out: `<type> value`

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

WARNING: If the zVariable has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `value` must be a `CHARACTER` Fortran variable. If the zVariable does not have one of the character data types, then `value` must NOT be a `CHARACTER` Fortran variable.

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: `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 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: `<type> value`

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

WARNING: If the zVariable has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `value` must be a `CHARACTER` Fortran variable. If the zVariable does not have one of the character data types, then `value` must NOT be a `CHARACTER` Fortran variable.

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: `INTEGER*4 s_arrays_type`

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

out: `INTEGER*4 s_arrays_parms(CDF_MAX_PARMS)`

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

out: `INTEGER*4 s_arrays_pct`

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: `INTEGER*4 s_records_type`

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

out: `INTEGER*4 max_rec`

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: `INTEGER*4 num_vars`

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

in: `INTEGER*4 var_nums(*)`

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

in: `<type> 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. `<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 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 Fortran STRUCTUREs to receive multiple full-physical zVariable 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 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 the `<SELECT_,zVAR_RECNUMBER_>` operation).

`<NULL_>`

Ends the argument list that is passed to an internal interface call. No more other arguments are allowed to follow except the returned status code as the very last argument.

`<OPEN_,CDF_>`

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

in: `CHARACTER CDF_name*(*)`

File name of the CDF to be opened. (Do not append an extension.) This can be at most `CDF_PATHNAME_LEN` characters. 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.

UNIX: For the proper operation of `CDF_lib`, `CDF_name` MUST be a Fortran `CHARACTER` variable or constant.

out: `INTEGER*4 id`

CDF identifier to be used in subsequent operations of 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: `CHARACTER attr_name*(*)`

New attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters.

UNIX: For the proper operation of `CDF_lib`, `attr_name` MUST be a Fortran `CHARACTER` variable or constant.

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: `INTEGER*4 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: `INTEGER*4 c_type`

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

in: `INTEGER*4 c_parms(*)`

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: `INTEGER*4 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: `INTEGER*4 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: `INTEGER*4 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: `INTEGER*4 data_type`

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

in: `INTEGER*4 num_elements`

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. For all other data types this is the number of elements in an array of that data type.

in: `<type> value`

The value(s). `<type>` depends on the data type of the gEntry. The value is written to the CDF from `value`.

WARNING: If the gEntry has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `value` must be a `CHARACTER` Fortran variable. If the gEntry does not have one of the character data types, then `value` must NOT be a `CHARACTER` Fortran variable.

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: `INTEGER*4 data_type`

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

in: `INTEGER*4 num_elements`

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_>`

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

in: `INTEGER*4 data_type`

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

in: `INTEGER*4 num_elements`

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. For all other data types this is the number of elements in an array of that data type.

in: `<type> value`

The value(s). `<type>` depends on the data type of the rEntry. The value is written to the CDF from `value`.

> **WARNING:** If the rEntry has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `value` must be a `CHARACTER` Fortran variable. If the rEntry does not have one of the character data types, then `value` must NOT be a `CHARACTER` Fortran variable.

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.

`<PUT_,rENTRY_DATASPEC_>`

Modifies the data specification (data type and number of elements) of the rEntry at the current rEntry 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: `INTEGER*4 data_type`

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

in: `INTEGER*4 num_elements`

Number of elements of the 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.

`<PUT_,rVAR_ALLOCATEBLOCK_>`

Specifies a range of records to allocate for the current rVariable (in the current CDF). This operation is only applicable to uncompressed rVariables in single-file CDFs. The Concepts chapter in the CDF User's Guide describes the allocation of variable records Required arguments are as follows:

in: `INTEGER*4 first_record`

The first record number to allocate.

in: `INTEGER*4 last_record`

The last record number to allocate.

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

`<PUT_,rVAR_ALLOCATERECS_>`

Specifies the number of records to allocate for the current rVariable (in the current CDF). The records are allocated beginning at record number `1` (one). This operation is only applicable to uncompressed rVariables in single-file CDFs. The Concepts chapter in the CDF User's Guide describes the allocation of variable records. Required arguments are as follows:

in: `INTEGER*4 num_records`

Number of records to allocate.

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

`<PUT_,rVAR_BLOCKINGFACTOR_>`[12]

Specifies the blocking factor for the current rVariable (in the current CDF). The Concepts chapter in the CDF User's Guide describes blocking factors. **NOTE:** The blocking factor has no effect for NRV variables or multi-file CDFs.

Required arguments are as follows:

in: `INTEGER*4 blocking_factor`

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

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

`<PUT_,rVAR_COMPRESSION_>`

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

in: `INTEGER*4 c_type`

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

in: `INTEGER*4 c_parms(*)`

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

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

`<PUT_,rVAR_DATA_>`

Writes one value to the current rVariable (in the current CDF). The value is written at the current record number and current dimension indices for the rVariables (in the current CDF). Required arguments are as follows:

in: `<type> value`

Value. `<type>` depends on the data type of the rVariable. The value is written to the CDF from `value`.

WARNING: If the rVariable has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `value` must be a `CHARACTER` Fortran variable. If the rVariable does not have one of the character data types, then `value` must NOT be a `CHARACTER` Fortran variable.

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.

`<PUT_,rVAR_DATASPEC_>`

Respecifies the data specification (data type and number of elements) of the current rVariable (in the current CDF). An rVariable's data specification may not be changed if the new data specification is not equivalent to the old data specification and any values (including the pad value) have been written. Data specifications are considered equivalent if the data types are equivalent (see the Concepts chapter in the CDF User's Guide) and the number of elements are the same. Required arguments are as follows:

in: `INTEGER*4 data_type`

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

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

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

`<PUT_,rVAR_DIMVARYS_>`

Respecifies the dimension variances of the current rVariable (in the current CDF). An rVariable's dimension variances may not be changed if any values have been written (except for an explicit pad value — it may have been written). For 0-dimensional rVariables this operation is not applicable. Required arguments are as follows:

in: `INTEGER*4 dim_varys(*)`

New dimension variances. Each element of `dim_varys` specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 4.9.

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

`<PUT_,rVAR_HYPERDATA_>`

Writes one or more values to the current rVariable (in the current CDF). The values are written 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:

in: `<type> buffer`

The values. `<type>` depends on the data type of the rVariable. The values in `buffer` are written to the CDF.

WARNING: If the rVariable has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `buffer` must be a `CHARACTER` Fortran variable. If the rVariable does not have one of the character data types, then `buffer` must NOT be a `CHARACTER` Fortran variable.

