IDL startup file provided with the CDF distribution. When IDL is started, this startup file is automatically executed (because `IDL_STARTUP` is defined). The startup file performs the following functions...

1. Makes known to IDL the functions/procedures that are used to call the CDF library (and perform other utility operations).
2. Sets IDL's internal paths. These are used when searching for a batch file to be executed or when locating online help files.
3. Executes a batch file (`cdf.pro`) that creates a number of local variables necessary for calling CDF functions/procedures and then interpreting the results.

A list of messages should be displayed indicating the CDF functions/procedures that have been compiled.

If you are already using your own startup file (pointed to by your definition of `IDL_STARTUP`), you should remove the definition of `IDL_STARTUP` from the CDF "definitions" file being used and do one of the following...

1. Add the contents of the CDF/IDL startup file to your own startup file.
2. Have your startup file execute the CDF/IDL startup file.

The definition of `IDL_STARTUP` in the CDF "definitions" file being used points to the location of the CDF/IDL startup file.

B.3 CDF "Include" Files

A number of parameters (macro constants) are necessary when calling functions in the CDF library from a C program. These parameters are defined in `cdf.h`. When calling the CDF functions/procedures in IDL, local variables are used to accomplish the same thing. An IDL batch[1] file, `cdf.pro`, is provided that creates/initializes the necessary local variables. `cdf.pro` is executed automatically by the startup file when you run IDL. All necessary CDF parameters will be available including the status codes commonly used (`CDF_OK` and `CDF_WARN`). Most users will not need the other status codes since the `CDFerror` function can be used to determine the explanation of a status code. If you do require the other status codes, execute the batch file `cdf1.pro` (the IDL internal path will know where to find it). If you are using the Internal Interface (`CDFlib`), you will also need the parameters (local variables) defined in `cdf2.pro`.

IDL limits the number of local variables that may be defined in any one function/procedure. This limit might be reached if you have to include one or more of the CDF "include" files. If that happens you will have to use one or more of the "mapping" functions provided with the CDF/IDL interface. The use of these "mapping" functions eliminates the need to include the corresponding 'include' file.

Include File	Mapping Function	Purpose
cdf.pro	mcp.pro	Map CDF parameters
cdf1.pro	msc.pro	Map status codes
cdf2.pro	mii.pro	Map Internal Interface

Each function returns the longword value that corresponds to the parameter passed in as a character string. For example, the status code value for `NO_SUCH_CDF` would be retrieved as follows...

[1] These are also known as include files.

```
IDL> status = CDFopen ('test1', id)
IDL> if (status eq MSC('NO_SUCH_CDF')) ...
```

In this case the batch file `cdf1.pro` would not have to be included.

B.4 On-Line Help

IDL provides an on-line help facility to which help for user written procedures/functions can be added. A specially formatted file is provided with the CDF distribution that contains on-line help for the procedures/functions in the CDF/IDL interface. This file is named `cdf25cdfif.help` and is located in the "help" directory of the CDF distribution. If you are using the appropriate "definitions" file, IDL will be able to locate this file when online help is requested. Choose the topic called `CDF25CDFIF` rather than `CDF`. (`CDF` provides online help for IDL's CDF interface — not the interface provided with the CDF distribution.)

B.5 Available Functions

The CDF functions/procedures available at the IDL command line exactly mirror the Standard and Internal Interface functions available for a program written in C.[2] The CDF C Reference Manual describes the functionality of each in detail. The syntax required when calling each function is as follows...

```
IDL> status = CDFlib (op1, ...)
IDL> status = CDFcreate (CDFpath, numDims, dimSizes, encoding, majority, $
IDL>                     id)
IDL> status = CDFopen (CDFpath, id)
IDL> status = CDFdoc (id, version, release, copyright)
IDL> status = CDFinquire (id, numDims, dimSizes, encoding, majority, $
IDL>                      maxRec, numVars, numAttrs)
IDL> status = CDFclose (id)
IDL> status = CDFdelete (id)
IDL> status = CDFerror (status, text)
IDL> status = CDFattrCreate (id, attrName, attrScope, attrNum)
IDL> status = CDFattrRename (id, attrNum, attrName)
IDL> status = CDFattrInquire (id, attrNum, attrName, attrScope, maxEntry)
IDL> status = CDFattrEntryInquire (id, attrNum, entryNum, dataType, $
IDL>                               numElems)
IDL> status = CDFattrPut (id, attrNum, entryNum, dataType, numElems, value)
IDL> status = CDFattrGet (id, attrNum, entryNum, value)
IDL> status = CDFvarCreate (id, varName, dataType, numElems, recVary, $
IDL>                        dimVarys, varNum)
IDL> status = CDFvarRename (id, varNum, varName)
```

