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.[4]

6.3 Returned Status

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

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

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

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

6.4 Indentation/Style

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

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

Note that the functions (`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 functions.

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

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

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

6.5 Syntax

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

The syntax for `CDFlib` is as follows:

```
status = CDFlib (fnc1, item1, arg1, arg2, ...argN,
                       item2, arg1, arg2, ...argN,
                       .
                       .
                       itemN, arg1, arg2, ...argN,
                 fnc2, item1, arg1, arg2, ...argN,
                       item2, arg1, arg2, ...argN,
                       .
                       .
                       itemN, arg1, arg2, ...argN,
```

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

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

6.6 Operations...

An operation consists of a function being performed on an item. The supported functions 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.
`NULL_`	Used to signal the end of the argument list of an internal interface call.
`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 function the supported items, required arguments, and required preselected objects/states are listed below. The required preselected objects/states are those objects/states that must be selected (typically with the `SELECT_` function) 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 is only applicable to multi-file CDFs.

There are no required arguments.

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

<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: `long *attrNum`

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: `char *attrName`

The attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters (excluding the `NUL` terminator).

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

<CONFIRM_,CDF_>

Confirms the current CDF. Required arguments are as follows:

out: `CDFid *id`

The current CDF.

There are no required preselected objects/states.

<CONFIRM_,CDF_ACCESS_>

Confirms the accessability of the current CDF. If a fatal error occurred while accessing the CDF the error code `NO_MORE_ACCESS` will be returned. If this is the case, the CDF should still be closed.

There are no required arguments.

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

<CONFIRM_,CDF_CACHESIZE_>

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

out: `long *numBuffers`

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: `long *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 file name of the current CDF. Required arguments are as follows:

out: `char CDFname[CDF_PATHNAME_LEN+1]`

File name of the CDF.

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: `long *mode`

The `-0.0` to `0.0` mode. The `-0.0` to `0.0` modes are described in Section 4.15.

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: `long *mode`

The read-only mode. The read-only modes are described in Section 4.13.

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

<CONFIRM_,CDF_STATUS_>

Confirms the current status code. Note that this is not the most recently returned status code but rather the most recently selected status code (see the <SELECT_,CDF_STATUS_> operation). Required arguments are as follows:

out: `CDFstatus *status`

The status code.

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

<CONFIRM_,zMODE_>

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

out: `long *mode`

The zMode. The zModes are described in Section 4.14.

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

`<CONFIRM_,COMPRESS_CACHESIZE_>`

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

out: `long *numBuffers`

The number of cache buffers being used.

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: `long *entryNum`

The 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: `long entryNum`

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: `long *entryNum`

The 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: `long entryNum`

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: `long *varNum`

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. Required arguments are as follows:

out: `long *numBuffers`

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: `char *varName`

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

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

<CONFIRM_,rVAR_PADVALUE_>

Confirms the existence of an explicitly specified pad value for the current rVariable (in the current CDF). If an explicit pad value has not been specified, the informational status code `NO_PADVALUE_SPECIFIED_` will be returned.

There are no required arguments.

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

<CONFIRM_,rVAR_RESERVEPERCENT_>

Confirms the reserve percentage being used for the current rVariable (of the current CDF). This operation is only applicable to compressed rVariables. The Concepts chapter in the CDF User's Guide describes the reserve percentage scheme used by the CDF library. Required arguments are as follows:

out: `long *percent`

The reserve percentage.

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

<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: `long *recNum`

Record number.

out: `long indices[CDF_MAX_DIMS]`

Dimension indices. Each element of `indices` receives the corresponding dimension index. For 0-dimensional rVariables this argument is ignored (but must be present).

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: `long counts[CDF_MAX_DIMS]`

Dimension counts. 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: `long indices[CDF_MAX_DIMS]`

Dimension indices. 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: `long intervals[CDF_MAX_DIMS]`

Dimension intervals. 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: `long *recCount`

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: `long *recInterval`

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: `long *recNum`

Record number.

