```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                            ! CDF identifier.
INTEGER*4 status                        ! Returned status code.
INTEGER*4 num_dims                      ! Number of dimensions.
INTEGER*4 dim_sizes(CDF_MAX_DIMS)       ! Dimension sizes (allocate to allow
                                        ! the maximum number of dimensions).
INTEGER*4 encoding                      ! Data encoding.
INTEGER*4 majority                      ! Variable majority.
INTEGER*4 max_rec                       ! Maximum record number in CDF.
INTEGER*4 num_vars                      ! Number of variables in CDF.
INTEGER*4 num_attrs                     ! Number of attributes in CDF.
INTEGER*4 attr_n                        ! Attribute number.
CHARACTER attr_name*(CDF_ATTR_NAME_LEN) ! Attribute name.
INTEGER*4 attr_scope                    ! Attribute scope.
INTEGER*4 max_entry                     ! Maximum entry number.
.
.
CALL CDF_inquire (id, num_dims, dim_sizes, encoding, majority,
1                 max_rec, num_vars, num_attrs, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

DO attr_n = 1, num_attrs
  CALL CDF_attr_inquire (id, attr_n, attr_name, attr_scope, max_entry,
  1                      status)
  IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

  IF (attr_scope .EQ. GLOBAL_SCOPE_ASSUMED) THEN
    CALL CDF_attr_correct_scope (id, attr_n, GLOBAL_SCOPE, status)
    IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
  ELSE
    IF (attr_scope .EQ. VARIABLE_SCOPE_ASSUMED) THEN
      CALL CDF_attr_correct_scope (id, attr_n, VARIABLE_SCOPE, status)
      IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
    END IF
  END IF
END DO
.
.
```

5.8.16 CDF_var_create (Standard Interface)

```
SUBROUTINE CDF_var_create (id, var_name, data_type, num_elements,
1                          rec_variance, dim_variances, var_num, status)

INTEGER*4 id                  ! in --  CDF identifier.
```

```
CHARACTER var_name*(*)       ! in --  rVariable name.
INTEGER*4 data_type          ! in --  Data type.
INTEGER*4 num_elements       ! in --  Number of elements (of the data type).
INTEGER*4 rec_variance       ! in --  Record variance.
INTEGER*4 dim_variances(*)   ! in --  Dimension variances.
INTEGER*4 var_num            ! out -- rVariable number.
INTEGER*4 status             ! out -- Completion status.
```

CDF_var_create 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 CDF_var_create are defined as follows:

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_name	The name of the rVariable to create. This may be at most CDF_VAR_NAME_LEN characters. Variable names are case-sensitive.
data_type	The data type of the new rVariable. Specify one of the data types defined in Section 5.7.3.
num_elements	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.
rec_variance	The variable's record variance. Specify one of the variances defined in Section 5.7.7.
dim_variances	The variable's dimension variances. dim_variances is a 1-dimensional array containing one element per rVariable dimension. Each element of dim_variances specifies the corresponding dimension variance. For each dimension specify one of the variances defined in Section 5.7.7. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
var_num	The 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 be determined with the CDF_var_num function.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

