Chapter 4

CDF C Programming Reference

4.1 Introduction

This chapter provides a reference for using the CDF library from C applications. A separate reference exists for writing CDF applications in Fortran (Chapter 5).

4.2 Compiling

Each source file that calls the CDF library or references CDF parameters must include `cdf.h`. On VMS systems a logical name, `CDF$INC`, that specifies the location of `cdf.h` is defined in the definitions file, `DEFINITIONS.COM`, provided with the CDF distribution. On UNIX systems an environment variable, `CDF_INC`, that serves the same purpose is defined in the definitions file `definitions.<shell>` (where `<shell>` is the shell being used: `csh`, `ksh`, `sh`, or `tcsh`). This section assumes that you are using the appropriate definitions file.

One of two methods may be used to include `cdf.h`. They are described in the following sections.

4.2.1 Specifying `cdf.h` Location in the Compile Command

The first method involves including the following line at/near the top of each source file:

```
#include "cdf.h"
```

Since the pathname of the disk/directory containing `cdf.h` was not specified, it must be specified when the source file is compiled.

VMS Systems

An example of the command to compile a source file on VMS systems would be as follows:

```
$ CC/INCLUDE_DIRECTORY=CDF$INC <source-name>
```

where `<source-name>` is the name of the source file being compiled. (The `.C` extension does not have to be specified.) The object module created will be named `<source-name>.OBJ`.

NOTE: If you are running OpenVMS on a DEC Alpha and are using a CDF distribution built for a default double-precision floating-point representation of `D_FLOAT`, you will also have to specify `/FLOAT=D_FLOAT` on the `CC` command line in order to correctly process double-precision floating-point values.

UNIX Systems

An example of the command to compile a source file on UNIX systems would be as follows:

```
% cc -c -I${CDF_INC} <source-name>.c
```

where `<source-name>.c` is the name of the source file being compiled (the `.c` extension is required). The `-c` option specifies that only an object module is to be produced. (The link step is described in Section 4.3.) The object module created will be named `<source-name>.o`.

MS-DOS Systems

An example of the command to compile a source file on MS-DOS systems would be as follows (for Microsoft C and Borland C, respectively):

```
> CL /c /AL /I<inc-path> <source-name>.c
> BCC -c -ml -I<inc-path> <source-name>.c
```

where `<source-name>.c` is the name of the source file being compiled (the `.c` extension is required) and `<inc-path>` is the pathname of the directory containing `cdf.h`. You will need to know where on your system `cdf.h` has been installed.

You may also need to specify the location of system include files. For Microsoft C this is usually accomplished by setting MS-DOS environment variables. For Borland C it may be necessary to also specify `-I<inc-path>` where `<inc-path>` is the location of the Borland C system include files. Consult the documentation for the compiler being used for more information.

The `/c` (for Microsoft C) and `-c` (for Borland C) options specify that only an object module is to be produced. (The link step is described in Section 4.3.) The object module will be named `<source-name>.obj`.

The `/AL` (for Microsoft C) and `-ml` (for Borland C) options specify that the object module is to be compiled using the large memory model. The huge memory model could also be used, if necessary. (The CDF library

can be built using either the large or huge memory model.) All of your application source files must be compiled using the same memory model as that used for the CDF library. The CDF libraries supplied with the CDF distribution for MS-DOS systems are compiled using the large memory model.

Most C compilers for MS-DOS systems have some type of environment in which you can develop your applications (Integrated Developers Environment in Borland C, Programmer's Workbench in Microsoft C, etc.). The options shown above for the command line compilers are specified in the development environment being used. Consult the documentation for the development environment you are using for the steps needed to compile (and link) your application.

4.2.2 Specifying `cdf.h` Location in the Source File

The second method involves specifying the pathname of the directory containing `cdf.h` in the actual source file. The following line would be included at/near the top of each source file:

```
#include "<inc-path>cdf.h"
```

where `<inc-path>` is the pathname of the directory containing `cdf.h`. The source file would then be compiled as shown in Section 4.2.1 but without specifying the location of `cdf.h` on the command line.

On VMS systems `CDF$INC:` may be used for `<inc-path>`. On UNIX and MS-DOS systems `<inc-path>` must be a relative or absolute pathname. (An environment variable may not be used for `<inc-path>` on UNIX systems.) You will need to know where on your system `cdf.h` has been installed.

4.3 Linking

Your applications must be linked with the CDF library (`LIBCDF.OLB` on VMS systems, `libcdf.a` on UNIX systems, and `libcdf.lib` on MS-DOS systems).[1] Both the Standard and Internal interfaces for C applications are built into the CDF library. On VMS systems a logical name, `CDF$LIB`, which specifies the location of the CDF library, is defined in the definitions file, `DEFINITIONS.COM`, provided with the CDF distribution. On UNIX systems an environment variable, `CDF_LIB`, which serves the same purpose, is defined in the definitions file `definitions.<shell>` (where `<shell>` is the shell being used: `csh`, `ksh`, `sh`, or `tcsh`). This section assumes that you are using the appropriate definitions file.