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.

`<PUT_,rVAR_INITIALRECS_>`

Specifies the number of records to initially write to the current rVariable (in the current CDF). The records are written beginning at record number `1` (one). This may be specified only once per rVariable and before any other records have been written to that rVariable. If a pad value has not yet been specified, the default is used (see the Concepts chapter in the CDF User's Guide). If a pad value has been explicitly specified, that value is written to the records. The Concepts chapter in the CDF User's Guide describes initial records. Required arguments are as follows:

in: `INTEGER*4 num_records`

Number of records to write.

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

<PUT_,rVAR_NAME_>

Renames the current rVariable (in the current CDF). A variable (rVariable or zVariable) with the same name must not already exist in the CDF. Required arguments are as follows:

in: `CHARACTER var_name*(*)`

New name of the rVariable. This may consist of at most `CDF_VAR_NAME_LEN` characters.

UNIX: For the proper operation of `CDF_lib`, `var_name` MUST be a Fortran `CHARACTER` variable or constant.

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

<PUT_,rVAR_PADVALUE_>

Specifies the pad value for the current rVariable (in the current CDF). An rVariable's pad value may be specified (or respecified) at any time without affecting already written values (including where pad values were used). The Concepts chapter in the CDF User's Guide describes variable pad values. Required arguments are as follows:

in: `<type> value`

The pad value. `<type>` depends on the data type of the rVariable. The pad value is written to the CDF from `value`.

WARNING: If the rVariable has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `value` must be a `CHARACTER` Fortran variable. If the rVariable does not have one of the character data types, then `value` must NOT be a `CHARACTER` Fortran variable.

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

<PUT_,rVAR_RECVARY_>

Respecifies the record variance of the current rVariable (in the current CDF). An rVariable's record variance may not be changed if any values have been written (except for an explicit pad value — it may have been written). Required arguments are as follows:

in: `INTEGER*4 rec_vary`

New record variance. Specify one of the variances described in Section 4.9.

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

<PUT_,rVAR_SEQDATA_>

Writes one value to the current rVariable (in the current CDF) at the current sequential value for that rVariable. After the write the current sequential value is automatically incremented to the next value (crossing a record boundary if necessary). If the current sequential value is past the last record for the rVariable, the rVariable is extended as necessary. Required arguments are as follows:

in: `<type> value`

Value. `<type>` depends on the data type of the rVariable. The value is written to the CDF from `value`.

WARNING: If the rVariable has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `value` must be a `CHARACTER` Fortran variable. If the rVariable does not have one of the character data types, then `value` must NOT be a `CHARACTER` Fortran variable.

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 written.

`<PUT_,rVAR_SPARSEARRAYS_>`

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

in: `INTEGER*4 s_arrays_type`

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

in: `INTEGER*4 s_arrays_parms(*)`

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

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

`<PUT_,rVAR_SPARSERECORDS_>`

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

in: `INTEGER*4 s_records_type`

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.

`<PUT_,rVARs_RECDATA_>`

Writes full-physical records to one or more rVariables (in the current CDF). The full-physical records are written 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 to which to write. This must be at least one (1).

in: `INTEGER*4 var_nums(*)`

The rVariables to which to write. 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 of full-physical rVariable records to be written. `<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 get the full-physical records being written.) The order of the full-physical rVariable records in this buffer must agree with the rVariable numbers listed in `varNums` and this buffer must be contiguous — there can be no spacing between full-physical rVariable records. Be careful if using Fortran STRUCTUREs to store 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.

`<PUT_,zENTRY_DATA_>`

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

in: `INTEGER*4 data_type`

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

in: `INTEGER*4 num_elements`

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. For all other data types this is the number of elements in an array of that data type.

in: `<type> value`

The value(s). `<type>` depends on the data type of the zEntry. The value is written to the CDF from `value`.

WARNING: If the zEntry has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `value` must be a `CHARACTER` Fortran variable. If the zEntry does not have one of the character data types, then `value` must NOT be a `CHARACTER` Fortran variable.

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.

`<PUT_,zENTRY_DATASPEC_>`

Modifies the data specification (data type and number of elements) of the zEntry at the current zEntry 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: `INTEGER*4 data_type`

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

in: `INTEGER*4 num_elements`

Number of elements of the 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.

`<PUT_,zVAR_ALLOCATEBLOCK_>`

Specifies a range of records to allocate for the current zVariable (in the current CDF). This operation is only applicable to uncompressed zVariables in single-file CDFs. The Concepts chapter in the CDF User's Guide describes the allocation of variable records. Required arguments are as follows:

in: `INTEGER*4 first_record`

The first record number to allocate.

in: `INTEGER*4 last_record`

The last record number to allocate.

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

`<PUT_,zVAR_ALLOCATERECS_>`

Specifies the number of records to allocate for the current zVariable (in the current CDF). The records are allocated beginning at record number `1` (one). This operation is only applicable to uncompressed zVariables in single-file CDFs. The Concepts chapter in the CDF User's Guide describes the allocation of variable records. Required arguments are as follows:

in: `INTEGER*4 num_records`

Number of records to allocate.

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

`<PUT_,zVAR_BLOCKINGFACTOR_>`[13]

Specifies the blocking factor for the current zVariable (in the current CDF). The Concepts chapter in the CDF User's Guide describes blocking factors. **NOTE:** The blocking factor has no effect for NRV variables or multi-file CDFs.

Required arguments are as follows:

in: `INTEGER*4 blocking_factor`

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

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

`<PUT_,zVAR_COMPRESSION_>`

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

in: `INTEGER*4 c_type`

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

in: `INTEGER*4 c_parms(*)`

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

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

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

<PUT_,zVAR_DATA_>

Writes one value to the current zVariable (in the current CDF). The value is written at the current record number and current dimension indices for that zVariable (in the current CDF). Required arguments are as follows:

in: `<type> value`

The value. `<type>` depends on the data type of the zVariable. The value is written to the CDF from `value`.

WARNING: If the zVariable has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `value` must be a `CHARACTER` Fortran variable. If the zVariable does not have one of the character data types, then `value` must NOT be a `CHARACTER` Fortran variable.

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.

