The NetCDF Installation and Porting Guide

NetCDF Version 4.0.1
Last Updated 24 March 2009

Ed Hartnett, Russ Rew, John Caron
Unidata Program Center

Table of Contents

1 Installing the NetCDF Binaries

The easiest way to get netCDF is through a package management program, such as rpm, yum, adept, and others. NetCDF is available from many different repositories, including the default Red Hat and Ubuntu repositories.

Another way to get netCDF is to get a pre-built binary distribution. To get them, see `http://www.unidata.ucar.edu/downloads/netcdf/index.jsp`.

To install the binary distribution, uncompress and unpack the tar file. You will end up with 4 subdirectories, lib, include, man, and bin.

The lib subdirectory holds the netCDF libraries (C, Fortran, and C++). The include directory holds the necessary netcdf.h file (for C), netcdf.inc (for Fortran), netcdfcpp.h (for C++), and the .mod files (for Fortran 90). The bin directory holds the ncgen and ncdump utilities, and the man directory holds the netCDF documentation.

You can have these directories anywhere you like, and use netCDF. But when compiling a netCDF program, you will have to tell the linker where to find the library (e.g. with the -L option of most C compilers), and you will also have to tell the C pre-processor where to find the include file (e.g. with the -I option).

If you are using shared libraries, you will also have to specify the library location for run-time dynamic linking. See your compiler documentation. For some general information see the netCDF FAQ "How do I use shared libraries" at `http://www.unidata.ucar.edu/netcdf/faq.html#using_shared`.

2 Quick Instructions for Installing NetCDF on Unix

Who has time to read long installation manuals these days?

When building netCDF-4, you must first decide whether to support the use of HDF5 as a storage format. Use of netCDF-4 and HDF5 is turned off by default. Most users should build without HDF5 support, until significant netCDF-4/HDF5 datasets are available.

2.1 Building NetCDF Without HDF5

If you don't want netCDF-4/HDF5, then build like this:

```
./configure --prefix=/home/ed/local
make check install
```

(Replace "/home/ed/local" with the name of the directory where netCDF is to be installed.)

If you get the message that netCDF installed correctly, then you are done!

2.2 Building NetCDF With HDF5

If you want to use the HDF5 storage format, you must have the HDF5 1.8.2 release. You must also have the zlib compression library, version 1.2.3 (or better). Both of these packages are available from the netCDF-4 ftp site at `ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4`.

Make sure you run "make check" for the HDF5 and zlib distributions. They are very well-behaved distributions, but sometimes the build doesn't work (perhaps because of something subtly misconfigured on the target machine). If one of these libraries is not working, netCDF will have serious problems.

Optionally, you can also use the szip 2.0 library (a.k.a. szlib). There are license restrictions on the use of szip, see the HDF5 web page: http://hdf.ncsa.uiuc.edu/doc˙resource/SZIP/Commercial˙szip.html.

These license restrictions seem to apply to commercial users who are writing data. (Data readers are not restricted.) But here at NetCDF World Headquarters, in Sunny Boulder, Colorado, there are no lawyers, only programmers, so please read the szip documents for the license agreement to see how it applies to your situation.

If you wish to use szip, get it from the HDF5 1.8.2 download page: http://hdfgroup.org/HDF5//HDF5/release/beta/obtain518.html.

If "make check" fails for either zlib or HDF5, the problem must be resolved before the netCDF-4 installation can continue. For HDF5 problems, send email to the HDF5 help desk: help@hdfgroup.org.

Build zlib like this:

```
./configure --prefix=/home/ed/local
make check install
```

(If you want a shared library build of HDF5, you must set CFLAGS to -fPIC before building zlib.)

Then you build HDF5, specifying the location of the zlib library:

```
./configure --with-zlib=/home/ed/local --prefix=/home/ed/local --disable-shared
make check install
```

This builds a static version of the HDF5 library. For a shared build, don't use –disable-shared. For a shared build to work you must have compiled zlib (and, optionally, szip) with CFLAGS=-fPIC.

If you are building HDF5 with szip, then include the –with-szip= option, with the directory holding the szip library.

After HDF5 is done, build netcdf-4, specifying the location of the HDF5, zlib, and (if built into HDF5) the szip libraries with the –with-hdf5, –with-zlib, and –with-szlib option. (If they are all in the same directory, you only need to use one of these options.) You must also provide the –enable-netcdf-4 option to configure.

```
./configure --enable-netcdf-4 --with-hdf5=/home/ed/local --with-zlib=/home/ed/local --prefix=/home/ed/local
make check install
```

The configure script will try to find necessary tools in your path. When you run configure you may optionally use the –prefix argument to change the default installation directory. For example, the above examples install the zlib, HDF5, and netCDF-4 libraries in /home/ed/local/lib, the header file in /home/ed/local/include, and the utilities in /home/ed/local/bin.

The default install root is /usr/local (so there's no need to use the prefix argument if you want the software installed there).

Without the –enable-netcdf-4 argument, the netCDF-4 HDF5 features will not be available. Without the –with-hdf5 and –with-zlib options, netCDF-4 will not know where to find HDF5.

By default the netCDF configuration will build static libraries only. For shared libraries as well, use the –enable-shared option to configure.

To use netCDF-4 you must link to all the libraries, netCDF, HDF5, zlib, and (if used with HDF5 build) szip. This will mean -L options to your build for the locations of the libraries, and -l (lower-case L) for the names of the libraries.

For example, one user reports that she can build other applications with netCDF-4 by setting the LIBS envoronment variable:

```
LIBS='-L/X/netcdf-4.0/lib -lnetcdf -L/X/hdf5-1.8.2/lib -lhdf5_hl -lhdf5 -lz -lm -L/X/szip-2.1/lib -lsz'
```

2.3 Building ncgen4

An experimental version of ncgen4 is now included in the installation. The ncgen4 program operates essentially the same as ncgen, but is capable of constructing netcdf-4 files using the enhanced data model that includes user defined types and groups.

3 Building and Installing NetCDF on Unix Systems

The latest version of this document is available at `http://www.unidata.ucar.edu/netcdf/docs/netcdf-install`.

This document contains instructions for building and installing the netCDF package from source on various platforms. Prebuilt binary releases are (or soon will be) available for various platforms from `http://www.unidata.ucar.edu/downloads/netcdf/index.jsp`.

A good general tutorial on how software is built from source on Linux platforms can me found at `http://www.tuxfiles.org/linuxhelp/softinstall.html`.

3.1 Installation Requirements

If you wish to build from source on a Windows (Win32) platform, different instructions apply. See Chapter 4 [Building on Windows], page 19.

Depending on the platform, you may need up to 25 Mbytes of free space to unpack, build, and run the tests. You will also need a Standard C compiler. If you have compilers for FORTRAN 77, FORTRAN 90, or C++, the corresponding netCDF language interfaces may also be built and tested. Compilers and associated tools will only be found if they are in your path, or if you specify the path and compiler in the appropriate environment variable. (Example for csh: setenv CC /some/directory/cc).

