.
CDFid     id;                           /* CDF identifier. */
CDFstatus status;                       /* Returned status code. */
char      varName[CDF_VAR_NAME_LEN+1];  /* rVariable name, +1
                                           for NUL terminator. */
long      dataType;                     /* Data type of the rVariable. */
long      numElems;                     /* Number of elements (of data
                                           type). */
long      recVary;                      /* Record variance. */
long      dimVarys[CDF_MAX_DIMS];       /* Dimension variances (allocate
                                           to allow the maximum number of
                                           dimensions). */
.
.
status = CDFvarInquire (id, CDFvarNum(id,"HEAT_FLUX"), varName, &dataType,
                        &numElems, &recVary, dimVarys);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.20 CDFvarPut (Standard Interface)

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

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

The arguments to `CDFvarPut` are defined as follows:

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

Example(s)

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

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

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

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

4.7.21 CDFvarGet (Standard Interface)

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

CDFvarGet is used to read a single value from an rVariable. CDFvarHyperGet may be used to read more than one rVariable value with a single call (see Section 4.7.23).

The arguments to CDFvarGet are defined as follows:

id	The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate or CDFopen.
varNum	The number of the rVariable from which to read. This number may be determined with a call to CDFvarNum (see Section 4.7.17).
recNum	The record number at which to read.
indices	The array indices within the specified record at which to read. indices is a 1-dimensional array containing one element per rVariable dimension. Each element of indices specifies the corresponding dimension index. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
value	The value read. This buffer must be large enough to hold the value. CDFvarInquire would be used to determine the rVariable's data type and number of elements (of that data type) at each value. The value is read from the CDF and placed at memory address value.

Example(s)

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

```
.
.
#include "cdf.h"
.
.
CDFid     id;               /* CDF identifier. */
```