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

```
.
.
INCLUDE '<path>cdf.inc'
```

```
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
INTEGER*4 EPOCH_rec_vary            ! EPOCH record variance.
INTEGER*4 LAT_rec_vary              ! LAT record variance.
INTEGER*4 LON_rec_vary              ! LON record variance.
INTEGER*4 TMP_rec_vary              ! TMP record variance.
INTEGER*4 EPOCH_dim_varys(2)        ! EPOCH dimension variances.
INTEGER*4 LAT_dim_varys(2)          ! LAT dimension variances.
INTEGER*4 LON_dim_varys(2)          ! LON dimension variances.
INTEGER*4 TMP_dim_varys(2)          ! TMP dimension variances.
INTEGER*4 EPOCH_var_num             ! EPOCH variable number.
INTEGER*4 LAT_var_num               ! LAT rVariable number.
INTEGER*4 LON_var_num               ! LON rVariable number.
INTEGER*4 TMP_var_num               ! TMP rVariable number.

DATA EPOCH_rec_vary/VARY/, LAT_rec_vary/NOVARY/,
1    LON_rec_vary/NOVARY/, TMP_rec_vary/VARY/
DATA EPOCH_dim_varys/NOVARY,NOVARY/, LAT_dim_varys/NOVARY,VARY/,
1    LON_dim_varys/VARY,NOVARY/, TMP_dim_varys/VARY,VARY/
.
.
CALL CDF_var_create (id, 'EPOCH', CDF_EPOCH, 1,
1                    EPOCH_rec_vary, EPOCH_dim_varys, EPOCH_var_num, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL CDF_var_create (id, 'LATITUDE', CDF_INT2, 1,
1                    LAT_rec_vary, LAT_dim_varys, LAT_var_num, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL CDF_var_create (id, 'LONGITUDE', CDF_INT2, 1,
1                    LON_rec_vary, LON_dim_varys, LON_var_num, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL CDF_var_create (id, 'TEMPERATURE', CDF_REAL4, 1,
1                    TMP_rec_vary, TMP_dim_varys, TMP_var_num, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.17 CDF_var_num (Standard Interface)

```
INTEGER*4 FUNCTION CDF_var_num (id, var_name)

INTEGER*4 id                ! in -- CDF identifier.
CHARACTER var_name*(*)      ! in -- rVariable name.
```

CDF_var_num is used to determine the number associated with a given rVariable name. If the rVariable is found, CDF_var_num returns its number — which will be equal to or greater than one (1). If an error occurs (e.g., the rVariable does not exist in the CDF), an error code is returned. Error codes are less than zero (0).

The arguments to CDF_var_num are defined as follows:

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_name	The name of the rVariable for which to search. This may be at most CDF_VAR_NAME_LEN characters. Variable names are case-sensitive.

CDF_var_num may be used as an embedded function call when an rVariable number is needed. CDF_var_num is declared in cdf.inc. (Fortran functions must be declared so that the returned value is interpreted correctly.)

NOTE: Some Fortran compilers give a warning if CDF_var_num is not used in a routine where cdf.inc has been included. This warning may be ignored. To eliminate the warning message, remove the declaration of CDF_var_num from cdf.inc, but be sure to declare CDF_var_num in all routines that use it.

Example(s)

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

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                            ! CDF identifier.
INTEGER*4 status                        ! Returned status code.
CHARACTER var_name*(CDF_VAR_NAME_LEN)   ! rVariable name.
INTEGER*4 data_type                     ! Data type of the rVariable.
INTEGER*4 num_elements                  ! Number of elements (of the
                                        ! data type).
INTEGER*4 rec_variances                 ! Record variance.
INTEGER*4 dim_variances(CDF_MAX_DIMS)   ! Dimension variances.
.
.
CALL CDF_var_inquire (id, CDF_var_num(id,'LATITUDE'), var_name, data_type,
1                     num_elements, rec_variance, dim_variances, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

In this example the rVariable named LATITUDE was inquired. Note that if LATITUDE did not exist in the CDF, the call to CDF_var_num would have returned an error code. Passing that error code to CDF_var_inquire as an rVariable number would have resulted in CDF_var_inquire also returning an error code. Also note that the name written into var_name is already known (LATITUDE). In some cases the rVariable names

will be unknown — CDF_var_inquire would be used to determine them. CDF_var_inquire is described in Section 5.8.19.

5.8.18 CDF_var_rename (Standard Interface)

```
SUBROUTINE CDF_var_rename (id, var_num, var_name, status)

INTEGER*4 id               ! in  -- CDF identifier.
INTEGER*4 var_num          ! in  -- rVariable number.
CHARACTER var_name*(*)     ! in  -- New name.
INTEGER*4 status           ! out -- Completion status.
```

CDF_var_rename 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 CDF_var_rename are defined as follows:

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable to rename. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
var_name	The new rVariable name. This may be at most CDF_VAR_NAME_LEN characters. Variables names are case-sensitive.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

In the following example the rVariable named TEMPERATURE is renamed TMP (if it exists). Note that if CDF_var_num returns a value less than one (1), then that value is not an rVariable number but rather a warning/error code.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
INTEGER*4 var_num                   ! rVariable number.
.
.
var_num = CDF_var_num (id, 'TEMPERATURE')
IF (var_num .LT. 1) THEN
  IF (var_num .NE. NO_SUCH_VAR) CALL UserStatusHandler (var_num)
```

```
ELSE
  CALL CDF_var_rename (id, var_num, 'TMP', status)
  IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
END IF
.
.
```