<PUT_,zVAR_DATASPEC_>

Respecifies the data specification (data type and number of elements) of the current zVariable (in the current CDF). A zVariable's data specification may not be changed if the new data specification is not equivalent to the old data specification and any values (including the pad value) have been written. Data specifications are considered equivalent if the data types are equivalent (see the Concepts chapter in the CDF User's Guide) and the number of elements are the same. Required arguments are as follows:

in: `INTEGER*4 data_type`

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

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

<PUT_,zVAR_DIMVARYS_>

Respecifies the dimension variances of the current zVariable (in the current CDF). A zVariable's dimension variances may not be changed if any values have been written (except for an explicit pad value — it may have been written). For 0-dimensional zVariables this operation is not applicable. Required arguments are as follows:

in: `INTEGER*4 dim_varys(*)`

New dimension variances. Each element of `dim_varys` specifies the corresponding dimension variance. For each dimension specify one of the variances described in Section 4.9.

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

<PUT_,zVAR_INITIALRECS_>

Specifies the number of records to initially write to the current zVariable (in the current CDF). The records are written beginning at record number `1` (one). This may be specified only once per zVariable and before any other records have been written to that zVariable. If a pad value has not yet been specified, the default is used (see the Concepts chapter in the CDF User's Guide). If a pad value has been explicitly specified, that value is written to the records. The Concepts chapter in the CDF User's Guide describes initial records. Required arguments are as follows:

in: `INTEGER*4 num_records`

Number of records to write.

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

`<PUT_,zVAR_HYPERDATA_>`

Writes one or more values to the current zVariable (in the current CDF). The values are written 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:

in: `<type> buffer`

The values. `<type>` depends on the data type of the zVariable. The values in `buffer` are written to the CDF.

WARNING: If the zVariable has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `buffer` must be a `CHARACTER` Fortran variable. If the zVariable does not have one of the character data types, then `buffer` must NOT be a `CHARACTER` Fortran variable.

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.

`<PUT_,zVAR_NAME_>`

Renames the current zVariable (in the current CDF). A variable (rVariable or zVariable) with the same name must not already exist in the CDF. Required arguments are as follows:

in: `CHARACTER var_name*(*)`

New name of the zVariable. This may consist of at most `CDF_VAR_NAME_LEN` characters.

UNIX: For the proper operation of `CDF_lib`, `var_name` MUST be a Fortran `CHARACTER` variable or constant.

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

`<PUT_,zVAR_PADVALUE_>`

Specifies the pad value for the current zVariable (in the current CDF). A zVariable's pad value may be specified (or respecified) at any time without affecting already written values (including where pad values were used). The Concepts chapter in the CDF User's Guide describes variable pad values. Required arguments are as follows:

in: `<type> value`

The pad value. `<type>` depends on the data type of the zVariable. The pad value is written to the CDF from `value`.

WARNING: If the zVariable has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `value` must be a `CHARACTER` Fortran variable. If the zVariable does not have one of the character data types, then `value` must NOT be a `CHARACTER` Fortran variable.

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

`<PUT_,zVAR_RECVARY_>`

Respecifies the record variance of the current zVariable (in the current CDF). A zVariable's record variance may not be changed if any values have been written (except for an explicit pad value — it may have been written). Required arguments are as follows:

in: `INTEGER*4 rec_vary`

New record variance. Specify one of the variances described in Section 4.9.

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

`<PUT_,zVAR_SEQDATA_>`

Writes one value to the current zVariable (in the current CDF) at the current sequential value for that zVariable. After the write the current sequential value is automatically incremented to the next value (crossing a record boundary if necessary). If the current sequential value is past the last record for the zVariable, the zVariable is extended as necessary. Required arguments are as follows:

in: `<type> value`

The value. `<type>` depends on the data type of the zVariable. The value is written to the CDF from `value`.

WARNING: If the zVariable has one of the character data types (`CDF_CHAR` or `CDF_UCHAR`), then `value` must be a `CHARACTER` Fortran variable. If the zVariable does not have one of the character data types, then `value` must NOT be a `CHARACTER` Fortran variable.

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 written.

`<PUT_,zVAR_SPARSEARRAYS_>`

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

in: `INTEGER*4 s_arrays_type`

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

in: `INTEGER*4 s_arrays_parms(*)`

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

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

`<PUT_,zVAR_SPARSERECORDS_>`

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

in: `INTEGER*4 s_records_type`

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.

`<PUT_,zVARs_RECDATA_>`

Writes full-physical records to one or more zVariables (in the current CDF). The full physical record for a particular zVariable is written 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: `INTEGER*4 num_vars`

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

in: `INTEGER*4 var_nums(*)`

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

in: `<type> buffer`

The buffer of full-physical zVariable records to be written. `<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 get the full-physical records being written.) The order of the full-physical zVariable records in this buffer must agree with the zVariable numbers listed in `varNums`, and this buffer must be contiguous — there can be no spacing between full-physical zVariable records. Be careful if using Fortran STRUCTUREs to store multiple full-physical zVariable 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 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 the `<SELECT_,zVAR_RECNUMBER_>` operation).

`<SELECT_,ATTR_>`

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

in: `INTEGER*4 attr_num`

Attribute number.

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

`<SELECT_,ATTR_NAME_>`

Explicitly selects the current attribute (in the current CDF) by name. **NOTE:** Selecting the current attribute by number (see `<SELECT_,ATTR_>`) is more efficient. 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 a Fortran `CHARACTER` variable or constant.

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

`<SELECT_,CDF_>`

Explicitly selects the current CDF. Required arguments are as follows:

in: `INTEGER*4 id`

Identifier of the CDF. This identifier must have been initialized by a successful `<CREATE_,CDF_>` or `<OPEN_,CDF_>` operation.

There are no required preselected objects/states.

`<SELECT_,CDF_CACHESIZE_>`

Selects the number of cache buffers to be 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:

in: `INTEGER*4 num_buffers`

The number of cache buffers to be used.

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

`<SELECT_,CDF_DECODING_>`

Selects a decoding (for the current CDF). Required arguments are as follows:

in: `INTEGER*4 decoding`

The decoding. Specify one of the decodings described in Section 4.7.

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

`<SELECT_,CDF_NEGtoPOSfp0_MODE_>`

Selects a `-0.0` to `0.0` mode (for the current CDF). Required arguments are as follows:

in: `INTEGER*4 mode`

The `-0.0` to `0.0` mode. Specify one of the `-0.0` to `0.0` modes described in Section 4.15.

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

`<SELECT_,CDF_READONLY_MODE_>`

Selects a read-only mode (for the current CDF). Required arguments are as follows:

in: `INTEGER*4 mode`

The read-only mode. Specify one of the read-only modes described in Section 4.13.