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

<CONFIRM_,STAGE_CACHESIZE_>

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

out: `long *numBuffers`

The number of cache buffers being used.

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: `long *entryNum`

The 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: `long entryNum`

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: `long *varNum`

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. Required arguments are as follows:

out: `long *numBuffers`

The number of cache buffers being used.

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

`<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: `long counts[CDF_MAX_DIMS]`

Dimension counts. 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: `long indices[CDF_MAX_DIMS]`

Dimension indices. 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: `long intervals[CDF_MAX_DIMS]`

Dimension intervals. Each element of `intervals` receives the corresponding dimension interval.

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: `char *varName`

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

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

<CONFIRM_,zVAR_PADVALUE_>

Confirms the existence of an explicitly specified pad value for the current zVariable (in the current CDF). If an explicit pad value has not been specified, the informational status code `NO_PADVALUE_SPECIFIED_` will be returned.

There are no required arguments.

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: `long *recCount`

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: `long *recInterval`

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: `long *recNum`

Record number.

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

<CONFIRM_,zVAR_RESERVEPERCENT_>

Confirms the reserve percentage being used for the current zVariable (of the current CDF). This operation is only applicable to compressed zVariables. The Concepts chapter in the CDF User's Guide describes the reserve percentage scheme used by the CDF library. Required arguments are as follows:

out: `long *percent`

The reserve percentage.

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: `long *recNum`

Record number.

out: `long indices[CDF_MAX_DIMS]`

Dimension indices. Each element of `indices` receives the corresponding dimension index. For 0-dimensional zVariables this argument is ignored (but must be present).

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: `char *attrName`

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

in: `long scope`

Scope of the new attribute. Specify one of the scopes described in Section 4.12.

out: `long *attrNum`

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

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

<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: `char *CDFname`

File name of the CDF to be created. (Do not append an extension.) This can be at most `CDF_PATHNAME_LEN` characters (excluding the `NUL` terminator). A CDF file name 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: File names are case-sensitive.

in: `long numDims`

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

in: `long dimSizes[]`

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

out: `CDFid *id`

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

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

A CDF must be closed with the <CLOSE_,CDF_> operation 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: `char *varName`

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

in: `long dataType`

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

in: `long numElements`

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

in: `long recVary`

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

in: `long dimVarys[]`

Dimension variances. Each element of `dimVarys` 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 must be present).

out: `long *varNum`

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

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

<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: `char *varName`

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

in: `long dataType`

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

in: `long numElements`

Number of elements of the data type at each value. For character data types (`CDF_CHAR` and `CDF_UCHAR`), this is the number of characters in each string (an array of characters).

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

in: `long numDims`

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

in: `long dimSizes[]`

The dimension sizes. Each element of `dimSizes` specifies the corresponding dimension size. Each dimension size must be greater than zero (0). For a 0-dimensional zVariable this argument is ignored (but must be present).

in: `long recVary`

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

in: `long dimVarys[]`

Dimension variances. Each element of `dimVarys` 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 must be present).

out: `long *varNum`

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

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

`<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. **NOTE:** This operation is only allowed on single-file CDFs.

There are no required arguments.

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

`<DELETE_,rVAR_RECORDS_>`

Deletes the specified range of records from the current rVariable (in the current CDF). If the rVariable has sparse records a gap of missing records will be created. If the rVariable does not have sparse records, the records following the range of deleted records are immediately renumbered beginning with the number of the first deleted record. **NOTE:** This operation is only allowed on single-file CDFs. Required arguments are as follows:

in: `long firstRecord`

The record number of the first record to be deleted.

in: `long lastRecord`

The record number of the last record to be deleted.

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 rEntries which numerically follow the rEntries being deleted are also immediately renumbered. When the zVariable is deleted, there is no longer a current zVariable. **NOTE:** This operation is only allowed on single-file CDFs.

There are no required arguments.

