The NetCDF Tutorial

NetCDF the Easy Way
NetCDF Version 4.1.1
Last Updated 30 September 2009

Ed Hartnett
Unidata Program Center

Table of Contents

1 What is NetCDF?

NetCDF is a set of data formats, programming interfaces, and software libraries that help read and write scientific data files.

NetCDF was developed and is maintained at Unidata, part of the University Corporation for Atmospheric Research (UCAR) Office of Programs (UOP). Unidata is funded primarily by the National Science Foundation.

Unidata provides data and software tools for use in geoscience education and research. For more information see the web sites of Unidata (`http://www.unidata.ucar.edu`), UOP (`http://www.uop.ucar.edu`), and UCAR (`http://www.ucar.edu`).

This tutorial may serve as an introduction to netCDF. Full netCDF documentation is available on-line (see Section 1.8 [Documentation], page 6).

1.1 The Classic NetCDF Data Model

The classic netCDF data model consists of *variables*, *dimensions*, and *attributes*. This way of thinking about data was introduced with the very first netCDF release, and is still the core of all netCDF files.

(In version 4.0, the netCDF data model has been expanded. See Section 1.2 [Common Data Model], page 2.)

`Variables`
: N-dimensional arrays of data. Variables in netCDF files can be one of six types (char, byte, short, int, float, double). For more information see Section "Variables" in *The NetCDF Users Guide*.

`Dimensions`
: describe the axes of the data arrays. A dimension has a name and a length. An unlimited dimension has a length that can be expanded at any time, as more data are written to it. NetCDF files can contain at most one unlimited dimension. For more information see Section "Dimensions" in *The NetCDF Users Guide*.

`Attributes`
: annotate variables or files with small notes or supplementary metadata. Attributes are always scalar values or 1D arrays, which can be associated with either a variable or the file as a whole. Although there is no enforced limit, the user is expected to keep attributes small. For more information see Section "Attributes" in *The NetCDF Users Guide*.

For more information on the netCDF data model see Section "The NetCDF Data Model" in *The NetCDF Users Guide*.

1.1.1 Meteorological Example

NetCDF can be used to store many kinds of data, but it was originally developed for the Earth science community.

NetCDF views the world of scientific data in the same way that an atmospheric scientist might: as sets of related arrays. There are various physical quantities (such as pressure and temperature) located at points at a particular latitude, longitude, vertical level, and time.

A scientist might also like to store supporting information, such as the units, or some information about how the data were produced.

The axis information (latitude, longitude, level, and time) would be stored as netCDF dimensions. Dimensions have a length and a name.

The physical quantities (pressure, temperature) would be stored as netCDF variables. Variables are N-dimensional arrays of data, with a name and an associated set of netCDF dimensions.

It is also customary to add one variable for each dimension, to hold the values along that axis. These variables are call "coordinate variables." The latitude coordinate variable would be a one-dimensional variable (with latitude as its dimension), and it would hold the latitude values at each point along the axis.

The additional bits of metadata would be stored as netCDF attributes.

Attributes are always single values or one-dimensional arrays. (This works out well for a string, which is a one-dimensional array of ASCII characters.)

The pres_temp_4D example in this tutorial shows how to write and read a file containing some four-dimensional pressure and temperature data, including all the metadata needed. See Section 2.3 [pres_temp_4D], page 55.

1.2 The Common Data Model and NetCDF-4

With netCDF-4, the netCDF data model has been extended, in a backwards compatible way.

The new data model, which is known as the "Common Data Model" is part of an effort here at Unidata to find a common engineering language for the development of scientific data solutions. It contains the variables, dimensions, and attributes of the classic data model, but adds:

- groups A way of hierarchically organizing data, similar to directories in a Unix file system.
- user-defined types The user can now define compound types (like C structures), enumeration types, variable length arrays, and opaque types.

These features may only be used when working with a netCDF-4/HDF5 file. Files created in classic or 64-bit offset format cannot support groups or user-defined types.

With netCDF-4/HDF5 files, the user may define groups, which may contain variables, dimensions, and attributes. In this way, a group acts as a container for the classic netCDF dataset. But netCDF-4/HDF5 files can have many groups, organized hierarchically.

Each file begins with at least one group, the root group. The user may then add more groups, receiving a new ncid for each group created.

Since each group functions as a complete netCDF classic dataset, it is possible to have variables with the same name in two or more different groups, within the same netCDF-4/HDF5 data file.

Dimensions have a special scope: they may me seen my all variables in their group, and all descendant groups. This allows the user to define dimensions in a top-level group, and use them in many sub-groups.

Since it may be necessary to write code which works with all types of netCDF data files, we also introduce the ability to create netCDF-4/HDF5 files which follow all the rules of the classic netCDF model. That is, these files are in HDF5, but will not support multiple unlimited dimensions, user-defined types, groups, etc. They act just like a classic netCDF file.

1.3 NetCDF Error Handling

Each netCDF function in the C, Fortran 77, and Fortran 90 APIs returns 0 on success, in the tradition of C. (For C++, see below).

