```
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);
      }
   }
}
.
.
```

4.7.22 CDFvarHyperPut (Standard Interface)

```
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. Section 2.3.11 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 4.7.17).
`recStart`	The record number at which to start writing.
`recCount`	The number of records to write.
`recInterval`	The interval between records for subsampling[4] (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).
`intervals`	For each dimension the interval between values for subsampling[5] (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.

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 4.7.20 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. */
```

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

```
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);
.
.
```

4.7.23 CDFvarHyperGet (Standard Interface)

```
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. Section 2.3.11 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 4.7.17).
`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`.

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 4.7.21 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.

4.7.24 CDFvarClose (Standard Interface)

```
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 Section 2.1.4. 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 4.7.17).

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);
.
.
```

4.7.25 CDFlib (Internal Interface)

```
CDFstatus 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.

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"
.
.
CDFid       id;                          /* CDF identifier (handle). */
CDFstatus   status;                      /* Status returned from CDF
                                            library. */
static char CDFpath[] = {"test1"};       /* Pathname of the CDF. */
long        numDims = 2;                 /* Number of dimensions. */
static long dimSizes[2] = {100,200};     /* Dimension sizes. */
long        encoding = HOST_ENCODING;    /* Data encoding. */
long        majority = ROW_MAJOR;        /* Variable data majority. */
long        format = SINGLE_FILE;        /* Format of CDF. */
```

```
.
.
status = CDFcreate (CDFpath, numDims, dimSizes, encoding, majority, &id);
if (status != CDF_OK) UserStatusHandler (status);

status = CDFlib (PUT_, CDF_FORMAT_, format,
                 NULL_);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

The call to `CDFcreate` created the CDF as expected but with a format of multi-file. The call to `CDFlib` is then used to change the format to single-file (which must be done before any variables are created in the CDF).

The arguments to `CDFlib` in this example are explained as follows:

`PUT_`	The first operation to be performed. In this case an item is going to be put to the "current" CDF (a new format). `PUT_` is defined in `cdf.h` (as are all CDF constants). It was not necessary to select a current CDF since the call to `CDFcreate` implicitly selected the CDF created as the current CDF.[6] This is the case since all of the Standard Interface functions actually call the Internal Interface to perform their operations.
`CDF_FORMAT_`	The item to be put. In this case it is the CDF's format.
`format`	The actual format for the CDF. Depending on the item being put, one or more arguments would have been necessary. In this case only one argument is necessary.
`NULL_`	This argument could have been one of two things. It could have been another item to put (followed by the arguments required for that item) or it could have been a new operation to perform. In this case it is a new operation to perform — the `NULL_` operation. `NULL_` indicates the end of the call to `CDFlib`. Specifying `NULL_` at the end of the argument list is required because not all compilers/operating systems provide the ability for a called function to determine how many arguments were passed in by the calling function.

The next example shows how the same CDF could have been created using only one call to `CDFlib`. (The declarations would be the same.)

```
.
.
status = CDFlib (CREATE_, CDF_, CDFpath, numDims, dimSizes, &id,
                 PUT_, CDF_ENCODING_, encoding,
                       CDF_MAJORITY_, majority,
                       CDF_FORMAT_, format,
```

[6] In previous releases of CDF, it was required that the current CDF be selected in each call to `CDFlib`. That requirement has been eliminated. The CDF library now maintains the current CDF from one call to the next of `CDFlib`.

```
                    NULL_);
  if (status != CDF_OK) UserStatusHandler (status);
  .
  .
```

The purpose of each argument is as follows:

