```
INTEGER*4 dim_sizes(2)            ! Dimension sizes.
INTEGER*4 encoding                ! Data encoding.
INTEGER*4 majority                ! Variable data majority.
INTEGER*4 format                  ! Format of CDF.

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

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

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

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

`PUT_`	The first operation to be performed. In this case an item is going to be put to the "current" CDF (a new format). `PUT_` is defined in `cdf.inc` (as are all CDF constants). It was not necessary to select a current CDF since the call to `CDF_create` implicitly selected the CDF created as the current CDF.[2] This is the case since all of the Standard Interface functions actually call the Internal Interface to perform their operations.
`CDF_FORMAT_`	The item to be put. In this case it is the CDF's format. One required argument for this operation follows.
`format`	The actual format for the CDF. Depending on the item being put, one or more arguments would have been necessary. In this case only one argument is necessary.
`NULL_`	This argument could have been one of two things. It could have been another item to put (followed by the arguments required for that item) or it could have been a new operation to perform. In this case it is a new operation to perform — the `NULL_` operation. `NULL_` indicates the end of the call to `CDF_lib`. Specifying `NULL_` at the end of the argument list is required because not all Fortran compilers/operating systems provide the ability for a called function to determine how many arguments were passed in by the calling function.
`status`	The completion status code. Chapter 7 explains how to interpret status codes.

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

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

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

The purpose of each argument is as follows.

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

`NULL_` This argument could have been either another item to put or a new operation to perform. Here it is another operation to perform — the `NULL_` operation which ends the call to `CDF_lib`.

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

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

6.2 Current Objects/States (Items)

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

CDF (object)

A CDF operation is always performed on the current CDF. The current CDF is implicitly selected whenever a CDF is opened or created. The current CDF may be explicitly selected using the `<SELECT_,CDF_>`[3] operation. There is no current CDF until one is opened or created (which implicitly selects it) or until one is explicitly selected.[4]

rVariable (object)

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

zVariable (object)

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

attribute (object)

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

gEntry number (state)

A gAttribute gEntry operation is always performed on the current gEntry number in the current CDF for the current attribute in that CDF. For each open CDF a current gEntry number is

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

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

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

rEntry number (state)

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

zEntry number (state)

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

record number, rVariables (state)

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

record count, rVariables (state)

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

record interval, rVariables (state)

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

dimension indices, rVariables (state)

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

dimension counts, rVariables (state)

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

dimension intervals, rVariables (state)

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

sequential value, rVariable (state)

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

record number, zVariable (state)

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

record count, zVariable (state)

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

record interval, zVariable (state)

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

dimension indices, zVariable (state)

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

dimension counts, zVariable (state)

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

dimension intervals, zVariable (state)

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

sequential value, zVariable (state)

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

status code (state)

> When inquiring the explanation of a CDF status code, the text returned is always for the current status code. One current status code is maintained for the entire CDF library (regardless of the number of open CDFs). The current status code may be selected using the `<SELECT_,CDF_STATUS_>` operation. There is no default current status code. Note that the current status code is NOT the status code from the last operation performed.[5]

6.3 Returned Status

`CDF_lib` returns a status code of type `INTEGER*4`. Since more than one operation may be performed with a single call to `CDF_lib`, the following rules apply.

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

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

Chapter 7 explains how to interpret status codes. Appendix A lists the possible status codes and the type of each: error, warning, or informational.

6.4 Indentation/Style

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

```
CALL CDF_lib (CREATE_, CDF_, CDF_path, num_dims, dim_sizes, id,
1             PUT_, CDF_FORMAT_, format,
2                   CDF_MAJORITY_, majority,
3             CREATE_, ATTR_, attr_name, scope, attr_num,
4                      rVAR_, var_name, data_type, num_elements,
5                             rec_vary, dim_varys, var_num,
6             NULL_, status)
```

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

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

```
CALL CDF_lib (CREATE_, CDF_, CDF_path, num_dims, dim_sizes, id, PUT_,
1             CDF_FORMAT_, format, CDF_MAJORITY_, majority, CREATE_,
2             ATTR_, attr_name, scope, attr_num, rVAR_, var_name, data_type,
3             num_elements, rec_vary, dim_varys, var_num, NULL_, status)
```

The need for proper indentation to ensure the readability of your applications should be obvious.[6]

6.5 Syntax

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

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

The syntax for CDF_lib is as follows:

```
   CALL CDF_lib (op1, item1, arg1, arg2, ...argN,
  +                   item2, arg1, arg2, ...argN,
                        .
                        .
  +                   itemN, arg1, arg2, ...argN,
  +              op2, item1, arg1, arg2, ...argN,
  +                   item2, arg1, arg2, ...argN,
                        .
                        .
  +                   itemN, arg1, arg2, ...argN,
                 .
                 .
  +              opN, item1, arg1, arg2, ...argN,
  +                   item2, arg1, arg2, ...argN,
                        .
                        .
  +                   itemN, arg1, arg2, ...argN,
  +              NULL_, status)
```

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

6.5.1 Macintosh, MPW Fortran

The MPW Fortran compiler does not allow variable length argument lists such as those used by `CDF_lib`.[7] For that reason, a number of additional Internal Interface subroutines are available named `CDF_lib_4`, `CDF_lib_5`, etc. Each of these subroutines expects the number of arguments indicated by their names. The maximum number of arguments is at least 25 (corresponding to `CDF_lib_25`) but can be increased if necessary by contacting CDFsupport. Using these subroutines, the second example shown in this section would be as follows:

```
   .
   .