[2] With the exception of the multiple variable access operations available via the Internal Interface (`<GET_,rVARs_RECDATA_>`, `<PUT_,rVARs_RECDATA_>`, `<GET_,zVARs_RECDATA_>`, and `<PUT_,zVARs_RECDATA_>`). Because these operations deal with variables having different data types, the use of structures to store the values would be the logical method to use. Unfortunately, IDL does not currently support the manipulation of structures from within a C application (which is how CDF's IDL interface is implemented). Support of these operations will hopefully be added in the future.

```
IDL> status = CDFvarInquire (id, varNum, varName, dataType, numElems, $
IDL>                          recVary, dimVarys)
IDL> status = CDFvarPut (id, varNum, recNum, indices, value)
IDL> status = CDFvarGet (id, varNum, recNum, indices, value)
IDL> status = CDFvarHyperPut (id, varNum, recStart, recCount, recInterval, $
IDL>                           indices, counts, intervals, buffer)
IDL> status = CDFvarHyperGet (id, varNum, recStart, recCount, recInterval, $
IDL>                           indices, counts, intervals, buffer)
IDL> status = CDFvarClose (id, varNum)
IDL> attrNum = CDFattrNum (id, attrName)
IDL> varNum = CDFvarNum (id, varName)
IDL> status = CDFlib (op1, ...)
IDL> epoch = computeEPOCH (year, month, day, hour, minute, second, $
IDL>                       millisecond)
IDL> EPOCHbreakdown, epoch, year, month, day, hour, minute, second, $
IDL>                 millisecond
IDL> epoch = parseEPOCH (epochString)
IDL> encodeEPOCH, epoch, epochString
IDL> encodeEPOCH1, epoch, epochString
IDL> encodeEPOCH2, epoch, epochString
IDL> encodeEPOCH3, epoch, epochString
```

The IDL data types to use for integer arguments are always longwords or arrays of longwords using IDL's data type of `TYP_LONG`.[3] Attribute entries and variable values should have the data types corresponding their CDF data types. CDF pathnames and variable/attribute names are IDL character strings. `CDFid` and `CDFstatus` are also longwords. The CDF C Reference Manual may be used as a guide (especially for `CDFlib`). Arguments passed to the CDF functions/procedures must obviously be created and initialized before the call. Arrays of longwords are necessary for dimension sizes, variances, etc.[4] Arguments returned by the CDF functions will be created to be of the proper data type and size (they do not have to be created before calling the CDF function).

NOTE: There is one difference between how the Internal Interface (`CDFlib`) is used in IDL and in a C/Fortran application. In IDL the current CDF must be selected each time `CDFlib` is called. Failure to select the current CDF may result in IDL crashing.

Also provided are two functions, `row_to_col` and `col_to_row`, that may be used to change the majority of an array (or arrays) of values. This function is necessary because CDF can store variable values in either row or column major order. The syntax of `row_to_col` is as follows...

```
IDL> status = row_to_col (inArray, outArray, numDims, dimSizes, numBytes, $
                          arrayCount
```

where...

[3] On DEC Alpha's running OSF/1, don't confuse the C data type of `long` with IDL's data type of `TYP_LONG`. The C `long` is eight bytes while IDL's `TYP_LONG` is four bytes.

[4] If an array in a call to a CDF function won't be used because a variable is 0-dimensional, a dummy array must still be passed to avoid an error from the CDF library. Note that embedding a call to `lonarr` [such as `lonarr(1)`] in the call to the CDF function will work but will result in a memory leak because the allocated array will not be freed.

`inArray`	The array (or arrays) of values to convert — row major ordering. Any valid IDL data type is allowed.
`outArray`	The converted array (or arrays) of values — column major ordering. This array will be created by `row_to_col`.
`numDims`	The number of dimensions in the array (or arrays). This value must be a longword.
`dimSizes`	The size of each dimension. This must be an array of longwords — one longword per dimension.
`numBytes`	The number of bytes in each value. This value must be a longword.
`arrayCount`	The number of arrays to convert. The order of the arrays will not be affected. This value must be a longword.
`status`	Completion status code. This value will be a longword. The possible values are defined in `cdf1.pro`.

Multiple arrays (records) of CDF variable values should always be stored with the last dimension of the IDL array corresponding to the record number. For example, assume a CDF variable with two dimensions whose sizes are `[180,360]`. If `50` records are to be stored, an IDL array with three dimensions whose sizes are `[180,360,50]` should be used. If the CDF variables have row major ordering, `row_to_col` would be called with `numDims` set to `2`, `dimSizes` set to `[180,360]`, and `arrayCount` set to `50`.

The syntax of `col_to_row` is as follows...

```
IDL> status = col_to_row (inArray, outArray, numDims, dimSizes, numBytes, $
                          arrayCount
```

where...

`inArray`	The array (or arrays) of values to convert — column major ordering. Any valid IDL data type is allowed.
`outArray`	The converted array (or arrays) of values — row major ordering. This array will be created by `col_to_row`.
`numDims`	The number of dimensions in the array (or arrays). This value must be a longword.
`dimSizes`	The size of each dimension. This must be an array of longwords — one longword per dimension.
`numBytes`	The number of bytes in each value. This value must be a longword.
`arrayCount`	The number of arrays to convert. The order of the arrays will not be affected. This value must be a longword.
`status`	Completion status code. This value will be a longword. The possible values are defined in `cdf1.pro`.

`col_to_row` would be used in those cases where an array of values created by IDL (in column-major ordering) are to be written to a row-major CDF.

