CDF
C Reference Manual

Version 2.5, January 10, 1995

National Space Science Data Center

Copyright © 1990 — 1995 NASA/GSFC/NSSDC

National Space Science Data Center
NASA/Goddard Space Flight Center
Greenbelt, Maryland 20771 (U.S.A.)

NSI/DECnet	— NSSDCA::CDFSUPPORT
Internet	— cdfsupport@nssdca.gsfc.nasa.gov

Permission is granted to make and distribute verbatim copies of this document provided this copyright and permission notice are preserved on all copies.

Contents

Chapter 1

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 on those systems. On MS-DOS and Macintosh (MacOS) systems, definitions files are not available. The location of cdf.h is specified as described in the appropriate sections for those systems.

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

1.1 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.

1.1.1 VMS/OpenVMS Systems

An example of the command to compile a source file on VMS/OpenVMS 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.

1.1.2 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 2.3.) The object module created will be named <source-name>.o. Note that in a "makefile" where CDF_INC is imported, $(CDF_INC) would be specified instead of ${CDF_INC}.

1.1.3 MS-DOS Systems, Microsoft C

An example of the command to compile a source file on MS-DOS systems using Microsoft C would be as follows:

```
> CL /c /AL /FPi /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. <inc-path> may be either an absolute or relative pathname.

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. Consult the Microsoft C documentation for more information.

The /c option specifies that only an object module is to be produced. (The link step and a combined compile/link step are described in Section 2.4.) The object module will be named <source-name>.obj.

The /AL option specifies that the object module is to be compiled using the large memory model. The CDF library for Microsoft C supplied with the CDF distribution is compiled using the large memory model. If you need to use the huge memory model for your application, you will also need to rebuild the CDF library for the huge memory model.

The /FPi option specifies how floating-point operations will be handled at run-time. With this option a math coprocessor will be used if it exists; otherwise, the emulation library will be called. Using this option allows your program to run on any MS-DOS system regardless of whether or not a math coprocessor exists. If you know that a math coprocessor exists, you may want to use a floating-point option that provides better performance.

You may instead want to use the Microsoft Programmer's Workbench (PWB) development environment to compile/link your applications. The options shown above for the command line compiler are specified in the development environment. Consult the documentation for the PWB for the steps necessary to compile/link your application.

1.1.4 MS-DOS Systems, Borland C

An example of the command to compile a source file on MS-DOS systems using Borland C would be as follows:

```
> 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. `<inc-path>` may be either an absolute or relative pathname.

You may also need to specify the location of system include files. For Borland C it may be necessary to also specify `-I<bc-inc-path>` where `<bc-inc-path>` is the location of the Borland C system include files. Consult the Borland C documentation for more information.

The `-c` option specifies that only an object module is to be produced. (The link step and a combined compile/link step are described in Section 2.5.) The object module will be named `<source-name>.obj`.

The `-ml` option specifies that the object module is to be compiled using the large memory model. The CDF library for Borland C supplied with the CDF distribution is compiled using the large memory model. If you need to use the huge memory model for your application, you will also need to rebuild the CDF library for the huge memory model.

You may instead want to use the Borland Integrated Developers Environment (IDE) to compile/link your applications. The options shown above for the command line compiler are specified in the development environment. Consult the documentation for the IDE for the steps necessary to compile/link your application.

1.1.5 Macintosh Systems, Symantec THINK C

Symantec THINK C has a development environment in which an application is compiled. The folder containing `cdf.h` must be in the project tree for your application (or in the THINK C project tree). You may also use an `Aliases` folder in your project tree to make known to THINK C the location of the folder containing `cdf.h`. Consult the THINK C documentation for complete details.

You should also set the following THINK C compile options:

1. Check the `#define __STDC__` check box.
2. Do not check the `4-byte ints` check box.
3. Check the `8-byte doubles` check box.
4. Do not check the `Native floating-point format` check box.
5. Check the `Far data` check box (under `Set Project Type...`).

