Chapter 5

Standard Interface

The following sections describe the Standard Interface routines callable from Fortran applications. Most routines return a status code of type `INTEGER*4` (see Chapter 7). The Internal Interface is described in Chapter 6. An application can use both interfaces when necessary. Note that zVariables and vAttribute zEntries are only accessible via the Internal Interface.

5.1 CDF_create

```
SUBROUTINE CDF_create (CDF_path, num_dims, dim_sizes, encoding, majority,
1                       id, status)

CHARACTER CDF_path*(*)        ! in  -- CDF pathname.
INTEGER*4 num_dims            ! in  -- Number of dimensions, rVariables.
INTEGER*4 dim_sizes(*)        ! in  -- Dimension sizes, rVariables.
INTEGER*4 encoding            ! in  -- Data encoding.
INTEGER*4 majority            ! in  -- Variable majority.
INTEGER*4 id                  ! out -- CDF identifier.
INTEGER*4 status              ! out -- Completion status.
```

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

The arguments to `CDF_create` are defined as follows:

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

UNIX: Pathnames are case-sensitive.

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

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

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

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

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

`status` The completion status code. Chapter 7 explains how to interpret status codes.

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

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

5.1.1 Example(s)

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

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id               ! CDF identifier.
INTEGER*4 status           ! Returned status code.
INTEGER*4 num_dims         ! Number of dimensions, rVariables.
INTEGER*4 dim_sizes(3)     ! Dimension sizes, rVariables.
INTEGER*4 majority         ! Variable majority.

DATA num_dims/3/, dim_sizes/180,360,10/, majority/ROW_MAJOR/
.
.
CALL CDF_create ('test1', num_dims, dim_sizes, NETWORK_ENCODING,
1                 majority, id, status)
```

```
   IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
   .
   .
```

`ROW_MAJOR` and `NETWORK_ENCODING` are defined in `cdf.inc`.

5.2 CDF_open

```
SUBROUTINE CDF_open (CDF_path, id, status)

CHARACTER CDF_path*(*)        ! in  -- CDF pathname.
INTEGER*4 id                  ! out -- CDF identifier.
INTEGER*4 status              ! out -- Completion status.
```

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

The arguments to `CDF_open` are defined as follows:

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

UNIX: Pathnames are case-sensitive.

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

`status` The completion status code. Chapter 7 explains how to interpret status codes.

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

5.2.1 Example(s)

The following example will open a CDF named `NOAA1`.

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

```
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
CHARACTER CDF_path*(CDF_PATHNAME_LEN)  ! Pathname of CDF.

DATA CDF_path/'NOAA1'/
.
.
CALL CDF_open (CDF_path, id, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.3 CDF_doc

```
SUBROUTINE CDF_doc (id, version, release, copyright, status)

INTEGER*4 id,                           ! in  -- CDF identifier.
INTEGER*4 version,                      ! out -- Version number.
INTEGER*4 release,                      ! out -- Release number.
CHARACTER copyright*(CDF_DOCUMENT_LEN)  ! out -- Copyright.
```

`CDF_doc` is used to inquire general documentation about a CDF. The version/release of the CDF that which created the CDF is provided (e.g., CDF V2.3 is version 2, release 3) along with the CDF copyright notice.

The arguments to `CDF_doc` 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`.
- `version` The version number of the CDF library that created the CDF.
- `release` The release number of the CDF library that created the CDF.
- `copyright` The copyright notice of the CDF library that created the CDF. This character string must be large enough to hold `CDF_COPYRIGHT_LEN` characters and will be blank padded if necessary. This string will contain a linefeed character after each line of the copyright notice. (It can be printed without modification but may contain many trailing blanks.)
- `status` The completion status code. Chapter 7 explains how to interpret status codes.