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

<DELETE_,zVAR_RECORDS_>

Deletes the specified range of records from the current zVariable (in the current CDF). If the zVariable has sparse records a gap of missing records will be created. If the zVariable does not have sparse records, the records following the range of deleted records are immediately renumbered beginning with the number of the first deleted record. **NOTE:** This operation is only allowed on single-file CDFs. Required arguments are as follows:

in: `long firstRecord`

The record number of the first record to be deleted.

in: `long lastRecord`

The record number of the last record to be deleted.

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

<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: `long *maxEntry`

The maximum gEntry number for the attribute. If no gEntries exist, then a value of `-1` 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: `long *maxEntry`

The maximum rEntry number for the attribute. If no rEntries exist, then a value of `-1` 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: `long *maxEntry`

The maximum zEntry number for the attribute. If no zEntries exist, then a value of `-1` 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: `char attrName[CDF_ATTR_NAME_LEN+1]`

Attribute name.

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: `char *attrName`

Attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters (excluding the `NUL` terminator).

out: `long *attrNum`

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: `long *numEntries`

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: `long *numEntries`

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: `long *numEntries`

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: `long *scope`

Attribute scope. The scopes are described in Section 4.12.

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

<GET_,CDF_COMPRESSION_>

Inquires the compression type/parameters of the current CDF. This refers to the compression of the CDF — not of any compressed variables. Required arguments are as follows:

out: `long *cType`

The compression type. The types of compressions are described in Section 4.10.

out: `long cParms[CDF_MAX_PARMS]`

The compression parameters. The compression parameters are described in Section 4.10.

out: `long *cPct`

If compressed, the percentage of the uncompressed size of the CDF needed to store the compressed CDF.

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

<GET_,CDF_COPYRIGHT_>

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

out: `char copyRight[CDF_COPYRIGHT_LEN+1`

CDF copyright text.

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: `long *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: `long *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: `long *increment`

Incremental number.

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

<GET_,CDF_INFO_>

Inquires the compression type/parameters of a CDF without having to open the CDF. This refers to the compression of the CDF — not of any compressed variables. Required arguments are as follows:

in: `char *CDFname`

File name of the CDF to be inquired. (Do not append an extension.) This can be at most `CDF_PATHNAME_LEN` characters (excluding the `NUL` terminator). A CDF file name 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: File names are case-sensitive.

out: `long *cType`

The CDF compression type. The types of compressions are described in Section 4.10.

out: `long cParms[CDF_MAX_PARMS]`

The compression parameters. The compression parameters are described in Section 4.10.

out: `long *cSize`

If compressed, size in bytes of the dotCDF file. If not compressed, set to zero (0).

out: `long *uSize`

If compressed, size in bytes of the dotCDF file when decompressed. If not compressed, size in bytes of the dotCDF file.

There are no required preselected objects/states.

<GET_,CDF_MAJORITY_>

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

out: `long *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: `long *numAttrs`

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: `long *numAttrs`

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: `long *numVars`

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: `long *numAttrs`

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: `long *numVars`

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: `long *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: `long *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: `long dataType`

Data type.

out: `long *numBytes`

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: `void *value`

Value. This buffer must be large to hold the value. The value is read from the CDF and placed into memory at address `value`.

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: `long *dataType`

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: `long *numElements`

Number of elements of the data type. For character data types (`CDF_CHAR` and `CDF_UCHAR`) this is the number of characters in the string (an array of characters). 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 of the CDF library being used. Required arguments are as follows:

out: `char copyRight[CDF_COPYRIGHT_LEN+1`

CDF library copyright text.

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: `long *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: `long *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: `char *subincrement`

Subincremental 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: `long *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: `void *value`

Value. This buffer must be large to hold the value. The value is read from the CDF and placed into memory at address `value`.

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: `long *dataType`

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: `long *numElements`

Number of elements of the data type. For character data types (`CDF_CHAR` and `CDF_UCHAR`) this is the number of characters in the string (an array of characters). 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_ALLOCATEDFROM_>`

Inquires the next allocated record at or after a given record for the current rVariable (in the current CDF). Required arguments are as follows:

in: `long startRecord`

The record number at which to begin searching for the next allocated record. If this record exists, it will be considered the next allocated record.

out: `long *nextRecord`

The number of the next allocated record.