1.1.6 Macintosh Systems, MPW C

Macintosh Programmer's Workshop (MPW) C uses a command line instruction to compile source files. This command may be entered either on the MPW `Worksheet` or in an MPW makefile. An example of the command to compile a source file using MPW C would be as follows:

```
C -i <inc-path> -model far <source-name>.c
```

where `<source-name>.c` is the name of the source file being compiled and `<inc-path>` is an absolute or relative pathname of the folder containing `cdf.h`. You will need to know where on your system `cdf.h` has been installed. Pathnames on a Macintosh are constructed by separating volume/folder names with colons and terminating the pathname with a colon if it is a folder rather than a file (e.g., `Disk1:cdf25-dist:include:`). The name of the object module produced will be `<source-name>.c.o` in the current directory. Note that this example also assumes that `<source-name>.c` is in the current directory.

The `-model far` option indicates that the 32K restrictions on the size of code segments, the jump table, and the global data area are to be removed. This option is necessary in order to successfully link to the CDF library provided for MPW applications. (See Section 2.5.2.)

If your application is fairly large, you may also find it necessary to use the `-s` option to place your compiled source code (object modules) into separate segments when linked. Consult the MPW C documentation for more details.

1.2 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 1.1 but without specifying the location of `cdf.h` on the command line (where applicable).

On VMS systems `CDF$INC:` may be used for `<inc-path>`. On UNIX, MS-DOS, and Macintosh 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. On Macintosh systems, pathnames are constructed by separating volume/folder names with colons.

Chapter 2

Linking

Your applications must be linked with the CDF library.[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 on those systems. On MS-DOS and Macintosh (MacOS) systems, definitions files are not available. The location of the CDF library is specified as described in the appropriate sections for those systems.

2.1 VAX/VMS & VAX/OpenVMS Systems

An example of the command to link your application with the CDF library (`LIBCDF.OLB`) on VAX/VMS and VAX/OpenVMS systems would be as follows:

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

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.

2.2 DEC Alpha/OpenVMS Systems

An example of the command to link your application with the CDF library (`LIBCDF.OLB`) on DEC Alpha/OpenVMS systems would be as follows:

[1] A shareable version of the CDF library is also available on VMS and some flavors of UNIX. Its use is described in Chapter 3. A dynamic link library (DLL), `LIBCDF.DLL`, is available on MS-DOS systems for Microsoft and Borland Windows applications. Consult the Microsoft and Borland documentation for details on using a DLL. Note that the DLL for Microsoft is created using Microsoft C 7.00.

```
$ 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.

2.3 UNIX Systems

An example of the command to link your application with the CDF library (`libcdf.a`) 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 run-time 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. Note that in a "makefile" where `CDF_INC` is imported, `$(CDF_INC)` would be specified instead of `${CDF_INC}`.

2.3.1 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.

2.4 MS-DOS Systems, Microsoft C

An example of the command to link your application with the CDF library (`libcdf.lib`) on MS-DOS systems using Microsoft C 6.00 would be as follows:[2]

[2] This example assumes you have properly set the MS-DOS environment variables (e.g., `INCLUDE` and `LIB`) used by the Microsoft C compiler and linker. Note that there are some differences between the Microsoft C 6.00 and Microsoft C 7.00 run-time 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.

```
> LINK /NOI /NOD /ST:<size> <objs>,<exe>,nul.map,<lib-path>libcdf+LLIBCE;
```

where `<objs>` is your application's object module(s) (the `.obj` extension is not necessary); `<exe>` is the name of the executable file to be created (with a default 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. `<lib-path>` may be either an absolute or relative pathname.

`<size>` is the size (in decimal bytes) of the executable's stack. A value large enough to prevent run-time 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.

The `/NOI` option specifies that function names are to remain case-sensitive. The `/NOD` option specifies that the default libraries (named in object files) should not be used. The needed libraries must instead be named in the link command. The C run-time library shown, `LLIBCE`, assumes the large memory model and emulated floating-point operations if a coprocessor does not exist at run-time. If Microsoft C 7.00 is being used, the library named `OLDNAMES` must also be specified to handle the function naming differences between the Microsoft C 6.00 and Microsoft C 7.00 run-time libraries.

NOTE: The same memory model must have been used to compile your application's source files and the CDF library. The CDF library for Microsoft C supplied with the CDF distribution is compiled using the large memory model. If you need to use the huge memory model for your application, you will also have to rebuild the CDF library for the huge memory model.

You may instead want to use the Microsoft Programmer's Workbench (PWB) development environment to compile/link your applications. The options shown above for the command line linker are specified in the development environment. Consult the documentation for the PWB for the steps necessary to compile/link your application.

2.5 MS-DOS Systems, Borland C

An example of the command to link your application with the CDF library (`libcdf.lib`) on MS-DOS systems using Borland C would be as follows:

```
> TLINK /x /L<bc>\lib COL <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 a default 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. `<inc-path>` may be either an absolute or relative pathname.