`CREATE_`	The first operation to be performed. In this case something will be created.
`CDF_`	The item to be created — a CDF in this case. There are four required arguments that must follow. When a CDF is created (with `CDFlib`), the format, encoding, and majority default to values specified when your CDF distribution was built and installed. Consult your system manager for these defaults.
`CDFpath`	The pathname of the CDF.
`numDims`	The number of dimensions in the CDF.
`dimSizes`	The dimension sizes.
`id`	The identifier to be used when referencing the created CDF in subsequent operations.
`PUT_`	This argument could have been one of two things. Another item to create or a new operation to perform. In this case it is another operation to perform — something will be put into the CDF.
`CDF_ENCODING_`	The item to be put — in this case the CDF's encoding. Note that the CDF did not have to be selected. It was implicitly selected as the current CDF when it was created.
`encoding`	The encoding to be put to the CDF.
`CDF_MAJORITY_`	This argument could have been one of two things. Another item to put or a new operation to perform. In this case it is another item to put — the CDF's majority.
`majority`	The majority to be put to the CDF.
`CDF_FORMAT_`	Once again this argument could have been either another item to put or a new operation to perform. It is another item to put — the CDF's format.
`format`	The format to be put to the CDF.
`NULL_`	This argument could have been either another item to put or a new operation to perform. Here it is another operation to perform — the `NULL_` operation that ends the call to `CDFlib`.

Note that the operations are performed in the order that they appear in the argument list. The CDF had to be created before the encoding, majority, and format could be specified (put).

Current Objects/States (Items)

The use of `CDFlib` requires that an application be aware of the current objects/states maintained by the CDF library. The following current objects/states are used by the CDF library when performing operations.

CDF (object)

A CDF operation is always performed on the current CDF. The current CDF is implicitly selected whenever a CDF is opened or created. The current CDF may be explicitly selected using the `<SELECT_,CDF_>`[7] operation. There is no current CDF until one is opened or created (which implicitly selects it) or until one is explicitly selected.[8]

rVariable (object)

An rVariable operation is always performed on the current rVariable in the current CDF. For each open CDF a current rVariable is maintained. This current rVariable is implicitly selected when an rVariable is created (in the current CDF) or it may be explicitly selected with the `<SELECT_,rVAR_>` or `<SELECT_,rVAR_NAME_>` operations. There is no current rVariable in a CDF until one is created (which implicitly selects it) or until one is explicitly selected.

zVariable (object)

A zVariable operation is always performed on the current zVariable in the current CDF. For each open CDF a current zVariable is maintained. This current zVariable is implicitly selected when a zVariable is created (in the current CDF) or it may be explicitly selected with the `<SELECT_,zVAR_>` or `<SELECT_,zVAR_NAME_>` operations. There is no current zVariable in a CDF until one is created (which implicitly selects it) or until one is explicitly selected.

attribute (object)

An attribute operation is always performed on the current attribute in the current CDF. For each open CDF a current attribute is maintained. This current attribute is implicitly selected when an attribute is created (in the current CDF) or it may be explicitly selected with the `<SELECT_,ATTR_>` or `<SELECT_,ATTR_NAME_>` operations. There is no current attribute in a CDF until one is created (which implicitly selects it) or until one is explicitly selected.

gEntry number (state)

A gAttribute gEntry operation is always performed on the current gEntry number in the current CDF for the current attribute in that CDF. For each open CDF a current gEntry number is maintained. This current gEntry number must be explicitly selected with the `<SELECT_,gENTRY_>` operation. (There is no implicit or default selection of the current gEntry number for a CDF.) Note that the current gEntry number is maintained for the CDF (not each attribute) — it applies to all of the attributes in that CDF.

rEntry number (state)

A vAttribute rEntry operation is always performed on the current rEntry number in the current CDF for the current attribute in that CDF. For each open CDF a current rEntry number is maintained. This current rEntry number must be explicitly selected with the `<SELECT_,rENTRY_>` operation. (There is no implicit or default selection of the current rEntry number for a CDF.) Note that the current rEntry number is maintained for the CDF (not each attribute) — it applies to all of the attributes in that CDF.

[7] This notation is used to specify an operation to be performed on an item. The syntax is `<operation,item>`.