4.3.1 VMS Systems

An example of the command to link your application with the CDF library on VMS systems would be as follows:

```
$ LINK <object-file(s)>, CDF$LIB:LIBCDF/LIBRARY
```

[1] A shareable version of the CDF library is also available on VMS and some flavors of UNIX. Its use is described in Section 4.4.

where <object-file(s)> is your application's object module(s). (The .OBJ extension is not necessary.) The name of the executable created will be the name part of the first object file listed with .EXE appended. A different executable name may be specified by using the /EXECUTABLE qualifier.

4.3.2 OpenVMS Systems

An example of the command to link your application with the CDF library on OpenVMS systems would be as follows:

```
$ LINK <object-file(s)>, CDF$LIB:LIBCDF/LIBRARY, SYS$LIBRARY:<crtl>/LIBRARY
```

where <object-file(s)> is your application's object module(s) (the .OBJ extension is not necessary) and <crtl> is VAXCRTL if your CDF distribution is built for a default double-precision floating-point representation of G_FLOAT or VAXCRTLD for a default of D_FLOAT. The name of the executable created will be the name part of the first object file listed with .EXE appended. A different executable name may be specified by using the /EXECUTABLE qualifier.

4.3.3 UNIX Systems

An example of the command to link your application with the CDF library on UNIX systems would be as follows:

```
% cc <object-file(s)>.o ${CDF_LIB}/libcdf.a
```

where <object-file(s)>.o is your application's object module(s). (The .o extension is required.) The name of the executable created will be a.out by default. It may also be explicitly specified using the -o option. Some UNIX systems may also require that -lc (the C runtime library), -lm (the math library), and/or -ldl (the dynamic linker library) be specified at the end of the command line. This may depend on the particular release of the operating system being used.

Combining the Compile and Link

On UNIX systems the compile and link may be combined into one step as follows:

```
% cc -I${CDF_INC} <source-name(s)>.c ${CDF_LIB}/libcdf.a
```

where <source-name(s)>.c is the name of the source file(s) being compiled/linked. (The .c extension is required.) Some UNIX systems may also require that -lc, -lm, and/or -ldl be specified at the end of the command line.

4.3.4 MS-DOS Systems

An example of the command to link your application with the CDF library on MS-DOS systems would be as follows (for Microsoft C[2] and Borland C, respectively):

```
> LINK /ST:<stack-size> <object-file(s)>,<executable>,nul.map,
       <lib-path>libcdf.lib;
> TLINK /x /L<bc>\lib C0L <object-file(s)>,<executable>,,
        <lib-path>libcdf.lib CL EMU MATHL
```

where `<object-file(s)>` is your application's object module(s) (the `.obj` extension is not necessary); `<executable>` is the name of the executable file to be created (with an extension of `.exe`); and `<lib-path>` is the pathname of the directory containing the CDF library. You will need to know where on your system the CDF library has been installed.

For Microsoft C `<stack-size>` is the size (in decimal bytes) of the executable's stack. A value large enough to prevent runtime stack overflow errors should be specified. (The default value is 2048 [decimal].) A map file is created by default unless the special name `nul.map` is used (as shown). If a map file is desired, the map file parameter should be omitted (in which case the name of the map file will be the name part of the executable file with `.map` appended) or a map file (other than `nul.map`) should be explicitly specified. Note that if `<executable>` is omitted, the name of the executable created will be the name part of the first object module listed with an extension of `.exe` appended.

For Borland C `<bc>` is the directory path of your Borland C software installation top-level directory. This helps Borland C find the necessary startup module (`C0L`) and system libraries (`CL`, `EMU`, and `MATHL`). This example assumes you are using the large memory model. (If the huge memory model were being used, `C0H` and `MATHH` would have been specified instead of `C0L` and `MATHL`, respectively.) The `EMU` library specified indicates that floating-point emulation should be used if a math coprocessor is not present at runtime. If you have a math coprocessor chip, you may want to specify `FP87` instead for increased performance (but the executable will not run on machines that do not have a math coprocessor chip). The omitted parameter (indicated by `,,`) is the name of the map file to be created. A map file is created by default unless the `/x` option is used (as shown). In either case the name of the map file will be the name part of the executable file with `.map` appended (unless a name for the map file is explicitly specified).

NOTE: The same memory model must have been used to compile your application's source files and the CDF library. The CDF library can be compiled using either the large or huge memory model. The CDF libraries supplied with the CDF distribution for MS-DOS systems are compiled using the large memory model.