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

`<SELECT_,CDF_SCRATCHDIR_>`

Selects a directory to be used for scratch files (by the CDF library) for the current CDF. The Concepts chapter in the CDF User's Guide describes how the CDF library uses scratch files. This scratch directory will override the directory specified by the the `CDF$TMP` logical name (on VMS systems) or `CDF_TMP` environment variable (on UNIX and MS-DOS systems). Required arguments are as follows:

in: `CHARACTER scratch_dir*(*)`

The directory to be used for scratch files. The length of this directory specification is limited only by the operating system being used.

UNIX: For the proper operation of `CDF_lib`, `scratch_dir` MUST be a Fortran `CHARACTER` variable or constant.

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

`<SELECT_,CDF_STATUS_>`

Selects the current status code. Required arguments are as follows:

in: `INTEGER*4 status`

CDF status code.

There are no required preselected objects/states.

`<SELECT_,CDF_zMODE_>`

Selects a zMode (for the current CDF). Required arguments are as follows:

in: `INTEGER*4 mode`

The zMode. Specify one of the zModes described in Section 4.14.

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

`<SELECT_,COMPRESS_CACHESIZE_>`

Selects the number of cache buffers to be used for the compression 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:

in: `INTEGER*4 num_buffers`

The number of cache buffers to be used.

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

`<SELECT_,gENTRY_>`

Selects the current gEntry number for all gAttributes in the current CDF. Required arguments are as follows:

in: `INTEGER*4 entry_num`

gEntry number.

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

<SELECT_,rENTRY_>

Selects the current rEntry number for all vAttributes in the current CDF. Required arguments are as follows:

in: `INTEGER*4 entry_num`

rEntry number.

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

<SELECT_,rENTRY_NAME_>

Selects the current rEntry number for all vAttributes (in the current CDF) by rVariable name. The number of the named rVariable becomes the current rEntry number. (The current rVariable is not changed.) **NOTE:** Selecting the current rEntry by number (see <SELECT_,rENTRY_>) is more efficient. Required arguments are as follows:

in: `CHARACTER var_name*(*)`

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

UNIX: For the proper operation of `CDF_lib`, `var_name` MUST be a Fortran `CHARACTER` variable or constant.

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

<SELECT_,rVAR_>

Explicitly selects the current rVariable (in the current CDF) by number. Required arguments are as follows:

in: `INTEGER*4 var_num`

rVariable number.

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

<SELECT_,rVAR_CACHESIZE_>

Selects the number of cache buffers to be 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:

in: `INTEGER*4 num_buffers`

The number of cache buffers to be used.

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

<SELECT_,rVAR_NAME_>

Explicitly selects the current rVariable (in the current CDF) by name. **NOTE:** Selecting the current rVariable by number (see <SELECT_,rVAR_>) is more efficient. Required arguments are as follows:

in: `CHARACTER var_name*(*)`

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

UNIX: For the proper operation of `CDF_lib`, `var_name` MUST be a Fortran `CHARACTER` variable or constant.

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

`<SELECT_,rVAR_RESERVEPERCENT_>`

Selects the reserve percentage to be used for the current rVariable (in 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:

in: `INTEGER*4 percent`

The reserve percentage.

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

`<SELECT_,rVAR_SEQPOS_>`

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

in: `INTEGER*4 rec_num`

Record number.

in: `INTEGER*4 indices(*)`

Dimension indices. Each element of `indices` specifies 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.

`<SELECT_,rVARs_CACHESIZE_>`

Selects the number of cache buffers to be used for all of the rVariable files (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:

in: `INTEGER*4 num_buffers`

The number of cache buffers to be used.

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

`<SELECT_,rVARs_DIMCOUNTS_>`

Selects 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:

in: `INTEGER*4 counts(*)`

Dimension counts. Each element of `counts` specifies the corresponding dimension count.

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

<SELECT_,rVARs_DIMINDICES_>

Selects 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:

in: `INTEGER*4 indices(*)`

Dimension indices. Each element of `indices` specifies the corresponding dimension index.

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

<SELECT_,rVARs_DIMINTERVALS_>

Selects 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:

in: `INTEGER*4 intervals(*)`

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

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

<SELECT_,rVARs_RECCOUNT_>

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

in: `INTEGER*4 rec_count`

Record count.

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

<SELECT_,rVARs_RECINTERVAL_>

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

in: `INTEGER*4 rec_interval`

Record interval.

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

<SELECT_,rVARs_RECNUMBER_>

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

in: `INTEGER*4 rec_num`

Record number.

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

<SELECT_,STAGE_CACHESIZE_>

Selects the number of cache buffers to be 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:

in: `INTEGER*4 num_buffers`

The number of cache buffers to be used.

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

`<SELECT_,zENTRY_>`

Selects the current zEntry number for all vAttributes in the current CDF. Required arguments are as follows:

in: `INTEGER*4 entry_num`

zEntry number.

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

`<SELECT_,zENTRY_NAME_>`

Selects the current zEntry number for all vAttributes (in the current CDF) by zVariable name. The number of the named zVariable becomes the current zEntry number. (The current zVariable is not changed.) **NOTE:** Selecting the current zEntry by number (see `<SELECT_,zENTRY_>`) is more efficient. Required arguments are as follows:

in: `CHARACTER var_name*(*)`

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

UNIX: For the proper operation of `CDF_lib`, `var_name` MUST be a Fortran `CHARACTER` variable or constant.

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

`<SELECT_,zVAR_>`

Explicitly selects the current zVariable (in the current CDF) by number. Required arguments are as follows:

in: `INTEGER*4 var_num`

zVariable number.

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

`<SELECT_,zVAR_CACHESIZE_>`

Selects the number of cache buffers to be 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:

in: `INTEGER*4 num_buffers`

The number of cache buffers to be used.

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

`<SELECT_,zVAR_NAME_>`

Explicitly selects the current zVariable (in the current CDF) by name. **NOTE:** Selecting the current zVariable by number (see `<SELECT_,zVAR_>`) is more efficient. Required arguments are as follows:

in: `CHARACTER var_name*(*)`

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

UNIX: For the proper operation of `CDF_lib`, `var_name` MUST be a Fortran `CHARACTER` variable or constant.

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

`<SELECT_,zVAR_DIMCOUNTS_>`

Selects 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:

in: `INTEGER*4 counts(*)`

Dimension counts. Each element of `counts` specifies the corresponding dimension count.