5.3.1 Example(s)

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

```
.
```

```
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                             ! CDF identifier.
INTEGER*4 status                         ! Returned status code.
INTEGER*4 version                        ! CDF version number.
INTEGER*4 release                        ! CDF release number.
CHARACTER copyright*(CDF_COPYRIGHT_LEN)  ! Copyright notice.
INTEGER*4 last_char                      ! Last character position
                                         ! actually used in the copyright.
INTEGER*4 start_char                     ! Starting character position
                                         ! in a line of the copyright.
CHARACTER lf*1                           ! Linefeed character.
.
.
CALL CDF_doc (id, version, release, copyright, status)
IF (status .LT. CDF_OK) THEN                ! INFO status codes ignored
  CALL UserStatusHandler (status)
ELSE
  WRITE (6,101) version, release
101   FORMAT (' ','Version: ',I3,' Release:',I3)

  last_char = CDF_COPYRIGHT_LEN
  DO WHILE (copyright(last_char:last_char) .EQ. ' ')
    last_char = last_char - 1
  END DO

  lf = CHAR(10)

  start_char = 1
  DO i = 1, last_char
    IF (copyright(i:i) .EQ. lf) THEN
      WRITE (6,301) copyright(start_char:i-1)
301       FORMAT (' ',A)
      start_char = i + 1
    END IF
  END DO
END IF
.
.
```

5.4 CDF_inquire

```
SUBROUTINE CDF_inquire (id, num_dims, dim_sizes, encoding, majority,
1                       max_rec, num_vars, num_attrs, status)

INTEGER*4 id                      ! in  -- CDF identifier.
INTEGER*4 num_dims                ! out -- Number of dimensions, rVariables.
```

```
INTEGER*4 dim_sizes(CDF_MAX_DIMS)  ! out -- Dimension sizes, rVariables.
INTEGER*4 encoding                 ! out -- Data encoding.
INTEGER*4 majority                 ! out -- Variable majority.
INTEGER*4 max_rec                  ! out -- Maximum record number in the CDF,
                                           rVariables.
INTEGER*4 num_vars                 ! out -- Number of rVariables in
                                           the CDF.
INTEGER*4 num_attrs                ! out -- Number of attributes in the CDF.
INTEGER*4 status                   ! out -- Completion status.
```

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

The arguments to `CDF_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`.
- `num_dims` — The number of dimensions in the CDF for the rVariables.
- `dim_sizes` — The dimension sizes for the rVariables. `dim_sizes` is a 1-dimensional array containing one element per dimension. Each element of `dim_sizes` receives the corresponding dimension size. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
- `encoding` — The encoding of the variable data and attribute entry data. The encodings are defined in Section 4.6.
- `majority` — The majority of the variable data. The majorities are defined in Section 4.8.
- `max_rec` — The maximum record number written to an rVariable in the CDF. Note that the maximum record number written is also kept separately for each rVariable in the CDF. The value of `max_rec` is the largest of these. Some rVariables may have fewer records actually written. `CDF_lib` (Internal Interface) may be used to inquire the maximum record written for an individual rVariable (see Section 6).
- `num_vars` — The number of rVariables in the CDF.
- `num_attrs` — The number of attributes in the CDF.
- `status` — The completion status code. Chapter 7 explains how to interpret status codes.

5.4.1 Example(s)

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

```
.
.
```

```
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                       ! CDF identifier.
INTEGER*4 status                   ! Returned status code.
INTEGER*4 num_dims                 ! Number of dimensions, rVariables.
INTEGER*4 dim_sizes(CDF_MAX_DIMS)  ! Dimension sizes, rVariables
                                   ! (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.
INTEGER*4 num_vars                 ! Number of rVariables in CDF.
INTEGER*4 num_attrs                ! Number of attributes in CDF.
.
.
CALL CDF_inquire (id, num_dims, dim_sizes, encoding, majority,
.                 max_rec, num_vars, num_attrs, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.5 CDF_close

```
SUBROUTINE CDF_close (id, status)

INTEGER*4 id            ! in  -- CDF identifier.
INTEGER*4 status        ! out -- Completion status.
```

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

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

The arguments to `CDF_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`.

`status` The completion status code. Chapter 7 explains how to interpret status codes.

5.5.1 Example(s)

The following example will close an open CDF.

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

5.6 CDF_delete

```
SUBROUTINE CDFdelete (id, status)

INTEGER*4 id             ! in  -- CDF identifier.
INTEGER*4 status         ! out -- Completion status.
```

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

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

The arguments to `CDF_delete` 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`.
`status`	The completion status code. Chapter 7 explains how to interpret status codes.

5.6.1 Example(s)

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

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
```

```
.
.
CALL CDF_open ('test2', id, status)
IF (status .LT. CDF_OK) THEN         ! INFO status codes ignored.
  CALL UserStatusHandler (status)
ELSE
  CALL CDF_delete (id, status)
  IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
END IF
.
.
```

5.7 CDF_error

```
SUBROUTINE CDF_error (status, message)

INTEGER*4 status                       ! in  -- Status code.
CHARACTER message*(CDF_STATUSTEXT_LEN)  ! out -- Explanation text for
                                                the status code.
```

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

The arguments to `CDF_error` are defined as follows:

`status` The status code to check.

