NEGtoPOSfp0on	Convert -0.0 to 0.0 when read from the CDF.
NEGtoPOSfp0off	Do not convert -0.0 to 0.0 when read from the CDF.

5.7.12 Operational Limits

These are limits within the CDF library. If you reach one of these limits, please contact CDF User Support.

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

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

5.7.13 Limits of Names and Other Character Strings

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

5.8 Fortran Function Reference

The following sections describe the CDF library routines callable from Fortran applications. Most routines return a status code of type INTEGER*4 (see Section 5.9). Each routine except CDF_lib is part of the Standard Interface — CDF_lib is part of the Internal Interface. An application can use both interfaces when necessary. Note that zVariables and vAttribute zEntries are only accessible via the Internal Interface.

5.8.1 CDF_create (Standard Interface)

```
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). **NOTE:** On UNIX systems 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 5.7.4.
majority	The majority for variable data. Specify one of the majorities described in Section 5.7.6.
id	The identifier for the created CDF. This identifier must be used in all subsequent operations on the CDF.
status	The completion status code. Section 5.9 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 to single-file.

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 5.8.5).

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`.

5.8.2 CDF_open (Standard Interface)

```
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).

NOTE: On UNIX systems 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. Section 5.9 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 5.8.5).

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)
.
.
```

5.8.3 CDF_doc (Standard Interface)

```
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. Section 5.9 explains how to interpret status codes.

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
    .
    .
```

5.8.4 CDF_inquire (Standard Interface)

```
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 5.7.4.
`majority`	The majority of the variable data. The majorities are defined in Section 5.7.6.
`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 5.8.25).

num_vars	The number of rVariables in the CDF.
num_attrs	The number of attributes in the CDF.
status	The completion status code. Section 5.9 explains how to interpret status codes.

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)
.
.
```

5.8.5 CDF_close (Standard Interface)

```
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. Section 5.9 explains how to interpret status codes.

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)
.
.
```

5.8.6 CDF_delete (Standard Interface)

```
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. Section 5.9 explains how to interpret status codes.

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
.
.
```

5.8.7 CDF_error (Standard Interface)

```
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). Section 5.9 explains how to interpret status codes, and Appendix C 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.

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
      .
      .
```

5.8.8 CDF_attr_create (Standard Interface)

```
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 5.7.8.
`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. Section 5.9 explains how to interpret status codes.

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)
.
.
```

5.8.9 CDF_attr_num (Standard Interface)

```
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.

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 5.8.10.

5.8.10 CDF_attr_rename (Standard Interface)

```
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 5.8.9).
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. Section 5.9 explains how to interpret status codes.

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)
.
.
```

5.8.11 CDF_attr_inquire (Standard Interface)

```
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 5.8.12).

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 5.8.9).
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 5.7.8.
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 5.8.25).
status	The completion status code. Section 5.9 explains how to interpret status codes.

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
   .
   .
```

5.8.12 CDF_attr_entry_inquire (Standard Interface)

```
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 5.8.11). 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 5.8.9).
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 5.7.3.
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. Section 5.9 explains how to interpret status codes.

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
```

5.8.13 CDF_attr_put (Standard Interface)

```
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 5.8.9).
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 5.7.3.
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 memory address value.
status	The completion status code. Section 5.9 explains how to interpret status codes.

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)
.
.
```

5.8.14 CDF_attr_get (Standard Interface)

```
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 5.8.9).

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 memory at address value.

Example(s)

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

```
.
.
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
   .
   .
```

5.8.15 CDF_attr_correct_scope (Standard Interface)

```
SUBROUTINE CDF_attr_correct_scope (id, attr_num, attr_scope, status)

INTEGER*4 id                  ! in  -- CDF identifier.
INTEGER*4 attr_num            ! in  -- Attribute number.
INTEGER*4 attr_scope          ! in  -- New scope of attribute.
INTEGER*4 status              ! out -- Completion status.
```

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

The arguments to `CDF_attr_correct_scope` 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 5.8.9).
`attr_scope`	The new scope for the attribute. Specify one of the definite scopes defined in Section 5.7.8.
`status`	The completion status code. Section 5.9 explains how to interpret status codes.

IMPORTANT NOTE

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

Example(s)

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