   CALL CDF_lib_15 (CREATE_, CDF_, CDF_path, num_dims, dim_sizes, id,
   1                PUT_, CDF_ENCODING_, encoding,
   2                      CDF_MAJORITY_, majority,
   3                      CDF_FORMAT_, format,
   4                NULL_, status)
   IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
   .
   .
```

[7] If you know of a way to make MPW Fortran accept variable length argument lists, by all means let us know. We don't like having to do this any more than you do.

Note that `CDF_lib` may still be used but with the same number of arguments for each occurrence.

6.6 Operations/Items/Arguments...

The supported operations are as follows:

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

For each operation the supported items, required arguments, and required preselected objects/states are listed in the following sections. The required preselected objects/states are those objects/states that must be selected (typically with the `SELECT_` operation) before a particular operation may be performed. Note that some of the required preselected objects/states have default values as described beginning on page 58.

`<CLOSE_,CDF_>`

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

There are no required arguments.

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

`<CLOSE_,rVAR_>`

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

There are no required arguments.

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

`<CLOSE_,zVAR_>`

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

There are no required arguments.

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

`<CONFIRM_,ATTR_>`

Confirms the current attribute (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 attr_num`

Attribute number.

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

<CONFIRM_,ATTR_EXISTENCE_>

Confirms the existence of the named attribute (in the current CDF). If the attribute does not exist, an error code will be returned. In any case the current attribute is not affected. Required arguments are as follows:

in: CHARACTER attr_name*(*)

The attribute name. This may be at most CDF_ATTR_NAME_LEN characters.

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

<CONFIRM_,CDF_CACHESIZE_>

Confirms the number of cache buffers being used for the ".cdf" file (for the current CDF). The Concepts chapter in the CDF User's Guide describes the caching scheme used by the CDF library in greater detail. Required arguments are as follows:

out: INTEGER*4 num_buffers

The number of cache buffers being used.

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

<CONFIRM_,CDF_DECODING_>

Confirms the decoding for the current CDF. Required arguments are as follows:

out: INTEGER*4 decoding

The decoding. The decodings are described in Section 4.7.

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

<CONFIRM_,CDF_NAME_>

Confirms the pathname of the current CDF. Required arguments are as follows:

out: CHARACTER CDF_path*(CDF_PATHNAME_LEN)

Pathname of the CDF. This character string must be large enough to hold CDF_PATHNAME_LEN characters and will be blank padded if necessary.

UNIX: For the proper operation of CDF_lib, CDF_path MUST be of type CHARACTER.

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

<CONFIRM_,CDF_NEGtoPOSfp0_MODE_>

Confirms the -0.0 to 0.0 mode for the current CDF. Required arguments are as follows:

out: INTEGER*4 mode

The -0.0 to 0.0 mode. The -0.0 to 0.0 modes are described in Section 4.13.

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

<CONFIRM_,CDF_READONLY_MODE_>

Confirms the read-only mode for the current CDF. Required arguments are as follows:

out: `INTEGER*4 mode`

The read-only mode. The read-only modes are described in Section 4.11.

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

`<CONFIRM_,CDF_zMODE_>`

Confirms the zMode for the current CDF. Required arguments are as follows:

out: `INTEGER*4 mode`

The zMode. The zModes are described in Section 4.12.

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

`<CONFIRM_,CURgENTRY_EXISTENCE_>`

Confirms the existence of the gEntry at the current gEntry number for the current attribute (in the current CDF). If the gEntry does not exist, an error code will be returned.

There are no required arguments.

The required preselected objects/states are the current CDF, its current attribute, and its current gEntry number.

NOTE: Only use this operation on gAttributes. An error will occur if used on a vAttribute.

`<CONFIRM_,CURrENTRY_EXISTENCE_>`

Confirms the existence of the rEntry at the current rEntry number for the current attribute (in the current CDF). If the rEntry does not exist, an error code will be returned.

There are no required arguments.

The required preselected objects/states are the current CDF, its current attribute, and its current rEntry number.

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<CONFIRM_,CURzENTRY_EXISTENCE_>`

Confirms the existence of the zEntry at the current zEntry number for the current attribute (in the current CDF). If the zEntry does not exist, an error code will be returned.

There are no required arguments.

The required preselected objects/states are the current CDF, its current attribute, and its current zEntry number.

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<CONFIRM_,gENTRY_>`

Confirms the current gEntry number for all attributes in the current CDF. Required arguments are as follows:

out: `INTEGER*4 entry_num`

gEntry number.

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

`<CONFIRM_,gENTRY_EXISTENCE_>`

Confirms the existence of the specified gEntry for the current attribute (in the current CDF). If the gEntry does not exist, an error code will be returned. In any case the current gEntry number is not affected. Required arguments are as follows:

in: `INTEGER*4 entry_num`

The gEntry number.

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

NOTE: Only use this operation on gAttributes. An error will occur if used on a vAttribute.

`<CONFIRM_,rENTRY_>`

Confirms the current rEntry number for all attributes in the current CDF. Required arguments are as follows:

out: `INTEGER*4 entry_num`

rEntry number.

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

`<CONFIRM_,rENTRY_EXISTENCE_>`

Confirms the existence of the specified rEntry for the current attribute (in the current CDF). If the rEntry does not exist, an error code will be returned. In any case the current rEntry number is not affected. Required arguments are as follows:

in: `INTEGER*4 entry_num`

The rEntry number.

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

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<CONFIRM_,rVAR_>`

Confirms the current rVariable (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 var_num`

rVariable number.