`message` The explanation of the status code. This character string must be large enough to hold `CDF_STATUSTEXT_LEN` characters and will be blank padded if necessary.

5.7.1 Example(s)

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

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                         ! CDF identifier.
INTEGER*4 status                     ! Returned status code.
CHARACTER text*(CDF_STATUSTEXT_LEN)  ! Explanation text.
INTEGER*4 last_char                  ! Last character position
                                     ! actually used in the copyright.
.
.
```

```
     CALL CDF_open ('giss_wetl', id, status)
     IF (status .LT. CDF_WARN) THEN         ! INFO and WARNING codes ignored.
       CALL CDF_error (status, text)
       last_char = CDF_STATUSTEXT_LEN
       DO WHILE (text(last_char:last_char) .EQ. ' ')
         last_char = last_char - 1
       END DO
       WRITE (6,101) text(1:last_char)
 101   FORMAT (' ','ERROR> ',A)
     END IF
     .
     .
```

5.8 CDF_attr_create

```
SUBROUTINE CDF_attr_create (id, attr_name, attr_scope, attr_num, status)
INTEGER*4 id                ! in  -- CDF identifier.
CHARACTER attr_name*(*)     ! in  -- Attribute name.
INTEGER*4 attr_scope        ! in  -- Scope of attribute.
INTEGER*4 attr_num          ! out -- Attribute number.
INTEGER*4 status            ! out -- Completion status.
```

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

The arguments to `CDF_attr_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`.
`attr_name`	The name of the attribute to create. This may be at most `CDF_ATTR_NAME_LEN` characters. Attribute names are case-sensitive.
`attr_scope`	The scope of the new attribute. Specify one of the scopes described in Section 4.10.
`attr_num`	The number assigned to the new attribute. This number must be used in subsequent CDF function calls when referring to this attribute. An existing attribute's number may be determined with the `CDF_attr_num` function.
`status`	The completion status code. Chapter 7 explains how to interpret status codes.

5.8.1 Example(s)

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

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
CHARACTER UNITS_attr_name*5         ! Name of "Units" attribute.
INTEGER*4 UNITS_attr_num            ! "Units" attribute number.
INTEGER*4 TITLE_attr_num            ! "TITLE" attribute number.
INTEGER*4 TITLE_attr_scope          ! "TITLE" attribute scope.

DATA UNITS_attr_name/'Units'/, TITLE_attr_scope/GLOBAL_SCOPE/
.
.
CALL CDF_attr_create (id, 'TITLE', TITLE_attr_scope, TITLE_attr_num, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL CDF_attr_create (id, UNITS_attr_name, VARIABLE_SCOPE, UNITS_attr_num,
1                     status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.9 CDF_attr_num

```
INTEGER*4 FUNCTION CDF_attr_num (id, attr_name)

INTEGER*4 id                ! in --  CDF id.
CHARACTER attr_name*(*)     ! in -- attribute name.
```

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

The arguments to `CDF_attr_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`.

`attr_name` The name of the attribute for which to search. This may be at most `CDF_ATTR_NAME_LEN` characters. Attribute names are case-sensitive.

`CDF_attr_num` may be used as an embedded function call when an attribute number is needed. `CDF_attr_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_attr_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_attr_num from cdf.inc, but be sure to declare CDF_attr_num in all routines that use it.

5.9.1 Example(s)

In the following example the attribute named pressure will be renamed PRESSURE with CDF_attr_num being used as an embedded function call.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
.
.
CALL CDF_attr_rename (id, CDF_attr_num(id,'pressure'), 'PRESSURE', status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

Note that if the attribute pressure did not exist in the CDF, the call to CDF_attr_num would have returned an error code. Passing that error code to CDF_attr_rename as an attribute number would have resulted in CDF_attr_rename also returning an error code. CDF_attr_rename is described in Section 5.10.

5.10 CDF_attr_rename

```
SUBROUTINE CDF_attr_rename (id, attr_num, attr_name, status)