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

`<GET_,rVAR_ALLOCATEDTO_>`

Inquires the last allocated record (before the next unallocated record) at or after a given record for the current rVariable (in the current CDF). Required arguments are as follows:

in: `long startRecord`

The record number at which to begin searching for the last allocated record.

out: `long *nextRecord`

The number of the last allocated record.

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

`<GET_,rVAR_BLOCKINGFACTOR_>`[5]

Inquires the blocking factor for the current rVariable (in the current CDF). Blocking factors are described in the Concepts chapter in the CDF User's Guide. Required arguments are as follows:

out: `long *blockingFactor`

The blocking factor. A value of zero (0) indicates that the default blocking factor is being used.

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

`<GET_,rVAR_COMPRESSION_>`

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

out: `long *cType`

The compression type. The types of compressions are described in Section 4.10.

out: `long cParms[CDF_MAX_PARMS]`

The compression parameters. The compression parameters are described in Section 4.10.

out: `long *cPct`

If compressed, the percentage of the uncompressed size of the rVariable's data values needed to store the compressed values.

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

`<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:

[5]The item `rVAR_BLOCKINGFACTOR_` was previously named `rVAR_EXTENDRECS_`.

out: `void *value`

Value. This buffer must be large enough to hold the value. The value is read from the CDF and placed into memory at address `value`.

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: `long *dataType`

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: `long dimVarys[CDF_MAX_DIMS]`

Dimension variances. Each element of `dimVarys` 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_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: `void *buffer`

Values. This buffer must be large enough to hold the values. The values are read from the CDF and placed into memory starting at address `buffer`.

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). Required arguments are as follows:

out: `long *varMaxRecAlloc`

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 zero (0). A value of negative one (-1) indicates that no records have been written. Required arguments are as follows:

out: `long *varMaxRec`

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: `char varName[CDF_VAR_NAME_LEN+1`

Name of the rVariable.

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: `long *numEntries`

Number of index entries.

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

<GET_,rVAR_nINDEXLEVELS_>

Inquires the number of index levels 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: `long *numLevels`

Number of index levels.

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: `long *numRecords`

Number of index records.

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

<GET_,rVAR_NUMallocRECS_>

Inquires the number of records allocated for the current rVariable (in the current CDF). The Concepts chapter in the CDF User's Guide describes the allocation of variable records in a single-file CDF. Required arguments are as follows:

out: `long *numRecords`

Number of allocated 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: `char *varName`

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

out: `long *varNum`

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: `long *numElements`

Number of elements of the data type at each value. For character 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 data types.

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

<GET_,rVAR_NUMRECS_>

Inquires the number of records written for the current rVariable (in the current CDF). This may not correspond to the maximum record written (see <GET_,rVAR_MAXREC_>) if the rVariable has sparse records. Required arguments are as follows:

out: `long *numRecords`

Number of records written.

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). If a pad value has not been explicitly specified for the rVariable (see <PUT_,rVAR_PADVALUE_>), the informational status code `NO_PADVALUE_SPECIFIED` will be returned and the default pad value for the rVariable's data type will be placed in the pad value buffer provided. Required arguments are as follows:

out: `void *value`

Pad value. This buffer must be large enough to hold the pad value. The pad value is read from the CDF and placed in memory at address `value`.

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: `long *recVary`

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: `void *value`

Value. This buffer must be large enough to hold the value. The value is read from the CDF and placed into memory at address `value`.

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_,rVAR_SPARSEARRAYS_>`

Inquires the sparse arrays type/parameters of the current rVariable (in the current CDF). Required arguments are as follows:

out: `long *sArraysType`

The sparse arrays type. The types of sparse arrays are described in Section 4.11.

out: `long sArraysParms[CDF_MAX_PARMS]`

The sparse arrays parameters. The sparse arrays parameters are described in Section 4.11.

out: `long *sArraysPct`

If sparse arrays, the percentage of the non-sparse size of the rVariable's data values needed to store the sparse values.