Most C compilers for MS-DOS systems have some type of environment in which you can develop your applications (Integrated Developers Environment in Borland C, Programmer's Workbench in Microsoft C, etc.). The options shown above for the command line linkers are specified in the development environment being used. Consult the documentation for the development environment you are using for the steps needed to link (and compile) your application.

[2] This example assumes you have properly set the MS-DOS environment variables used by the Microsoft C compiler and linker. There were some changes to the Microsoft C 6.00 and Microsoft C 7.00 runtime libraries (regarding system function names). Because the CDF library is compiled (and distributed) using Microsoft C 6.00, you should be able to successfully link your application with either Microsoft C compiler (since Microsoft C 7.00 is backwards compatible with object modules/libraries created with Microsoft C 6.00).

Combining the Compile and Link

On MS-DOS systems the compile and link may also be combined into one step as follows (for Microsoft C and Borland C, respectively):

```
> CL /AL /F <stack-size> /I<inc-path> <source-name(s)>.c <lib-path>libcdf.lib
> BCC -ml -I<inc-path> <source-name(s)>.c <lib-path>libcdf.lib
```

where `<source-name(s)>.c` is the name of the source file(s) being compiled (the `.c` extension is required); `<inc-path>` is the pathname of the directory containing `cdf.h`; and `<lib-path>` is the pathname of the directory containing the CDF library. You will need to know where on your system `cdf.h` and the CDF library have been installed.

For Microsoft C `<stack-size>` is the size (in hexadecimal bytes) of the executable's stack. A value large enough to prevent runtime stack overflow errors should be specified. (The default value is 800 [hexadecimal].)

4.4 Linking, Shared CDF Library

A shareable version of the CDF library is also available on VMS systems and some flavors of UNIX. The shared version is put in the same directory as the non-shared version and is named as follows:

Machine/Operating System	Shared CDF Library
VAX/VMS & DEC Alpha/OpenVMS	`LIBCDF.EXE`
Sun/SunOS & Sun/SOLARIS	`libcdf.so`
HP 9000/HP-UX	`libcdf.sl`
IBM RS6000/AIX	`libcdf.o`

The commands necessary to link to a shareable library vary among operating systems. Examples are shown in the following sections.

4.4.1 VAX/VMS

```
$ ASSIGN CDF$LIB:LIBCDF.EXE CDF$LIBCDFEXE
$ LINK <object-file(s)>, SYS$INPUT:/OPTIONS
  CDF$LIBCDFEXE/SHAREABLE
  SYS$SHARE:VAXCRTL/SHAREABLE
  <Control-Z>
$ DEASSIGN CDF$LIBCDFEXE
```

where `<object-file(s)>` is your application's object module(s). (The `.OBJ` extension is not necessary.) The name of the executable created will be the name part of the first object file listed with `.EXE` appended. A different executable name may be specified by using the `/EXECUTABLE` qualifier.

NOTE: On VMS systems the shareable CDF library may also be installed in `SYS$SHARE`.

4.4.2 DEC Alpha/OpenVMS

```
$ ASSIGN CDF$LIB:LIBCDF.EXE CDF$LIBCDFEXE
$ LINK <object-file(s)>, SYS$INPUT:/OPTIONS
  CDF$LIBCDFEXE/SHAREABLE
  SYS$LIBRARY:<crtl>/LIBRARY
  <Control-Z>
$ DEASSIGN CDF$LIBCDFEXE
```

where <object-file(s)> is your application's object module(s) (the .OBJ extension is not necessary) and <crtl> is VAXCRTL if your CDF distribution is built for a default double-precision floating-point representation of G_FLOAT or VAXCRTLD for a default of D_FLOAT. The name of the executable created will be the name part of the first object file listed with .EXE appended. A different executable name may be specified by using the /EXECUTABLE qualifier.

NOTE: On OpenVMS systems the shareable CDF library may also be installed in SYS$SHARE.

4.4.3 Sun/SunOS

```
% cc -o <exe-file> <object-file(s)>.o ${CDF_LIB}/libcdf.so -lm -ldl
```

where <object-file(s)>.o is your application's object module(s) (the .o extension is required) and <exe-file> is the name of the executable file created.

4.4.4 Sun/SOLARIS

```
% cc -o <exe-file> <object-file(s)>.o ${CDF_LIB}/libcdf.so -lc -lm
```

where <object-file(s)>.o is your application's object module(s) (the .o extension is required) and <exe-file> is the name of the executable file created.

4.4.5 HP 9000/HP-UX

```
% cc -o <exe-file> <object-file(s)>.o ${CDF_LIB}/libcdf.sl
```

where <object-file(s)>.o is your application's object module(s) (the .o extension is required) and <exe-file> is the name of the executable file created.