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

`<CONFIRM_,rVAR_CACHESIZE_>`

Confirms the number of cache buffers being used for the current rVariable's file (of the current CDF). This operation is not applicable to a single-file CDF. The Concepts chapter in the CDF User's Guide describes the caching scheme used by the CDF library in greater detail. Required arguments are as follows:

out: `INTEGER*4 num_buffers`

The number of cache buffers being used.

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

<CONFIRM_,rVAR_EXISTENCE_>

Confirms the existence of the named rVariable (in the current CDF). If the rVariable does not exist, an error code will be returned. In any case the current rVariable is not affected. Required arguments are as follows:

in: `CHARACTER var_name*(*)`

The rVariable name. This may be at most `CDF_VAR_NAME_LEN` characters.

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

<CONFIRM_,rVAR_SEQPOS_>

Confirms the current sequential value for sequential access for the current rVariable (in the current CDF). Note that a current sequential value is maintained for each rVariable individually. Required arguments are as follows:

out: `INTEGER*4 rec_num`

Record number.

out: `INTEGER*4 indices(*)`

Dimension indices. `indices` is a 1-dimensional array containing one element per rVariable dimension. Each element of `indices` receives the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).

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

<CONFIRM_,rVARs_DIMCOUNTS_>

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

out: `INTEGER*4 counts(*)`

Dimension counts. `counts` is a 1-dimensional array containing one element per dimension. Each element of `counts` receives the corresponding dimension count.

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

<CONFIRM_,rVARs_DIMINDICES_>

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

out: `INTEGER*4 indices(*)`

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

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

<CONFIRM_,rVARs_DIMINTERVALS_>

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

out: `INTEGER*4 intervals(*)`

Dimension intervals. `intervals` is a 1-dimensional array containing one element per dimension. Each element of `intervals` receives the corresponding dimension interval.

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

<CONFIRM_,rVARs_RECCOUNT_>

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

out: `INTEGER*4 rec_count`

Record count.

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

<CONFIRM_,rVARs_RECINTERVAL_>

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

out: `INTEGER*4 rec_interval`

Record interval.

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

<CONFIRM_,rVARs_RECNUMBER_>

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

out: `INTEGER*4 rec_num`

Record number.

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

<CONFIRM_,zENTRY_>

Confirms the current zEntry number for all attributes in the current CDF. Required arguments are as follows:

out: `INTEGER*4 entry_num`

zEntry number.

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

<CONFIRM_,zENTRY_EXISTENCE_>

Confirms the existence of the specified zEntry for the current attribute (in the current CDF). If the zEntry does not exist, an error code will be returned. In any case the current zEntry number is not affected. Required arguments are as follows:

in: `INTEGER*4 entry_num`

The zEntry number.

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

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<CONFIRM_,zVAR_>`

Confirms the current zVariable (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 var_num`

zVariable number.

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

`<CONFIRM_,zVAR_CACHESIZE_>`

Confirms the number of cache buffers being used for the current zVariable's file (of the current CDF). This operation is not applicable to a single-file CDF. The Concepts chapter in the CDF User's Guide describes the caching scheme used by the CDF library in greater detail. Required arguments are as follows:

out: `INTEGER*4 num_buffers`

The number of cache buffers being used.

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

`<CONFIRM_,zVAR_EXISTENCE_>`

Confirms the existence of the named zVariable (in the current CDF). If the zVariable does not exist, an error code will be returned. In any case the current zVariable is not affected. Required arguments are as follows:

in: `CHARACTER var_name*(*)`

The zVariable name. This may be at most `CDF_VAR_NAME_LEN` characters.

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

`<CONFIRM_,zVAR_DIMCOUNTS_>`

Confirms the current dimension counts for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are as follows:

out: `INTEGER*4 counts(*)`

Dimension counts. `counts` is a 1-dimensional array containing one element per dimension. Each element of `counts` receives the corresponding dimension count.

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

`<CONFIRM_,zVAR_DIMINDICES_>`

Confirms the current dimension indices for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are as follows:

out: `INTEGER*4 indices(*)`

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

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

<CONFIRM_,zVAR_DIMINTERVALS_>

Confirms the current dimension intervals for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are as follows:

out: `INTEGER*4 intervals(*)`

Dimension intervals. `intervals` is a 1-dimensional array containing one element per dimension. Each element of `intervals` receives the corresponding dimension interval.

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

<CONFIRM_,zVAR_RECCOUNT_>

Confirms the current record count for the current zVariable in the current CDF. Required arguments are as follows:

out: `INTEGER*4 rec_count`

Record count.

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

<CONFIRM_,zVAR_RECINTERVAL_>

Confirms the current record interval for the current zVariable in the current CDF. Required arguments are as follows:

out: `INTEGER*4 rec_interval`

Record interval.

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

<CONFIRM_,zVAR_RECNUMBER_>

Confirms the current record number for the current zVariable in the current CDF. Required arguments are as follows:

out: `INTEGER*4 rec_num`

Record number.

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

<CONFIRM_,zVAR_SEQPOS_>

Confirms the current sequential value for sequential access for the current zVariable (in the current CDF). Note that a current sequential value is maintained for each zVariable individually. Required arguments are as follows:

out: `INTEGER*4 rec_num`

Record number.

out: `INTEGER*4 indices(*)`

Dimension indices. `indices` is a 1-dimensional array containing one element per dimension. Each element of `indices` receives the corresponding dimension index. For 0-dimensional zVariables this argument is ignored (but a place holder is necessary).