[8] In previous releases of CDF, it was required that the current CDF be selected in each call to `CDFlib`. That requirement no longer exists. The CDF library now maintains the current CDF from one call to the next of `CDFlib`.

zEntry number (state)

A vAttribute zEntry operation is always performed on the current zEntry number in the current CDF for the current attribute in that CDF. For each open CDF a current zEntry number is maintained. This current zEntry number must be explicitly selected with the <SELECT_,zENTRY_> operation. (There is no implicit or default selection of the current zEntry number for a CDF.) Note that the current zEntry number is maintained for the CDF (not each attribute) — it applies to all of the attributes in that CDF.

record number, rVariables (state)

An rVariable read or write operation is always performed at (for single and multiple variable reads and writes) or starting at (for hyper reads and writes) the current record number for the rVariables in the current CDF. When a CDF is opened or created, the current record number for its rVariables is initialized to zero (0). It may then be explicitly selected using the <SELECT_,rVARs_RECNUMBER_> operation. Note that the current record number for rVariables is maintained for a CDF (not each rVariable) — it applies to all of the rVariables in that CDF.

record count, rVariables (state)

An rVariable hyper read or write operation is always performed using the current record count for the rVariables in the current CDF. When a CDF is opened or created, the current record count for its rVariables is initialized to one (1). It may then be explicitly selected using the <SELECT_,rVARs_RECCOUNT_> operation. Note that the current record count for rVariables is maintained for a CDF (not each rVariable) — it applies to all of the rVariables in that CDF.

record interval, rVariables (state)

An rVariable hyper read or write operation is always performed using the current record interval for the rVariables in the current CDF. When a CDF is opened or created, the current record interval for its rVariables is initialized to one (1). It may then be explicitly selected using the <SELECT_,rVARs_RECINTERVAL_> operation. Note that the current record interval for rVariables is maintained for a CDF (not each rVariable) — it applies to all of the rVariables in that CDF.

dimension indices, rVariables (state)

An rVariable read or write operation is always performed at (for single reads and writes) or starting at (for hyper reads and writes) the current dimension indices for the rVariables in the current CDF. When a CDF is opened or created, the current dimension indices for its rVariables are initialized to zeroes (0,0,...). They may then be explicitly selected using the <SELECT_,rVARs_DIMINDICES_> operation. Note that the current dimension indices for rVariables are maintained for a CDF (not each rVariable) — they apply to all of the rVariables in that CDF. For 0-dimensional rVariables the current dimension indices are not applicable.

dimension counts, rVariables (state)

An rVariable hyper read or write operation is always performed using the current dimension counts for the rVariables in the current CDF. When a CDF is opened or created, the current dimension counts for its rVariables are initialized to the dimension sizes of the rVariables (which specifies the entire array). They may then be explicitly selected using the <SELECT_,rVARs_DIMCOUNTS_> operation. Note that the current dimension counts for rVariables are maintained for a CDF (not each rVariable) — they apply to all of the rVariables in that CDF. For 0-dimensional rVariables the current dimension counts are not applicable.

dimension intervals, rVariables (state)

An rVariable hyper read or write operation is always performed using the current dimension intervals

for the rVariables in the current CDF. When a CDF is opened or created, the current dimension intervals for its rVariables are initialized to ones (1,1,...). They may then be explicitly selected using the <SELECT_,rVARs_DIMINTERVALS_> operation. Note that the current dimension intervals for rVariables are maintained for a CDF (not each rVariable) — they apply to all of the rVariables in that CDF. For 0-dimensional rVariables the current dimension intervals are not applicable.

sequential value, rVariable (state)

An rVariable sequential read or write operation is always performed at the current sequential value for that rVariable. When an rVariable is created (or for each rVariable in a CDF being opened), the current sequential value is set to the first physical value (even if no physical values exist yet). It may then be explicitly selected using the <SELECT_,rVAR_SEQPOS_> operation. Note that a current sequential value is maintained for each rVariable in a CDF.

record number, zVariable (state)

A zVariable read or write operation is always performed at (for single reads and writes) or starting at (for hyper reads and writes) the current record number for the current zVariable in the current CDF. A multiple variable read or write operation is performed at the current record number of each of the zVariables involved. (The record numbers do not have to be the same.) When a zVariable is created (or for each zVariable in a CDF being opened), the current record number for that zVariable is initialized to zero (0). It may then be explicitly selected using the <SELECT_,zVAR_RECNUMBER_> operation (which only affects the current zVariable in the current CDF). Note that a current record number is maintained for each zVariable in a CDF.