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

`<SELECT_,zVAR_DIMINDICES_>`

Selects 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:

in: `INTEGER*4 indices(*)`

Dimension indices. Each element of `indices` specifies the corresponding dimension index.

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

`<SELECT_,zVAR_DIMINTERVALS_>`

Selects 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:

in: `INTEGER*4 intervals(*)`

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

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

`<SELECT_,zVAR_RECCOUNT_>`

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

in: `INTEGER*4 rec_count`

Record count.

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

<SELECT_,zVAR_RECINTERVAL_>

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

in: `INTEGER*4 rec_interval`

Record interval.

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

<SELECT_,zVAR_RECNUMBER_>

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

in: `INTEGER*4 rec_num`

Record number.

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

<SELECT_,zVAR_RESERVEPERCENT_>

Selects the reserve percentage to be used for the current zVariable (in 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:

in: `INTEGER*4 percent`

The reserve percentage.

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

<SELECT_,zVAR_SEQPOS_>

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

in: `INTEGER*4 rec_num`

Record number.

in: `INTEGER*4 indices(*)`

Dimension indices. Each element of `indices` specifies 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.

<SELECT_,zVARs_CACHESIZE_>

Selects the number of cache buffers to be used for all of the zVariable files (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:

in: `INTEGER*4 num_buffers`

The number of cache buffers to be used.

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

<SELECT_,zVARs_RECNUMBER_>

Selects the current record number for each zVariable in the current CDF. This operation is provided to simplify the selection of the current record numbers for the zVariables involved in a multiple variable access operation (see the Concepts chapter in the CDF User's Guide). Required arguments are as follows:

in: `INTEGER*4 rec_num`

Record number.

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

6.7 More Examples

Several more examples of the use of `CDF_lib` follow. In each example it is assumed that the current CDF has already been selected (either implicitly by creating/opening the CDF or explicitly with the `<SELECT_,CDF_>` operation).

6.7.1 rVariable Creation.

In this example an rVariable will be created with a pad value being specified; initial records will be written; and the rVariable's blocking factor will be specified. Note that the pad value was specified before the initial records. This results in the specified pad value being written. Had the pad value not been specified first, the initial records would have been written with the default pad value. It is assumed that the current CDF has already been selected.

```
.
.
INCLUDE '<path>cdf.h'
.
.
INTEGER*4 status                      ! Status returned from CDF library.
INTEGER*4 dim_varys(2)                ! Dimension variances.
INTEGER*4 var_num                     ! rVariable number.
REAL*4    pad_value                   ! Pad value.

DATA pad_value/-999.9/
.
.
dim_varys(1) = VARY
dim_varys(2) = VARY
status = CDF_lib (CREATE_, rVAR_, 'HUMIDITY', CDF_REAL4, 1, VARY,
1                                  dim_varys, var_num,
```

```
2                 PUT_, rVAR_PADVALUE_, pad_value,
3                       rVAR_INITIALRECS_, 500,
4                       rVAR_BLOCKINGFACTOR_, 50,
5                 NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

6.7.2 zVariable Creation (Character Data Type).

In this example a zVariable with a character data type will be created with a pad value being specified. It is assumed that the current CDF has already been selected.

```
.
.
INCLUDE '<path>CDF.INC'
.
.
INTEGER*4 status                     ! Status returned from CDF library.
INTEGER*4 dim_varys(1)               ! Dimension variances.
INTEGER*4 var_num                    ! zVariable number.
INTEGER*4 num_dims                   ! Number of dimension.
INTEGER*4 dim_sizes(1)               ! Dimension sizes.
INTEGER*4 num_elems                  ! Number of elements (of the data
                                     ! type).
CHARACTER*10 pad_value               ! Pad value.

DATA pad_value/'**********'/,
0    num_dims/1/,
1    dim_sizes/20/,
2    num_elems/10/
.
.
dim_varys(1) = VARY
status = CDF_lib (CREATE_, zVAR_, 'Station', CDF_CHAR, num_elems, num_dims,
1                          dim_sizes, NOVARY, dim_varys, var_num,
2                 PUT_, zVAR_PADVALUE_, pad_value,
3                 NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

6.7.3 Hyper Read with Subsampling.

In this example an rVariable will be subsampled in a CDF whose rVariables are 2-dimensional and have dimension sizes [100,200]. The CDF is column major, and the data type of the rVariable is CDF_INT2. It is assumed that the current CDF has already been selected.

```
.
.
INCLUDE '<path>CDF.INC'
.
.
INTEGER*4 status               ! Status returned from CDF library.
INTEGER*2 values(50,100)       ! Buffer to receive values.
INTEGER*4 rec_count            ! Record count, one record per hyper get.
INTEGER*4 rec_interval         ! Record interval, set to one to indicate
                               ! contiguous records (really meaningless
                               ! since record count is one).
INTEGER*4 indices(2)           ! Dimension indices, start each read
                               ! at 1,1 of the array.
INTEGER*4 counts(2)            ! Dimension counts, half of the values along
                               ! each dimension will be read.
INTEGER*4 intervals(2)         ! Dimension intervals, every other value
                               ! along each dimension will be read.
INTEGER*4 rec_num              ! Record number.
INTEGER*4 max_rec              ! Maximum rVariable record in the
                               ! CDF - this was determined with a call
                               ! to CDF_inquire.

DATA rec_count/1/, rec_interval/1/, indices/1,1/, counts/50,100/,
1    intervals/2,2/
.
.
status = CDF_lib (SELECT_, rVAR_NAME_, 'BRIGHTNESS',
1                          rVARs_RECCOUNT_, rec_count,
2                          rVARs_RECINTERVAL_, rec_interval,
3                          rVARs_DIMINDICES_, indices,
4                          rVARs_DIMCOUNTS_, counts,
5                          rVARs_DIMINTERVALS_, intervals,
6                 NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

DO rec_num = 1, max_rec
  status = CDF_lib (SELECT_, rVARs_RECNUMBER_, rec_num,
  1                 GET_, rVAR_HYPERDATA_, values,
  2                 NULL_, status)
  IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
  .
  .
  ! process values
  .
  .
END DO
.
.
```

6.7.4 Attribute Renaming.

In this example the attribute named Tmp will be renamed to TMP. It is assumed that the current CDF has already been selected.

```
.
.
INCLUDE '<path>CDF.INC'
.
.
INTEGER*4 status               ! Status returned from CDF library.
.
.
status = CDF_lib (SELECT_, ATTR_NAME_, 'Tmp',
1                 PUT_, ATTR_NAME, 'TMP',
2                 NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

6.7.5 Sequential Access.

In this example the values for a zVariable will be averaged. The values will be read using the sequential access method (see the Concepts chapter in the CDF User's Guide). Each value in each record will be read and averaged. It is assumed that the data type of the zVariable has been determined to be CDF_REAL4. It is assumed that the current CDF has already been selected.