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

`<CREATE_,ATTR_>`

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

in: `CHARACTER attr_name*(*)`

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

UNIX: For the proper operation of `CDF_lib`, `attr_name` MUST be of type `CHARACTER`.

in: `INTEGER*4 scope`

Scope of the new attribute. Specify one of the scopes described in Section 4.10.

out: `INTEGER*4 attr_num`

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 `<GET_,ATTR_NUMBER_>` operation.

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

`<CREATE_,CDF_>`

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

in: `CHARACTER CDF_path*(*)`

Pathname of the CDF to be created. (Do not append an extension.) This can 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.

UNIX: For the proper operation of `CDF_lib`, `CDF_path` MUST be of type `CHARACTER`.

in: `INTEGER*4 num_dims`

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

in: `INTEGER*4 dim_sizes(*)`

Dimension sizes for the rVariables. `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). For 0-dimensional rVariables this argument is ignored (but a place holder is necessary). Note that this must be specified even if the CDF will contain only zVariables.

out: `INTEGER*4 id`

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

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

`<CLOSE_,CDF_>` MUST be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk.

There are no required preselected objects/states.

`<CREATE_,rVAR_>`

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

in: `CHARACTER var_name*(*)`

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

UNIX: For the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

in: `INTEGER*4 data_type`

Data type of the new rVariable. Specify one of the data types described in Section 4.5.

in: `INTEGER*4 num_elements`

Number of elements of the data type at each value. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in each string. A string exists at each value of the variable. For the non-character string data types this must be one (1) — multiple elements are not allowed for non-character string data types.

in: `INTEGER*4 rec_vary`

Record variance. Specify one of the variances described in Section 4.9.

in: `INTEGER*4 dim_varys(*)`

Dimension variances. `dim_varys` is a 1-dimensional array containing one element per rVariable dimension. Each element of `dim_varys` specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 4.9. For 0-dimensional rVariables this argument is ignored (but a place holder is necessary).

out: `INTEGER*4 var_num`

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 `<GET_,rVAR_NUMBER_>` operation.

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

`<CREATE_,zVAR_>`

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

in: `CHARACTER var_name*(*)`

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

UNIX: For the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

in: `INTEGER*4 data_type`

Data type of the new zVariable. Specify one of the data types described in Section 4.5.

in: `INTEGER*4 num_elements`

Number of elements of the data type at each value. For character string data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in each string. A string exists at each value of the variable. For the non-character string data types this must be one (1) — multiple elements are not allowed for non-character string data types.

in: `INTEGER*4 num_dims`

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

in: `INTEGER*4 dim_sizes(*)`

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

in: `INTEGER*4 rec_vary`

Record variance. Specify one of the variances described in Section 4.9.

in: `INTEGER*4 dim_varys(*)`

Dimension variances. `dim_varys` is a 1-dimensional array containing one element per dimension. Each element of `dim_varys` specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 4.9. For a 0-dimensional zVariable this argument is ignored (but a place holder is necessary).

out: `INTEGER*4 var_num`

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

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

`<DELETE_,ATTR_>`

Deletes the current attribute (in the current CDF). Note that the attribute's entries are also deleted. The attributes which numerically follow the attribute being deleted are immediately renumbered. When the attribute is deleted, there is no longer a current attribute.

There are no required arguments.

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

`<DELETE_,CDF_>`

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

There are no required arguments.

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

`<DELETE_,gENTRY_>`

Deletes the gEntry at the current gEntry number of the current attribute (in the current CDF). Note that this does not affect the current gEntry number.

There are no required arguments.

The required preselected objects/states are the current CDF, its current attribute, and its current gEntry number.

NOTE: Only use this operation on gAttributes. An error will occur if used on a vAttribute.

`<DELETE_,rENTRY_>`

Deletes the rEntry at the current rEntry number of the current attribute (in the current CDF). Note that this does not affect the current rEntry number.

There are no required arguments.

The required preselected objects/states are the current CDF, its current attribute, and its current rEntry number.

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<DELETE_,rVAR_>`

Deletes the current rVariable (in the current CDF). Note that the rVariable's corresponding rEntries are also deleted (from each vAttribute). The rVariables which numerically follow the rVariable being deleted are immediately renumbered. The rEntries which numerically follow the rEntries being deleted are also immediately renumbered. When the rVariable is deleted, there is no longer a current rVariable.

There are no required arguments.

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

`<DELETE_,zENTRY_>`

Deletes the zEntry at the current zEntry number of the current attribute (in the current CDF). Note that this does not affect the current zEntry number.

There are no required arguments.

The required preselected objects/states are the current CDF, its current attribute, and its current zEntry number.

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<DELETE_,zVAR_>`

Deletes the current zVariable (in the current CDF). Note that the zVariable's corresponding zEntries are also deleted (from each vAttribute). The zVariables which numerically follow the zVariable being deleted are immediately renumbered. The zEntries which numerically follow the zEntries being deleted are also immediately renumbered. When the zVariable is deleted, there is no longer a current zVariable.

There are no required arguments.

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

`<GET_,ATTR_MAXgENTRY_>`

Inquires the maximum gEntry number used for the current attribute (in the current CDF). This does not necessarily correspond with the number of gEntries for the attribute. Required arguments are as follows:

out: `INTEGER*4 max_entry`

The maximum gEntry number for the attribute. If no gEntries exist, then a value of zero (0) will be passed back.

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

NOTE: Only use this operation on gAttributes. An error will occur if used on a vAttribute.