B.6 Example IDL Session

The following IDL session will open a CDF (unsuccessfully at first — typo), inquire about a variable named `LATITUDE`, and then close the CDF.

```
.
.
IDL> status=CDFopen('yrdy3',id)
IDL> status=CDFerror(status,text)
IDL> print,text
NO_SUCH_CDF: The specified CDF does not exist.
IDL> status=CDFopen('test2',id)
IDL> status=CDFerror(status,text)
IDL> print,text
CDF_OK: Function completed successfully.
IDL> status=CDFvarInquire(id,CDFvarNum(id,'LATITUDE'),varName,dataType,$
IDL> numElems,recVary,dimVarys)
IDL> status=CDFerror(status,text)
IDL> print,text
CDF_OK: Function completed successfully.
IDL> print,dataType,numElems
          21            1
IDL> status=CDFclose(id)
IDL> status=CDFerror(status,text)
IDL> print,text
CDF_OK: Function completed successfully.
.
.
```

As you can see, checking the return status from each call to the CDF library can be fairly tedious. A procedure such as the following could be used to ease status code checking. . .

```
pro checkstatus, status
@cdf
if (status lt CDF_OK) then begin
  statusT = CDFerror (status, text)
  print, text
endif
return
end
```

The sample IDL session could then be as follows. . .

```
.
.
IDL> status=CDFopen('yrdy3',id)
IDL> checkstatus,status
NO_SUCH_CDF: The specified CDF does not exist.
IDL> checkstatus,CDFopen('test2',id)
CDF_OK: Function completed successfully.
IDL> status=CDFvarInquire(id,CDFvarNum(id,'LATITUDE'),varName,dataType,$
IDL> numElems,recVary,dimVarys)
IDL> checkstatus,status
CDF_OK: Function completed successfully.
IDL> print,dataType,numElems
          21           1
IDL> checkstatus,CDFclose(id)
CDF_OK: Function completed successfully.
.
.
```

As you can see, a call to a CDF function can be embedded in a call to **checkstatus**.

B.7 Example IDL Function

The following example function is also available in the "samples" subdirectory of the CDF distribution in a file named `idlload1.pro`.

```
;
;
; This function will load all data for a specified rVariable from a CDF into
; an IDL variable.  All of the IDL/CDF interface routines may be entered at
; the IDL command line but for demonstration purposes they will be used in
; an IDL function.
;
; Argument descriptions:                           data type
; CDF_name      input CDF name                     string
; var_name      variable name in CDF               string
; data          array to be returned to IDL        (see creation below)
;
; Executing
; status = get_var('SAMPLE_CDF','VARNAME',data)
;
function get_var, CDF_name, var_name, data
;
; include all CDF definitions
;
@cdf
;
; Open the CDF, note that CDF_id will be returned as the proper data type
```

```
;
status = CDFopen(CDF_name,CDF_id)
;
; check the status of the open, if a error is detected print a error and
; return
;
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; Inquire about the structure of the CDF
;
status = CDFinquire(CDF_id,num_dims,dim_sizes,encoding,majority,$
                    max_record_num,num_vars,num_attrs)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; Get the number in the CDF of the variable
;
var_num = CDFvarNum(CDF_id, var_name)
if(var_num lt CDF_WARN)then begin
   status = CDFerror(var_num, error_mes)
   print,error_mes
   return,status
endif
;
; Inquire about the variable
;
status = CDFvarInquire(CDF_id, var_num, out_var_name, data_type, $
                       num_bytes, record_variance, dim_variances)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; We will retrieve some of the metadata from this variable.
; FIELDNAM is a CDF standard attribute and is a description of the
; variable.
;
attr_num = CDFattrNum(CDF_id, "FIELDNAM")
if(attr_num lt CDF_WARN)then begin
   status = CDFerror(attr_num, error_mes)
   print,error_mes
   return,status
endif
```

```
status = CDFattrInquire(CDF_id, attr_num, out_attr_name, $
                        attr_scope, max_entry)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; Need the data type and size of this attribute
;
status = CDFattrEntryInquire(CDF_id, attr_num, var_num, $
                             attr_data_type, num_elements)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
status = CDFattrGet(CDF_id, attr_num, var_num, attr_value)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
print,'Loading variable ',var_name,': ',attr_value
;
; We will now get ready to load all data for this variable,
; use the CDF structure to initialize all parameters to perform this
; operation
;
recStart = 0l
recCount = max_record_num+1
recInterval=1l
indices = lonarr(num_dims)
counts = lonarr(num_dims)
intervals = lonarr(num_dims)
for dim_num=0, num_dims-1 do begin
    indices(dim_num) = 0;
    counts(dim_num) = dim_sizes(dim_num)
    intervals(dim_num) = 1
endfor
status = CDFvarHyperGet(CDF_id,var_num, recStart,recCount,recInterval, $
                        indices,counts,intervals,inbuf)
if(status lt CDF_WARN)then begin
   st = CDFerror(status, error_mes)
   print,error_mes
   return,status
endif
;
; IDL needs to have the CDF in column majority in order to plot properly,
; we will supply a routine that will convert from ROW to COLUMN major.
; Converting character data is not supported in this release
```

```
;
if((majority eq ROW_MAJOR) and $
   (data_type ne CDF_CHAR and data_type ne CDF_UCHAR))then begin
   print,'Converting from ROW to COLUMN major'
;
; Need to know the number of bytes of this data type (e.g. 1,2,4,8)
;
   nbytes,data_type,num_bytes_dt
;
; Now convert
;
   status = row_to_col (inbuf, data, num_dims, dim_sizes, num_bytes_dt, $
                        recCount)
endif else begin
;
; No conversion necessary, load input buffer into return data buffer
;
   data = inbuf
endelse
status=CDFclose(CDF_id)
if (status lt CDF_WARN) then begin
  st = CDFerror (status, error_mes)
  print, error_mes
  return, status
endif
return,CDF_OK
end

pro nbytes,data_type,num_bytes
@cdf
num_bytes = 0l
Case 1 of
 (data_type eq CDF_INT1)   : num_bytes = 1l
 (data_type eq CDF_INT2)   : num_bytes = 2l
 (data_type eq CDF_INT4)   : num_bytes = 4l
 (data_type eq CDF_UINT1)  : num_bytes = 1l
 (data_type eq CDF_UINT2)  : num_bytes = 2l
 (data_type eq CDF_UINT4)  : num_bytes = 4l
 (data_type eq CDF_REAL4)  : num_bytes = 4l
 (data_type eq CDF_REAL8)  : num_bytes = 8l
 (data_type eq CDF_EPOCH)  : num_bytes = 8l
 (data_type eq CDF_BYTE)   : num_bytes = 1l
 (data_type eq CDF_FLOAT)  : num_bytes = 4l
 (data_type eq CDF_DOUBLE) : num_bytes = 8l
 (data_type eq CDF_CHAR)   : num_bytes = 1l
 (data_type eq CDF_UCHAR)  : num_bytes = 1l
endcase
return
end
```