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

`<GET_,rVAR_SPARSERECORDS_>`

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

out: `long *sRecordsType`

The sparse records type. The types of sparse records are described in Section 4.11.

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

<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: `long dimSizes[CDF_MAX_DIMS]`

Dimension sizes. Each element of `dimSizes` 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. Note that this is not the number of records but rather the maximum record number (which is one less than the number of records). A value of negative one (-1) indicates that the rVariables contain no records. The maximum record number for an individual rVariable may be inquired using the <GET_,rVAR_MAXREC_> operation. Required arguments are as follows:

out: `long *maxRec`

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: `long *numDims`

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: `long numVars`

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

in: `long varNums[]`

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

in: `void *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. 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 C `struct` objects to receive multiple full-physical rVariable records. C compilers on some operating systems will pad between the elements of a `struct` in order to prevent memory alignment errors (i.e., the elements of a `struct` may not be contiguous). See the Concepts chapter in the CDF User's Guide for more details on how to allocate 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 from the last operation performed. Required arguments are as follows:

out: `char text[CDF_STATUSTEXT_LEN+1`

Text explaining the status code.

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: `void *value`

Value. This buffer must be large to hold the value. The value is read from the CDF and placed into memory at address `value`.

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: `long *dataType`

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: `long *numElements`

Number of elements of the data type. For character data types (`CDF_CHAR` and `CDF_UCHAR`) this is the number of characters in the string (an array of characters). 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_ALLOCATEDFROM_>`

Inquires the next allocated record at or after a given record for the current zVariable (in the current CDF). Required arguments are as follows:

in: `long startRecord`

The record number at which to begin searching for the next allocated record. If this record exists, it will be considered the next allocated record.

out: `long *nextRecord`

The number of the next allocated record.

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

`<GET_,zVAR_ALLOCATEDTO_>`

Inquires the last allocated record (before the next unallocated record) at or after a given record for the current zVariable (in the current CDF). Required arguments are as follows:

in: `long startRecord`

The record number at which to begin searching for the last allocated record.

out: `long *nextRecord`

The number of the last allocated record.

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

`<GET_,zVAR_BLOCKINGFACTOR_>`[6]

Inquires the blocking factor for the current zVariable (in the current CDF). Blocking factors are described in the Concepts chapter in the CDF User's Guide. Required arguments are as follows:

out: `long *blockingFactor`

The blocking factor. A value of zero (0) indicates that the default blocking factor is being used.

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

`<GET_,zVAR_COMPRESSION_>`

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

out: `long *cType`

The compression type. The types of compressions are described in Section 4.10.

[6] The item `zVAR_BLOCKINGFACTOR_` was previously named `zVAR_EXTENDRECS_`.

out: `long cParms[CDF_MAX_PARMS]`

The compression parameters. The compression parameters are described in Section 4.10.

out: `long *cPct`

If compressed, the percentage of the uncompressed size of the zVariable's data values needed to store the compressed values.

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

`<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: `void *value`

Value. This buffer must be large enough to hold the value. The value is read from the CDF and placed into memory at address `value`.

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: `long *dataType`

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: `long dimSizes[CDF_MAX_DIMS]`

Dimension sizes. Each element of `dimSizes` 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: `long dimVarys[CDF_MAX_DIMS]`

Dimension variances. Each element of `dimVarys` 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_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: `void *buffer`

Values. This buffer must be large enough to hold the values. The values are read from the CDF and placed into memory starting at address `buffer`.

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). Required arguments are as follows:

out: `long *varMaxRecAlloc`

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 zero (0). A value of negative one (-1) indicates that no records have been written. Required arguments are as follows:

out: `long *varMaxRec`

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: `char varName[CDF_VAR_NAME_LEN+1`

Name of the zVariable.

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 only has significance 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: `long *numEntries`

Number of index entries.