`<GET_,ATTR_MAXrENTRY_>`

Inquires the maximum rEntry number used for the current attribute (in the current CDF). This does not necessarily correspond with the number of rEntries for the attribute. Required arguments are as follows:

out: `INTEGER*4 max_entry`

The maximum rEntry number for the attribute. If no rEntries exist, then a value of zero (0) will be passed back.

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

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<GET_,ATTR_MAXzENTRY_>`

Inquires the maximum zEntry number used for the current attribute (in the current CDF). This does not necessarily correspond with the number of zEntries for the attribute. Required arguments are as follows:

out: `INTEGER*4 max_entry`

The maximum zEntry number for the attribute. If no zEntries exist, then a value of zero (0) will be passed back.

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

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<GET_,ATTR_NAME_>`

Inquires the name of the current attribute (in the current CDF). Required arguments are as follows:

out: `CHARACTER attr_name*(CDF_ATTR_NAME_LEN)`

Attribute name. This character string must be large enough to hold `CDF_ATTR_NAME_LEN` characters and will be blank padded if necessary.

UNIX: For the proper operation of `CDF_lib`, `attr_name` MUST be of type `CHARACTER`.

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

`<GET_,ATTR_NUMBER_>`

Gets the number of the named attribute (in the current CDF). Note that this operation does not select the current attribute. Required arguments are as follows:

in: `CHARACTER attr_name*(*)`

Attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters.

UNIX: For the proper operation of `CDF_lib`, `attr_name` MUST be of type `CHARACTER`.

out: `INTEGER*4 attr_num`

The attribute number.

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

`<GET_,ATTR_NUMgENTRIES_>`

Inquires the number of gEntries for the current attribute (in the current CDF). This does not necessarily correspond with the maximum gEntry number used. Required arguments are as follows:

out: `INTEGER*4 num_entries`

The number of gEntries for the attribute.

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

NOTE: Only use this operation on gAttributes. An error will occur if used on a vAttribute.

`<GET_,ATTR_NUMrENTRIES_>`

Inquires the number of rEntries for the current attribute (in the current CDF). This does not necessarily correspond with the maximum rEntry number used. Required arguments are as follows:

out: `INTEGER*4 num_entries`

The number of rEntries for the attribute.

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

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<GET_,ATTR_NUMzENTRIES_>`

Inquires the number of zEntries for the current attribute (in the current CDF). This does not necessarily correspond with the maximum zEntry number used. Required arguments are as follows:

out: `INTEGER*4 num_entries`

The number of zEntries for the attribute.

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

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<GET_,ATTR_SCOPE_>`

Inquires the scope of the current attribute (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 scope`

Attribute scope. The scopes are described in Section 4.10.

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

`<GET_,CDF_COPYRIGHT_>`

Reads the copyright notice for the CDF library that created the current CDF. Required arguments are as follows:

out: `CHARACTER copy_right*(CDF_COPYRIGHT_LEN)`

CDF copyright text. This character string must be large enough hold `CDF_COPYRIGHT_LEN` characters and will be blank padded if necessary.

UNIX: For the proper operation of `CDF_lib`, `copy_right` MUST be of type `CHARACTER`.

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

`<GET_,CDF_ENCODING_>`

Inquires the data encoding of the current CDF. Required arguments are as follows:

out: `INTEGER*4 encoding`

Data encoding. The encodings are described in Section 4.6.

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

`<GET_,CDF_FORMAT_>`

Inquires the format of the current CDF. Required arguments are as follows:

out: `INTEGER*4 format`

CDF format. The formats are described in Section 4.4.

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

`<GET_,CDF_INCREMENT_>`

Inquires the incremental number of the CDF library that created the current CDF. Required arguments are as follows:

out: `INTEGER*4 increment`

Incremental number.

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

`<GET_,CDF_MAJORITY_>`

Inquires the variable majority of the current CDF. Required arguments are as follows:

out: `INTEGER*4 majority`

Variable majority. The majorities are described in Section 4.8.

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

`<GET_,CDF_NUMATTRS_>`

Inquires the number of attributes in the current CDF. Required arguments are as follows:

out: `INTEGER*4 num_attrs`

Number of attributes.

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

`<GET_,CDF_NUMgATTRS_>`

Inquires the number of gAttributes in the current CDF. Required arguments are as follows:

out: `INTEGER*4 num_attrs`

Number of gAttributes.

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

`<GET_,CDF_NUMrVARS_>`

Inquires the number of rVariables in the current CDF. Required arguments are as follows:

out: `INTEGER*4 num_vars`

Number of rVariables.

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

`<GET_,CDF_NUMvATTRS_>`

Inquires the number of vAttributes in the current CDF. Required arguments are as follows:

out: `INTEGER*4 num_attrs`

Number of vAttributes.

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

`<GET_,CDF_NUMzVARS_>`

Inquires the number of zVariables in the current CDF. Required arguments are as follows:

out: `INTEGER*4 num_vars`

Number of zVariables.

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

`<GET_,CDF_RELEASE_>`

Inquires the release number of the CDF library that created the current CDF. Required arguments are as follows:

out: `INTEGER*4 release`

Release number.

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

<GET_,CDF_VERSION_>

Inquires the version number of the CDF library that created the current CDF. Required arguments are as follows:

out: `INTEGER*4 version`

Version number.

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

<GET_,DATATYPE_SIZE_>

Inquires the size (in bytes) of an element of the specified data type. Required arguments are as follows:

in: `INTEGER*4 data_type`

Data type.

out: `INTEGER*4 num_bytes`

Number of bytes per element.

There are no required preselected objects/states.