INTEGER*4 id              ! in  -- CDF identifier.
INTEGER*4 attr_num        ! in  -- Attribute number.
CHARACTER attr_name*(*)   ! in  -- New attribute name.
INTEGER*4 status          ! out -- Completion status.
```

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

The arguments to CDF_attr_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.

attr_num — The number of the attribute to rename. This number may be determined with a call to CDF_attr_num (see Section 5.9).

attr_name The new attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters. Attribute names are case-sensitive.

status The completion status code. Chapter 7 explains how to interpret status codes.

5.10.1 Example(s)

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

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
.
.
CALL CDF_attr_rename (id, CDF_attr_num(id,'LAT'), 'LATITUDE', status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.11 CDF_attr_inquire

```
SUBROUTINE CDF_attr_inquire (id, attr_num, attr_name, attr_scope, max_entry,
1                            status)
INTEGER*4 id                              ! in  -- CDF identifier.
INTEGER*4 attr_num                        ! in  -- Attribute number.
CHARACTER attr_name*(CDF_ATTR_NAME_LEN)   ! out -- Attribute name.
INTEGER*4 attr_scope                      ! out -- Attribute scope.
INTEGER*4 max_entry                       ! out -- Maximum entry
                                          !        number for attribute.
INTEGER*4 status                          ! out -- Completion status.
```

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

The arguments to `CDF_attr_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`.

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

attr_name The attribute's name. This character string must be large enough to hold `CDF_ATTR_NAME_LEN` characters and will be blank padded if necessary.

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

max_entry For gAttributes this is the maximum entry number used. For vAttributes this is the maximum rEntry number used. In either case this may not correspond with the number of entries (if some entry numbers were not used). The number of entries actually used may be inquired with the `CDF_lib` function (see Section 6). If no entries exist for the attribute, then a value of zero (0) will be passed back.

status The completion status code. Chapter 7 explains how to interpret status codes.

5.11.1 Example(s)

The following example displays the name of each attribute in a CDF. The number of attributes in the CDF is first determined using `CDF_inquire`. Note that attribute numbers start at one (1) and are consecutive.

```
.
.
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 .LT. CDF_OK) THEN      ! INFO status codes ignored.
    CALL UserStatusHandler (status)
  ELSE
```

```
      WRITE (6,10) attr_name
10    FORMAT (' ',A)
    END IF
  END DO
  .
  .
```

5.12 CDF_attr_entry_inquire

```
SUBROUTINE CDF_attr_entry_inquire (id, attr_num, entry_num, data_type,
1                                  num_elements, status)
INTEGER*4 id               ! in  -- CDF identifier.
INTEGER*4 attr_num         ! in  -- Attribute number.
INTEGER*4 entry_num        ! in  -- Entry number.
INTEGER*4 data_type        ! out -- Data type.
INTEGER*4 num_elements     ! out -- Number of elements (of the data type).
INTEGER*4 status           ! out -- Completion status.
```

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

The arguments to `CDF_attr_entry_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`.
`attr_num`	The attribute number for which to inquire an entry. This number may be determined with a call to `CDF_attr_num` (see Section 5.9).
`entry_num`	The entry number to inquire. If the attribute is global in scope, this is simply the entry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the entry).
`data_type`	The data type of the specified entry. The data types are defined in Section 4.5.
`num_elements`	The number of elements of the data type. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in the string. For all other data types, this is the number of elements in an array of that data type.
`status`	The completion status code. Chapter 7 explains how to interpret status codes.

5.12.1 Example(s)

The following example inquires each entry for an attribute. Note that entry numbers need not be consecutive — not every entry number between one (1) and the maximum entry number must exist. For this reason NO_SUCH_ENTRY is an expected error code. Note also that if the attribute is variable in scope, then the rEntry numbers are actually rVariable numbers.

```
   .
   .
   INCLUDE '<path>cdf.inc'
   .
   .
   INTEGER*4 id                         ! CDF identifier.
   INTEGER*4 status                     ! Returned status code.
   INTEGER*4 attr_n                     ! Attribute number.
   INTEGER*4 entryN                     ! Entry number.
   CHARACTER attr_name*(CDF_ATTR_NAME_LEN)  ! Attribute name.
   INTEGER*4 attr_scope                 ! Attribute scope.
   INTEGER*4 max_entry                  ! Maximum entry number used.
   INTEGER*4 data_type                  ! Data type.
   INTEGER*4 num_elems                  ! Number of elements (of the
                                        ! data type).
   .
   .
   attr_n = CDF_attr_num (id, 'TMP')
   IF (attr_n .LT. 1) CALL UserStatusHandler (attr_n)  ! If less than one (1),
                                                        ! then it must be a
                                                        ! warning/error code.