5.8.19 CDF_var_inquire (Standard Interface)

```
SUBROUTINE CDF_var_inquire (id, var_num, var_name, data_type, num_elements,
1                           rec_variance, dim_variances, status)

INTEGER*4 id                            ! in  -- CDF identifier.
INTEGER*4 var_num                       ! in  -- rVariable number.
CHARACTER var_name*(CDF_VAR_NAME_LEN)   ! out -- rVariable name.
INTEGER*4 data_type                     ! out -- Data type.
INTEGER*4 num_elements                  ! out -- Number of elements (of the
                                        !        data type).
INTEGER*4 rec_variance                  ! out -- Record variance.
INTEGER*4 dim_variances(CDF_MAX_DIMS)   ! out -- Dimension variances.
INTEGER*4 status                        ! out -- Completion status.
```

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

The arguments to CDF_var_inquire are defined as follows:

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable to inquire. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
var_name	The rVariable's name. This character string must be large enough to hold CDF_VAR_NAME_LEN characters and will be blank padded if necessary.
data_type	The data type of the rVariable. The data types are defined in Section 5.7.3.
num_elements	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 will always be one (1) — multiple elements at each value are not allowed for non-character string data types.
rec_variance	The record variance. The record variances are defined in Section 5.7.7.
dim_variances	The dimension variances. dim_variances is a 1-dimensional array containing one element per rVariable dimension. Each element of dim_variances receives the corresponding dimension variance. The dimension variances are defined in

Section 5.7.7. For 0-dimensional rVariable this argument is ignored (but a place holder is necessary).

`status` The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

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

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                            ! CDF identifier.
INTEGER*4 status                        ! Returned status code.
CHARACTER var_name*(CDF_VAR_NAME_LEN)   ! rVariable name.
INTEGER*4 data_type                     ! Data type.
INTEGER*4 num_elems                     ! Number of elements (of data type).
INTEGER*4 rec_vary                      ! Record variance.
INTEGER*4 dim_varys(CDF_MAX_DIMS)       ! Dimension variances (allocate
                                        ! to allow the maximum number of
                                        ! dimensions).
.
.
CALL CDF_var_inquire (id, CDF_var_num(id,'HEAT_FLUX'), var_name, data_type,
1                     num_elems, rec_vary, dim_varys, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.20 CDF_var_put (Standard Interface)

```
SUBROUTINE CDF_var_put (id, var_num, rec_num, indices, value, status)

INTEGER*4 id             ! in -- CDF identifier.
INTEGER*4 var_num        ! in -- rVariable number.
INTEGER*4 rec_num        ! in -- Record number.
INTEGER*4 indices(*)     ! in -- Dimension indices.
<type>    value          ! in -- Value. (<type> depends on the data type
                         !       of the rVariable).
INTEGER*4 status         ! out -- Completion status.
```

`CDF_var_put` is used to write a single value to an rVariable. `CDF_var_hyper_put` may be used to write more than one rVariable value with a single call.

The arguments to `CDF_var_put` are defined as follows:

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable to which to write. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
rec_num	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.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example writes values to the rVariable LATITUDE of a CDF whose rVariables are 2-dimensional and have 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 '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
INTEGER*2 lat                       ! Latitude value.
INTEGER*4 var_n                     ! rVariable number.
INTEGER*4 rec_num                   ! Record number.
INTEGER*4 indices(2)                ! Dimension indices.