4.4.6 IBM RS6000/AIX

```
% cc -o <exe-file> <object-file(s)>.o -L${CDF_LIB} ${CDF_LIB}/libcdf.o -lc -lm
```

where <object-file(s)>.o is your application's object module(s) (the .o extension is required) and <exe-file> is the name of the executable file created.

4.5 Item Referencing

For C applications all items are referenced starting at zero (0). These include variable, attribute, and attribute entry numbers, record numbers, dimensions, and dimension indices. Note that both rVariables and zVariables are numbered starting at zero (0).

4.6 CDF Parameters — Predefined Constants and Types

The following sections describe constants and types defined for use by all CDF application programs written in C. These constants and types are defined in cdf.h. The file cdf.h should be #include'd in all application source files referencing CDF functions/parameters.

4.6.1 Defined Types

The following typedef's are provided. They should be used when declaring or defining the corresponding items.

CDFstatus	All CDF functions except CDFvarNum and CDFattrNum are of type CDFstatus. They return a status code indicating the completion status of the function. The CDFerror function can be used to inquire the meaning of any status code. Appendix C lists the possible status codes along with their explanations. Section 4.8 describes how to interpret status codes.
CDFid	An identifier (or handle) for a CDF that must be used when referring to a CDF. A new CDFid is established whenever a CDF is created or opened, establishing a connection to that CDF on disk. The CDFid is used in all subsequent operations on a particular CDF. The CDFid must not be altered by an application.

4.6.2 CDFstatus Constants

These constants are of type CDFstatus.

CDF_OK	A status code indicating the normal completion of a CDF function.
CDF_WARN	Threshold constant for testing severity of non-normal CDF status codes.

Section 4.8 describes how to use these constants to interpret status codes.

4.6.3 CDF Formats

SINGLE_FILE	The CDF consists of only one file.
MULTI_FILE	The CDF consists of one header file for control and attribute data and one additional file for each variable in the CDF.

4.6.4 CDF Data Types

One of the following constants must be used when specifying a CDF data type for an attribute entry or variable.

CDF_BYTE	1-byte, signed integer.
CDF_CHAR	1-byte, signed character.
CDF_INT1	1-byte, signed integer.
CDF_UCHAR	1-byte, unsigned character.
CDF_UINT1	1-byte, unsigned integer.
CDF_INT2	2-byte, signed integer.
CDF_UINT2	2-byte, unsigned integer.
CDF_INT4	4-byte, signed integer.
CDF_UINT4	4-byte, unsigned integer.
CDF_REAL4	4-byte, floating point.
CDF_FLOAT	4-byte, floating point.
CDF_REAL8	8-byte, floating point.
CDF_DOUBLE	8-byte, floating point.
CDF_EPOCH	8-byte, floating point.

CDF_CHAR and CDF_UCHAR are considered character string data types. These are significant because only variables of these data types may have more than one element per value (where each element is a character).

NOTE: When using a DEC Alpha running OSF/1 keep in mind that a long is 8 bytes and that an int is 4 bytes. Use int C variables with the CDF data types CDF_INT4 and CDF_UINT4 rather than long C variables.

NOTE: When using an IBM PC (MS-DOS) keep in mind that an int is 2 bytes and that a long is 4 bytes. Use long C variables with the CDF data types CDF_INT4 and CDF_UINT4 rather than int C variables.

4.6.5 Data Encodings

A CDF's data encoding affects how its attribute entry and variable data values are stored (on disk). Attribute entry and variable values passed into the CDF library (to be written to a CDF) should always be in the host machine's native encoding. Attribute entry and variable values read from a CDF by the CDF library and passed out to an application will be in the currently selected decoding for that CDF (see Section 2.2.8).

`HOST_ENCODING`	Indicates host machine data representation (native). This encoding will provide the greatest performance when reading/writing on a machine of the same type.
`NETWORK_ENCODING`	Indicates network transportable data representation (XDR).
`VAX_ENCODING`	Indicates VAX data representation. Double-precision floating-point values are encoded in Digital's `D_FLOAT` representation.
`ALPHAVMSd_ENCODING`	Indicates DEC Alpha running OpenVMS data representation. Double-precision floating-point values are encoded in Digital's `D_FLOAT` representation.
`ALPHAVMSg_ENCODING`	Indicates DEC Alpha running OpenVMS data representation. Double-precision floating-point values are encoded in Digital's `G_FLOAT` representation.
`SUN_ENCODING`	Indicates SUN data representation.
`SGi_ENCODING`	Indicates Silicon Graphics Iris and Power Series data representation.
`DECSTATION_ENCODING`	Indicates DECstation data representation.
`ALPHAOSF1_ENCODING`	Indicates DEC Alpha running OSF/1 data representation.
`IBMRS_ENCODING`	Indicates IBMRS data representation (IBM RS6000 series).
`HP_ENCODING`	Indicates HP data representation (HP 9000 series).
`IBMPC_ENCODING`	Indicates IBM PC data representation.
`NeXT_ENCODING`	Indicates NeXT data representation.