```
.
.
INCLUDE '<path>CDF.INC'
.
.
INTEGER*4 status               ! Status returned from CDF library.
INTEGER*4 var_num              ! zVariable number.
INTEGER*4 rec_num              ! Record number, start at first record.
INTEGER*4 indices(2)           ! Dimension indices.
REAL*4    value                ! Value read.
REAL*8    sum                  ! Sum of all values.
INTEGER*4 count                ! Number of values.
REAL*4    ave                  ! Average value.

DATA indices/1,1/, sum/0.0/, count/0/, rec_num/1/
.
.
status = CDF_lib (GET_, zVAR_NUMBER_, 'FLUX', var_num,
1                 NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
```

```
status = CDF_lib (SELECT_, zVAR_, var_num,
1                          zVAR_SEQPOS_, rec_num, indices,
2                 GET_, zVAR_SEQDATA_, value,
3                 NULL_, status)

DO WHILE (status .GE. CDF_OK)
  sum = sum + value
  count = count + 1
  status = CDF_lib (GET_, zVAR_SEQDATA_, value,
  1                 NULL_, status)
END DO

IF (status .NE. END_OF_VAR) CALL UserStatusHandler (status)

ave = sum / count
.
.
```

6.7.6 Attribute rEntry Writes.

In this example a set of attribute rEntries for a particular rVariable will be written. It is assumed that the current CDF has already been selected.

```
.
.
INCLUDE '<path>CDF.INC'
.
.
INTEGER*4 status                   ! Status returned from CDF library.
REAL*4    scale(2)                 ! Scale, minimum/maximum.

DATA scale/-90.0,90.0/
.
.
status = CDF_lib (SELECT_, rENTRY_NAME_, 'LATITUDE',
1                          ATTR_NAME_, 'FIELDNAM',
2                 PUT_, rENTRY_DATA_, CDF_CHAR, 20, 'Latitude            ',
3                 SELECT_, ATTR_NAME_, 'SCALE',
4                 PUT_, rENTRY_DATA_, CDF_REAL4, 2, scale,
5                 SELECT_, ATTR_NAME_, 'UNITS',
6                 PUT_, rENTRY_DATA_, CDF_CHAR, 20, 'Degrees north       ',
7                 NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

6.7.7 Multiple zVariable Write.

In this example full-physical records will be written to the zVariables in a CDF. Note the ordering of the zVariables (see the Concepts chapter in the CDF User's Guide). It is assumed that the current CDF has already been selected.

```
.
.
INCLUDE '<path>CDF.INC'
.
.
INTEGER*4 status                    ! Status returned from CDF library.
INTEGER*2 time                      ! ‘Time’ value.
BYTE      vector_a(3)               ! ‘vectorA’ values.
REAL*8    vector_b(5)               ! ‘vectorB’ values.
INTEGER*4 rec_number                ! Record number.
BYTE      buffer(45)                ! Buffer of full-physical records.
INTEGER*4 var_numbers(3)            ! Variable numbers.

EQUIVALENCE (vector_b, buffer(1))
EQUIVALENCE (time, buffer(41))
EQUIVALENCE (vector_a, buffer(43))
.
.
status = CDF_lib (GET_, zVAR_NUMBER_, 'vectorB', var_numbers(1),
1                       zVAR_NUMBER_, 'time', var_numbers(2),
2                       zVAR_NUMBER_, 'vectorA', var_numbers(3),
3                 NULL_, status);
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
DO rec_number = 1, 100
  .
  /* read values from input file */
  .
  status = CDF_lib (SELECT_, zVARs_RECNUMBER_, rec_number,
  1                 PUT_, zVARs_RECDATA_, 3L, var_numbers, buffer,
  2                 NULL_, status);
  IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
END DO
.
.
```

Chapter 7

Interpreting CDF Status Codes

Most CDF functions return a status code of type INTEGER*4. The symbolic names for these codes are defined in cdf.inc and should be used in your applications rather than using the true numeric values. Appendix A explains each status code. When the status code returned from a CDF function is tested, the following rules apply.

status > CDF_OK	Indicates successful completion but some additional information is provided. These are informational codes.
status = CDF_OK	Indicates successful completion.
CDF_WARN < status < CDF_OK	Indicates that the function completed but probably not as expected. These are warning codes.
status < CDF_WARN	Indicates that the function did not complete. These are error codes.

The following example shows how you could check the status code returned from CDF functions.

```
INTEGER*4 status
.
.
CALL CDF_function (..., status)   ! any CDF function returning status
IF (status .NE. CDF_OK) THEN
  CALL UserStatusHandler (status, ...)
  .
  .
END IF
```

In your own status handler you can take whatever action is appropriate to the application. An example status handler follows. Note that no action is taken in the status handler if the status is CDF_OK.

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

```
   SUBROUTINE UserStatusHandler (status)
   INTEGER*4 status

   CHARACTER message*(CDF_STATUSTEXT_LEN)

   IF (status .LT. CDF_WARN) THEN
     WRITE (6,10)
10   FORMAT (' ','An error has occurred, halting...')
     CALL CDF_error (status, message)
     WRITE (6,11) message
11   FORMAT (' ',A)
     STOP
   ELSE
     IF (status .LT. CDF_OK) THEN
       WRITE (6,12)
12     FORMAT (' ','Warning, function may not have completed as expected...')
       CALL CDF_error (status, message)
       WRITE (6,13) message
13     FORMAT (' ',A)
     ELSE
       IF (status .GT. CDF_OK) THEN
         WRITE (6,14)
14       FORMAT (' ','Function completed successfully, but be advised that...')
         CALL CDF_error (status, message)
         WRITE (6,15) message
15       FORMAT (' ',A)
       END IF
     END IF
   END IF

   RETURN
   END
```

Explanations for all CDF status codes are available to your applications through the function `CDF_error`. `CDF_error` writes to a text string an explanation of a given status code.