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

   DO entryN = 1, max_entry
     CALL CDF_attr_entry_inquire (id, attr_n, entryN, data_type, num_elems,
     1                            status)
     IF (status .LT. CDF_OK) THEN
       IF (status .NE. NO_SUCH_ENTRY) CALL UserStatusHandler (status)
     ELSE
C      (process entries)
       .
       .
     END IF
   END DO
```

5.13 CDF_attr_put

```
SUBROUTINE CDF_attr_put (id, attr_num, entry_num, data_type, num_elements,
1                        value, status)
```

```
INTEGER*4 id,             ! in --  CDF identifier.
INTEGER*4 attr_num        ! in --  Attribute number.
INTEGER*4 entry_num       ! in --  Entry number.
INTEGER*4 data_type       ! in --  Data type of the entry.
INTEGER*4 num_elements    ! in --  Number of elements (of the data type).
<type>    value           ! in --  Value (<type> depends on the data type
                          !        of the entry).
INTEGER*4 status          ! out -- Completion status.
```

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

The arguments to `CDF_attr_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`.

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

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

`data_type` The data type of the entry. Specify one of the data types defined in Section 4.5.

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

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

`status` The completion status code. Chapter 7 explains how to interpret status codes.

5.13.1 Example(s)

The following example writes two attribute entries. The first is to gEntry number one (1) of the gAttribute `TITLE`. The second is to vAttribute `VALIDs` for the rEntry that corresponds to the rVariable `TMP`.

```
.
.
INCLUDE '<path>cdf.inc'
.
.
PARAMETER TITLE_LEN = 10          ! Length of CDF title.
.
```

```
.
INTEGER*4 id                        ! CDF identifier.
INTEGER*4 status                    ! Returned status code.
INTEGER*4 entry_num                 ! Entry number.
INTEGER*4 num_elements              ! Number of elements (of data type).
CHARACTER title*(TITLE_LEN)         ! Value of TITLE attribute, rEntry
                                    ! number 1.
INTEGER*2 TMPvalids(2)              ! Value(s) of VALIDs attribute,
                                    ! rEntry for rVariable TMP

DATA title/'CDF title.'/, TMPvalids/15,30/
.
.
entry_num = 1
CALL CDF_attr_put (id, CDF_attr_num(id,'TITLE'), entry_num, CDF_CHAR,
1                  TITLE_LEN, title, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
num_elements = 2
CALL CDF_attr_put (id, CDF_attr_num(id,'VALIDs'), CDF_var_num(id,'TMP'),
1                  CDF_INT2, num_elements, TMPvalids, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

5.14 CDF_attr_get

```
SUBROUTINE CDF_attr_get (id, attr_num, entry_num, value, status)

INTEGER*4 id             ! in  -- CDF identifier.
INTEGER*4 attr_num       ! in  -- Attribute number.
INTEGER*4 entry_num      ! in  -- Entry number.
<type>    value          ! out -- Value (<type> is dependent on the data
                         !        type of the entry).
INTEGER*4 status         ! out -- Completion status.
```

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

The arguments to CDF_attr_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.

attr_num The attribute number. This number may be determined with a call to CDF_attr_num (see Section 5.9).

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

`value` The value read. This buffer must be large enough to hold the value. The subroutine `CDF_attr_entry_inquire` would be used to determine the entry data type and number of elements (of that data type). The value is read from the CDF and placed into `value`.

`status` The completion status code. Chapter 7 explains how to interpret status codes.