<GET_,gENTRY_DATA_>

Reads the gEntry data value from the current attribute at the current gEntry number (in the current CDF). Required arguments are as follows:

out: `<type> value`

Value. `<type>` depends on the data type of the gEntry. The value is read from the CDF and placed into `value`. This buffer must be large enough to hold the value.

UNIX: If the gEntry has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

The required preselected objects/states are the current CDF, its current attribute, and its current gEntry number.

NOTE: Only use this operation on gAttributes. An error will occur if used on a vAttribute.

<GET_,gENTRY_DATATYPE_>

Inquires the data type of the gEntry at the current gEntry number for the current attribute (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 data_type`

Data type. The data types are described in Section 4.5.

The required preselected objects/states are the current CDF, its current attribute, and its current gEntry number.

NOTE: Only use this operation on gAttributes. An error will occur if used on a vAttribute.

<GET_,gENTRY_NUMELEMS_>

Inquires the number of elements (of the data type) of the gEntry at the current gEntry number for the current attribute (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 num_elements`

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.

The required preselected objects/states are the current CDF, its current attribute, and its current gEntry number.

NOTE: Only use this operation on gAttributes. An error will occur if used on a vAttribute.

`<GET_,LIB_COPYRIGHT_>`

Reads the copyright notice for the CDF library being used. Required arguments are as follows:

out: `CHARACTER copy_right*(CDF_COPYRIGHT_LEN)`

CDF library copyright text. This character string must be large enough hold `CDF_COPYRIGHT_LEN` characters and will be blank padded if necessary.

UNIX: For the proper operation of `CDF_lib`, `copy_right` MUST be of type `CHARACTER`.

There are no required preselected objects/states.

`<GET_,LIB_INCREMENT_>`

Inquires the incremental number of the CDF library being used. Required arguments are as follows:

out: `INTEGER*4 increment`

Incremental number.

There are no required preselected objects/states.

`<GET_,LIB_RELEASE_>`

Inquires the release number of the CDF library being used. Required arguments are as follows:

out: `INTEGER*4 release`

Release number.

There are no required preselected objects/states.

`<GET_,LIB_subINCREMENT_>`

Inquires the subincremental character of the CDF library being used. Required arguments are as follows:

out: `CHARACTER*1 subincrement`

Subincremental character.

UNIX: For the proper operation of `CDF_lib`, `subincrement` MUST be of type `CHARACTER`.

There are no required preselected objects/states.

`<GET_,LIB_VERSION_>`

Inquires the version number of the CDF library being used. Required arguments are as follows:

out: `INTEGER*4 version`

Version number.

There are no required preselected objects/states.

`<GET_,rENTRY_DATA_>`

Reads the rEntry data value from the current attribute at the current rEntry number (in the current CDF). Required arguments are as follows:

out: `<type> value`

Value. `<type>` depends on the data type of the rEntry. The value is read from the CDF and placed into `value`. This buffer must be large enough to hold the value.

UNIX: If the rEntry has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

The required preselected objects/states are the current CDF, its current attribute, and its current rEntry number.

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<GET_,rENTRY_DATATYPE_>`

Inquires the data type of the rEntry at the current rEntry number for the current attribute (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 data_type`

Data type. The data types are described in Section 4.5.

The required preselected objects/states are the current CDF, its current attribute, and its current rEntry number.

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<GET_,rENTRY_NUMELEMS_>`

Inquires the number of elements (of the data type) of the rEntry at the current rEntry number for the current attribute (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 num_elements`

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.

The required preselected objects/states are the current CDF, its current attribute, and its current rEntry number.

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

<GET_,rVAR_DATA_>

Reads a value from the current rVariable (in the current CDF). The value is read at the current record number and current dimension indices for the rVariables (in the current CDF). Required arguments are as follows:

out: `<type> value`

Value. `<type>` depends on the data type of the rVariable. This buffer must be large enough to hold the value. The value is read from the CDF and placed into `value`.

UNIX: If the rVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

The required preselected objects/states are the current CDF, its current rVariable, its current record number for rVariables, and its current dimension indices for rVariables.

<GET_,rVAR_DATATYPE_>

Inquires the data type of the current rVariable (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 data_type`

Data type. The data types are described in Section 4.5.

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

<GET_,rVAR_DIMVARYS_>

Inquires the dimension variances of the current rVariable (in the current CDF). For 0-dimensional rVariables this operation is not applicable. Required arguments are as follows:

out: `INTEGER*4 dim_varys(*)`

Dimension variances. `dim_varys` is a 1-dimensional array containing one element per rVariable dimension. Each element of `dim_varys` receives the corresponding dimension variance. The variances are described in Section 4.9.

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

<GET_,rVAR_EXTENDRECS_>

Inquires the number of extend records for the current rVariable (in the current CDF). Extend records are described in the Concepts chapter in the CDF User's Guide. Required arguments are as follows:

out: `INTEGER*4 n_extend_recs`

Number of records to extend. A value of zero (0) indicates that the default number of records is being used.

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

<GET_,rVAR_HYPERDATA_>

Reads one or more values from the current rVariable (in the current CDF). The values are read based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals for the rVariables (in the current CDF). Required arguments are as follows:

out: `<type> buffer`

Value(s). `<type>` depends on the data type of the rVariable. The values are read from the CDF and placed into `buffer`. This buffer must be large enough to hold the values.

UNIX: If the rVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

The required preselected objects/states are the current CDF, its current rVariable, its current record number, record count, and record interval for rVariables, and its current dimension indices, dimension counts, and dimension intervals for rVariables.