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

<GET_,zVAR_nINDEXLEVELS_>

Inquires the number of index levels for the current zVariable (in the current CDF). This only has significance 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: `long *numLevels`

Number of index levels.

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

<GET_,zVAR_nINDEXRECORDS_>

Inquires the number of index records for the current zVariable (in the current CDF). This only has significance 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: `long *numRecords`

Number of index records.

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

<GET_,zVAR_NUMallocRECS_>

Inquires the number of records allocated for the current zVariable (in the current CDF). The Concepts chapter in the CDF User's Guide describes the allocation of variable records in a single-file CDF. Required arguments are as follows:

out: `long *numRecords`

Number of allocated records.

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

<GET_,zVAR_NUMBER_>

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

in: `char *varName`

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

out: `long *varNum`

The zVariable number.

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

<GET_,zVAR_NUMDIMS_>

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

out: `long *numDims`

Number of dimensions.

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

`<GET_,zVAR_NUMELEMS_>`

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

out: `long *numElements`

Number of elements of the data type at each value. For character 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 data types.

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

`<GET_,zVAR_NUMRECS_>`

Inquires the number of records written for the current zVariable (in the current CDF). This may not correspond to the maximum record written (see `<GET_,zVAR_MAXREC_>`) if the zVariable has sparse records. Required arguments are as follows:

out: `long *numRecords`

Number of records written.

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

`<GET_,zVAR_PADVALUE_>`

Inquires the pad value of the current zVariable (in the current CDF). If a pad value has not been explicitly specified for the zVariable (see `<PUT_,zVAR_PADVALUE_>`), the informational status code `NO_PADVALUE_SPECIFIED` will be returned and the default pad value for the zVariable's data type will be placed in the pad value buffer provided. Required arguments are as follows:

out: `void *value`

Pad value. This buffer must be large enough to hold the pad value. The pad value is read from the CDF and placed in memory at address `value`.

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

`<GET_,zVAR_RECVARY_>`

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

out: `long *recVary`

Record variance. The variances are described in Section 4.9.

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

`<GET_,zVAR_SEQDATA_>`

Reads one value from the current zVariable (in the current CDF) at the current sequential value for that zVariable. 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 zVariable. Required arguments are as follows:

out: `void *value`

Value. This buffer must be large enough to hold the value. The value is read from the CDF and placed into memory at address `value`.

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

`<GET_,zVAR_SPARSEARRAYS_>`

Inquires the sparse arrays type/parameters of the current zVariable (in the current CDF). Required arguments are as follows:

out: `long *sArraysType`

The sparse arrays type. The types of sparse arrays are described in Section 4.11.

out: `long sArraysParms[CDF_MAX_PARMS]`

The sparse arrays parameters. The sparse arrays parameters are described in Section 4.11.

out: `long *sArraysPct`

If sparse arrays, the percentage of the non-sparse size of the zVariable's data values needed to store the sparse values.

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

`<GET_,zVAR_SPARSERECORDS_>`

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

out: `long *sRecordsType`

The sparse records type. The types of sparse records are described in Section 4.11.

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

`<GET_,zVARs_MAXREC_>`

Inquires the maximum record number of the zVariables in the current CDF. Note that this is not the number of records but rather the maximum record number (which is one less than the number of records). A value of negative one (-1) indicates that the zVariables contain no records. The maximum record number for an individual zVariable may be inquired using the `<GET_,zVAR_MAXREC_>` operation. Required arguments are as follows:

out: `long *maxRec`

Maximum record number.

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

`<GET_,zVARs_RECDATA_>`

Reads full-physical records from one or more zVariables (in the current CDF). The full-physical record for a particular zVariable is read at the current record number for that zVariable. (The record numbers do not have to be the same but in most cases probably will be.) This operation does not affect the current zVariable (in the current CDF). Required arguments are as follows:

in: `long numVars`

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

in: `long varNums[]`

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

in: `void *buffer`

The buffer into which the full-physical zVariable records being read are to be placed. This buffer must be large enough to hold the full-physical records. The order of the full-physical zVariable records in this buffer will correspond to the zVariable numbers listed in `varNums`, and this buffer will be contiguous — there will be no spacing between full-physical zVariable records. Be careful if using C `struct` objects to receive multiple full-physical zVariable records. C compilers on some operating systems will pad between the elements of a `struct` in order to prevent memory alignment errors (i.e., the elements of a `struct` may not be contiguous). See the Concepts chapter in the CDF User's Guide for more details on how to allocate this buffer.