DATA rec_num/1/, indices/1,1/
.
.
var_n = CDF_var_num (id, 'LATITUDE')
IF (var_n .LT. 1) CALL UserStatusHandler (var_n)  ! If less than one (1),
                                                  ! then not an rVariable
                                                  ! number but rather a
                                                  ! warning/error code.
DO lat = -90, 90
   indices(2) = 91 + lat
   CALL CDF_var_put (id, var_n, rec_num, indices, lat, status)
   IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
END DO
.
.
```

Since the record variance is NOVARY, the record number (rec_num) is set to one (1). 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.)

5.8.21 CDF_var_get (Standard Interface)

```
SUBROUTINE CDF_var_get (id, var_num, rec_num, indices, value, status)

INTEGER*4 id             ! in  -- CDF identifier.
INTEGER*4 var_num        ! in  -- rVariable number.
INTEGER*4 rec_num        ! in  -- Record number.
INTEGER*4 indices(*)     ! in  -- Dimension indices.
<type>    value          ! out -- Value (<type> depends on the data type
                         !        of the rVariable).
INTEGER*4 status         ! out -- Completion status.
```

CDF_var_get is used to read a single value from an rVariable. CDF_var_hyper_get may be used to read more than one rVariable value with a single call.

The arguments to CDF_var_get are defined as follows:

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable from which to read. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
rec_num	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. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).
value	The value read. This buffer must be large enough to hold the value. CDF_var_inquire 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 into memory at address value.
status	The completion status code. Section 5.9 explains how to interpret status codes.

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 the variable the record variance is VARY; the dimension variances are [VARY,VARY,VARY]; and the data type is CDF_REAL4.

.

```
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
REAL*4    tmp(180,91,10)            ! Temperature values.
INTEGER*4 indices(3)                ! Dimension indices.
INTEGER*4 var_n                     ! rVariable number.
INTEGER*4 rec_num                   ! Record number.
INTEGER*4 d1, d2, d3                ! Dimension index values.
.
.
var_n = CDF_var_num (id, 'Temperature')
IF (var_n .LT. 1) CALL UserStatusHandler (var_n)  ! If less than one (1),
                                                  ! then it is actually a
                                                  ! warning/error code.

rec_num = 13

DO d1 = 1, 180
  indices(1) = d1
  DO d2 = 1, 91
    indices(2) = d2
    DO d3 = 1, 10
      indices(3) = d3
      CALL CDF_var_get (id, var_n, rec_num, indices, tmp(d1,d2,d3), status)
      IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
    END DO
  END DO
END DO
.
.
```

5.8.22 CDF_var_hyper_put (Standard Interface)

```
SUBROUTINE CDF_var_hyper_put (id, var_num, rec_start, rec_count, rec_interval,
1                             indices, counts, intervals, buffer, status)

INTEGER*4 id               ! in -- CDF identifier.
INTEGER*4 var_num          ! in -- rVariable number.
INTEGER*4 rec_start        ! in -- Starting record number.
INTEGER*4 rec_count        ! in -- Number of records.
INTEGER*4 rec_interval     ! in -- Interval between records.
INTEGER*4 indices(*)       ! in -- Dimension indices of starting value.
INTEGER*4 counts(*)        ! in -- Number of values along each dimension.
INTEGER*4 intervals(*)     ! in -- Interval between values along each
                           !       dimension.
<type>    buffer           ! in -- Buffer of values (<type> depends on the
```