record count, zVariable (state)

A zVariable hyper read or write operation is always performed using the current record count for the current zVariable in the current CDF. When a zVariable created (or for each zVariable in a CDF being opened), the current record count for that zVariable is initialized to one (1). It may then be explicitly selected using the <SELECT_,zVAR_RECCOUNT_> operation (which only affects the current zVariable in the current CDF). Note that a current record count is maintained for each zVariable in a CDF.

record interval, zVariable (state)

A zVariable hyper read or write operation is always performed using the current record interval for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened), the current record interval for that zVariable is initialized to one (1). It may then be explicitly selected using the <SELECT_,zVAR_RECINTERVAL_> operation (which only affects the current zVariable in the current CDF). Note that a current record interval is maintained for each zVariable in a CDF.

dimension indices, zVariable (state)

A zVariable read or write operation is always performed at (for single reads and writes) or starting at (for hyper reads and writes) the current dimension indices for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened), the current dimension indices for that zVariable are initialized to zeroes (0,0,...). They may then be explicitly selected using the <SELECT_,zVAR_DIMINDICES_> operation (which only affects the current zVariable in the current CDF). Note that current dimension indices are maintained for each zVariable in a CDF. For 0-dimensional zVariables the current dimension indices are not applicable.

dimension counts, zVariable (state)

A zVariable hyper read or write operation is always performed using the current dimension counts for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable

in a CDF being opened), the current dimension counts for that zVariable are initialized to the dimension sizes of that zVariable (which specifies the entire array). They may then be explicitly selected using the <SELECT_,zVAR_DIMCOUNTS_> operation (which only affects the current zVariable in the current CDF). Note that current dimension counts are maintained for each zVariable in a CDF. For 0-dimensional zVariables the current dimension counts are not applicable.

dimension intervals, zVariable (state)

A zVariable hyper read or write operation is always performed using the current dimension intervals for the current zVariable in the current CDF. When a zVariable is created (or for each zVariable in a CDF being opened), the current dimension intervals for that zVariable are initialized to ones (1,1,...). They may then be explicitly selected using the <SELECT_,zVAR_DIMINTERVALS_> operation (which only affects the current zVariable in the current CDF). Note that current dimension intervals are maintained for each zVariable in a CDF. For 0-dimensional zVariables the current dimension intervals are not applicable.

sequential value, zVariable (state)

A zVariable sequential read or write operation is always performed at the current sequential value for that zVariable. When a zVariable is created (or for each zVariable in a CDF being opened), the current sequential value is set to the first physical value (even if no physical values exist yet). It may then be explicitly selected using the <SELECT_,zVAR_SEQPOS_> operation. Note that a current sequential value is maintained for each zVariable in a CDF.

status code (state)

When inquiring the explanation of a CDF status code, the text returned is always for the current status code. One current status code is maintained for the entire CDF library (regardless of the number of open CDFs). The current status code may be selected using the <SELECT_,CDF_STATUS_> operation. There is no default current status code. Note that the current status code is NOT the status code from the last operation performed.[9]

Returned Status

`CDFlib` returns a status code of type `CDFstatus`. Since more than one operation may be performed with a single call to `CDFlib`, the following rules apply:

1. The first error detected aborts the call to `CDFlib`, and the corresponding status code is returned.
2. In the absence of any errors, the status code for the last warning detected is returned.
3. In the absence of any errors or warnings, the status code for the last informational condition is returned.
4. In the absence of any errors, warnings, or informational conditions, `CDF_OK` is returned.

Section 4.8 explains how to interpret status codes. Appendix C lists the possible status codes and the type of each: error, warning, or informational.

[9] The CDF library now maintains the current status code from one call to the next of `CDFlib`.