Appendix C

Status Codes

C.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.

C.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_CACHESIZE`	An illegal number of cache buffers was specified. The value must be at least one (1). [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_DATA_TYPE`	Unknown data type specified. The CDF data types are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]
`BAD_DECODING`	Unknown decoding 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. [Error]
`BAD_DIM_INDEX`	One or more dimension index is out of range. A valid value must be specified regardless of the dimension variance. Note also that the combination of dimension index, count, and in-

	terval must not specify an element beyond the end of the dimension. [Error]
`BAD_DIM_INTERVAL`	Illegal dimension interval specified. Dimension intervals must be at least one (1). [Error]
`BAD_DIM_SIZE`	Illegal dimension size specified. A dimension size must be at least one (1). [Error]
`BAD_ENCODING`	Unknown data encoding specified. The CDF encodings are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]
`BAD_ENTRY_NUM`	Illegal attribute entry number specified. Entry numbers must be at least zero (0) for C applications and at least one (1) for Fortran applications. [Error]
`BAD_EXTEND_RECS`	An illegal number of extend records has been specified. The number of extend records must be at least zero (0). A value of zero indicates that the default should be used. Note that extend records are only significant for variables that are not NRV and are in single-file CDFs. [Error]
`BAD_FNC_OR_ITEM`	The specified function or item is illegal. Check that the proper number of arguments are specified for each operation being performed. Also make sure that `NULL_` is specified as the last operation. [Error]
`BAD_FORMAT`	Unknown format specified. The CDF formats are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]
`BAD_INITIAL_RECS`	An illegal number of records to initially write has been specified. The number of initial records must be at least one (1). [Error]
`BAD_MAJORITY`	Unknown variable majority specified. The CDF variable majorities are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]
`BAD_MALLOC`	Unable to allocate dynamic memory — system limit reached. Contact CDF User Support if this error occurs. [Error]
`BAD_NEGtoPOSfp0_MODE`	An illegal `-0.0` to `0.0` mode was specified. The `-0.0` to `0.0` modes are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]
`BAD_NUM_DIMS`	The number of dimensions specified is out of the allowed range. Zero (0) through `CDF_MAX_DIMS` dimensions are allowed. If more are needed, contact CDF User Support. [Error]
`BAD_NUM_ELEMS`	The number of elements of the data type is illegal. The number of elements must be at least one (1). For non-character string variables, the number of elements must always be one (1). [Error]

`BAD_NUM_VARS`	Illegal number of variables in a record access operation. [Error]
`BAD_READONLY_MODE`	Illegal read-only mode specified. The CDF read-only modes are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]
`BAD_REC_COUNT`	Illegal record count specified. A record count must be at least one (1). [Error]
`BAD_REC_INTERVAL`	Illegal record interval specified. A record interval must be at least one (1). [Error]
`BAD_REC_NUM`	Record number is out of range. Record numbers must be at least zero (0) for C applications and at least one (1) for Fortran applications. Note that a valid value must be specified regardless of the record variance. [Error]
`BAD_SCOPE`	Unknown attribute scope specified. The attribute scopes are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]
`BAD_VAR_NAME`	Illegal variable name specified. Variable names must contain at least one character and each character must be printable. [Error]
`BAD_VAR_NUM`	Illegal variable number specified. Variable numbers must be zero (0) or greater for C applications and one (1) or greater for Fortran applications. [Error]
`BAD_zMODE`	Illegal zMode specified. The CDF zModes are defined in `cdf.h` for C applications and in `cdf.inc` for Fortran applications. [Error]
`CANNOT_CHANGE`	Because of dependencies on the value, it cannot be changed. Some possible causes of this error follow:

1. Changing a CDF's data encoding after a variable value (including a pad value) or an attribute entry has been written.
2. Changing a CDF's format after a variable has been created.
3. Changing a CDF's variable majority after a variable value (excluding a pad value) has been written.
4. Changing a variable's data specification after a value (including the pad value) has been written to that variable.
5. Changing a variable's record variance after a value (excluding the pad value) has been written to that variable.
6. Changing a variable's dimension variances after a value (excluding the pad value) has been written

to that variable.

