CDF
Fortran Reference Manual

Version 2.5, January 10, 1995

National Space Science Data Center

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

Contents

Chapter 1

Compiling

Each program, subroutine, or function that calls the CDF library or references CDF parameters must include one or more CDF include files. On VMS systems a logical name, CDF$INC, that specifies the location of the CDF include files 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 named 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.

On VMS and UNIX systems the following line would be included at/near the top of each routine:

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

where <inc-path> is the pathname of the directory containing cdf.inc. On VMS systems CDF$INC: may be used for <inc-path>. On UNIX systems <inc-path> must be a relative or absolute pathname. (An environment variable may not be used.) Another option would be to create a symbolic link to cdf.inc (using ln -s) making cdf.inc appear to be in the same directory as the source file to be compiled. In that case specifying <inc-path> would not be necessary. On UNIX systems you will need to know where on your system cdf.inc has been installed.

On MS-DOS systems using Microsoft Fortran the following lines would be included at/near the top of each routine/source file:

```
INCLUDE 'cdfmsf.inc'
.
. (PROGRAM, SUBROUTINE, or FUNCTION statement)
.
INCLUDE 'cdf.inc'
```

The include file cdfmsf.inc contains an INTERFACE statement for each subroutine/function in the CDF library. Including this file is absolutely essential if you are using the Internal Interface (CDF_lib). cdfmsf.inc is located in the same directory as cdf.inc.

NOTE: There are limitations on where `cdfmsf.inc` can be included. It must generally be included before the `PROGRAM`, `SUBROUTINE`, or `FUNCTION` statement of a routine. If a source file contains multiple routines, `cdfmsf.inc` only needs to be included once (at the very top of the source file). `cdf.inc`, however, may need to be included inside each routine.

On Macintosh systems using Macintosh Programmer's Workshop (MPW) Fortran the following line would be included at/near the top of each routine:

```
INCLUDE 'cdf.inc'
```

The location of `cdf.inc` will be made known to MPW Fortran when the routine is compiled.

1.1 VMS/OpenVMS Systems

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

```
$ FORTRAN <source-name>
```

where `<source-name>` is the name of the source file being compiled. (The `.FOR` extension is not necessary.) 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.

1.2 UNIX Systems

An example of the command to compile a source file on UNIX systems would be as follows:[1]

```
% f77 -c <source-name>.f
```

where `<source-file>.f` is the name of the source file being compiled. (The `.f` extension is required.)

The `-c` option specifies that only an object module is to be produced. (The link step is described in Chapter 2.) The object module created will be named `<source-name>.o`.

1.3 MS-DOS Systems, Microsoft Fortran

NOTE: Even though your application is written in Fortran and compiled with a Fortran compiler, compatible C run-time system libraries will be necessary to successfully link your application. This is because the CDF library is written in C and calls C run-time system functions.

[1] The name of the Fortran compiler may be different depending on the flavor of UNIX being used.

An example of the command to compile a source file on MS-DOS systems using Microsoft Fortran would be as follows:[2]

```
> FL /c /AL /FPi /I<inc-path> <source-name>.for
```

where `<source-name>.for` is the name of the source file being compiled (the `.for` extension is required) and `<inc-path>` is the pathname of the directory containing `cdfmsf.inc` and `cdf.inc`. You will need to know where on your system `cdfmsf.inc` and `cdf.inc` have been installed. `<inc-path>` may be either an absolute or relative pathname.

The `/c` option specifies that only an object module is to be produced. (The link step is described in Chapter 2.) 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 Fortran 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.

NOTE: The CDF library is written in C. The Fortran compiler used to compile your applications must be compatible with the C compiler used to build the CDF library. (Microsoft C and Microsoft Fortran have been shown in the examples in this document.) The linker used must also be configured to allow a Fortran application to call C routines (in the CDF library). Your Fortran applications, however, should not be concerned with calling functions written in C. (They can assume that they are calling Fortran.) The CDF library is written to handle the default Fortran calling conventions.

1.4 Macintosh Systems, MPW Fortran

Macintosh Programmer's Workshop (MPW) Fortran 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 Fortran would be as follows:

```
Fortran -i <inc-path> <source-name>.f
```

[2] This example assumes you have properly set the MS-DOS environment variables used by the Microsoft Fortran compiler and linker. It is also assumed that the environment variables are set such that the linker will be able to find both the Fortran and C run-time system libraries that are needed.

where `<source-name>.f` is the name of the source file being compiled and `<inc-path>` is an absolute or relative pathname of the folder containing `cdf.inc`. You will need to know where on your system `cdf.inc` 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>.f.o` in the current directory. Note that this example also assumes that `<source-name>.f` is in the current directory.