If you want to run the large file tests, you will need about 13 GB of free disk space, as some very large files are created. The created files are immediately deleted after the tests complete. These large file tests are not run unless the –enable-large-file-tests option is used with configure. (The –with-temp-large option may also be used to specify a directory to create the large files in).

Unlike the output from other netCDF test programs, each large test program deletes its output before successfully exiting.

To use the netCDF-4 features you will also need to have a HDF5-1.8.2 release installed. HDF5, in turn, must have been built with zlib, version 1.2.3 (or better).

A tested version of HDF5 and zlib can be found at the netCDF-4 ftp site at `ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4`.

For more information about HDF5 see the HDF5 web site at `http://hdfgroup.org/HDF5/`. For more information about zlib see the zlib web site at `http://www.zlib.net`.

To use the DAP features you will also need to have a version of libcurl (version 7.18.0 or later) installed. Depending on how this library were built, you may also need zib (version 1.2.3 or later). Information about libcurl may be obtained at `http://curl.haxx.se`.

3.2 Specifying the Environment for Building

The netCDF configure script searches your path to find the compilers and tools it needed. To use compilers that can't be found in your path, set their environment variables.

When finding compilers, vendor compilers will be preferred to GNU compilers. Not because we don't like GNU, but because we assume if you purchased a compiler, you want to use it. Setting CC allows you to over-ride this preference. (Alternatively, you could temporarily remove the compiler's directories from your PATH.)

For example, on an AIX system, configure will first search for xlc, the AIX compiler. If not found, it will try gcc, the GNU compiler. To override this behavior, set CC to gcc (in sh: export CC=gcc). (But don't forget to also set CXX to g++, or else configure will try and use xlC, the AIX C++ compiler to build the netCDF C++ API.)

By default, the netCDF library is built with assertions turned on. If you wish to turn off assertions, set CPPFLAGS to -DNDEBUG (csh ex: setenv CPPFLAGS -DNDEBUG).

If GNU compilers are used, the configure script sets CPPFLAGS to "-g -O2". If this is not desired, set CPPFLAGS to nothing, or to whatever other value you wish to use, before running configure.

For cross-compiles, the following environment variables can be used to override the default fortran/C type settings like this (in sh):

```
export NCBYTE_T=''integer(selected_int_kind(2))''
export NCSHORT_T=''integer*2''
export NF_INT1_T=''integer(selected_int_kind(2))''
export NF_INT2_T=''integer*2''
export NF_INT1_IS_C_SHORT=1
export NF_INT2_IS_C_SHORT=1
export NF_INT_IS_C_INT=1
export NF_REAL_IS_C_FLOAT=1
export NF_DOUBLEPRECISION_IS_C_DOUBLE=1
```

In this case you will need to run configure with –disable-fortran-compiler-check and –disable-fortran-type-check.

3.2.1 Variable Description Notes

CC	C compiler	If you don't specify this, the configure script will try to find a suitable C compiler such as cc, c89, xlc, or gcc.
FC	Fortran compiler (if any)	If you don't specify this, the configure script will try to find a suitable Fortran and Fortran 77 compiler. Set FC to "" explicitly, or provide the –disable-f77 option to configure, if no Fortran interface is desired. Use –disable-f90 to disable the netCDF Fortran 90 API, but build the netCDF Fortran 77 API.
F77	Fortran 77 compiler (if any)	Only specify this if your platform explicitly needs a different Fortran 77 compiler. Otherwise use FC to specify the Fortran compiler. If you don't specify this, the configure script will try to find a suitable Fortran compiler. For vendor compilers, make sure you're using the same vendor's Fortran 90 compiler. Using Fortran compilers from different vendors, or mixing vendor compilers with g77, the GNU F77 compiler, is not supported and may not work.

CXX	C++ compiler	If you don't specify this, the configure script will try to find a suitable C++ compiler. Set CXX to "" explicitly, or use the –disable-cxx configure option, if no C++ interface is desired. If using a vendor C++ compiler, use that vendor's C compiler to compile the C interface. Using different vendor compilers for C and C++ may not work.
CFLAGS	C compiler flags	"-O" or "-g", for example. If you don't set this, configure may set it based on your platform's needs (unless you have used the –disable-flag-setting option is used with configure).
CPPFLAGS	C preprocessor options	"-DNDEBUG" to omit assertion checks, for example. If you don't set this, configure may set it based on your platform's needs (unless you have used the –disable-flag-setting option is used with configure).
FCFLAGS	Fortran 90 compiler flags	"-O" or "-g", for example. These flags will be used for FORTRAN 90. If setting these you may also need to set FFLAGS for the FORTRAN 77 test programs.
FFLAGS	Fortran 77 compiler flags	"-O" or "-g", for example. If you need to pass the same arguments to the FORTRAN 90 build, also set FCFLAGS.
CXXFLAGS	C++ compiler flags	"-O" or "-g", for example. If you don't set this, configure may set it based on your platform's needs (unless you have used the –disable-flag-setting option is used with configure).
ARFLAGS, NMFLAGS, FPP, M4FLAGS, LIBS, FLIBS, FLDFLAGS	Miscellaneous	One or more of these were needed for some platforms, as specified below. Unless specified, you should not set these environment variables, because that may interfere with the configure script.
HDF5DIR	configure option	Set this to the directory you wish to specify in the –with-hdf5 argument of configure.

The section marked Tested Systems below contains a list of systems on which we have built this package, the environment variable settings we used, and additional commentary.

3.3 Building on 64 Bit Platforms

The compiler options for SunOS, Irix, and AIX are listed below. The zlib and HDF5 libraries must also be built with 64-bit options.

`AIX` Set -q64 option in all compilers, and set NMFLAGS to -X64, and ARFLAGS to '-X64 cru'. Alternatively, set environment variable OBJECT_MODE to 64 before running configure.

`IRIX` Set the -64 option in all compilers.

`SunOS` Use the -xarch=v9 or -m64 flag on all compilers for Sparc, or -m64 on x86 platforms.

3.4 Building on Platforms with Parallel I/O

NetCDF makes available the parallel I/O features of HDF5, allowing parallel I/O from netCDF-4 linked programs.

For parallel I/O to work, HDF5 must be installed with –enable-parallel, and an MPI library (and related libraries) must be made available to the HDF5 configure. This can be accomplished with the mpicc wrapper script, in the case of MPICH2.

The following works to build HDF5 with parallel I/O on our netCDF testing system:

```
CC=mpicc ./configure --enable-parallel --prefix=/shecky/local_par --with-zlib=/shecky/local_par --disable-shared && make check install
```

There is no need to specify –enable-parallel for netCDF. The netCDF configure program will recognize that HDF5 was built for parallel I/O and will turn on netCDF-4's parallel I/O features.

```
CC=mpicc ./configure --enable-netcdf-4 --prefix=/shecky/local_par --with-hdf5=/shecky/local_par && make check install
```

To enable the parallel tests, specify –enable-parallel-tests as an option to configure. These tests will be run as mpiexec calls. This may not be appropriate on all systems, especially those which use some queue for jobs.

