that the current sequential value for an rVariable increments automatically as values are read.

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

in: `INTEGER*4 num_vars`

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

in: `INTEGER*4 var_nums(*)`

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

in: `<type> buffer`

The buffer into which the full-physical rVariable records being read are to be placed. This buffer must be large enough to hold the full-physical records. `<type>` must be a Fortran variable that will be passed by reference and cannot be of type `CHARACTER`. (The CDF library is expecting an address at which to place the full-physical records being read.) The order of the full-physical rVariable records in this buffer will correspond to the rVariable numbers listed in `varNums`, and this buffer will be contiguous — there will be no spacing between full-physical rVariable records. Be careful if using Fortran STRUCTUREs to receive multiple full-physical rVariable records. Fortran compilers on some operating systems will pad between the elements of a STRUCTURE in order to prevent memory alignment errors (i.e., the elements of a STRUCTURE may not be contiguous). See Section 2.3.15 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.

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

out: `CHARACTER var_name*(CDF_VAR_NAME_LEN)`

Name of the zVariable. This character string must be large enough to hold `CDF_VAR_NAME_LEN` characters and will be blank padded if necessary. **NOTE:** On UNIX systems, for the proper operation of `CDF_lib`, `var_name` MUST be of type `CHARACTER`.

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

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

out: INTEGER*4 data_type

Data type. The data types are described in Section 5.7.3.

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

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: INTEGER*4 num_elements

Number of elements of the data type at each value. For character string data types (CDF_CHAR and CDF_UCHAR) this is the number of characters in the string. (Each value consists of the entire string.) For all other data types this will always be one (1) — multiple elements at each value are not allowed for non-character string data types.

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

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 5.7.7.

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

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

out: INTEGER*4 dim_varys(*)

Dimension variances. dim_varys is a 1-dimensional array containing one element per dimension. Each element of dim_varys receives the corresponding dimension variance. The variances are described in Section 5.7.7.

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

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: CHARACTER var_name*(*)

The zVariable name. This may be at most CDF_VAR_NAME_LEN characters. **NOTE:** On UNIX systems, for the proper operation of CDF_lib, var_name MUST be of type CHARACTER.

out: INTEGER*4 var_num

The rVariable number.

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

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

out: `<type> value`

Pad value. `<type>` depends on the data type of the zVariable. The pad value is read from the CDF and placed into memory at address `value`. This buffer must be large enough to hold the pad value. **NOTE:** On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

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

`zVAR_EXTENDRECS_` Inquires the number of extend records for the current zVariable (in the current CDF). Extend records are described in Section 2.3.7. Required arguments are as follows:

out: `INTEGER*4 n_extend_recs`

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

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

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

out: `INTEGER*4 var_max_rec`

Maximum record number.

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

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

out: `INTEGER*4 var_max_rec_alloc`

Maximum record number allocated.

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

`zVAR_nINDEXRECORDS_` Inquires the number of index records for the current zVariable (in the current CDF). This has significance only for zVariables that are in single-file CDFs. Section 2.2.6 describes the indexing scheme used

for variable records in a single-file CDF. Required arguments are as follows:

out: `long *num_records`

Number of index records.

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

`zVAR_nINDEXENTRIES_` Inquires the number of index entries for the current zVariable (in the current CDF). This has significance only for zVariables that are in single-file CDFs. Section 2.2.6 describes the indexing scheme used for variable records in a single-file CDF. Required arguments are as follows:

out: `long *num_entries`

Number of index entries.

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

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

out: `<type> value`

Value. `<type>` depends on the data type of the zVariable. This buffer must be large enough to hold the value. The value is read from the CDF and placed into memory at address `value`. **NOTE:** On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

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

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

out: `<type> buffer`

Value(s). `<type>` depends on the data type of the zVariable. The values are read from the CDF and placed into memory starting at address `buffer`. This buffer must be large enough to hold the values. **NOTE:** On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

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

`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`

Value. `<type>` depends on the data type of the zVariable. The value is read from the CDF and placed into memory at address `value`. This buffer must be large enough to hold the value. **NOTE:** On UNIX systems, if the zVariable has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

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

`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 For-

tran 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 Section 2.3.15 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 `<SELECT_,zVAR_RECNUMBER_>`).