Chapter 2

Linking

Your applications must be linked with the CDF library.[1] Both the Standard and Internal interfaces for Fortran applications are built into the CDF library. On VMS systems a logical name, `CDF$LIB`, that 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`, 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 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 used to link an application to the CDF library (`LIBCDF.OLB`) on a VAX/VMS or VAX/OpenVMS system 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 module 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 used to link an application to the CDF library (`LIBCDF.OLB`) on a DEC Alpha/OpenVMS system 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 Windows applications. Consult the Microsoft documentation for details on using a DLL. Note that the DLL for Microsoft is created using Microsoft 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`. (You must specify a VAX C run-time library because the CDF library is written in C.) 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 used to link an application to the CDF library (`libcdf.a`) on a UNIX system would be as follows:

```
% f77 <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_LIB` is imported, `$(CDF_LIB)` would be specified instead of `${CDF_LIB}`.

2.3.1 Combining the Compile and Link

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

```
% f77 <source-file(s)>.f ${CDF_LIB}/libcdf.a
```

where `<source-file(s)>.f` is the name of the source file(s) being compiled/linked. (The `.f` extension is required.) Some UNIX systems may also require that `-lc`, `-lm`, and/or `-ldl` be specified at the end of the command line. Note that in a "makefile" where `CDF_LIB` is imported, `$(CDF_LIB)` would be specified instead of `${CDF_LIB}`.

2.4 MS-DOS Systems, Microsoft Fortran

NOTE: Even though your application is written in Fortran and compiled with a Fortran compiler, compatible C run-time system libraries (as supplied with Microsoft C) will be necessary to successfully link your application. This is because the CDF library is written in C and calls C run-time system functions.

An example of the command used to link an application to the CDF library (libcdf.lib) on MS-DOS systems using Microsoft Fortran 5.1 and Microsoft C 6.00 would be as follows:[2]

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

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 (.exe will be appended by default); and <lib-path> is the pathname of the directory containing libcdf.lib. You will need to know where on your system libcdf.lib has been installed. <lib-path> may be either an absolute or relative pathname.

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 should be explicitly specified.

The /NOE option specifies that the linker should not search extended dictionaries of library symbols. This is necessary to suppress errors that would be generated because of multiply defined symbols between the Microsoft Fortran and Microsoft C system libraries.

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, and the Fortran run-time library shown, LLIBFORE, assume 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 (immediately after LLIBCE) to handle the function naming differences between the Microsoft C 6.00 and Microsoft C 7.00 run-time libraries. **NOTE:** Specify the libraries in the order shown or errors involving multiply defined symbols may result.

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 Fortran 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 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>.f.o <object-file>.f.o ... <object-file>.f.o δ
```

[2] This example assumes you have properly set the MS-DOS environment variables (e.g., INCLUDE and LIB) used by the Microsoft Fortran (and 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 of the Microsoft C run-time libraries.

```
<lib-path>libcdf.o δ
"{FLibraries}"FORTRANlib.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>.cf.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>.f.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 `"{FLibraries}"`, `"{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 shared 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`. (You must specify a VAX C run-time library [RTL] because the CDF library is written in C.) 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)

```
% f77 -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)

```
% f77 -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)

```
% f77 -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)

```
% f77 -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)

```
% f77 -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 Fortran programming interface for CDF applications. These include constants and types defined for use by all CDF application programs written in Fortran. These constants and types are defined in `cdf.inc`. The file `cdf.inc` should be `INCLUDE`ed in all application source files referencing CDF routines/parameters.

4.1 Argument Passing

The CDF library is written entirely in C. Most computer systems have Fortran and C compilers that allow a Fortran application to call a C function without being concerned that different programming languages are involved. The CDF library takes advantage of the mechanisms provided by these compilers so that your Fortran application can appear to be calling another Fortran subroutine/function (in actuality the CDF library written in C). Pass all arguments exactly as shown in the description of each CDF function. This includes character strings (i.e., `%REF(...)` is not required). Be aware, however, that trailing blanks on variable and attribute names will be considered as part of the name. If the trailing blanks are not desired, pass only the part of the character string containing the name (e.g., `VAR_NAME(1:8)`).

NOTE: Unfortunately, the Microsoft C and Microsoft Fortran compilers on the IBM PC and the C and Fortran compilers on the NeXT computer do not provide the needed mechanism to pass character strings from Fortran to C without explicitly `NUL` terminating the strings. Your Fortran application must place an ASCII `NUL` character after the last character of a CDF, variable, or attribute name. An example of this follows:

```
.
.
CHARACTER ATTR_NAME*9                ! Attribute name
.
.
ATTR_NAME(1:8) = 'VALIDMIN'          ! Actual attribute name
ATTR_NAME(9:9) = CHAR(0)             ! ASCII NUL character
.
.
```

CHAR(0) is an intrinsic Fortran function that returns the ASCII character for the numerical value passed in (0 is the numerical value for an ASCII NUL character). ATTR_NAME could then be passed to one of the CDF library functions.

When the CDF library passes out a character string on an IBM PC (using the Microsoft compilers) or on a NeXT computer, the number of characters written will be exactly as shown in the description of the function called. You must declare your Fortran variable to be exactly that size.

4.2 Item Referencing

For Fortran applications, all items are referenced starting at one (1). These include variable, attribute, and attribute entry numbers, record numbers, dimensions, and dimension indices. Note that both rVariables and zVariables are numbered starting at one (1).

4.3 Status Code Constants

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).

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).

[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.

`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.

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 rVariables 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 Fortran applications the compiler defined majority for arrays is column major. The first dimension of multidimensional arrays varies the fastest 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 will be 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 CDF.

4.14 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 rVariable 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.

4.15 Limits of Names and Other Character Strings

`CDF_PATHNAME_LEN`	Maximum length of a CDF pathname (excluding 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 system being used (including logical names on VMS systems and environment variables on UNIX systems). **UNIX:** Pathnames are case-sensitive.
`CDF_VAR_NAME_LEN`	Maximum length of a variable name.
`CDF_ATTR_NAME_LEN`	Maximum length of an attribute name.
`CDF_COPYRIGHT_LEN`	Maximum length of the CDF copyright text.
`CDF_STATUSTEXT_LEN`	Maximum length of the explanation text for a status code.