For parallel builds the netCDF examples are not built. This is to avoid cluttering them with MPI_Init/Finalize calls.

3.5 Running the configure Script

To create the Makefiles needed to build netCDF, you must run the provided configure script. Go to the top-level netCDF directory.

Decide where you want to install this package. Use this for the "–prefix=" argument to the configure script below. The default installation prefix is "/usr/local," which will install the package's files in usr/local/bin, usr/local/lib, and usr/local/man. The default can be overridden with the –prefix argument to configure.

(Note that this is a new default location for version 3.6.2 of netCDF. Previous versions used the directory in which netCDF was built as the default install directory).

Here's how to execute the configure script with a different installation directory:

```
./configure --prefix=/whatever/you/decided --enable-netcdf-4 --with-hdf5=/home/ed/local
```

The above would cause the netCDF libraries to be installed in /whatever/you/decided/lib, the header files in /whatever/you/decided/include, the utilities (ncdump/ncgen) in /whatever/you/decided/bin, and the man pages in /whatever/you/decided/man.

The –enable-netcdf-4 option tells configure that you wish to build the netCDF-4/HDF5 features. The –with-hdf4= argument tells configure where HDF5 is installed. It must be version 1.8.0-beta2, and it must be built with zlib, version 1.2.3.

There are other options for the configure script. The most useful ones are listed below. Use the –help option to get the full list.

`--prefix` Specify the directory under which netCDF will be installed. Subdirectories lib, bin, include, and man will be created there, if they don't already exist.

`--enable-netcdf-4`
Turn on netCDF-4 features.

`--with-hdf5=/location`
Specify the location of the HDF5 library.

`--with-zlib=/location`
Specify the location of the zlib library. NetCDF-4 requires that HDF5 be built with zlib, for variable compression.

`--with-szlib=/location`
Specify the location of the szlib (a.k.a. szip) library. This is optional, and netCDF-4 does not make use of the szlib, due to licence issues. However, if HDF5 is build with szlib, then you must provide the location of szlib when building netCDF-4.

`--enable-shared`
Build shared libraries (as well as static) on platforms which support them.

`--enable-docs-install`
By default, the netCDF distribution will install man pages, but not any other documentation.

The latest documentation is available on-line at the the netCDF website. `http://www.unidata.ucar.edu/netcdf`. However, for users who cannot always reach the web, all netCDF documentation is included in the distribution in the "man" subdirectory, in PDF, HTML, postscript, info, ASCII text, and other formats.

Using the –enable-docs-install causes "make install" to install the netCDF documentation under $(prefix)/doc/netcdf-4.0.1.

–enable-docs-install does not cause the documentation to be built from source; it causes the documentation which was shipped with the distribution to be installed.

Users who wish to contribute to the documentation development are urged to make any suggested changes to the documentation source files, which have the .texi filename extension (netcdf.text, netcdf-c.texi, etc.). Building the netCDF documentation from source requires recent versions of the open-source tools sed, m4, texinfo, and tex.

`--disable-largefile`
This omits OS support for large files (i.e. files larger than 2 GB).

`--disable-f77`
This turns off building of the F77 and F90 APIs. (The F90 API cannot be built without the F77 API). This also disables some of the configure tests that relate to fortran, including the test of the F90 compiler. Setting the environment variables FC or F77 to NULL will have the same effect as –disable-f77.

`--disable-f90`
: This turns off the building of the F90 API. Setting the environment variable F90 to null for configure will have the same effect.

`--disable-cxx`
: This turns off the building of the C++ API. Setting the environment variable CXX to null for configure will have the same effect.

`--disable-v2`
: This turns of the V2 API. The V2 API is completely replaced with the V3 API, but is usually built with netCDF for backwards compatibility, and also because the C++ API depends on the V2 API. Setting this has the effect of automatically turning off the CXX API, as if –disable-cxx had also been specified.

`--enable-cxx4`
: Turns on the new C++ API, which is currently under development, and will expose the full expanded model in the C++ API. The cxx4 API is experiemental, unfinished, and untested. It is provided for experiemental purposes only.

`--enable-large-file-tests`
: Turn on tests for large files. These tests create files over 2 GB in size, and need about 13 GB of free disk space to run. These files are deleted after the test successfully completes. They will be created in the netCDF nc_test directory, unless the –with-temp-large option is used to specify another location (see below).

`--with-temp-large`
: Normally large files are not created during the netCDF build, but they will be if –enable-large-file-tests is specified (see above). In that case, this configure parameter can be used to specify a location to create these large files, for example: –with-large-files=/tmp/ed.

`--enable-benchmarks`
: Turn on tests of the speed of various netCDF operations. Some of these operations take a long time to run (minutes, on a reasonable workstation).

`--enable-valgrind-tests`
: Causes some tests to be re-run under valgrind, the memory testing tool. Valgrind must be present for this to work.

`--disable-fortran-compiler-check`
: Normally the netCDF configure checks the F77 and F90 compilers with small test programs to see if they work. This is very helpful in supporting netCDF installations on different machines, but won't work with cross-compilation. Use the –disable-fortran-compiler-check to turn off the fortran compiler tests, and just assume that the compilers will work.

`--disable-fortran-type-check`
: The netCDF configure compiles and runs some programs to test fortran vs. C type sizes. Setting this option turns off those test, and uses a set of default values (which can be overridden by environment variables see Section 3.2 [Environment], page 5).

`--disable-compiler-recover`
: Normally, if the netCDF configure finds a F90 compiler, and it fails to build the test program described in –disable-f90-check, it will print a warning, and then continue to build without the F90 API, as if the user has specified –disable-f90. With the –disable-compiler-recover option set, it will not continue, but will just stop if the fortran 90 compiler doesn't work. This is useful for automatic testing, where we want the tests to fail if something causes the fortran compiler to break.

`--disable-examples`
: Starting with version 3.6.2, netCDF comes with some examples in the "examples" directory. By default, the examples are all built during a "make check" unless the –disable-examples option is provided.

`--enable-separate-fortran`
: This will cause the Fortran 77 and Fortran 90 APIs to be built into their own separate library, instead of being included in the C library. This is useful for supporting more than one fortran compiler with the same netCDF C library. This is turned on by default for shared library builds.

`--enable-extra-tests`
: During the beta release phase, this option may turn on tests which are known to fail (i.e. bugs that we are currently working to fix).

`--enable-dap`
: Enable DAP support. The configure script will attempt to locate a usable curl library, but if it fails, then it is necessary to specify their location using –with-curl=/some/path. It is assumed that the directories /some/path/include and /some/path/lib exist. This feature is currently provided for experimental purposes until more detailed testing is accomplished. Note that when DAP is enabled, this can be tested for in a configure script by looking for the function "nc__opendap".