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

out: `INTEGER*4 scope`

Attribute scope. The scopes are described in Section 5.7.8.

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

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

out: `CHARACTER attr_name*(CDF_ATTR_NAME_LEN)`

Attribute name. This character string must be large enough to hold `CDF_ATTR_NAME_LEN` characters and will be blank padded if necessary. **NOTE:** On UNIX systems, for the proper operation of `CDF_lib`, `attr_name` MUST be of type `CHARACTER`.

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

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

in: `CHARACTER attr_name*(*)`

Attribute name. This may be at most `CDF_ATTR_NAME_LEN` characters. **NOTE:** On UNIX systems, for the proper operation of `CDF_lib`, `attr_name` MUST be of type `CHARACTER`.

out: `INTEGER*4 attr_num`

The attribute number.

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

`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. **NOTE:** Use this operation with gAttributes only. Required arguments are as follows:

out: `INTEGER*4 max_entry`

The maximum gEntry number for the attribute.

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

`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. **NOTE:** Use this operation with gAttributes only. Required arguments are as follows:

out: `INTEGER*4 num_entries`

The number of gEntries for the attribute.

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

`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. **NOTE:** Use this operation with vAttributes only. Required arguments are as follows:

out: `INTEGER*4 max_entry`

The maximum rEntry number for the attribute.

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

`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. **NOTE:** Use this operation with vAttributes only. Required arguments are as follows:

out: `INTEGER*4 num_entries`

The number of rEntries for the attribute.

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

`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. **NOTE:** Use this operation with vAttributes only. Required arguments are as follows:

out: `INTEGER*4 max_entry`

The maximum zEntry number for the attribute.

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

`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. **NOTE:** Use this operation with vAttributes only. Required arguments are as follows:

out: `INTEGER*4 num_entries`

The number of zEntries for the attribute.

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

`gENTRY_DATATYPE_` Inquires the data type of the gEntry at the current gEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with gAttributes only. Required arguments are as follows:

out: `INTEGER*4 data_type`

Data type. The data types are described in Section 5.7.3.

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

`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). **NOTE:** Use this operation with gAttributes only. Required arguments are as follows:

out: `INTEGER*4 num_elements`

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

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

`gENTRY_DATA_` Reads the gEntry data value from the current attribute at the current gEntry number (in the current CDF). **NOTE:** Use this operation with gAttributes only. Required arguments are as follows:

out: `<type> value`

Value. `<type>` depends on the data type of the gEntry. The value is read from the CDF and placed into memory at address `value`. This buffer must be large enough to hold the value. **NOTE:** On UNIX systems, if the gEntry has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

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

`rENTRY_DATATYPE_` Inquires the data type of the rEntry at the current rEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are as follows:

out: `INTEGER*4 data_type`

Data type. The data types are described in Section 5.7.3.

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

`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). **NOTE:** Use this operation with vAttributes only. Required arguments are as follows:

out: `INTEGER*4 num_elements`

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

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

`rENTRY_DATA_` Reads the rEntry data value from the current attribute at the current rEntry number (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are as follows:

out: `<type> value`

Value. `<type>` depends on the data type of the rEntry. The value is read from the CDF and placed into memory at address `value`. This buffer must be large enough to hold the value. **NOTE:** On UNIX systems, if the rEntry has a character string data type, then `<type>` MUST be `CHARACTER` to ensure the proper operation of `CDF_lib`.

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

`zENTRY_DATATYPE_` Inquires the data type of the zEntry at the current zEntry number for the current attribute (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are as follows:

out: `INTEGER*4 data_type`

Data type. The data types are described in Section 5.7.3.