5.14.1 Example(s)

The following example displays the value of the `UNITS` attribute for the rEntry corresponding to the `PRES_LVL` rVariable (but only if the data type is `CDF_CHAR`).

```
.
.
INCLUDE '<path>cdf.inc'
.
.
INTEGER*4 id                          ! CDF identifier.
INTEGER*4 status                      ! Returned status code.
INTEGER*4 attr_n                      ! Attribute number.
INTEGER*4 entryN                      ! Entry number.
INTEGER*4 data_type                   ! Data type.
INTEGER*4 num_elems                   ! Number of elements (of data type).
CHARACTER buffer*100                  ! Buffer to receive value (in
                                      ! this case it is assumed that 100
                                      ! characters is enough).
.
.
attr_n = CDF_attr_Num (id, 'UNITS')
IF (attr_n .LT. 0) CALL UserStatusHandler (attr_n)  ! If less than one (1),
                                                     ! then it must be a
                                                     ! warning/error code.

entryN = CDF_var_num (id, 'PRES_LVL')              ! The rEntry number is
                                                    ! the rVariable number.
IF (entryN .LT. 0) CALL UserStatusHandler (entryN) ! If less than one (1),
                                                    ! then it must be a
                                                    ! warning/error code.

CALL CDF_attr_entry_inquire (id, attr_n, entryN, data_type, num_elems,
1                            status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

IF (data_type .EQ. CDF_CHAR) THEN
  CALL CDF_attr_get (id, attr_n, entryN, buffer, status)
```

```
     IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
     WRITE (6,10) buffer(1:num_elems)
10   FORMAT (' ',A)
   END IF
   .
   .
```

5.15 CDF_var_create

```
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 4.5.

`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 4.9.

`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 4.9. 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. Chapter 7 explains how to interpret status codes.

5.15.1 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)
.
.
```

5.16 CDF_var_num

```
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.

5.16.1 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 5.18.

5.17 CDF_var_rename

```
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 5.16).
`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. Chapter 7 explains how to interpret status codes.

5.17.1 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
.
.
```

5.18 CDF_var_inquire

```
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 5.16).
`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 4.5.
`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 4.9.
`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 4.9. For 0-dimensional rVariable this argument is ignored (but a place holder is necessary).
`status`	The completion status code. Chapter 7 explains how to interpret status codes.

5.18.1 Example(s)

The following example inquires about an rVariable named `HEAT_FLUX` in a CDF. Note that the rVariable name returned by `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)
.
```

.

5.19 CDF_var_put

```
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 5.16).
`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 `value`.
`status`	The completion status code. Chapter 7 explains how to interpret status codes.

5.19.1 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.)

5.20 CDF_var_get

```
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 5.16).
`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 `value`.
`status`	The completion status code. Chapter 7 explains how to interpret status codes.

5.20.1 Example(s)

The following example will read and hold an entire record of data from an rVariable. The CDF's rVariables are 3-dimensional with sizes `[180,91,10]`. For 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
  .
  .
```

5.21 CDF_var_hyper_put

```
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. The Concepts chapter in the CDF User's Guide 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 5.16).

`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[1] (e.g., an interval of 2 means write to every other record).

`indices` The indices (within each record) at which to start writing. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each ele-

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

ment 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[2] (e.g., an interval of 2 means write to every other value). `intervals` is a 1-dimensional array containing one element per rVariable dimension. Each element of `intervals` specifies the corresponding dimension interval. 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 in `buffer` are written to the CDF.

`status` The completion status code. Chapter 7 explains how to interpret status codes.

5.21.1 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 are `[NOVARY,VARY]`; and the data type is `CDF_INT2`. This example is similar to the example in Section 5.19 except that it uses a single call to `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),
```

[2] Again, not the best term.

```
                                                   ! 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)
.
.
```

5.22 CDF_var_hyper_get

```
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. The Concepts chapter in the CDF User's Guide 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 5.16).

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 `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. Chapter 7 explains how to interpret status codes.

5.22.1 Example(s)

The following example will read an entire record of data from an rVariable. The CDF's rVariables are 3-dimensional with sizes `[180,91,10]`, and 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 5.20 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.

5.23 CDF_var_close

```
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 the Concepts chapter in the CDF User's Guide. Note that there is not a function that opens an rVariable. The CDF library automatically opens an rVariable when it is accessed by an application (unless it is already open).

The arguments to `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 5.16).

`status` The completion status code. Chapter 7 explains how to interpret status codes.

5.23.1 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)
.
.
```

Chapter 6

Internal Interface — CDF_lib

The Internal Interface consists of only one function, CDF_lib.[1] 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.

6.1 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.
```

[1] See Section 6.5.1 for an ugly exception to this.