Chapter 8

EPOCH Utility Routines

Several subroutines exist that compute, decompose, parse, and encode CDF_EPOCH values. These subroutines may be called by applications using the CDF_EPOCH data type and are included in the CDF library. The Concepts chapter in the CDF User's Guide describes EPOCH values.

8.1 compute_EPOCH

compute_EPOCH calculates a CDF_EPOCH value given the individual components. If an illegal component is detected, the value returned will be -1.0.

```
      SUBROUTINE compute_EPOCH (year, month, day, hour, minute, second, msec,
     .                          epoch)
      INTEGER*4 year               ! In  -- Year (AD, e.g., 1994).
      INTEGER*4 month              ! In  -- Month (1-12).
      INTEGER*4 day                ! In  -- Day (1-31).
      INTEGER*4 hour               ! In  -- Hour (0-23).
      INTEGER*4 minute             ! In  -- Minute (0-59).
      INTEGER*4 second             ! In  -- Second (0-59).
      INTEGER*4 msec               ! In  -- Millisecond (0-999).
      REAL*8    epoch              ! Out -- CDF_EPOCH value.
```

NOTE: There are two variations on how compute_EPOCH may be used. If the month argument is 0 (zero), then the day argument is assumed to be the day of the year (DOY) having a range of 1 through 366. Also, if the hour, minute, and second arguments are all 0 (zero), then the msec argument is assumed to be the millisecond of the day having a range of 0 through 86400000.

8.2 EPOCH_breakdown

EPOCH_breakdown decomposes a CDF_EPOCH value into the individual components.

```
SUBROUTINE EPOCH_breakdown (epoch, year, month, day, hour, minute, second,
.                           msec)
REAL*8     epoch            ! In  -- The CDF_EPOCH value.
INTEGER*4  year             ! Out -- Year (AD, e.g., 1994).
INTEGER*4  month            ! Out -- Month (1-12).
INTEGER*4  day              ! Out -- Day (1-31).
INTEGER*4  hour             ! Out -- Hour (0-23).
INTEGER*4  minute           ! Out -- Minute (0-59).
INTEGER*4  second           ! Out -- Second (0-59).
INTEGER*4  msec             ! Out -- Millisecond (0-999).
```

8.3 encode_EPOCH

encode_EPOCH encodes a CDF_EPOCH value into the standard date/time character string. The format of the string is dd-mmm-yyyy hh:mm:ss.ccc where dd is the day of the month (1-31), mmm is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec), yyyy is the year, hh is the hour (0-23), mm is the minute (0-59), ss is the second (0-59), and ccc is the millisecond (0-999).

```
SUBROUTINE encode_EPOCH (epoch, ep_string)
REAL*8    epoch                          ! In  -- The CDF_EPOCH value.
CHARACTER ep_string*(EPOCH_STRING_LEN)   ! Out -- The standard date/time
                                         !        character string.
```

EPOCH_STRING_LEN is defined in cdf.inc.

8.4 encode_EPOCH1

encode_EPOCH1 encodes a CDF_EPOCH value into an alternate date/time character string. The format of the string is yyyymmdd.ttttttt, where yyyy is the year, mm is the month (1-12), dd is the day of the month (1-31), and ttttttt is the fraction of the day (e.g., 5000000 is 12 o'clock noon).

```
SUBROUTINE encode_EPOCH1 (epoch, ep_string)
REAL*8    epoch                          ! In  -- The CDF_EPOCH value.
CHARACTER ep_string*(EPOCH1_STRING_LEN)  ! Out -- The alternate date/time
                                         !        character string.
```

EPOCH1_STRING_LEN is defined in cdf.inc.

8.5 encode_EPOCH2

encode_EPOCH2 encodes a CDF_EPOCH value into an alternate date/time character string. The format of the string is yyyymoddhhmmss where yyyy is the year, mo is the month (1-12), dd is the day of the month (1-31), hh is the hour (0-23), mm is the minute (0-59), and ss is the second (0-59).

```
SUBROUTINE encode_EPOCH2 (epoch, ep_string)
REAL*8    epoch                          ! In  -- The CDF_EPOCH value.
CHARACTER ep_string*(EPOCH2_STRING_LEN)  ! Out -- The alternate date/time
                                         !        character string.
```

EPOCH2_STRING_LEN is defined in cdf.inc.

8.6 encode_EPOCH3

encode_EPOCH3 encodes a CDF_EPOCH value into an alternate date/time character string. The format of the string is yyyy-mo-ddThh:mm:ss.cccZ where yyyy is the year, mo is the month (1-12), dd is the day of the month (1-31), hh is the hour (0-23), mm is the minute (0-59), ss is the second (0-59), and ccc is the millisecond (0-999).

```
SUBROUTINE encode_EPOCH3 (epoch, ep_string)
REAL*8    epoch                          ! In  -- The CDF_EPOCH value.
CHARACTER ep_string*(EPOCH3_STRING_LEN)  ! Out -- The alternate date/time
                                         !        character string.
```

EPOCH3_STRING_LEN is defined in cdf.inc.

8.7 encode_EPOCHx

encode_EPOCHx encodes a CDF_EPOCH value into a custom date/time character string. The format of the encoded string is specified by a format string.

```
SUBROUTINE encode_EPOCHx (epoch, format, encoded)
REAL*8    epoch                          ! In  -- The CDF_EPOCH value.
CHARACTER format*(EPOCHx_FORMAT_MAX)     ! In  -- The format string.
CHARACTER encoded*(EPOCHx_STRING_MAX)    ! Out -- The custom date/time
                                         !        character string.
```

The format string consists of EPOCH components which are encoded and text which is simply copied to the encoded custom string. Components are enclosed in angle brackets and consist of a component token and an optional width. The syntax of a component is: <token[.width]>. If the optional width contains a leading zero, then the component will be encoded with leading zeroes (rather than leading blanks).

The supported component tokens and their default widths are as follows. . .

Token	Meaning	Default
dom	Day of month (1-31)	<dom.0>
doy	Day of year (001-366)	<doy.03>
month	Month ('Jan','Feb',...,'Dec')	<month>
mm	Month (1,2,...,12)	<mm.0>
year	Year (4-digit)	<year.04>
yr	Year (2-digit)	<yr.02>
hour	Hour (00-23)	<hour.02>
min	Minute (00-59)	<min.02>
sec	Second (00-59)	<sec.02>
fos	Fraction of second.	<fos.3>
fod	Fraction of day.	<fod.8>

Note that a width of zero indicates that as many digits as necessary should be used to encoded the component. The <month> component is always encoded with three characters. The <fos> and <fod> components are always encoded with leading zeroes.