`--enable-dap-remote-tests`
: If DAP support is enabled, it is possible to also enable extra dap tests that utilize an external server at opendap.org. Since that server may be inaccessible for a variety of reasons, it is only used if this flag is enabled.

`--enable-ncgen4`
: Turns on the build of a version of ncgen, called ncgen4, that can process CDL files using the netcdf-4 data model. It is currently provided for experimental purposes only. All comments in this document about ncgen will also apply to ncgen4. If compiled with the –enable-netcdf-4 option, its full range of functionality is available. If compiled without the –enable-netcdf-4 option, its functionality is limited to more-or-less that of the existing ncgen utility, except that FORTRAN output is not supported.

The configure script will examine your computer system – checking for attributes that are relevant to building the netCDF package. It will print to standard output the checks that it makes and the results that it finds.

The configure script will also create the file "config.log", which will contain error messages from the utilities that the configure script uses in examining the attributes of your system. Because such an examination can result in errors, it is expected that "config.log" will contain error messages. Therefore, such messages do not necessarily indicate a problem (a better indicator would be failure of the subsequent "make"). One exception, however, is an error message in "config.log" that indicates that a compiler could not be started. This indicates a severe problem in your compilation environment – one that you must fix.

3.6 Running make

Run "make". This will build one or more netCDF libraries. It will build the basic netCDF library libnetcdf.a. If you have Fortran 77 or Fortran 90 compilers, then the Fortran library will also be built (libnetcdff.a). If you have a C++ compiler, then the C++ interface will be built (libnetcdf_c++.a.)

A “make” will also build the netCDF utilities ncgen(1) and ncdump(1).

Run make like this:

```
make
```

3.7 Testing the Build

Run “make check” to verify that the netCDF library and executables have been built properly (you can instead run “make test” which does the same thing).

A make check will build and run various test programs that test the C, Fortran, and C++ interfaces as well as the "ncdump" and "ncgen" utility programs.

Lines in the output beginning with "***" report on success or failure of the tests; any failures will be reported before halting the test. Compiler and linker warnings during the testing may be ignored.

Run the tests like this:

```
make check
```

If you plan to use the 64-bit offset format (introduced in version 3.6.0) to create very large files (i.e. larger than 2 GB), you should probably specify the –enable-large-file-tests to configure, which tests the large file features. You must have 13 GB of free disk space for these tests to successfully run.

If you are running the large file tests, you may wish to use the –with-temp-large option to specify a temporary directory for the large files. (You may also set the environment variable TEMP_LARGE before running configure).

The default is to create the large files in the nc_test subdirectory of the netCDF build.

Run the large file tests like this:

```
./configure --enable-large-file-tests --with-temp-large=/home/ed/tmp
make check
```

All of the large files are removed on successful completion of tests. If the test fails, you may wish to make sure that no large files have been left around.

If any of the the large file tests test fail, check to ensure that your file system can handle files larger than 2 GiB by running the following command:

```
dd if=/dev/zero bs=1000000 count=3000 of=$(TEMP_LARGE)/largefile
```

If your system does not have a /dev/zero, this test will fail. Then you need to find some other way to create a file larger than 2 GiB to ensure that your system can handle them.

See Chapter 5 [Build Problems], page 25.

3.8 Installing NetCDF

To install the libraries and executables, run "make install". This will install to the directory specified in the configure step.

Run the installation like this:

```
make install
```

The install will put files in the following subdirectories of the directory you provided as a prefix, creating the subdirectories if needed:

`lib` Libraries will be installed here. If static libraries are built, without separate fortran libraries, then libnetcdf.a and libnetcdf.la will be installed. If the C++ API is built, libnetcdf_c++.a and libnetcdf_c++.la will be added. If separate fortran libraries are built, libnetcdff.a and libnetcdff.la will also be added.

Static library users should ignore the .la files, and link to the .a files.

Shared library builds will add some .so files to this directory, as well.

`include` Header files will be installed here. The C library header file is netcdf.h. If the C++ library is built, ntcdfcpp.h, ncvalues.h and netcdf.hh will be installed here. If the F77 API is built, netcdf.inc will be copied here. If the F90 API is built, the netcdf.mod and typesizes.mod files will be copied here as well.

`bin` Utilities ncdump and ncgen will be installed here. If enabled, ncgen4 will also be installed.

`man` The ncdump/ncgen man pages will be installed in subdirectory man1, and the three man pages netcdf.3, netcdf_f77.3, and netcdf_f90.3 will be installed in the man3 subdirectory.

`share` If the configure is called with the –enable-docs option, the netCDF documentation set will be built, and will be installed under the share directory, under the netcdf subdirectory. This will include postscript, PDF, info and text versions of all netCDF manuals. These manuals are also available at the netCDF web site.

Try linking your applications. Let us know if you have problems (see Section 5.4 [Reporting Problems], page 28).

3.9 Using NetCDF

To use netCDF you must link to the netCDF library, and, if using the netCDF-4/HDF5 features, also two HDF5, at least one compression library, and (on some systems) the math library.

Use the -L option to your linker to pass the directories in which netCDF, HDF5, and zlib are installed.

Use the -l (lower-case L) option to list the libraries, which must be listed in the correct order:

```
-lnetcdf -lhdf5_hl -lhdf5 -lz
```

On some systems you must also include -lm for the math library.

If szip was used when building HDF5, you must also use -lsz.

3.10 Platform Specific Notes

The following platform-specific note may be helpful when building and installing netCDF. Consult your vendor manuals for information about the options listed here. Compilers can change from version to version; the following information may not apply to your platform.

Full output from some of the platforms of the test platforms for netCDF 4.0.1 can be found at `http://www.unidata.ucar.edu/netcdf/builds`.

3.10.1 AIX

We found the vendor compilers in /usr/vac/bin, and included this in our PATH. Compilers were xlc, xlf, xlf90, xlC.

The F90 compiler requires the qsuffix option to believe that F90 code files can end with .f90. This is automatically turned on by configure when needed:

```
FCFLAGS=-qsuffix=f=f90
```

We had to use xlf for F77 code, and xlf90 for F90 code.

To compile 64-bit code, set the appropriate environment variables (documented below).

The environment variable OBJECT_MODE can be set to 64, or use the -q64 option on all AIX compilers by setting CFLAGS, FFLAGS, and CXXFLAGS to -q64.

The following is also necessary on an IBM AIX SP system for 64-bit mode:

```
ARFLAGS='-X64 cru'
NMFLAGS='-X64'
```

There are thread-safe versions of the AIX compilers. For example, xlc_r is the thread-safe C compiler. The NetCDF configure script ignores these compilers. To use thread-safe compilers, override the configure script by setting CC to xlc_r; similarly for FC and CXX.