Indentation/Style

Indentation should be used to make calls to CDFlib readable. The following example shows a call to CDFlib using proper indentation.

```
status = CDFlib (CREATE_, CDF_, CDFpath, numDims, dimSizes, &id,
                 PUT_, CDF_FORMAT_, format,
                       CDF_MAJORITY_, majority,
                 CREATE_, ATTR_, attrName, scope, &attrNum,
                          rVAR_, varName, dataType, numElements,
                                 recVary, dimVarys, &varNum,
                 NULL_);
```

Note that the operations (CREATE_, PUT_, and NULL_) are indented the same and that the items (CDF_, CDF_FORMAT_, CDF_MAJORITY_, ATTR_, and rVAR_) are indented the same under their corresponding operations.

The following example shows the same call to CDFlib without the proper indentation.

```
status = CDFlib (CREATE_, CDF_, CDFpath, numDims, dimSizes, &id, PUT_,
                 CDF_FORMAT_, format, CDF_MAJORITY_, majority, CREATE_,
                 ATTR_, attrName, scope, &attrNum, rVAR_, varName, dataType,
                 numElements, recVary, dimVarys, &varNum, NULL_);
```

The need for proper indentation to ensure the readability of your applications should be obvious.[10]

Syntax

CDFlib takes a variable number of arguments. There must always be at least one argument. The maximum number of arguments is not limited by CDF but rather the C compiler and operating system being used. Under normal circumstances that limit would never be reached (or even approached). Note also that a call to CDFlib with a large number of arguments can always be broken up into two or more calls to CDFlib with fewer arguments.

The syntax for CDFlib is as follows:

```
status = CDFlib (op1, item1, arg1, arg2, ...argN,
                      item2, arg1, arg2, ...argN,
                      .
                      .
                      itemN, arg1, arg2, ...argN,
                 op2, item1, arg1, arg2, ...argN,
                      item2, arg1, arg2, ...argN,
                      .
                      .
```

[10] If it isn't, call us — we can help you.

```
        itemN, arg1, arg2, ...argN,
    .
    .
    .
    opN, item1, arg1, arg2, ...argN,
        item2, arg1, arg2, ...argN,
        .
        .
        itemN, arg1, arg2, ...argN,
    NULL_);
```

where opx is an operation to perform, itemx is the item on which to perform the operation, and argx is a required argument for the operation. The NULL_ operation must be used to end the call to CDFlib. The completion status, status, is returned.

Operations/Items/Arguments...

The supported operations are as follows:

CREATE_	Used to create an item.
OPEN_	Used to open an item.
DELETE_	Used to delete an item.
CLOSE_	Used to close an item.
SELECT_	Used to select the value of an item.
CONFIRM_	Used to confirm the value of an item.
PUT_	Used to put (write) something to an item.
GET_	Used to get (read) something from an item.

For each operation the supported items, required arguments, and required preselected objects/states are listed in the following sections. The required preselected objects/states are those objects/states that must be selected (typically with the SELECT_ operation) before a particular operation may be performed. Note that some of the required preselected objects/states have default values as described beginning on page 149.

CREATE_ The following items may be created:

CDF_ A new CDF will be created. It is illegal to create a CDF that already exists. The created CDF implicitly becomes the current CDF. Required arguments are as follows:

in: char *CDFpath

Pathname of the CDF to be created. (Do not append an extension.) This can 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). **NOTE:** On UNIX systems pathnames are case-sensitive.

in: `long numDims`

Number of dimensions for the rVariables. This can be as few as zero (0) and at most `CDF_MAX_DIMS`. Note that this must be specified even if the CDF will contain only zVariables.

in: `long dimSizes[]`

Dimension sizes for the rVariables. `dimSizes` is a 1-dimensional array containing `numDims` elements. Each element of `dimSizes` specifies the corresponding dimension size. Each dimension size must be greater than zero (0). For 0-dimensional rVariables this argument is ignored (but a place holder is necessary). Note that this must be specified even if the CDF will contain only zVariables.

out: `CDFid *id`

CDF identifier to be used in subsequent operations on the CDF.

A CDF is created with the default format, encoding, and variable majority as specified in the configuration file of your CDF distribution. Consult your system manager to determine these defaults. These defaults can be changed with the corresponding `PUT_` operations of `CDF_FORMAT_`, `CDF_ENCODING_`, and `CDF_MAJORITY_`.

A CDF must be closed with `<CLOSE_,CDF_>` to ensure that the CDF will be correctly written to disk.