If a left angle bracket is desired in the encoded string, then simply specify two left angle brackets (<<) in the format string (character stuffing).

For example, the format string used to encode the standard EPOCH date/time character string (see Section 8.3) would be...

```
<dom.02>-<month>-<year> <hour>:<min>:<sec>.<fos>
```

EPOCHx_FORMAT_LEN and EPOCHx_STRING_MAX are defined in cdf.inc.

8.8 parse_EPOCH

parse_EPOCH parses a standard date/time character string and returns a CDF_EPOCH value. The format of the string is that produces by the encode_EPOCH function described in Section 8.3. If an illegal field is detected in the string, the value returned will be -1.0.

```
SUBROUTINE parse_EPOCH (ep_string, epoch)
CHARACTER ep_string*(EPOCH_STRING_LEN)    ! In  -- The standard date/time
                                          !        character string.
REAL*8    epoch                           ! Out -- CDF_EPOCH value.
```

EPOCH_STRING_LEN is defined in cdf.inc.

8.9 parse_EPOCH1

parse_EPOCH1 parses an alternate date/time character string and returns a CDF_EPOCH value. The format of the string is that produces by the encode_EPOCH1 function described in Section 8.4. If an illegal field is detected in the string, the value returned will be -1.0.

```
SUBROUTINE parse_EPOCH1 (ep_string, epoch)
CHARACTER ep_string*(EPOCH1_STRING_LEN)   ! In  -- The alternate date/time
                                          !        character string.
REAL*8    epoch                           ! Out -- CDF_EPOCH value.
```

EPOCH1_STRING_LEN is defined in cdf.inc.

8.10 parse_EPOCH2

parse_EPOCH2 parses an alternate date/time character string and returns a CDF_EPOCH value. The format of the string is that produces by the encode_EPOCH2 function described in Section 8.5. If an illegal field is detected in the string, the value returned will be -1.0.

```
SUBROUTINE parse_EPOCH2 (ep_string, epoch)
CHARACTER ep_string*(EPOCH2_STRING_LEN)   ! In  -- The alternate date/time
                                          !        character string.
REAL*8    epoch                           ! Out -- CDF_EPOCH value.
```

EPOCH2_STRING_LEN is defined in cdf.inc.

8.11 parse_EPOCH3

parse_EPOCH3 parses an alternate date/time character string and returns a CDF_EPOCH value. The format of the string is that produces by the encode_EPOCH3 function described in Section 8.6. If an illegal field is detected in the string, the value returned will be -1.0.

```
SUBROUTINE parse_EPOCH3 (ep_string, epoch)
CHARACTER ep_string*(EPOCH3_STRING_LEN)   ! In  -- The alternate date/time
                                          !        character string.
REAL*8    epoch                           ! Out -- CDF_EPOCH value.
```

EPOCH3_STRING_LEN is defined in cdf.inc.

Appendix A

Status Codes

A.1 Introduction

A status code is returned from most CDF functions. The `cdf.h` (for C) and `CDF.INC` (for Fortran) include files contain the numerical values (constants) for each of the status codes (and for any other constants referred to in the explanations). The CDF library Standard Interface functions `CDFerror` (for C) and `CDF_error` (for Fortran) can be used within a program to inquire the explanation text for a given status code. The Internal Interface can also be used to inquire explanation text.

There are three classes of status codes: informational, warning, and error. The purpose of each is as follows:

Informational	Indicates success but provides some additional information that may be of interest to an application.
Warning	Indicates that the function completed but possibly not as expected.
Error	Indicates that a fatal error occurred and the function aborted.

Status codes fall into classes as follows:

Error codes `<` `CDF_WARN` `<` Warning codes `<` `CDF_OK` `<` Informational codes

`CDF_OK` indicates an unqualified success (it should be the most commonly returned status code). `CDF_WARN` is simply used to distinguish between warning and error status codes.

A.2 Status Codes and Messages

The following list contains an explanation for each possible status code. Whether a particular status code is considered informational, a warning, or an error is also indicated.

`ATTR_EXISTS`	Named attribute already exists — cannot create or rename.

Each attribute in a CDF must have a unique name. Note that trailing blanks are ignored by the CDF library when comparing attribute names. [Error]

ATTR_NAME_TRUNC
: Attribute name truncated to CDF_ATTR_NAME_LEN characters. The attribute was created but with a truncated name. [Warning]

BAD_ALLOCATE_RECS
: An illegal number of records to allocate for a variable was specified. For RV variables the number must be one or greater. For NRV variables the number must be exactly one. [Error]

BAD_ARGUMENT
: An illegal/undefined argument was passed. Check that all arguments are properly declared and initialized. [Error]

BAD_ATTR_NAME
: Illegal attribute name specified. Attribute names must contain at least one character, and each character must be printable. [Error]

BAD_ATTR_NUM
: Illegal attribute number specified. Attribute numbers must be zero (0) or greater for C applications and one (1) or greater for Fortran applications. [Error]

BAD_BLOCKING_FACTOR[1]
: An illegal blocking factor was specified. Blocking factors must be at least zero (0). [Error]

BAD_CACHESIZE
: An illegal number of cache buffers was specified. The value must be at least zero (0). [Error]

BAD_CDF_EXTENSION
: An illegal file extension was specified for a CDF. In general, do not specify an extension except possibly for a single-file CDF which has been renamed with a different file extension or no file extension. [Error]

BAD_CDF_ID
: CDF identifier is unknown or invalid. The CDF identifier specified is not for a currently open CDF. [Error]

BAD_CDF_NAME
: Illegal CDF name specified. CDF names must contain at least one character, and each character must be printable. Trailing blanks are allowed but will be ignored. [Error]

BAD_CDFSTATUS
: Unknown CDF status code received. The status code specified is not used by the CDF library. [Error]

BAD_COMPRESSION_PARM
: An illegal compression parameter was specified. [Error]

BAD_DATA_TYPE
: An unknown data type was specified or encountered. The CDF data types are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]

BAD_DECODING
: An unknown decoding was specified. The CDF decodings are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]

BAD_DIM_COUNT
: Illegal dimension count specified. A dimension count must be at least one (1) and not greater than the size of the dimension.

[1] The status code BAD_BLOCKING_FACTOR was previously named BAD_EXTEND_RECS.