For large file support, AIX requires that the macro _LARGE_FILES be defined. The configure script does this using AC_SYS_LARGEFILES. Unfortunately, this misfires when OBJECT_MODE is 64, or the q64 option is used. The netCDF tries to fix this by turning on _LARGE_FILES anyway in these cases.

The GNU C compiler does not mix successfully with the AIX fortran compilers.

3.10.2 Cygwin

NetCDF builds under Cygwin tools on Windows just as with Linux.

3.10.3 HPUX

The HP Fortran compiler (f77, a.k.a. fort77, also f90) requires FLIBS to include -lU77 for the fortran tests to work. The configure script does this automatically.

For the c89 compiler to work, CPPFLAGS must include -D_HPUX_SOURCE. This isn't required for the cc compiler. The configure script adds this as necessary.

For large file support, HP-UX requires _FILE_OFFSET_BITS=64. The configure script sets this automatically.

The HPUX C++ compiler doesn't work on netCDF code. It's too old for that. So either use GNU to compile netCDF, or skip the C++ code by setting CXX to " (in csh: setenv CXX ").

Building a 64 bit version may be possible with the following settings:

```
CC=/bin/cc
CPPFLAGS='-D_HPUX_SOURCE -D_FILE_OFFSET_BITS=64'    # large file support
CFLAGS='-g +DD64'                                    # 64-bit mode
FC=/opt/fortran90/bin/f90                            # Fortran-90 compiler
FFLAGS='-w +noppu +DA2.0W'                           # 64-bit mode, no "_" suffixes
FLIBS=-lU77
CXX=''                                               # no 64-bit mode C++ compiler
```

Sometimes quotas or configuration causes HPUX disks to be limited to 2 GiB files. In this cases, netCDF cannot create very large files. Rather confusingly, HPUX returns a system error that indicates that a value is too large to be stored in a type. This may cause scientists to earnestly check for attempts to write floats or doubles that are too large. In fact, the problem seems to be an internal integer problem, when the netCDF library attempts to read beyond the 2 GiB boundary. To add to the confusion, the boundary for netCDF is slightly less than 2 GiB, since netCDF uses buffered I/O to improve performance.

3.10.4 Irix

A 64-bit version can be built by setting the appropriate environment variables.

Build 64-bit by setting CFLAGS, FFLAGS, and CXXFLAGS to -64.

On our machine, there is a /bin/cc and a /usr/bin/cc, and the -64 option only works with the former.

3.10.5 Linux

The f2cFortran flag is required with GNU fortran:

```
CPPFLAGS=-Df2cFortran
```

For Portland Group Fortran, set pgiFortran instead:

```
CPPFLAGS=-DpgiFortran
```

Portland Group F90/F95 does not mix with GNU g77.

The netCDF configure script should notice which fortran compiler is being used, and set these automatically.

For large file support, _FILE_OFFSET_BITS must be set to 64. The netCDF configure script should set this automatically.

3.10.6 Macintosh

The f2cFortran flag is required with GNU fortran (CPPFLAGS=-Df2cFortran). The NetCDF configure script should and set this automatically.

For IBM compilers on the Mac, the following may work (we lack this test environment):

```
CC=/usr/bin/cc
CPPFLAGS=-DIBMR2Fortran
F77=xlf
FC=xlf90
FCFLAGS=-qsuffix=cpp=f90
```

3.10.7 OSF1

NetCDF builds out of the box on OSF1.

3.10.8 SunOS

PATH should contain /usr/ccs/bin to find make, nm, ar, etc.

For large file support, _FILE_OFFSET_BITS must be 64, also _LARGEFILE64_SOURCE and _LARGEFILE_SOURCE must be set. Configure will turn this on automatically for netCDF, but not for HDF5-1.8.2. So when building HDF5-1.8.2, set these variables before running configure, or HDF5 will not be capable of producing large files.

To compile in 64-bit mode, set -m64 (formerly -xarch=v9, which works on SPARC platforms only) on all compilers (i.e. in CFLAGS, FFLAGS, FCFLAGS, and CXXFLAGS).

When compiling with GNU Fortran (g77), the -Df2cFortran flag is required for the Fortran interface to work. The NetCDF configure script turns this on automatically if needed.

3.10.9 Handling Fortran Compilers

Commercial fortran compilers will generally require at least one flag in the CPPFLAGS variable. The netCDF configure script tries to set this for you, but won't try if you have used –disable-flag-setting, or if you have already set CPPFLAGS, CFLAGS, CXXFLAGS, FCFLAGS, or FFLAGS yourself.

The first thing to try is to set nothing and see if the netCDF configure script finds your fortran compiler, and sets the correct flags automatically.

If it doesn't find the correct fortran compiler, you can next try setting the FC environment variable to the compiler you wish to use, and then see if the configure script can set the correct flags for that compiler.

If all that fails, you must set the flags yourself.

The Intel compiler likes the pgiFortran flag, as does the Portland Group compiler. (Automatically turned on if your fortran compiler is named "ifort" or "pgf90").

Alternatively, Intel has provided a web page on "Building netCDF with the Intel compilers" at `http://www.intel.com/support/performancetools/sb/CS-027812.htm`, providing instructions for building netCDF (without using the pgiFortran flag).

The Portland Group also has a "PGI Guide to NetCDF" at `http://www.pgroup.com/resources/netcdf/netcdf362_pgi71.htm`.

3.11 Additional Porting Notes

The configure and build system should work on any system which has a modern "sh" shell, "make", and so on. The configure and build system is less portable than the "C" code itself, however. You may run into problems with the "include" syntax in the Makefiles.

You can use GNU make to overcome this, or simply manually include the specified files after running configure.

Instruction for building netCDF on other platforms can be found at `http://www.unidata.ucar.edu/netcdf/other-builds.html`. If you build netCDF on a new platform, please send your environment variables and any other important notes to support-netcdf@unidata.ucar.edu and we will add the information to the other builds page, with a credit to you.

If you can't run the configure script, you will need to create config.h and fortran/nfconfig.inc. Start with ncconfig.in and fortran/nfconfig.in and set the defines as appropriate for your system.

Operating system dependency is isolated in the "ncio" module. We provide two versions. posixio.c uses POSIX system calls like "open()", "read()" and "write(). ffio.c uses a special library available on CRAY systems. You could create other versions for different operating systems. The program "t_ncio.c" can be used as a simple test of this layer.

Note that we have not had a Cray to test on for some time. In particular, large file support is not tested with ffio.c.

Numerical representation dependency is isolated in the "ncx" module. As supplied, ncx.m4 (ncx.c) supports IEEE floating point representation, VAX floating point, and CRAY floating point. BIG_ENDIAN vs LITTLE_ENDIAN is handled, as well as various sizes of "int", "short", and "long". We assume, however, that a "char" is eight bits.