```
                          !       data type of the rVariable.)
INTEGER*4 status          ! out -- Completion status.
```

CDF_var_hyper_put 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 CDF_var_hyper_put because the values in the buffer to be written must be in the same majority. CDF_inquire is used to determine the variable majority of a CDF. Section 2.3.11 describes the variable majorities.

The arguments to CDF_var_hyper_put are defined as follows:

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable to which to write. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
rec_start	The record number at which to start writing.
rec_count	The number of records to write.
rec_interval	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. For 0-dimensional rVariables 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 counts specifies the corresponding dimension count. For 0-dimensional rVariables 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. For 0-dimensional rVariables 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.
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

The following example writes values to the rVariable LATITUDE in a CDF whose rVariables are 2-dimensional and have dimension sizes [360,181]. For LATITUDE the record variance is NOVARY; the dimension variances

[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.

are [NOVARY,VARY]; and the data type is CDF_INT2. This example is similar to the example in Section 5.8.20 except that it uses a single call to CDF_var_hyper_put rather than numerous calls to CDF_var_put.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
INTEGER*2 lat                       ! Latitude value.
INTEGER*2 lats(181)                 ! Buffer of latitude values.
INTEGER*4 var_n                     ! rVariable number.
INTEGER*4 rec_start                 ! Record number.
INTEGER*4 rec_count                 ! Record counts.
INTEGER*4 rec_interval              ! Record interval.
INTEGER*4 indices(2)                ! Dimension indices.
INTEGER*4 counts(2)                 ! Dimension counts.
INTEGER*4 intervals(2)              ! Dimension intervals.

DATA rec_start/1/, rec_count/1/, rec_interval/1/,
1    indices/1,1/, counts/1,181/, intervals/1,1/
.
.
var_n = CDF_var_num (id, 'LATITUDE')
IF (var_n .LT. 1) CALL UserStatusHandler (var_n)  ! If less than one (1),
                                                  ! then not an rVariable
                                                  ! number but rather a
                                                  ! warning/error code
DO lat = -90, 90
  lats(91+lat) = lat
END DO

CALL CDF_var_hyper_put (id, var_n, rec_start, rec_count, rec_interval,
1                       indices, counts, intervals, lats, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.23 CDF_var_hyper_get (Standard Interface)

```
SUBROUTINE CDF_var_hyper_get (id, var_num, rec_start, rec_count, rec_interval,
1                             indices, counts, intervals, buffer, status)

INTEGER*4 id                ! in -- CDF identifier.
INTEGER*4 var_num           ! in -- rVariable number.
INTEGER*4 rec_start         ! in -- Starting record number.
INTEGER*4 rec_count         ! in -- Number of records.
INTEGER*4 rec_interval      ! in -- Subsampling interval between records.
```

```
INTEGER*4 indices(*)        ! in -- Dimension indices of starting value.
INTEGER*4 counts(*)         ! in -- Number of values along each dimension.
INTEGER*4 intervals(*)      ! in -- Subsampling intervals along each dimension.
<type>    buffer            ! out -- Buffer of values (<type> depends on the
                            !        data type of the rVariable).
INTEGER*4 status            ! out -- Completion status.
```

CDF_var_hyper_get 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 CDF_var_hyper_get because the values placed into the buffer will be in that majority. CDF_var_inquire is used to determine the variable majority of a CDF. Section 2.3.11 describes the variable majorities.

The arguments to CDF_var_hyper_get are defined as follows:

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable from which to read. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
rec_start	The record number at which to start reading.
rec_count	The number of records to read.
rec_interval	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. For 0-dimensional rVariables 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. For 0-dimensional rVariables 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. For 0-dimensional rVariables 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 enough to hold the values. The values are read from the CDF and placed into memory starting at address buffer. CDF_var_inquire would be used to determine the variable's data type and number of elements (of that data type) at each value.
status	The completion status code. Section 5.9 explains how to interpret status codes.

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 the CDF's variable majority is `COL_MAJOR`. For the variable 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 5.8.21 except that it uses a single call to `CDF_var_hyper_get` rather than numerous calls to `CDF_var_get`.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
REAL*4    tmp(180,91,10)            ! Temperature values.
INTEGER*4 var_n                     ! rVariable number.
INTEGER*4 rec_start                 ! Record number.
INTEGER*4 rec_count                 ! Record counts.
INTEGER*4 rec_interval              ! Record interval.
INTEGER*4 indices(3)                ! Dimension indices.
INTEGER*4 counts(3)                 ! Dimension counts.
INTEGER*4 intervals(3)              ! Dimension intervals.

DATA rec_start/13/, rec_count/1/, rec_interval/1/,
1    indices/1,1,1/, counts/180,91,10/, intervals/1,1,1/
.
.
var_n = CDF_var_num (id, 'Temperature')
IF (var_n .LT. 1) CALL UserStatusHandler (var_n)  ! If less than one (1),
                                                  ! then it is actually a
                                                  ! warning/error code.

CALL CDF_var_hyper_get (id, var_n, rec_start, rec_count, rec_interval,
1                       indices, counts, intervals, tmp, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

Note that if the CDF's variable majority had been `ROW_MAJOR`, the `tmp` array would have been declared `REAL*4 tmp(10,91,180)` for proper indexing.

5.8.24 CDF_var_close (Standard Interface)