When creating a CDF (via the Standard Interface) or respecifying a CDF's encoding (via the Internal Interface), you must specify `NETWORK_ENCODING`, `HOST_ENCODING`, or the specific encoding of the host machine (e.g., `VAX_ENCODING`). Specifying the host machine's encoding explicitly has the same effect as specifying `HOST_ENCODING`.

When inquiring the encoding of a CDF, either `NETWORK_ENCODING` or the specific machine encoding will be returned. (`HOST_ENCODING` is never returned.)

4.6.6 Data Decodings

A CDF's decoding affects how its attribute entry and variable data values are passed out to a calling application. The decoding for a CDF may be selected and reselected any number of times while the CDF is open. Selecting a decoding does not affect how the values are stored in the CDF file(s) — only how the

values are decoded by the CDF library. Any decoding may be used with any of the supported encodings. Section 2.2.8 describes a CDF's decoding in more detail.

HOST_DECODING	Indicates host machine data representation (native). This is the default decoding.
NETWORK_DECODING	Indicates network transportable data representation (XDR).
VAX_DECODING	Indicates VAX data representation. Double-precision floating-point values will be in Digital's D_FLOAT representation.
ALPHAVMSd_DECODING	Indicates DEC Alpha running OpenVMS data representation. Double-precision floating-point values will be in Digital's D_FLOAT representation.
ALPHAVMSg_DECODING	Indicates DEC Alpha running OpenVMS data representation. Double-precision floating-point values will be in Digital's G_FLOAT representation.
SUN_DECODING	Indicates SUN data representation.
SGi_DECODING	Indicates Silicon Graphics Iris and Power Series data representation.
DECSTATION_DECODING	Indicates DECstation data representation.
ALPHAOSF1_DECODING	Indicates DEC Alpha running OSF/1 data representation.
IBMRS_DECODING	Indicates IBMRS data representation (IBM RS6000 series).
HP_DECODING	Indicates HP data representation (HP 9000 series).
IBMPC_DECODING	Indicates IBM PC data representation.
NeXT_DECODING	Indicates NeXT data representation.

The default decoding is HOST_DECODING. The other decodings may be selected via the Internal Interface with the <SELECT_,CDF_DECODING_> operation. Section 2.2.8 describes those situations in which a decoding other than HOST_DECODING may be desired.

4.6.7 Variable Majorities

A CDF's variable majority determines the order in which variable values (within the variable arrays) are stored in the CDF file(s). The majority is the same for rVariable and zVariables.

ROW_MAJOR	C-like array ordering for variable storage. The first dimension in each variable array varies the slowest.
COL_MAJOR	Fortran-like array ordering for variable storage. The first dimension in each variable array varies the fastest.

For C applications the compiler defined majority for arrays is row major. The first dimension of multi-dimensional arrays varies the slowest in memory.

Knowing the majority of a CDF's variables is necessary when performing hyper reads and writes. During a hyper read the CDF library will place the variable data values into the memory buffer in the same majority

as that of the variables. The buffer must then be processed according to that majority. Likewise, during a hyper write, the CDF library will expect to find the variable data values in the memory buffer in the same majority as that of the variables.

The majority must also be considered when performing sequential reads and writes. When sequentially reading a variable, the values passed out by the CDF library will be ordered according to the majority. When sequentially writing a variable, the values passed into the CDF library are assumed (by the CDF library) to be ordered according to the majority.

4.6.8 Record/Dimension Variances

Record and dimension variances affect how variable data values are physically stored.

`VARY`	True record or dimension variance.
`NOVARY`	False record or dimension variance.

If a variable has a record variance of `VARY`, then each record for that variable is physically stored. If the record variance is `NOVARY`, then only one record is physically stored. (All of the other records are virtual and contain the same values.)

If a variable has a dimension variance of `VARY`, then each value/subarray along that dimension is physically stored. If the dimension variance is `NOVARY`, then only one value/subarray along that dimension is physically stored. (All other values/subarrays along that dimension are virtual and contain the same values.)

4.6.9 Attribute Scopes

Attribute scopes are simply a way to explicitly declare the intended use of an attribute by user applications (and the CDF toolkit).