`<bc>` is the directory path of your Borland C software installation top-level directory. This allows Borland C to find the necessary startup module (`COL`) 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, `COH` and `MATHH` would have been specified instead of `COL` and `MATHL`, respectively.) The `EMU` library specified indicates that floating-point emulation should be used if a math coprocessor is not present at run-time. 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).

You may instead want to use the Borland C Integrated Developers Environment (IDE) to compile/link your applications. The options shown above for the command line compiler are specified in the development environment. Consult the documentation for the IDE for the steps necessary to compile/link your application.

NOTE: The same memory model must have been used to compile your application's source files and the CDF library. The CDF library for Borland C supplied with the CDF distribution is compiled using the large memory model. If you need to use the huge memory model for your application, you will also have to rebuild the CDF library for the huge memory model.

2.5.1 Macintosh Systems, Symantec THINK C

The CDF library for Symantec THINK C is distributed as five project files named **libcdf1.**π, **libcdf2.**π, **libcdf3.**π, **libcdf4.**π, and **libcdf5.**π. This is necessary because the CDF library must be split into five separate segments. Each CDF library project file must be placed in a separate segment in your application project.

Symantec THINK C has a development environment in which an application is linked. The folder containing the CDF library project files must be in the project tree for your application (or in the THINK C project tree). You may also use an `Aliases` folder in your project tree to make known to THINK C the location of the folder containing the CDF library project files. Consult the THINK C documentation for complete details.

You should also set the following THINK C link option(s):

1. Check the `Far data` check box.

The CDF library expects to be linked with an ANSI library of C run-time functions built using 2-byte `int`'s and 8-byte `float`'s. The ANSI library supplied with THINK C does not use 8-byte `float`'s. You will either need to build a version of the ANSI C run-time library with the above requirements or use the ANSI library supplied with the CDF distribution named **ANSIcdf.**π (which is the ANSI library used to link the CDF toolkit and test programs).

The CDF library does not use Macintosh resources. If your application uses resources, they must be compiled/linked as described in the THINK C documentation.

2.5.2 Macintosh Systems, MPW

Macintosh Programmer's Workshop (MPW) uses a command line instruction to link an application. This command may be entered either on the MPW `Worksheet` or in an MPW makefile. An example of the command to link an application with the CDF library (`libcdf.o`) using MPW would be as follows:

```
Link -t APPL -c '????' -model far δ
     <object-file>.c.o <object-file>.c.o ... <object-file>.c.o δ
     <lib-path>libcdf.o δ
     "{CLibraries}"<c-lib> "{CLibraries}"<c-lib> ... "{CLibraries}"<c-lib> δ
     "{Libraries}"<mac-lib> "{Libraries}"<mac-lib> ... "{Libraries}"<mac-lib> δ
     -o <appl-path>
```

where `<object-file>.c.o` is the name of one or more object modules being linked; `<lib-path>` is an absolute or relative pathname of the folder containing `libcdf.o`; `<c-lib>` is the name of one or more needed C libraries; `<mac-lib>` is the name of one or more needed Macintosh libraries; and `<appl-path>` is the pathname of the application being linked. You will need to know where on your system `libcdf.o` has been installed. Pathnames on a Macintosh are constructed by separating volume/folder names with colons and terminating the pathname with a colon if it is a folder rather than a file (e.g., `Disk1:cdf25-dist:lib:`). Note that this example assumes that `<object-file>.c.o` is in the current directory.

The C libraries that may be needed for the link are `StdCLib.o`, `Math.o`, and `CSANELib.o`. The Macintosh libraries that may be needed are `Runtime.o` and `Interface.o`. Note that `"{CLibraries}"` and `"{Libraries}"` are predefined by MPW.

The `-model far` option indicates that the 32K restrictions on the size of code segments, the jump table, and the global data area are to be removed. This option is necessary in order to successfully link to the CDF library provided for MPW applications.