7. Allocating or writing "initial" records to a variable after a value (excluding the pad value) has been written to that variable.
8. Changing an attribute entry's data specification where the new specification is not equivalent to the old specification.

[Error]

CDF_CLOSE_ERROR Error detected while trying to close CDF. Check that sufficient disk space exists for the `.cdf` file and that it has not been corrupted. [Error]

CDF_CREATE_ERROR Cannot create CDF specified — error from file system. Make sure sure that sufficient privilege exists to create the `.cdf` file in the disk/directory location specified and that an open file quota has not already been reached. [Error]

CDF_EXISTS The CDF named already exists — cannot create it. The CDF library will not overwrite an existing CDF. [Error]

CDF_INTERNAL_ERROR An unexpected condition has occurred in the CDF library. Report this error to CDFsupport. [Error]

CDF_NAME_TRUNC CDF pathname truncated to `CDF_PATHNAME_LEN` characters. The CDF was created but with a truncated name. [Warning]

CDF_OK Function completed successfully.

CDF_OPEN_ERROR Cannot open CDF specified — error from file system. Check that the `.cdf` file is not corrupted and that sufficient privilege exists to open it. Also check that an open file quota has not already been reached. [Error]

CDF_READ_ERROR Failed to read CDF file — error from file system. Check that the `.cdf` file is not corrupted. [Error]

CDF_WRITE_ERROR Failed to write CDF file — error from file system. Check that the `.cdf` file is not corrupted. [Error]

CORRUPTED_V2_CDF This Version 2 CDF is corrupted. An error has been detected in the CDF's control information. If the CDF file(s) are known to be valid, please contact CDF User Support. [Error]

END_OF_VAR The sequential access current value is at the end of the variable. Reading beyond the end of the last physical value for a variable is not allowed (when performing sequential access). [Error]

FUTURE_CDF The CDF distribution being used is older than the CDF distribution which created/modified the CDF. In this case, there is no guarantee that the CDF will be successfully accessed. [Warning]

IBM_PC_OVERFLOW	An operation involving a buffer greater than 64k bytes in size has been specified. [Error]
ILLEGAL_FOR_SCOPE	The operation is illegal for the attribute's scope. For example, only gEntries may be written for gAttributes — not rEntries or zEntries. [Error]
ILLEGAL_IN_zMODE	The attempted operation is illegal while in zMode. Most operations involving rVariables or rEntries will be illegal. [Error]
ILLEGAL_ON_V1_CDF	The specified operation (i.e., opening) is not allowed on Version 1 CDFs. [Error]
MULTI_FILE_FORMAT	The specified operation is not applicable to CDFs with the multi-file format. For example, it does not make sense to inquire indexing statistics for a variable in a multi-file CDF (indexing is only used in single-file CDFs). [Informational]
NEGATIVE_FP_ZERO	One or more of the values read/written are `-0.0` (an illegal value on VAXes and DEC Alphas running OpenVMS). [Warning]
NO_ATTR_SELECTED	An attribute has not yet been selected. First select the attribute on which to perform the operation. [Error]
NO_CDF_SELECTED	A CDF has not yet been selected. First select the CDF on which to perform the operation. [Error]
NO_DELETE_ACCESS	Deleting is not allowed (read-only access). Make sure that delete access is allowed on the CDF file(s). [Error]
NO_ENTRY_SELECTED	An attribute entry has not yet been selected. First select the entry number on which to perform the operation. [Error]
NO_MORE_ACCESS	Further access to the CDF is not allowed because of a severe error. If the CDF was being modified, closing the CDF may save some of the modifications made since the CDF was opened. [Error]
NO_PADVALUE_SPECIFIED	A pad value has not yet been specified. The default pad value is currently being used for the variable. The default pad value was returned. [Informational]
NO_STATUS_SELECTED	A CDF status code has not yet been selected. First select the status code on which to perform the operation. [Error]
NO_SUCH_ATTR	The named attribute was not found. Note that attribute names are case-sensitive. [Error]
NO_SUCH_CDF	The specified CDF does not exist. Check that the pathname specified is correct. [Error]
NO_SUCH_ENTRY	No such entry for specified attribute. [Error]
NO_SUCH_VAR	The named variable was not found. Note that variable names are case-sensitive. [Error]