`GLOBAL_SCOPE`	Indicates that an attribute's scope is global (applies to the CDF as a whole).
`VARIABLE_SCOPE`	Indicates that an attribute's scope is by-variable. (Each rEntry or zEntry corresponds to an rVariable or zVariable, respectively.)
`GLOBAL_SCOPE_ASSUMED`	Indicates that an attribute's scope is assumed global.
`VARIABLE_SCOPE_ASSUMED`	Indicates that an attribute's scope is assumed by-variable.

`GLOBAL_SCOPE` and `VARIABLE_SCOPE` are definite scopes. `GLOBAL_SCOPE_ASSUMED` and `VARIABLE_SCOPE_ASSUMED` are assumed scopes.

Section 2.4.3 explains attribute scopes and how assumed scopes can arise in a CDF.

4.6.10 Read-Only Modes

Once a CDF has been opened, it may be placed into a read-only mode to prevent accidental modification (such as when the CDF is simply being browsed). Read-only mode is selected via the Internal Interface using the <SELECT_,CDF_READONLY_MODE_> operation.

READONLYon	Turns on read-only mode.
READONLYoff	Turns off read-only mode.

4.6.11 zModes

Once a CDF has been opened, it may be placed into one of two variations of zMode. zMode is fully explained in Section 2.1.2. A zMode is selected for a CDF via the Internal Interface using the <SELECT_,CDF_zMODE_> operation.

zMODEoff	Turns off zMode.
zMODEon1	Turns on zMode/1.
zMODEon2	Turns on zMode/2.

4.6.12 -0.0 to 0.0 Modes

Once a CDF has been opened, the CDF library may be told to convert -0.0 to 0.0 when read from that CDF. This mode is selected via the Internal Interface using the <SELECT_,CDF_NEGtoPOSfp0_MODE_> operation.

NEGtoPOSfp0on	Convert -0.0 to 0.0 when read from the CDF.
NEGtoPOSfp0off	Do not convert -0.0 to 0.0 when read from the CDF.

4.6.13 Operational Limits

These are limits within the CDF library. If you reach one of these limits, please contact CDF User Support.

CDF_MAX_DIMS	Maximum number of dimensions for the rVariables or a zVariable.

The CDF library imposes no limit on the number of variables, attributes, or attribute entries that a CDF may have. On the IBM PC, however, the number of rVariables and zVariables will be limited to 100 of each in a multi-file CDF because of the 8.3 naming convention imposed by MS-DOS.

4.6.14 Limits of Names and Other Character Strings

CDF_PATHNAME_LEN	Maximum length of a CDF pathname (excluding the NUL[3] terminator and the .cdf or .vnn appended by the CDF library to construct file names). A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating systems being used (including logical names on VMS systems and environment variables on UNIX systems). **NOTE:** On UNIX systems pathnames are case-sensitive.
CDF_VAR_NAME_LEN	Maximum length of a variable name (excluding the NUL terminator).
CDF_ATTR_NAME_LEN	Maximum length of an attribute name (excluding the NUL terminator).
CDF_COPYRIGHT_LEN	Maximum length of the CDF copyright text (excluding the NUL terminator).
CDF_STATUSTEXT_LEN	Maximum length of the explanation text for a status code (excluding the NUL terminator).

4.7 C Function Reference

The following sections describe the CDF library functions callable from C applications. Most functions return a status code of type CDFstatus (see Section 4.8). Each function except CDFlib is part of the Standard Interface — CDFlib is part of the Internal Interface. An application can use both interfaces when necessary. Note that zVariables and vAttribute zEntries are only accessible via the Internal Interface.

Each section begins with an ANSI-style function prototype. The include file cdf.h contains function prototypes for each of the functions in the Standard and Internal interfaces (as well as the EPOCH utility routines). Note that many of the Standard Interface functions are implemented as macros (which call the Internal Interface).

4.7.1 CDFcreate (Standard Interface)

```
CDFstatus CDFcreate(
char  *CDFpath,        /* in  -- CDF pathname. */
long  numDims,         /* in  -- Number of dimensions, rVariables. */
long  dimSizes[],      /* in  -- Dimension sizes, rVariables. */
long  encoding,        /* in  -- Data encoding. */
long  majority,        /* in  -- Variable majority. */
CDFid *id);            /* out -- CDF identifier. */
```

CDFcreate creates a CDF as defined by the arguments. A CDF cannot be created if it already exists. (The existing CDF will not be overwritten.) If you want to overwrite an existing CDF, you must first open it with CDFopen, delete it with CDFdelete, and then recreate it with CDFcreate. If the existing CDF is corrupted, the call to CDFopen will fail. (An error code will be returned.) In this case you must delete the CDF at the command line. Delete the .cdf file, and if the CDF has the multi-file format, delete all of the .v0,.v1,... files and .z0,.z1,... files.

The arguments to CDFcreate are defined as follows:

[3] The ASCII null character, 0 octal.