`<GET_,rVAR_MAXallocREC_>`

Inquires the maximum record number allocated for the current rVariable (in the current CDF). This only has significance for rVariables that are not NRV and are in single-file CDFs. Required arguments are as follows:

out: `INTEGER*4 var_max_rec_alloc`

Maximum record number allocated.

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

`<GET_,rVAR_MAXREC_>`

Inquires the maximum record number for the current rVariable (in the current CDF). For rVariables with a record variance of `NOVARY`, this will be at most one (1). A value of zero (0) indicates that no records have been written. Required arguments are as follows:

out: `INTEGER*4 var_max_rec`

Maximum record number.

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

`<GET_,rVAR_NAME_>`

Inquires the name of the current rVariable (in the current CDF). Required arguments are as follows:

out: `CHARACTER var_name*(CDF_VAR_NAME_LEN)`

Name of the rVariable. This character string must be large enough to hold `CDF_VAR_NAME_LEN` characters and will be blank padded if necessary.

UNIX: For the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

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

`<GET_,rVAR_nINDEXENTRIES_>`

Inquires the number of index entries for the current rVariable (in the current CDF). This only has significance for rVariables that are in single-file CDFs. The Concepts chapter in the CDF User's Guide describes the indexing scheme used for variable records in a single-file CDF. Required arguments are as follows:

out: `INTEGER*4 num_entries`

Number of index entries.

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

<GET_,rVAR_nINDEXRECORDS_>

Inquires the number of index records for the current rVariable (in the current CDF). This only has significance for rVariables that are in single-file CDFs. The Concepts chapter in the CDF User's Guide describes the indexing scheme used for variable records in a single-file CDF. Required arguments are as follows:

out: `INTEGER*4 num_records`

Number of index records.

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

<GET_,rVAR_NUMBER_>

Gets the number of the named rVariable (in the current CDF). Note that this operation does not select the current rVariable. Required arguments are as follows:

in: `CHARACTER var_name*(*)`

The rVariable name. This may be at most `CDF_VAR_NAME_LEN` characters.

UNIX: For the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

out: `INTEGER*4 var_num`

The rVariable number.

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

<GET_,rVAR_NUMELEMS_>

Inquires the number of elements (of the data type) for the current rVariable (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 num_elements`

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.

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

<GET_,rVAR_PADVALUE_>

Inquires the pad value of the current rVariable (in the current CDF). Required arguments are as follows:

out: `<type> value`

Pad value. `<type>` depends on the data type of the rVariable. The pad value is read from the CDF and placed into `value`. This buffer must be large enough to hold the pad value.

UNIX: If the rVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

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

<GET_,rVAR_RECVARY_>

Inquires the record variance of the current rVariable (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 rec_vary`

Record variance. The variances are described in Section 4.9.

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

<GET_,rVAR_SEQDATA_>

Reads one value from the current rVariable (in the current CDF) at the current sequential value for that rVariable. After the read the current sequential value is automatically incremented to the next value (crossing a record boundary if necessary). An error is returned if the current sequential value is past the last record for the rVariable. Required arguments are as follows:

out: `<type> value`

Value. `<type>` depends on the data type of the rVariable. The value is read from the CDF and placed into `value`. This buffer must be large enough to hold the value.

UNIX: If the rVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

The required preselected objects/states are the current CDF, its current rVariable, and the current sequential value for the rVariable. Note that the current sequential value for an rVariable increments automatically as values are read.

<GET_,rVARs_DIMSIZES_>

Inquires the size of each dimension for the rVariables in the current CDF. For 0-dimensional rVariables this operation is not applicable. Required arguments are as follows:

out: `INTEGER*4 dim_sizes(*)`

Dimension sizes. `dim_sizes` is a 1-dimensional array containing one element per rVariable dimension. Each element of `dim_sizes` receives the corresponding dimension size.

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

<GET_,rVARs_MAXREC_>

Inquires the maximum record number of the rVariables in the current CDF. A value of zero (0) indicates that no records have been written to the rVariables in the CDF. The maximum record number for an individual rVariable may be inquired using `<GET_,rVAR_MAXREC_>`. Required arguments are as follows:

out: `INTEGER*4 max_rec`

Maximum record number.

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

<GET_,rVARs_NUMDIMS_>

Inquires the number of dimensions for the rVariables in the current CDF. Required arguments are as follows:

out: `INTEGER*4 num_dims`

Number of dimensions.

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

`<GET_,rVARs_RECDATA_>`

Reads full-physical records from one or more rVariables (in the current CDF). The full-physical records are read at the current record number for rVariables. This operation does not affect the current rVariable (in the current CDF). Required arguments are as follows:

in: `INTEGER*4 num_vars`

The number of rVariables from which to read. This must be at least one (1).

in: `INTEGER*4 var_nums(*)`

The rVariables from which to read. This array, whose size is determined by the value of `num_vars`, contains rVariable numbers. The rVariable numbers can be listed in any order.

in: `<type> buffer`

The buffer into which the full-physical rVariable records being read are to be placed. This buffer must be large enough to hold the full-physical records. `<type>` must be a Fortran variable that will be passed by reference and cannot be of type `CHARACTER`. (The CDF library is expecting an address at which to place the full-physical records being read.) The order of the full-physical rVariable records in this buffer will correspond to the rVariable numbers listed in `varNums`, and this buffer will be contiguous — there will be no spacing between full-physical rVariable records. Be careful if using Fortran STRUCTUREs to receive multiple full-physical rVariable records. Fortran compilers on some operating systems will pad between the elements of a STRUCTURE in order to prevent memory alignment errors (i.e., the elements of a STRUCTURE may not be contiguous). See the Concepts chapter in the CDF User's Guide for more details on how to create this buffer.