There is a separate implementation of the ncx interface available as ncx_cray.c which contains optimizations for CRAY vector architectures. Move the generic ncx.c out of the way and rename ncx_cray.c to ncx.c to use this module. By default, this module does not use the IEG2CRAY and CRAY2IEG library calls. When compiled with aggressive in-lining and optimization, it provides equivalent functionality with comparable speed and clearer error semantics. If you wish to use the IEG library functions, compile this module with -DUSE_IEG.

3.12 Contributing to NetCDF Source Code Development

Most users will not be interested in working directly with the netCDF source code. Rather, they will write programs which call netCDF functions, and delve no further. However some intrepid users may wish to dig into the netCDF code to study it, to try and spot bugs, or to make modifications for their own purposes.

To work with the netCDF source code, several extra utilities are required to fully build everything from source. If you are going to modify the netCDF source code, you will need some or all of the following Unix tools.

`m4` Macro processing language used heavily in libsrc, nc_test. Generates (in these cases) C code from m4 source. Version 1.4 works fine with release 3.5.1 through 3.6.2.

`flex` and `yacc`
: Used in ncgen directory to parse CDL files. Generates C files from .y and .l files. You only need to use this to modify ncgen's understanding of CDL grammar.

`makeinfo` Generates all documentation formats (except man pages) from texinfo source. I'm using makeinfo version 4.8, as of release 3.6.2. If you have trouble with makeinfo, upgrade to this version and try again. You only need makeinfo if you want to modify the documentation.

`tex` Knuth's venerable typesetting system. The version I am running (for netCDF release 3.6.2) is TeX 3.141592 (Web2C 7.5.4). I have found that some recent installations of TeX will not build the netCDF documentation - it's not clear to me why.

The user generally will just want to download the latest version of netCDF documents at the netCDF website. `http://www.unidata.ucar.edu/netcdf/docs`.

`autoconf` Generates the configure script. Version 2.59 or later is required.

`automake` Since version 3.6.2 of netCDF, automake is used to generate the Makefile.in files needed by the configure script to build the Makefiles.

`libtool` Since version 3.6.2 of netCDF, libtool is used to help generate shared libraries platforms which support them. Version 2.1a of libtool is required.

`sed` This text processing tool is used to process some of the netCDF examples before they are included in the tutorial. This is only needed to build the documentation, which the user generally will not need to do.

NetCDF has a large and enterprising user community, and a long history of accepting user modifications into the netCDF code base. Examples include the 64-bit offset format, and the F90 API.

All suggested changes and additions to netCDF code can be sent to support-netcdf@unidata.ucar.edu.

4 Building and Installing NetCDF on Windows

NetCDF can be built and used from a variety of development environments on Windows. The netCDF library is implemented as a Windows dynamic link library (DLL). The simplest way to get started with netCDF under Windows is to download the pre-built DLL from the Unidata web site.

Building under the Cygwin port of GNU tools is treated as a Unix install. See Section 3.10 [Platform Notes], page 14.

Instructions are also given for building the netCDF DLL from the source code.

VC++ documentation being so voluminous, finding the right information can be a chore. There's a good discussion of using DLLs called "About Dynamic-Link Libraries" at (perhaps) `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/dynamic_link_libraries.asp`.

From the .NET point of view, the netCDF dll is unmanaged code. As a starting point, see the help topic "Consuming Unmanaged DLL Functions" which may be found at `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconConsumingUnmanagedDLLFunctions.asp`, unless the page has been moved.

4.1 Getting Prebuilt netcdf.dll

We have pre-built Win32 binary versions of the netcdf dll and static library, as well as ncgen.exe and ncdump.exe (dll and static versions). You can get them from `ftp://ftp.unidata.ucar.edu/pub/netcdf/contrib/win32/netcdf-3.6.1-beta1-win32dll.zip`. (Note: we don't have a C++ interface here).

4.2 Installing the DLL

Whether you get the pre-built DLL or build your own, you'll then have to install it somewhere so that your other programs can find it and use it.

To install a DLL, you just have to leave it in some directory, and (possibly) tell your compiler in which directory to look for it.

A DLL is a library, and functions just like libraries under the Unix operating system. As with any library, the point of the netCDF DLL is to provide functions that you can call from your own code. When you compile that code, the linker needs to be able to find the library, and then it pulls out the functions that it needs. In the Unix world, the -L option tells the compiler where to look for a library. In Windows, library search directories can be added to the project's property dialog.

Similarly, you will need to put the header file, netcdf.h, somewhere that you compiler can find it. In the Unix world, the -I option tells the compiler to look in a certain directory to find header files. In the Windows world, you set this in the project properties dialog box of your integrated development environment.

Therefore, installing the library means nothing more than copying the DLL somewhere that your compiler can find it, and telling the compiler where to look for them.

The standard place to put DLLs is Windows\System32 folder (for Windows2000/XP) or the Windows\System folder (for Windows 98/ME). If you put the DLL there, along with the ncgen and ncdump executables, you will be able to use the DLL and utilities without further work, because compilers already look there for DLLs and EXEs.

Instead of putting the DLL and EXEs into the system directory, you can leave them wherever you want, and every development project that uses the dll will have to be told to search the netCDF directory when it's linking, or, the chosen directory can be added to your path.

On the .NET platform, you can also try to use the global assembly cache. (To learn how, see MSDN topic "Global Assembly Cache", at `www.msdn.microsoft.com`).

Following Windows conventions, the netCDF files belong in the following places:

File(s)	Description	Location
netcdf.dll	C and Fortran function in DLL	Windows\System (98/ME) or Windows\System32 (2000/XP)
netcdf.lib	Library file	Windows\System (98/ME) or Windows\System32 (2000/XP)
ncgen.exe, ncdump.exe	NetCDF utilities	Windows\System (98/ME) or Windows\System32 (2000/XP)
netcdf-3	netCDF source code	Program Files\Unidata

4.3 Building netcdf.dll with VC++ 6.0

The most recent releases of netCDF aren't tested under VC++ 6.0. (They are tested with VC++.NET). Older versions of the library, notably 3.5.0, did compile with VC++ 6.0, and the instructions for doing so are presented below.

Note that the introduction of better large file support (for files larger than 2 GiB) in version 3.6.0 and greater requires an off_t type of 8 bytes, and it's not clear how, or if, this can be found in VC++ 6.0.

To build the library yourself, get the file ftp://ftp.unidata.ucar.edu/pub/netcdf/contrib/win32/netcdf-3.5.0.win32make.VC6.zip

The makefiles there describe how to build netcdf-3.5 using the using Microsoft Visual C++ 6.x and (optionally) Digital Visual Fortran 6.x. Because of difficulties in getting Microsoft Visual Studio to fall in line with our existing source directory scheme, we chose _not_ to build the system "inside" Visual Studio. Instead, we provide a simple group of "msoft.mak" files which can be used. If you wish to work in Visual Studio, go ahead. Read the section called "Macros" at the end of this discussion.