CDFpath	The pathname of the CDF to create. (Do not specify an extension.) This may be at most CDF_PATHNAME_LEN characters (excluding the NUL terminator). A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on VMS systems and environment variables on UNIX systems). **NOTE:** On UNIX systems pathnames are case-sensitive.
numDims	Number of dimensions the rVariables in the CDF are to have. This may be as few as zero (0) and at most CDF_MAX_DIMS.
dimSizes	The size of each dimension. dimSizes is a 1-dimensional array containing numDims elements. Each element of dimSizes specifies the corresponding dimension size. Each size must be greater then zero (0). If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
encoding	The encoding for variable data and attribute entry data. Specify one of the encodings described in Section 4.6.5.
majority	The majority for variable data. Specify one of the majorities described in Section 4.6.7.
id	The identifier for the created CDF. This identifier must be used in all subsequent operations on the CDF.

When a CDF is created, both read and write access are allowed. The default format for a CDF created with CDFcreate is specified in the configuration file of your CDF distribution. Consult your system manager for this default. The CDFlib function (Internal Interface) may be used to change a CDF's format to single-file.

NOTE: CDFclose must be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk (see Section 4.7.5).

Example(s)

The following example will create a CDF named test1 with network encoding and row majority.

```
.
.
#include "cdf.h"
.
.
CDFid       id;                              /* CDF identifier. */
CDFstatus   status;                          /* Returned status code. */
static long numDims = 3;                     /* Number of dimensions,
                                                rVariables. */
static long dimSizes[3] = {180,360,10};      /* Dimension sizes,
                                                rVariables. */
static long majority = ROW_MAJOR;            /* Variable majority. */
.
.
status = CDFcreate ("test1", numDims, dimSizes, NETWORK_ENCODING,
```

```
                        majority, &id);
   if (status != CDF_OK) UserStatusHandler (status);
   .
   .
```

ROW_MAJOR and NETWORK_ENCODING are defined in cdf.h.

4.7.2 CDFopen (Standard Interface)

```
CDFstatus CDFopen(
char  *CDFpath,        /* in  -- CDF pathname. */
CDFid *id);            /* out -- CDF identifier. */
```

CDFopen opens an existing CDF. The CDF is initially opened with only read access. This allows multiple applications to read the same CDF simultaneously. When an attempt to modify the CDF is made, it is automatically closed and reopened with read/write access. (The function will fail if the application does not have or cannot get write access to the CDF.)

The arguments to CDFopen are defined as follows:

CDFpath	The pathname of the CDF to open. (Do not specify an extension.) This may be at most CDF_PATHNAME_LEN characters (excluding the NUL terminator). A CDF pathname may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on VMS systems and environment variables on UNIX systems). **NOTE:** On UNIX systems pathnames are case-sensitive.
id	The identifier for the opened CDF. This identifier must be used in all subsequent operations on the CDF.

NOTE: CDFclose must be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk (see Section 4.7.5).

Example(s)

The following example will open a CDF named NOAA1.

```
   .
   .
   #include "cdf.h"
   .
   .
   CDFid       id;                     /* CDF identifier. */
   CDFstatus   status;                 /* Returned status code. */
```

```
static char CDFpath[] = { "NOAA1" };   /* Pathname of CDF. */
.
.
status = CDFopen (CDFpath, &id);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.3 CDFdoc (Standard Interface)

```
CDFstatus CDFdoc(
CDFid id,                              /* in  -- CDF identifier. */
long  *version,                        /* out -- Version number. */
long  *release,                        /* out -- Release number. */
char  copyRight[CDF_DOCUMENT_LEN+1]);  /* out -- Copyright. */
```

`CDFdoc` is used to inquire general documentation about a CDF. The version/release of the CDF library that created the CDF is provided (e.g., CDF V2.4 is version 2, release 4) along with the CDF copyright notice.

The arguments to `CDFdoc` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`version`	The version number of the CDF library that created the CDF.
`release`	The release number of the CDF library that created the CDF.
`copyRight`	The copyright notice of the CDF library that created the CDF. This character string must be large enough to hold `CDF_COPYRIGHT_LEN + 1` characters (including the `NUL` terminator). This string will contain a newline character after each line of the copyright notice.

The copyright notice is formatted for printing without modification. The version and release are used together (e.g., CDF V2.4 is version 2, release 4).

Example(s)

The following example will inquire and display the version/release and copyright notice.