`NO_VAR_SELECTED`	A variable has not yet been selected. First select the variable on which to perform the operation. [Error]
`NO_VARS_IN_CDF`	This CDF contains no rVariables. The operation performed is not applicable to a CDF with no rVariables. [Informational]
`NO_WRITE_ACCESS`	Write access is not allowed on the CDF file(s). Make sure that the CDF file(s) have the proper file system privileges and ownership. [Error]
`NOT_A_CDF`	Named CDF is corrupted or not actually a CDF. Contact CDF User Support if you are sure that the specified file is a CDF. [Error]
`READ_ONLY_DISTRIBUTION`	Your CDF distribution has been built to allow only read access to CDFs. Check with your system manager if you require write access. [Error]
`READ_ONLY_MODE`	The CDF is in read-only mode — modifications are not allowed. [Error]
`SINGLE_FILE_FORMAT`	The specified operation is not applicable to CDFs with the single-file format. For example, it does not make sense to close a variable in a single-file CDF. [Informational]
`TOO_MANY_VARS`	A multi-file CDF on an IBM PC may contain only a limited number of variables because of the 8.3 file naming convention of MS-DOS. This consists of 100 rVariables and 100 zVariables. [Error]
`UNSUPPORTED_OPERATION`	The attempted operation is not supported at this time. [Error]
`VAR_ALREADY_CLOSED`	The specified variable is already closed. [Informational]
`VAR_CLOSE_ERROR`	Error detected while trying to close variable file. Check that sufficient disk space exists for the variable file and that it has not been corrupted. [Error]
`VAR_CREATE_ERROR`	An error occurred while creating a variable file in a multi-file CDF. Check that a file quota has not been reached. [Error]
`VAR_EXISTS`	Named variable already exists - cannot create or rename. Each variable in a CDF must have a unique name (rVariables and zVariables can not share names). Note that trailing blanks are ignored by the CDF library when comparing variable names. [Error]
`VAR_NAME_TRUNC`	Variable name truncated to `CDF_VAR_NAME_LEN` characters. The variable was created but with a truncated name. [Warning]
`VAR_OPEN_ERROR`	An error occurred while opening variable file. Check that sufficient privilege exists to open the variable file. Also make sure that the associated variable file exists. [Error]
`VAR_READ_ERROR`	Failed to read variable as requested — error from file system. Check that the associated file is not corrupted. [Error]

`VAR_WRITE_ERROR`	Failed to write variable as requested — error from file system. Check that the associated file is not corrupted. [Error]
`VIRTUAL_RECORD_DATA`	One or more of the records are virtual (never actually written to the CDF). Virtual records do not physically exist in the CDF file(s) but are part of the conceptual view of the data provided by the CDF library. Virtual records are described in the Concepts chapter in the CDF User's Guide. [Informational]

Appendix D

Release Notes

D.1 Introduction

This chapter should provide you with all of the information you need to know to begin using CDF V2.5 assuming that you are already familar with CDF V2.4. For additional information contact CDF User Support.

D.2 Supported Systems

CDF V2.5 is currently supported on the following computers/operating systems.

1. VAX (VMS)
2. Sun (SunOS & SOLARIS)
3. DECstation (ULTRIX)
4. Silicon Graphics Iris & Power Series (IRIX)
5. IBM RS6000 series (AIX)
6. HP 9000 series (HP-UX)
7. IBM PC (MS-DOS)
8. NeXT (Mach)
9. DEC Alpha (OSF/1 & OpenVMS)
10. Macintosh (MacOS 7.0)

Additional ports are planned for the future. They include Cray (UNICOS), Macintosh (AUX), and a 32-bit distribution for the IBM PC (MS-DOS).

D.3 Changes for CDF V2.5

Several changes have been made to the CDF distribution for V2.5. CDF V2.5 differs from CDF V2.4 in the following ways.

1. The distribution has been ported to the Macintosh (MacOS 7.0). A version of the CDF library is available for Symantec THINK C and Macintosh Programmer's Workshop (MPW) C and Fortran applications. The CDF toolkit programs have user interfaces typical of Macintosh applications.
2. A shareable version of the CDF library is available on DEC Alphas running OSF/1. Because of this, support for CDF's IDL interface has also been added for that platform.
3. Create/write access for a CDF with any supported encoding is now supported on all platforms. Previous CDF distributions only supported host and network encodings when creating/writing a CDF.
4. Support for Version 1 CDFs has been eliminated. If necessary, use CDF V2.4 to convert your Version 1 CDFs to Version 2 CDFs before accessing them with CDF V2.5.
5. A cache size qualifier has been added to most of the CDF toolkit programs. With this qualifier, the cache size to be used with open CDFs can be adjusted for increased performance.
6. Most of the functionality of `CDFinquire` has been moved to `SkeletonTable`. `CDFinquire` may still be used to inquire the version of the CDF distribution being used and the default qualifiers for the CDF toolkit programs. This was done in an effort to eliminate redundant capabilities between the two programs.
7. The ability to delete variables, attributes, and attribute entries from a CDF has been added.
8. An operation has been added to the Internal Interface which allows variable records to be allocated in a single-file CDF (but not yet written). The application program is then expected to write to the allocated variable records before closing the CDF. This method is more efficient than using "initial records".
9. The option of converting `-0.0` to `0.0` by the CDF library when reading from a CDF now also applies to writing to a CDF. When `-0.0` to `0.0` mode is enabled, `-0.0` would be converted to `0.0` by the CDF library before being written to a CDF.
10. An encoding qualifier has been added to the `CDFconvert` toolkit program to allow any supported encoding to be selected for the destination CDF of a conversion.
11. Support for the Obsolete Interface has been eliminated. Applications written for CDF Version 1 must now be modified for the Standard (or Internal) Interface of CDF Version 2.
12. The CDF V2.5 distribution is configured for a default format of single-file (previous CDF distributions defaulted to multi-file). This may be changed before building/installing the CDF distribution if desired.
13. A statistics qualifier has been added to some of the CDF toolkit programs. When selected, caching statistics are displayed when a CDF is closed.
14. `CDFskeleton` has been renamed to `SkeletonCDF`. This should ease the confusion experienced by some users. `SkeletonTable` creates a skeleton table and `SkeletonCDF` creates a skeleton CDF. (Note that the symbol `CDFSKELETON` on VMS systems and the environment variable `cdfskeleton` on UNIX systems is still defined in the corresponding "definitions" file for compatibility with existing user systems.)