The required preselected objects/states are the current CDF and the current record number for each of the zVariables specified. A convenience operation exists, `<SELECT_,zVARs_RECNUMBER_>`, that allows the current record number for each zVariable to be selected at one time (as opposed to selecting the current record numbers one at a time using `<SELECT_,zVAR_RECNUMBER_>`).

`<NULL_>`

Marks the end of the argument list that is passed to an internal interface call. No other arguments are allowed after it.

`<OPEN_,CDF_>`

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

in: `char *CDFname`

File name of the CDF to be opened. (Do not append an extension.) This can be at most `CDF_PATHNAME_LEN` characters (excluding the `NUL` terminator). A CDF file name 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: File names are case-sensitive.

out: `CDFid *id`

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

There are no required preselected objects/states.

`<PUT_,ATTR_NAME_>`

Renames the current attribute (in the current CDF). An attribute with the same name must not already exist in the CDF. Required arguments are as follows:

in: `char *attrName`

New attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters (excluding the `NUL` terminator).

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

`<PUT_,ATTR_SCOPE_>`

Respecifies the scope for the current attribute (in the current CDF). Required arguments are as follows:

in: `long scope`

New attribute scope. Specify one of the scopes described in Section 4.12.

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

`<PUT_,CDF_COMPRESSION_>`

Specifies the compression type/parameters for the current CDF. This refers to the compression of the CDF — not of any variables. Required arguments are as follows:

in: `long cType`

The compression type. The types of compressions are described in Section 4.10.

in: `long cParms[]`

The compression parameters. The compression parameters are described in Section 4.10.

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

`<PUT_,CDF_ENCODING_>`

Respecifies the data encoding of the current CDF. A CDF's data encoding may not be changed after any variable values (including the pad value) or attribute entries have been written. Required arguments are as follows:

in: `long encoding`

New data encoding. Specify one of the encodings described in Section 4.6.

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

`<PUT_,CDF_FORMAT_>`

Respecifies the format of the current CDF. A CDF's format may not be changed after any variables have been created. Required arguments are as follows:

in: `long format`

New CDF format. Specify one of the formats described in Section 4.4.

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

`<PUT_,CDF_MAJORITY_>`

Respecifies the variable majority of the current CDF. A CDF's variable majority may not be changed after any variable values have been written. Required arguments are as follows:

in: `long majority`

New variable majority. Specify one of the majorities described in Section 4.8.

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

`<PUT_,gENTRY_DATA_>`

Writes a gEntry to the current attribute at the current gEntry number (in the current CDF). An existing gEntry may be overwritten with a new gEntry having the same data specification (data type and number of elements) or a different data specification. Required arguments are as follows:

in: `long dataType`

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

in: `long numElements`

Number of elements of the data type. This may be greater than one (1) for any of the supported data types. For character data types (`CDF_CHAR` and `CDF_UCHAR`) this is the number of characters in the string (an array of characters). For all other data types this is the number of elements in an array of that data type.

in: `void *value`

Value(s). The entry value is written to the CDF from memory address `value`.

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.

`<PUT_,gENTRY_DATASPEC_>`

Modifies the data specification (data type and number of elements) of the gEntry at the current gEntry number of the current attribute (in the current CDF). The new and old data types must be equivalent, and the number of elements must not be changed. Equivalent data types are described in the Concepts chapter in the CDF User's Guide. Required arguments are as follows:

in: `long dataType`

New data type of the gEntry. Specify one of the data types described in Section 4.5.

in: `long numElements`

Number of elements of the 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.

`<PUT_,rENTRY_DATA_>`