```
.
.
#include "cdf.h"
.
.
CDFid     id;                          /* CDF identifier. */
CDFstatus status;                      /* Returned status code. */
```

```
long      version;                          /* CDF version number. */
long      release;                          /* CDF release number. */
char      copyRight[CDF_COPYRIGHT_LEN+1];   /* Copyright notice -- +1 for
                                               NUL terminator. */
.
.
status = CDFdoc (id, &version, &release, copyRight);

if (status < CDF_OK)                    /* INFO status codes ignored */
  UserStatusHandler (status);
else {
  printf ("CDF V%d.%d\n", version, release);
  printf("%s\n", copyRight);
}
.
.
```

4.7.4 CDFinquire (Standard Interface)

```
CDFstatus CDFinquire(
CDFid id,                       /* in  -- CDF identifier */
long  *numDims,                 /* out -- Number of dimensions, rVariables. */
long  dimSizes[CDF_MAX_DIMS],   /* out -- Dimension sizes, rVariables. */
long  *encoding,                /* out -- Data encoding. */
long  *majority,                /* out -- Variable majority. */
long  *maxRec,                  /* out -- Maximum record number in the
                                           CDF, rVariables. */
long  *numVars,                 /* out -- Number of rVariables in
                                           the CDF. */
long  *numAttrs);               /* out -- Number of attributes in the CDF. */
```

`CDFinquire` inquires the basic characteristics of a CDF. An application needs to know the number of rVariable dimensions and their sizes before it can access rVariable data. Knowing the variable majority can be used to optimize performance and is necessary to properly use the variable hyper functions (for both rVariables and zVariables).

The arguments to `CDFinquire` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.
`numDims`	The number of dimensions for the rVariables in the CDF.
`dimSizes`	The dimension sizes of the rVariables in the CDF. `dimSizes` is a 1-dimensional array containing one element per dimension. Each element of `dimSizes` receives the corresponding dimension size. If there are zero (0) dimensions, this argument is ignored (but a place holder is necessary).
`encoding`	The encoding of the variable data and attribute entry data. The encodings are defined in Section 4.6.5.

majority	The majority of the variable data. The majorities are defined in Section 4.6.7.
maxRec	The maximum record number written to an rVariable in the CDF. Note that the maximum record number written is also kept separately for each rVariable in the CDF. The value of maxRec is the largest of these. Some rVariables may have fewer records actually written. CDFlib (Internal Interface) may be used to inquire the maximum record written for an individual rVariable (see Section 4.7.25).
numVars	The number of rVariables in the CDF.
numAttrs	The number of attributes in the CDF.

Example(s)

The following example will inquire the basic information about a CDF.

```
.
.
#include "cdf.h"
.
.
CDFid id;                       /* CDF identifier. */
CDFstatus status;               /* Returned status code. */
long numDims;                   /* Number of dimensions, rVariables. */
long dimSizes[CDF_MAX_DIMS];    /* Dimension sizes, rVariables
                                   (allocate to allow the maximum number
                                   of dimensions). */
long encoding;                  /* Data encoding. */
long majority;                  /* Variable majority. */
long maxRec;                    /* Maximum record number, rVariables. */
long numVars;                   /* Number of rVariables in CDF. */
long numAttrs;                  /* Number of attributes in CDF. */
.
.
status = CDFinquire (id, &numDims, dimSizes, &encoding, &majority,
                     &maxRec, &numVars, &numAttrs);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.5 CDFclose (Standard Interface)

```
CDFstatus CDFclose(
CDFid id);              /* in -- CDF identifier. */
```

CDFclose closes the specified CDF. The CDF's cache buffers are flushed; the CDF's open file is closed (or files in the case of a multi-file CDF); and the CDF identifier is made available for reuse.

NOTE: You must close a CDF with `CDFclose` to guarantee that all modifications you have made will actually be written to the CDF's file(s). If your program exits, normally or otherwise, without a successful call to `CDFclose`, the CDF's cache buffers are left unflushed.

The arguments to `CDFclose` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.

Example(s)

The following example will close an open CDF.

```
  .
  .
 #include "cdf.h"
  .
  .
CDFid id;                       /* CDF identifier. */
CDFstatus status;               /* Returned status code. */
.
.
status = CDFclose (id);
if (status != CDF_OK) UserStatusHandler (status);
.
.
```

4.7.6 CDFdelete (Standard Interface)

```
CDFstatus CDFdelete(
CDFid id);              /* in -- CDF identifier. */
```

`CDFdelete` deletes the specified CDF. The CDF files deleted include the `.cdf` file and, if the CDF consists of multiple files, the `.v0`,`.v1`,... files and `.z0`,`.z1`,... files.

You must open a CDF before you are allowed to delete it. If you have no privilege to delete the CDF files, they will not be deleted. If the CDF is corrupted and cannot be opened, the CDF file(s) must be deleted at the command line.

The arguments to `CDFdelete` are defined as follows:

`id`	The identifier of the CDF. This identifier must have been initialized by a call to `CDFcreate` or `CDFopen`.