```
SUBROUTINE CDF_var_close (id, var_num, status)

INTEGER*4 id                ! in -- CDF identifier.
INTEGER*4 var_num           ! in -- rVariable number.
```

```
INTEGER*4 status            ! out -- Completion status.
```

CDF_var_close is used to close an rVariable in a multi-file CDF. This function is not applicable to single-file CDFs. The use of CDF_var_close 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. CDF_var_close 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 CDF_var_close are defined as follows:

id	The identifier of the CDF. This identifier must have been initialized by a call to CDF_create or CDF_open.
var_num	The number of the rVariable to close. This number may be determined with a call to CDF_var_num (see Section 5.8.17).
status	The completion status code. Section 5.9 explains how to interpret status codes.

Example(s)

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

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
.
.
CALL CDF_var_close (id, CDF_var_num(id,'Flux'), status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.8.25 CDF_lib (Internal Interface)

```
SUBROUTINE CDF_lib (..., status)

INTEGER*4 status                ! out -- Completion status.
```

The Internal Interface consists of only one function, `CDF_lib`. `CDF_lib` 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. `CDF_lib` 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 `CDF_lib` can also be used to perform certain operations more efficiently than with the Standard Interface functions.

`CDF_lib` 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 `CDF_lib` would be to start with a few examples. The following example will show how a CDF would be created with the single-file format (assuming multi-file is the default):

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
CHARACTER CDF_path*5                ! Name of the CDF.
INTEGER*4 num_dims                  ! Number of dimensions.
INTEGER*4 dim_sizes(2)              ! Dimension sizes.
INTEGER*4 encoding                  ! Data encoding.
INTEGER*4 majority                  ! Variable data majority.
INTEGER*4 format                    ! Format of CDF.

DATA CDF_path/'test1'/, num_dims/2/, dim_sizes/100,200/,
1    encoding/HOST_ENCODING/, majority/ROW_MAJOR/, format/SINGLE_FILE/
.
.
CALL CDF_create (CDF_path, num_dims, dim_sizes, encoding, majority,
1                id, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL CDF_lib (PUT_, CDF_FORMAT_, format,
2             NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

The call to `CDF_create` created the CDF as expected but with a format of multi-file. The call to `CDF_lib`

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 `CDF_lib` 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.inc` (as are all CDF constants). It was not necessary to select a current CDF since the call to `CDF_create` 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. One required argument for this operation follows.
`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 `CDF_lib`. Specifying `NULL_` at the end of the argument list is required because not all Fortran compilers/operating systems provide the ability for a called function to determine how many arguments were passed in by the calling function.
`status`	The completion status code. Section 5.9 explains how to interpret status codes.

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

```
.
.
CALL CDF_lib (CREATE_, CDF_, CDF_path, num_dims, dim_sizes, id,
1             PUT_, CDF_ENCODING_, encoding,
2                   CDF_MAJORITY_, majority,
3                   CDF_FORMAT_, format,
4             NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

The purpose of each argument is as follows.

`CREATE_`	The first operation to be performed. In this case something will be created.

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

CDF_	The item to be created — a CDF in this case. There are four required argument that must follow. When a CDF is created (with CDF_lib), the format, encoding, and majority default to values specified when your CDF distribution was built and installed. Consult your system manager for these defaults.
CDF_path	The pathname of the CDF.
num_dims	The number of dimensions in the CDF.
dim_sizes	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. One required argument follows.
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. One required argument follows.
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. One required argument follows.
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 which ends the call to CDF_lib.
status	The completion status code. Section 5.9 explains how to interpret status codes.

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 CDF_lib 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.

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.

[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 CDF_lib. That requirement no longer exists. The CDF library now maintains the current CDF from one call to the next of CDF_lib.

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