The CDF library does not use Macintosh resources. If your application uses resources, they must be compiled/linked as described in the MPW documentation.

Chapter 3

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 & OpenVMS)	`LIBCDF.EXE`
DEC Alpha (OpenVMS)	`LIBCDF.EXE`
Sun (SunOS)	`libcdf.so`
Sun (SOLARIS)	`libcdf.so`
HP 9000 (HP-UX)	`libcdf.sl`
IBM RS6000 (AIX)	`libcdf.o`
DEC Alpha (OSF/1)	`libcdf.so`

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

3.1 VAX (VMS & OpenVMS)

```
$ 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 VAX/VMS and VAX/OpenVMS systems the shareable CDF library may also be installed in `SYS$SHARE`. If that is the case, the link command would be as follows:

```
$ LINK <object-file(s)>, SYS$INPUT:/OPTIONS
  SYS$SHARE:LIBCDF/SHAREABLE
  SYS$SHARE:VAXCRTL/SHAREABLE
  <Control-Z>
```

3.2 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 DEC Alpha/OpenVMS systems the shareable CDF library may also be installed in `SYS$SHARE`. If that is the case, the link command would be as follows:

```
$ LINK <object-file(s)>, SYS$INPUT:/OPTIONS
  SYS$SHARE:LIBCDF/SHAREABLE
  SYS$LIBRARY:<crtl>/LIBRARY
  <Control-Z>
```

3.3 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. Note that in a "makefile" where `CDF_LIB` is imported, `$(CDF_LIB)` would be specified instead of `${CDF_LIB}`.

3.4 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. Note that in a "makefile" where `CDF_LIB` is imported, `$(CDF_LIB)` would be specified instead of `${CDF_LIB}`.

3.5 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. Note that in a "makefile" where `CDF_LIB` is imported, `$(CDF_LIB)` would be specified instead of `${CDF_LIB}`.

3.6 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. Note that in a "makefile" where `CDF_LIB` is imported, `$(CDF_LIB)` would be specified instead of `${CDF_LIB}`.

3.7 DEC Alpha (OSF/1)

```
% cc -o <exe-file> <object-file(s)>.o ${CDF_LIB}/libcdf.so -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. Note that in a "makefile" where `CDF_LIB` is imported, `$(CDF_LIB)` would be specified instead of `${CDF_LIB}`.

On a DEC Alpha running OSF/1, when executing a program linked to the shareable CDF library, the environment variable `LD_LIBRARY_PATH` must be set to include the directory containing `libcdf.so`.

Chapter 4

Programming Interface

The following sections describe various aspects of the C programming interface for CDF applications. These include 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 routines/parameters.

4.1 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).

4.2 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 A lists the possible status codes along with their explanations. Chapter 7 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.

4.3 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.

Chapter 7 describes how to use these constants to interpret status codes.

4.4 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.

4.5 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 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 the Concepts chapter in the CDF User's Guide).

`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.
`MAC_ENCODING`	Indicates Macintosh data representation.

When creating a CDF (via the Standard Interface) or respecifying a CDF's encoding (via the Internal Interface), you may specify any of the encodings listed above.[1] 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 a specific machine encoding will be returned. (`HOST_ENCODING` is never returned.)

4.7 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. The Concepts chapter in the CDF User's Guide 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.
`MAC_DECODING`	Indicates Macintosh data representation.

The default decoding is `HOST_DECODING`. The other decodings may be selected via the Internal Interface with the `<SELECT_,CDF_DECODING_>` operation. The Concepts chapter in the CDF User's Guide describes those situations in which a decoding other than `HOST_DECODING` may be desired.

[1] CDF distributions prior to CDF V2.5 did not allow an encoding other than `NETWORK_ENCODING`, `HOST_ENCODING`, or the specific encoding of the host machine.

4.8 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.

4.9 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.)

4.10 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.)

4.11 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.

4.12 zModes

Once a CDF has been opened, it may be placed into one of two variations of zMode. zMode is fully explained in the Concepts chapter in the CDF User's Guide. 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.

4.13 `-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 or written to 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 or written to a CDF.
`NEGtoPOSfp0off`	Do not convert `-0.0` to `0.0` when read from or written to a CDF.

4.14 Operational Limits

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