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

`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). **NOTE:** Use this operation with vAttributes only. Required arguments are as follows:

out: `INTEGER*4 num_elements`

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

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

zENTRY_DATA_ Reads the zEntry data value from the current attribute at the current zEntry number (in the current CDF). **NOTE:** Use this operation with vAttributes only. Required arguments are as follows:

out: <type> value

Value. <type> depends on the data type of the zEntry. The value is read from the CDF and placed into memory at address value. This buffer must be large enough to hold the value. **NOTE:** On UNIX systems, if the zEntry has a character string data type, then <type> MUST be CHARACTER to ensure the proper operation of CDF_lib.

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

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

out: CHARACTER text*(CDF_STATUSTEXT_LEN)

Text explaining the status code. This string must be large enough to hold CDF_STATUSTEXT_LEN characters and will be blank padded if necessary. **NOTE:** On UNIX systems, for the proper operation of CDF_lib, text MUST be of type CHARACTER.

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

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 <SELECT_,CDF_>).

rVariable Creation. In this example an rVariable will be created with a pad value being specified; initial records will be written; and the number of records to extend the rVariable when necessary 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
CALL 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_EXTENDRECS_, 50,
5             NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
```

zVariable Creation (Character String Data Type). In this example a zVariable with a character string 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 pad_value                  ! Pad value.

DATA pad_value/'**********'/,
0    num_dims/1/,
1    dim_sizes/20/,
2    num_elems/10/
.
.
dim_varys(0) = VARY
CALL 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)
.
.
```

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/
.
.
CALL 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
  CALL 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
.
```

```
.
```

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

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

Sequential Access. In this example the values for a zVariable will be averaged. The values will be read using the sequential access method (see Section 2.3.14). 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/
.
.
CALL CDF_lib (GET_, zVAR_NUMBER_, 'FLUX', var_num,
1             NULL_, status)
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)

CALL 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
  CALL CDF_lib (GET_, zVAR_SEQDATA_, value,
  1             NULL_, status)
END DO

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

ave = sum / count
.
.
```

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/
.
.
CALL 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)
.
.
```

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 Section 2.3.15). 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))
.
.
CALL CDF_lib (GET_, zVAR_NUMBER_, 'vectorB', var_numbers(0),
1                  zVAR_NUMBER_, 'time', var_numbers(1),
2                  zVAR_NUMBER_, 'vectorA', var_numbers(2),
3             NULL_, status);
IF (status .NE. CDF_OK) CALL UserStatusHandler (status)
.
.
DO rec_number = 1, 100
  .
  /* read values from input file */
  .
  CALL CDFlib (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
.
.
```

5.9 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 C 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.

5.10 CDF_EPOCH Values

Several subroutines exist that compute, break down, 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. Section 2.2.9 describes EPOCH values.

5.10.1 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.
```

5.10.2 EPOCH_breakdown

EPOCH_breakdown breaks down 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).
```

5.10.3 parse_EPOCH

parse_EPOCH parses a standard date/time character string and returns a CDF_EPOCH value. The format of the string is defined in Section 2.2.9 (and is also the format produced by the encode_EPOCH function). 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.

5.10.4 encode_EPOCH

encode_EPOCH encodes a CDF_EPOCH value into a standard date/time character string. The format of the string is defined in Section 2.2.9 (and is also the format expected by the parse_EPOCH function).

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

EPOCH_STRING_LEN is defined in cdf.inc.

5.10.5 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  -- CDF_EPOCH value.
CHARACTER ep_string*(EPOCH1_STRING_LEN)   ! Out -- The alternate date/time
                                          !        character string.
```

EPOCH1_STRING_LEN is defined in cdf.inc.

5.10.6 encode_EPOCH2

encode_EPOCH2 encodes a CDF_EPOCH value into an alternate date/time character string. The format of the string is yyyymmddhhmmss where yyyy is the year, mm 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  -- CDF_EPOCH value.
CHARACTER ep_string*(EPOCH2_STRING_LEN)  ! Out -- The alternate date/time
                                         !        character string.
```

`EPOCH2_STRING_LEN` is defined in `cdf.inc`.