The required preselected objects/states are the current CDF and its current record number for rVariables.

`<GET_,STATUS_TEXT_>`

Inquires the explanation text for the current status code. Note that the current status code is NOT the status code from the last operation performed. Required arguments are as follows:

out: `CHARACTER text*(CDF_STATUSTEXT_LEN)`

Text explaining the status code. This string must be large enough to hold `CDF_STATUSTEXT_LEN` characters and will be blank padded if necessary.

UNIX: For the proper operation of `CDF_lib`, `text` MUST be of type `CHARACTER`.

The only required preselected object/state is the current status code.

`<GET_,zENTRY_DATA_>`

Reads the zEntry data value from the current attribute at the current zEntry number (in the current CDF). Required arguments are as follows:

out: `<type> value`

Value. `<type>` depends on the data type of the zEntry. The value is read from the CDF and placed into `value`. This buffer must be large enough to hold the value.

UNIX: If the zEntry has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

The required preselected objects/states are the current CDF, its current attribute, and its current zEntry number.

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<GET_,zENTRY_DATATYPE_>`

Inquires the data type of the zEntry at the current zEntry number for the current attribute (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 data_type`

Data type. The data types are described in Section 4.5.

The required preselected objects/states are the current CDF, its current attribute, and its current zEntry number.

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<GET_,zENTRY_NUMELEMS_>`

Inquires the number of elements (of the data type) of the zEntry at the current zEntry number for the current attribute (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 num_elements`

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.

The required preselected objects/states are the current CDF, its current attribute, and its current zEntry number.

NOTE: Only use this operation on vAttributes. An error will occur if used on a gAttribute.

`<GET_,zVAR_DATA_>`

Reads a value from the current zVariable (in the current CDF). The value is read at the current record number and current dimension indices for that zVariable (in the current CDF). Required arguments are as follows:

out: `<type> value`

Value. `<type>` depends on the data type of the zVariable. This buffer must be large enough to hold the value. The value is read from the CDF and placed into `value`.

UNIX: If the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

The required preselected objects/states are the current CDF, its current zVariable, the current record number for the zVariable, and the current dimension indices for the zVariable.

<GET_,zVAR_DATATYPE_>

Inquires the data type of the current zVariable (in the current CDF). Required arguments are as follows:

out: `INTEGER*4 data_type`

Data type. The data types are described in Section 4.5.

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

<GET_,zVAR_DIMSIZES_>

Inquires the size of each dimension for the current zVariable in the current CDF. For 0-dimensional zVariables this operation is not applicable. Required arguments are as follows:

out: `INTEGER*4 dim_sizes(*)`

Dimension sizes. `dim_sizes` is a 1-dimensional array containing one element per dimension. Each element of `dim_sizes` receives the corresponding dimension size.

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

<GET_,zVAR_DIMVARYS_>

Inquires the dimension variances of the current zVariable (in the current CDF). For 0-dimensional zVariables this operation is not applicable. Required arguments are as follows:

out: `INTEGER*4 dim_varys(*)`

Dimension variances. `dim_varys` is a 1-dimensional array containing one element per dimension. Each element of `dim_varys` receives the corresponding dimension variance. The variances are described in Section 4.9.

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

<GET_,zVAR_EXTENDRECS_>

Inquires the number of extend records for the current zVariable (in the current CDF). Extend records are described in the Concepts chapter in the CDF User's Guide. Required arguments are as follows:

out: `INTEGER*4 n_extend_recs`

Number of records to extend. A value of zero (0) indicates that the default number of records is being used.

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

<GET_,zVAR_HYPERDATA_>

Reads one or more values from the current zVariable (in the current CDF). The values are read based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals for that zVariable (in the current CDF). Required arguments are as follows:

out: `<type> buffer`

Value(s). `<type>` depends on the data type of the zVariable. The values are read from the CDF and placed into `buffer`. This buffer must be large enough to hold the values.

UNIX: If the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

The required preselected objects/states are the current CDF, its current zVariable, the current record number, record count, and record interval for the zVariable, and the current dimension indices, dimension counts, and dimension intervals for the zVariable.

`<GET_,zVAR_MAXallocREC_>`

Inquires the maximum record number allocated for the current zVariable (in the current CDF). This has significance only for zVariables that are not NRV and are in single-file CDFs. Required arguments are as follows:

out: `INTEGER*4 var_max_rec_alloc`

Maximum record number allocated.

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

`<GET_,zVAR_MAXREC_>`

Inquires the maximum record number for the current zVariable (in the current CDF). For zVariables with a record variance of `NOVARY`, this will be at most one (1). A value of zero (0) indicates that no records have been written. Required arguments are as follows:

out: `INTEGER*4 var_max_rec`

Maximum record number.

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

`<GET_,zVAR_NAME_>`

Inquires the name of the current zVariable (in the current CDF). Required arguments are as follows:

out: `CHARACTER var_name*(CDF_VAR_NAME_LEN)`

Name of the zVariable. This character string must be large enough to hold `CDF_VAR_NAME_LEN` characters and will be blank padded if necessary.

UNIX: For the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

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

`<GET_,zVAR_nINDEXENTRIES_>`

Inquires the number of index entries for the current zVariable (in the current CDF). This has significance only for zVariables that are in single-file CDFs. The Concepts chapter in the CDF User's Guide describes the indexing scheme used for variable records in a single-file CDF. Required arguments are as follows:

out: `INTEGER*4 num_entries`

Number of index entries.

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

`<GET_,zVAR_nINDEXRECORDS_>`