There are no required preselected objects/states.

`rVAR_`

A new rVariable will be created in the current CDF. A variable (rVariable or zVariable) with the same name must not already exist in the CDF. The created rVariable implicitly becomes the current rVariable (in the current CDF). Required arguments are as follows:

in: `char *varName`

Name of the rVariable to be created. This can be at most `CDF_VAR_NAME_LEN` characters (excluding the `NUL`). Variable names are case-sensitive.

in: `long dataType`

Data type of the new rVariable. Specify one of the data types described in Section 4.6.4.

in: `long numElements`

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 each string (an array of characters). A string exists at each value of the variable. For the non-character string data types this must be one (1) — multiple elements are not allowed for non-character string data types.

in: `long recVary`

Record variance. Specify one of the variances described in

Section 4.6.8.

in: `long dimVarys[]`

Dimension variances. `dimVarys` is a 1-dimensional array containing one element per rVariable dimension. Each element of `dimVarys` specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 4.6.8. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).

out: `long *varNum`

Number assigned to the new rVariable. This number must be used in subsequent CDF function calls when referring to this rVariable. An existing rVariable's number may also be determined with the `<GET_,rVAR_NUMBER_>` operation.

The only required preselected object/state is the current CDF.

`zVAR_` A new zVariable will be created in the current CDF. A variable (rVariable or zVariable) with the same name must not already exist in the CDF. The created zVariable implicitly becomes the current zVariable (in the current CDF). Required arguments are as follows:

in: `char *varName`

Name of the zVariable to be created. This can be at most `CDF_VAR_NAME_LEN` characters (excluding the `NUL` terminator). Variable names are case-sensitive.

in: `long dataType`

Data type of the new zVariable. Specify one of the data types described in Section 4.6.4.

in: `long numElements`

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 each string (an array of characters). A string exists at each value of the variable. For the non-character string data types this must be one (1) — multiple elements are not allowed for non-character string data types.

in: `long numDims`

Number of dimensions for the zVariable. This may be as few as zero and at most `CDF_MAX_DIMS`.

in: `long dimSizes[]`

The dimension sizes. `dimSizes` is a 1-dimensional array containing `numDims` elements. Each element of `dimSizes` specifies the corresponding dimension size. Each dimension size must be greater than zero (0). For a 0-dimensional zVariable this argument is ignored (but a place holder is necessary).

in: `long recVary`

Record variance. Specify one of the variances described in Section 4.6.8.

in: `long dimVarys[]`

Dimension variances. `dimVarys` is a 1-dimensional array containing one element per dimension. Each element of `dimVarys` specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 4.6.8. For a 0-dimensional zVariable this argument is ignored (but a place holder is necessary).

out: `long *varNum`

Number assigned to the new zVariable. This number must be used in subsequent CDF function calls when referring to this zVariable. An existing zVariable's number may also be determined with the `<GET_,zVAR_NUMBER_>` operation.

The only required preselected object/state is the current CDF.

`ATTR_` A new attribute will be created in the current CDF. An attribute with the same name must not already exist in the CDF. The created attribute implicitly becomes the current attribute (in the current CDF). Required arguments are as follows:

in: `char *attrName`

Name of the attribute to be created. This can be at most `CDF_ATTR_NAME_LEN` characters (excluding the `NUL` terminator). Attribute names are case-sensitive.

in: `long scope`

Scope of the new attribute. Specify one of the scopes described in Section 4.6.9.

out: `long *attrNum`

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 also be determined with the `<GET_,ATTR_NUMBER_>` operation.

The only required preselected object/state is the current CDF.

`OPEN_` The following item(s) may be opened:

`CDF_` Opens the named CDF. The opened CDF implicitly becomes the current CDF. Required arguments are as follows:

in: `char *CDFpath`

Pathname of the CDF to be opened. (Do not append an extension.) This can 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). **NOTE:** On UNIX systems pathnames are case-sensitive.

out: `CDFid *id`

CDF identifier to be used in subsequent operations on the CDF.

There are no required preselected objects/states.