As of this writing, we have not tried compiling the C++ interface in this environment.

nmake is a Microsoft version of make, which comes with VC 6.0 (and VC 7.0) in directory C:\Program Files\Microsoft Visual Studio\VC98\Bin (or, for VC 7.0, C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin).

To build netcdf, proceed as follows:

`unpack source distribution.`

`copy netcdf-3.5.0.win32make.VC6.zip`
: copy netcdf-3.5.0.win32make.VC6.zip into the netcdf-3.5.0/src directory, and unzip it from there.

`cd src\libsrc; nmake /f msoft.mak`
: Run this command in src\libsrc. This will build netcdf.lib and netcdf.dll Note: This makefiles make DLLs. To make static libraries see section on static libraries.

`nmake /f msoft.mak test`
: Optionally, in src\libsrc, make and run the simple test.

`cd ..\fortran; nmake /f msoft.mak`
: Optionally build the fortran interface and rebuild dll in ..\libsrc to include the fortran interface. Note Bene: We don't provide a .DEF file, so this step changes the "ordinals" by which entry points in the DLL found. Some sites may wish to modify the msoft.mak file(s) to produce a separate library for the fortran interface.

`nmake /f msoft.mak test`
: (necessary if you want to use fortran code) While you are in src\fortran; nmake /f msoft.mak test This tests the netcdf-2 fortran interface.

`cd ..\nctest; nmake /f msoft.mak test`
: (optional, but recommended) In src\nctest; nmake /f msoft.mak test This tests the netcdf-2 C interface.

`cd ..\nc_test; nmake /f msoft.mak test`
: (optional, but highly recommended) In src\nc_test; nmake /f msoft.mak test This tortures the netcdf-3 C interface.

`cd ..\nf_test; nmake /f msoft.mak test`
: (optional, but highly recommended if you built the fortran interface) In src\nf_test; nmake /f msoft.mak test This tortures the netcdf-3 fortran interface.

`..\ncdump; nmake /f msoft.mak`
: In src\ncdump; nmake /f msoft.mak This makes ncdump.exe.

`..\ncgen; nmake /f msoft.mak`
: In src\ncgen; nmake /f msoft.mak This makes ncgen.exe.

`..\ncdump; nmake /f msoft.mak test`
: (optional) In src\ncdump; nmake /f msoft.mak test This tests ncdump. Both ncgen and ncdump need to be built prior to this test. Note the makefile sets the path so that ..\libsrc\netcdf.dll can be located.

`..\ncgen; nmake /f msoft.mak test`
: (optional) In src\ncgen; nmake /f msoft.mak test This tests ncgen. Both ncgen and ncdump need to be built prior to this test. Note the makefile sets the path so that ..\libsrc\netcdf.dll can be located.

`To Install`
: Copy libsrc\netcdf.lib to a LIBRARY directory. Copy libsrc\netcdf.h and fortran/netcdf.inc to an INCLUDE directory. Copy libsrc\netcdf.dll, ncdump/ncdump.exe, and ncgen/ncgen.exe to a BIN directory (someplace in your PATH).

4.4 Using netcdf.dll with VC++ 6.0

To use the netcdf.dll:

1. Place these in your include directory: netcdf.h C include file netcdf.inc Fortran include file

2a. To use the Dynamic Library (shared) version of the netcdf library: Place these in a directory that's in your PATH: netcdf.dll library dll ncgen.exe uses the dll ncdump.exe uses the dll

Place this in a library directory to link against: netcdf.lib library

2b. Alternatively, to use a static version of the library

Place this in a library directory to link against: netcdfs.lib library

Place these in a directory that's in your PATH: ncgens.exe statically linked (no DLL needed) ncdumps.exe statically linked (no DLL needed)

4.5 Building netcdf.dll with VC++.NET

To build the netCDF dll with VC++.NET open the win32/NET/netcdf.sln file with Visual Studio. Both Debug and Release configurations are available - select one and build.

The resulting netcdf.dll file will be in subdirectory Release or Debug.

The netCDF tests will be built and run as part of the build process. The Fortran 77 interface will be built, but not the Fortran 90 or C++ interfaces.

Unfortunately, different fortran compilers require different flag settings in the netCDF configuration files. (In UNIX builds, this is handled by the configure script.)

The quick_large_files test program is provided as an extra project, however it is not run during the build process, but can be run from the command line or the IDE. Note that, despite its name, it is not quick. On Unix systems, this program runs in a few seconds, because of some features of the Unix file system apparently not present in Windows. Nonetheless, the program does run, and creates (then deletes) some very large files. (So make sure you have at least 15 GiB of space available). It takes about 45 minutes to run this program on our Windows machines, so please be patient.

4.6 Using netcdf.dll with VC++.NET

Load-time linking to the DLL is the most straightforward from C++. This means the netcdf.lib file has to end up on the compile command line. This being Windows, that's hidden by a GUI.

In Visual Studio 2003 this can be done by modifying three of the project's properties.

Open the project properties window from the project menu. Go to the linker folder and look at the general properties. Modify the property "Additional Library Directories" by adding the directory which contains the netcdf.dll and netcdf.lib files. Now go to the linker input properties and set the property "Additional Dependencies" to netcdf.lib.

Finally, still within the project properties window, go to the C/C++ folder, and look at the general properties. Modify "Additional Include Directories" to add the directory with the netcdf.h file.

Now use the netCDF functions in your C++ code. Of course any C or C++ file that wants to use the functions will need:

```
#include <netcdf.h>
```

5 If Something Goes Wrong

The netCDF package is designed to build and install on a wide variety of platforms, but doesn't always. It's a crazy old world out there, after all.

5.1 The Usual Build Problems

5.1.1 Taking the Easy Way Out

Why not take the easy way out if you can?

Precompiled binaries for many platforms can be found at `http://www.unidata.ucar.edu/downloads/netcdf/index.jsp`. Click on your platform, and copy the files from the bin, include, lib, and man directories into your own local equivalents (Perhaps /usr/local/bin, /usr/local/include, etc.).

5.1.2 How to Clean Up the Mess from a Failed Build

If you are trying to get the configure or build to work, make sure you start with a clean distribution for each attempt. If netCDF failed in the "make" you must clean up the mess before trying again. To clean up the distribution:

```
make distclean
```

5.1.3 Platforms On Which NetCDF is Known to Work

At NetCDF World Headquarters (in sunny Boulder, Colorado), as part of the wonderful Unidata organization, we have a wide variety of computers, operating systems, and compilers. At night, house elves test netCDF on all these systems.

Output for the netCDF test platforms can be found at `http://www.unidata.ucar.edu/netcdf/builds`.

Compare the output of your build attempt with ours. Are you using the same compiler? The same flags? Look for the configure output that lists the settings of CC, FC, CXX, CFLAGS, etc.

On some systems you have to set environment variables to get the configure and build to work.