15. CDFs created with CDF V2.5 will use less disk space than CDFs created with previous CDF distributions. The savings will consist of 1689 bytes per CDF regardless of its contents plus an additional savings of 128 bytes per variable. **NOTE:** A side effect of this is that V2.5 CDFs may only be read with the CDF V2.5 distribution. Previous CDF distributions will not be able to read CDFs created with CDF V2.5.

16. The CDF library has been modified to use less memory for each open CDF (unless a large cache size is specified). This should ease the memory limitation problems encountered on the IBM PC and Macintosh. Access to attribute entry data may be slower because of this change.

17. Assumed attribute scopes are no longer supported. The CDF library will now automatically convert an assumed scope to its definite equivalent scope. Because of this the `CDFscope` toolkit program has also been eliminated. If necessary, use CDF V2.4 to correct attribute scopes. Note that the Internal Interface of CDF V2.5 may also be used to correct an attribute scope.

18. Dynamic-link libraries (DLLs) are now supplied with the MS-DOS distribution for Microsoft and Borland Windows applications.

19. Qualifiers have been added to `CDFedit` which allow attribute entries to either be displayed with the attributes or on separate menus (with one for each attribute).

Appendix E

Glossary

attribute	A CDF object with which entries of metadata are associated.
big-endian	The byte ordering in which the most significant byte (MSB) is stored in the lowest memory location.
caching	The method used by the CDF library to improve performance when accessing a file. An attempt is made to keep commonly accessed blocks of the file in memory rather than repeatedly reading them from disk.
CDF	This term is used in more than one way... 1. The actual files that contain your data/metadata. For example: The CDF library must be used to create a "CDF." 2. The software distribution containing the CDF library, include files, and toolkit. For example: We like using "CDF" to store our data.
CDF distribution	The directory of software consisting of the CDF library, include files, and toolkit.
CDF library	The software library that is used to access a CDF.
CDF toolkit	A set of utility programs which ease the creation, modification, and verification of CDFs.
`CDF_OK`	A completion status code indicating unqualified success.
`cdf.h`	An include file used in C applications.
`cdf.inc`	An include file used in Fortran applications.
`cdfmsf.inc`	An include file used in Microsoft Fortran applications.
column-major	The variable majority where the first index of a multidimensional array of values increments the fastest.

conceptual view	The way that values along a dimension having a variance of `NOVARY` are made to appear as if they do actually exist (only one value is actually physically stored). This also applies to records beyond the last record actually stored. The conceptual view of a variable consists of "virtual" records and values (in addition to the physical records and values actually stored).
current	When the Internal Interface is used, current objects/states are those items affected when an operation is performed. For example, a current CDF is selected and then any operation performed involving a CDF is performed on that CDF (until a different current CDF is selected).
data specification	For a variable or attribute entry the data type and number of elements of that data.
data type	For a variable or attribute entry, the type of data being stored (e.g., integer, floating-point, character).
decoding	The integer/floating-point representation of data values passed to an application by the CDF library as they are read from a CDF. This is independent of the way the data values are physically stored in the CDF.
dimensionality	The number of dimensions and the dimension sizes for the rVariables or a zVariable.
encoding	The integer/floating-point representation of the data values physically stored in a CDF.
entry	A CDF object in which metadata is stored. An entry is associated with an attribute.
`EPOCH`	A variable of the NSSDC Standard used to store "time" values. The `CDF_EPOCH` data type is used with this variable.
error code	A status code indicating that a fatal condition was encountered. The operation was aborted.
extend records	The number of records actually allocated for a variable in a single-file CDF when a new record is written. More records may be allocated than are actually needed in order to keep a variable's records as contiguous as possible (with the assumption that the records will eventually be written by an application).
format	In reference to a CDF, the way in which files are used to store the CDF's control/data/metadata. This may be single-file or multi-file.
full-physical record	A variable record consisting of values exactly as physically stored in the CDF.
gAttribute	A global scoped attribute.
gEntry	An entry for a gAttribute.