`DELETE_` The following item(s) may be deleted:

`CDF_` Deletes the current CDF. A CDF must be opened before it can be deleted. When the CDF is deleted, there is no longer a current CDF.

There are no required arguments.

The only required preselected object/state is the current CDF.

`CLOSE_` The following items may be closed:

`CDF_` Closes the current CDF. When the CDF is closed, there is no longer a current CDF. A CDF must be closed to ensure that it will be properly written to disk.

There are no required arguments.

The only required preselected object/state is the current CDF.

`rVAR_` Closes the current rVariable (in the current CDF). This operation is only applicable to multi-file CDFs.

There are no required arguments.

The required preselected objects/states are the current CDF and its current rVariable.

`zVAR_` Closes the current zVariable (in the current CDF). This operation is only applicable to multi-file CDFs.

There are no required arguments.

The required preselected objects/states are the current CDF and its current zVariable.

`SELECT_` The following items may be selected:

`CDF_` Explicitly selects the current CDF. Required arguments are as follows:

in: `CDFid id`

Identifier of the CDF. This identifier must have been initialized by a successful `<CREATE_,CDF_>` or `<OPEN_,CDF_>` operation.

There are no required preselected objects/states.

`CDF_STATUS_` Selects the current status code. Required arguments are as follows:

in: `CDFstatus status`

CDF status code.

There are no required preselected objects/states.

`CDF_READONLY_MODE_` Selects a read-only mode (for the current CDF). Required arguments are as follows:

in: `long mode`

The read-only mode. Specify one of the read-only modes described in Section 4.6.10.

The only required preselected object/state is the current CDF.

`CDF_zMODE_` Selects a zMode (for the current CDF). Required arguments are as follows:

in: `long mode`

The zMode. Specify one of the zModes described in Section 4.6.11.

The only required preselected object/state is the current CDF.

`CDF_NEGtoPOSfp0_MODE_`

Selects a `-0.0` to `0.0` mode (for the current CDF). Required arguments are as follows:

in: `long mode`

The `-0.0` to `0.0` mode. Specify one of the `-0.0` to `0.0` modes described in Section 4.6.12.

The only required preselected object/state is the current CDF.

`CDF_DECODING_` Selects a decoding (for the current CDF). Required arguments are as follows:

in: `long decoding`

The decoding. Specify one of the decodings described in Section 4.6.6.

The only required preselected object/state is the current CDF.

`CDF_CACHESIZE_` Selects the number of cache buffers to be used for the "`.cdf`" file (for the current CDF). Section 2.1.4 describes the caching scheme used by the CDF library in greater detail. Required arguments are as follows:

in: `long numBuffers`

The number of cache buffers to be used.

The only required preselected object/state is the current CDF.

`rVAR_` Explicitly selects the current rVariable (in the current CDF) by number. Required arguments are as follows:

in: `long varNum`

rVariable number.

The only required preselected object/state is the current CDF.

`rVAR_NAME_` Explicitly selects the current rVariable (in the current CDF) by name. Required arguments are as follows:

in: `char *varName`

rVariable name. This may be at most `CDF_VAR_NAME_LEN` characters (excluding the `NUL` terminator).

The only required preselected object/state is the current CDF.

`rVARs_RECNUMBER_` Selects the current record number for all rVariables in the current CDF. Required arguments are as follows:

in: `long recNum`

Record number.

The only required preselected object/state is the current CDF.

`rVARs_RECCOUNT_` Selects the current record count for all rVariables in the current CDF. Required arguments are as follows:

in: `long recCount`

Record count.

The only required preselected object/state is the current CDF.

`rVARs_RECINTERVAL_` Selects the current record interval for all rVariables in the current CDF. Required arguments are as follows:

in: `long recInterval`

Record interval.

The only required preselected object/state is the current CDF.

`rVARs_DIMINDICES_` Selects the current dimension indices for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are as follows:

in: `long indices[]`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` specifies the corresponding dimension index.

The only required preselected object/state is the current CDF.

`rVARs_DIMCOUNTS_` Selects the current dimension counts for all rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are as follows:

in: `long counts[]`