For example, for a 64-bit IRIX install of the netCDF-3.6.2 release, the variables are set before netCDF is configured or built. In this case we set CFLAGS, CXXFLAGS, FCFLAGS, and FFLAGS.

```
flip% uname -a
IRIX64 flip 6.5 07080050 IP30 mips
flip% setenv CFLAGS -64
flip% setenv CXXFLAGS -64
flip% setenv FFLAGS -64
flip% setenv FCFLAGS -64
flip% make distclean;./configure;make check
```

5.1.4 Platforms On Which NetCDF is Reported to Work

If your platform isn't listed on the successful build page, see if another friendly netCDF user has sent in values for environment variables that are reported to work: (`http://www.unidata.ucar.edu/netcdf/other-builds.html`).

If you build on a system that we don't have at Unidata (particularly if it's something interesting and exotic), please send us the settings that work (and the entire build output would be nice too). Send them to support-netcdf@unidata.ucar.edu.

5.1.5 If You Have a Broken Compiler

For netCDF to build correctly, you must be able to compile C from your environment, and, optionally, Fortran 77, Fortran 90, and C++. If C doesn't work, netCDF can't compile.

What breaks a C compiler? Installation or upgrade mistakes when the C compiler was installed, or multiple versions or compilers installed on top of each other. Commercial compilers frequently require some environment variables to be set, and some directories to appear ahead of others in your path. Finally, if you have an expired or broken license, your C compiler won't work.

If you have a broken C compiler and a working C compiler in your PATH, netCDF might only find the broken one. You can fix this by explicitly setting the CC environmental variable to a working C compiler, and then trying to build netCDF again. (Don't forget to do a "make distclean" first!)

If you can't build a C program, you can't build netCDF. Sorry, but that's just the way it goes. (You can get the GNU C compiler - search the web for "gcc").

If netCDF finds a broken Fortran 90, Fortran 77, or C++ compiler, it will report the problem during the configure, and then drop the associated API. For example, if the C++ compiler can't compile a very simple test program, it will drop the C++ interface. If you really want the C++ API, set the CXX environment variable to a working C++ compiler.

5.1.6 What to Do If NetCDF Still Won't Build

If none of the above help, try our troubleshooting section: See Section 5.2 [Troubleshooting], page 26.

Also check to see of your problem has already been solved by someone else (see Section 5.3 [Finding Help], page 27).

If you still can't get netCDF to build, report your problem to Unidata, but please make sure you submit all the information we need to help (see Section 5.4 [Reporting Problems], page 28).

5.2 Troubleshooting

5.2.1 Problems During Configuration

If the ./configure; make check fails, it's a good idea to turn off the C++ and Fortran interfaces, and try to build the C interface alone. All other interfaces depend on the C interface, so nothing else will work until the C interface works. To turn off C++ and Fortran, set environment variables CXX and FC to NULL before running the netCDF configure script (with csh: setenv FC ";setenv CXX ").

Turning off the Fortran and C++ interfaces results in a much shorter build and test cycle, which is useful for debugging problems.

If the netCDF configure fails, most likely the problem is with your development environment. The configure script looks through your path to find all the tools it needs to build netCDF, including C compiler and linker, the ar, ranlib, and others. The configure script will tell you what tools it found, and where they are on your system. Here's part of configure's output on a Linux machine:

```
checking CPPFLAGS...  -Df2cFortran
checking CC CFLAGS... cc -g
checking which cc... /usr/bin/cc
checking CXX... c++
checking CXXFLAGS... -g -O2
checking which c++... /usr/local/bin/c++
checking FC... f77
checking FFLAGS...
checking which f77... /usr/bin/f77
checking F90... unset
checking AR... ar
checking ARFLAGS... cru
checking which ar... /usr/bin/ar
checking NM... nm
checking NMFLAGS...
checking which nm... /usr/bin/nm
```

Make sure that the tools, directories, and flags are set to reasonable values, and compatible tools. For example the GNU tools may not inter-operate well with vendor tools. If you're using a vendor compiler, use the ar, nm, and ranlib that the vendor supplied.

As configure runs, it creates a config.log file. If configure crashes, do a text search of config.log for thing it was checking before crashing. If you have a licensing or tool compatibility problem, it will be obvious in config.log.

5.2.2 Problems During Compilation

If the configure script runs, but the compile step doesn't work, or the tests don't complete successfully, the problem is probably in your CFLAGS or CPPFLAGS.

Frequently shared libraries are a rich source of problems. If your build is not working, and you are using the –enable-shared option to generate shared libraries, then try to build without –enable-shared, and see if the static library build works.

5.2.3 Problems During Testing

If you are planning on using large files (i.e. > 2 GiB), then you may wish to rerun configure with –enable-large-file-tests to ensure that large files work on your system.

Some DAP tests (in the directory ncdap_test) attempt to access an external server at opendap.org. It is possible that the DAP server may not be running at test time, or the network access may be faulty. In this case, the DAP tests may fail. Because of this, the use of these tests is controlled by the –enable-dap-remote-tests option.

5.3 Finding Help On-line

The latest netCDF documentation (including this manual) can be found at `http://www.unidata.ucar.edu/netcdf/docs`.

The output of successful build and test runs for recent versions of netCDF can be found at `http://www.unidata.ucar.edu/netcdf/builds`.

A list of known problems with netCDF builds, and suggested fixes, can be found at `http://www.unidata.ucar.edu/netcdf/docs/known_problems.html`.

Reportedly successful settings for platforms unavailable for netCDF testing can be found at `http://www.unidata.ucar.edu/netcdf/other-builds.html`. If you build netCDF on a system that is not listed, please send your environment settings, and the full output of your configure, compile, and testing, to support-netcdf@unidata.ucar.edu. We will add the information to the other-builds page, with a credit to you.

The replies to all netCDF support emails are on-line and can be searched. Before reporting a problem to Unidata, please search this on-line database to see if your problem has already been addressed in a support email. If you are having build problems it's usually useful to search on your system host name. On Unix systems, use the uname command to find it.

The netCDF Frequently Asked Questions (FAQ) list can be found at `http://www.unidata.ucar.edu/netcdf/faq.html`.

To search the support database, see `/search.jsp?support&netcdf`.

The netCDF mailing list also can be searched; see `/search.jsp?netcdfgroup`.

5.4 Reporting Problems

To help us solve your problem, please include the following information in your email to `support-netcdf@unidata.ucar.edu`.

Unfortunately, we can't solve build questions without this information; if you ask for help without providing it, we're just going to have to ask for it.

So why not send it immediately, and save us both the extra trouble?

1. the exact version of netCDF - see the VERSION file.
2. the *complete* output of "./configure", "make", and "make check. Yes, it's long, but it's all important.
3. if the configure failed, the contents of config.log.
4. if you are having problems with very large files (larger than 2GiB), send the output of "make check" after first running "make distclean" and invoking the configure script with the –enable-large-file-tests option included.

Although responses to your email will be available in our support database, your email address is not included, to provide spammers with one less place to harvest it from.

Index

M

N

O

P

Q

R

S

T

V

W