global scope	Global scope indicates that an attribute describes some property of the entire CDF.
host decoding	The decoding of the computer currently being used.
host encoding	The encoding of the computer currently being used.
hyper access	A variable access method in which multiple records/values are read/written for a variable.
IDL Interface	A set of functions callable from within IDL (Interactive Data Language) that allow access to CDFs. The CDF distribution contains an IDL interface in addition to the CDF interface built into IDL by Research Systems, Inc. (RSI — the distributors of IDL).
IEEE 754	The floating-point representation of XDR.
include file	A file, included by a C or Fortran application, that contains constants recognized by the CDF library pertaining to various aspects of CDF objects/states.
indexing	The method used in a single-file CDF to keep track of where each variable's records are located.
informational code	A status code indicating success but providing some additional information that may be of interest.
Internal Interface	A set of routines in the CDF library callable from C and Fortran applications that provide all types of access to CDFs.
item	When the Internal Interface is used, an object or state on which an operation is performed.
`libcdf.a`	The CDF library on UNIX systems.
`libcdf.lib`	The CDF library on MS-DOS systems.
`LIBCDF.OLB`	The CDF library on VMS and OpenVMS systems.
little-endian	The byte ordering in which the least significant byte (LSB) is stored in the lowest memory location.
majority	The order in which the values of a multidimensional array are stored. This may be either row-major or column-major.
monotonicity	The property of a variable that specifies whether or not that variable's values increment or decrement (or neither) along a dimension or from record to record.
multi-file	A CDF format. Multi-file CDFs consist of one file for control/metadata and one file per variable of data.
multiple variable access	A variable access method in which one full-physical record is read/written for each of one or more variables.
network encoding	The encoding that uses the XDR representation.

`NOVARY`	A record/dimension variance indicating that the values do not change from record to record or along a dimension.
NRV variable	Non-record variant variable. A variable whose values do not change from record to record (a record variance of `NOVARY`).
NSSDC	National Space Science Data Center.
NSSDC Standard	A set of guidelines for creating a CDF containing attributes expected by NSSDC tools.
number of elements	For a variable the number of instances of the data type at each value. For an attribute entry the number of instances of the data type for that entry.
object	When the Internal Interface is used, an item that exists and may be accessed/manipulated (e.g., a CDF or variable).
operation	When the Internal Interface is used, an action performed on an item (e.g., creating or writing).
pad value	A value written to a variable by the CDF library in those cases where a physical record must be written but not all of its values have been specified by an application. For example, when a single value is written to a new record, all of the other values are written using the pad value.
physical record	A variable record actually stored in a CDF.
physical value	A variable value actually stored in a CDF.
read-only	A mode of the CDF library in which modifications to a CDF are not allowed.
rEntry	An entry for a vAttribute corresponding to an rVariable.
row-major	The variable majority where the last index of a multidimensional array of values increments the fastest.
RV variable	Record variant variable. A variable whose values change from record to record (a record variance of `VARY`).
rVariable	"R" variable. A CDF object in which data values are stored. All rVariables have the same dimensionality.
scope	The intended use for an attribute. This may be global scope or variable scope.
sequential access	A variable access method in which values are read/written in the physical order in which they are stored in the CDF.
single-file	A CDF format. Single-file CDFs are entirely contained within one file.
single value access	A variable access method in which exactly one value is read/written for a variable.

skeleton CDF	A CDF consisting of only control, metadata (attributes and attribute entries), and NRV variable values.
skeleton table	A text file containing the control, metadata, and NRV variable values of a skeleton CDF. A skeleton table is read by the `SkeletonCDF` toolkit program, which then creates the corresponding skeleton CDF. The `SkeletonTable` toolkit program can be used to create a skeleton table from a CDF.
Standard Interface	A set of routines in the CDF library callable from C and Fortran applications that provide access to a commonly used subset of the capabilities of the Internal Interface. This interface was defined with the release of CDF V2.0 and has not changed since. New features since that time are available only through the Internal Interface (e.g., zVariables and zMode).
state	When the Internal Interface is used, a property pertaining to an object (e.g., a CDF's format or variable's data specification).
status code	The result of a CDF function/subroutine call. `CDF_OK` indicates unqualified success.
status handler	A function/subroutine that acts upon a status code received from the CDF library.
variable scope	Variable scope indicates that an attribute describes some property of each variable.
variance, record	The property of a variable that specifies whether or not its values change from record to record.
variance, dimension	The property of a variable that specifies whether or not the values along a dimension change or stay the same.
`VARY`	A record/dimension variance indicating that the values change from record to record or along a dimension.
vAttribute	A variable scoped attribute.
virtual record	A variable record that is not actually stored in a CDF but does appear in the conceptual view of the CDF. Virtual records would be those records beyond the first record of an NRV variable and those records beyond the last record actually written to an RV variable.
virtual value	A variable value this is not actually stored in a CDF but does appear in the conceptual view of the CDF. Virtual values would be those values beyond the first value of a dimension whose variance is `NOVARY`.
warning code	A status code indicating that the operation did complete but probably not as expected.
XDR	External Data Representation. An integer/floating-point representation using big-endian byte ordering and the IEEE 754 floating-point representation.

zEntry	An entry for a vAttribute corresponding to a zVariable.
zMode	A mode of the CDF library in which rVariables are made to appear as zVariables (and rEntries appear as zEntries).
zVariable	"Z" variable. A CDF object in which data values are stored. zVariables can have dimensionalities that are different than those of the rVariables (and each other).

Index