When programming with netCDF in these languages, always check return values of every netCDF API call. The return code can be looked up in netcdf.h (for C programmers) or netcdf.inc (for Fortran programmers), or you can use the strerror function to print out an error message. (See Section "nc_strerror" in *The NetCDF C Interface Guide*/Section "NF_STRERROR" in *The NetCDF Fortran 77 Interface Guide*/Section "NF90_STRERROR" in *The NetCDF Fortran 90 Interface Guide*).

In general, if a function returns an error code, you can assume it didn't do what you hoped it would. The exception is the NC_ERANGE error, which is returned by any of the reading or writing functions when one or more of the values read or written exceeded the range for the type. (For example if you were to try to read 1000 into an unsigned byte.)

In the case of NC_ERANGE errors, the netCDF library completes the read/write operation, and then returns the error. The type conversion is handled like a C type conversion, whether or not it is within range. This may yield bad data, but the netCDF library just returns NC_ERANGE and leaves it up to the user to handle. (For more information about type conversion see Section "Type Conversion" in *The NetCDF C Interface Guide*).

Error handling in C++ is different. For some objects, the is_valid() method should be called. Other error handling is controlled by the NcError class. For more information see Section "Class NcError" in *The NetCDF C++ Interface Guide*.

For a complete list of netCDF error codes see Section "Error Codes" in *The NetCDF C Interface Guide*.

1.4 Unlimited Dimensions

Sometimes you don't know the size of all dimensions when you create a file, or you would like to arbitrarily extend the file along one of the dimensions.

For example, model output usually has a time dimension. Rather than specifying that there will be forty-two output times when creating the file, you might like to create it with one time, and then add data for additional times, until you wanted to stop.

For this purpose netCDF provides the unlimited dimension. By specifying a length of "unlimited" when defining a dimension, you indicate to netCDF that the dimension may be extended, and its length may increase.

In netCDF classic files, there can only be one unlimited dimension, and it must be declared first in the list of dimensions for a variable.

For programmers, the unlimited dimension will correspond with the slowest-varying dimension. In C this is the first dimension of an array, in Fortran, the last.

The third example in this tutorial, pres_temp_4D, demonstrates how to write and read data one time step at a time along an unlimited dimension in a classic netCDF file. See Section 2.3 [pres_temp_4D], page 55.

In netCDF-4/HDF5 files, any number of unlimited dimensions may be used, and there is no restriction as to where they appear in a variable's list of dimension IDs.

For more detailed information about dimensions see Section "Dimensions" in *The NetCDF Users Guide*.

1.5 Fill Values

Sometimes there are missing values in the data, and some value is needed to represent them.

For example, what value do you put in a sea-surface temperature variable for points over land?

In netCDF, you can create an attribute for the variable (and of the same type as the variable) called "_FillValue" that contains a value that you have used for missing data. Applications that read the data file can use this to know how to represent these values.

Using attributes it is possible to capture metadata that would otherwise be separated from the data. Various conventions have been established. By using a set of conventions, a data producer is more likely to produce files that can be easily shared within the research community, and that contain enough details to be useful as a long-term archive.

For more information on _FillValue and other attribute conventions, see Section "Attribute Conventions" in *The NetCDF Users Guide*.

Climate and meteorological users are urged to follow the Climate and Forecast (CF) metadata conventions when producing data files. For more information about the CF conventions, see `http://cf-pcmdi.llnl.gov`.

For information about creating attributes, see Section 3.1 [Creation], page 87.

1.6 Tools for Manipulating NetCDF Files

Many existing software applications can read and manipulate netCDF files. Before writing your own program, check to see if any existing programs meet your needs.

Two utilities come with the netCDF distribution: ncdump and ncgen. The ncdump command reads a netCDF file and outputs ASCII in a format called CDL. The ncgen command reads an ASCII file in CDL format, and generates a netCDF data file.

One common use for ncdump is to examine the metadata of a netCDF file, to see what it contains. At the beginning of each example in this tutorial, an ncdump of the resulting data file is shown. See Section 2.1 [simple_xy], page 9.

For more information about ncdump and ncgen see Section "NetCDF Utilities" in *The NetCDF Users Guide*.

The following general-purpose tools have been found to be useful in many situations. Some of the tools on this list are developed at Unidata. The others are developed elsewhere, and we can make no guarantees about their continued availability or success. All of these tools are open-source.

UDUNITS	Unidata library to help with scientific units.	`http://www.unidata.ucar.edu/software/udunits`
IDV	Unidata's Integrated Data Viewer, a 3D visualization and analysis package (Java based).	`http://www.unidata.ucar.edu/software/idv`
NCL	NCAR Command Language, a graphics and data manipulation package.	`http://www.ncl.ucar.edu`
GrADS	The Grid Analysis and Display System package.	`http://grads.iges.org/grads/grads.html`
NCO	NetCDF Command line Operators, tools to manipulate netCDF files.	`http://nco.sourceforge.net`

For a list of netCDF tools that we know about see `http://www.unidata.ucar.edu/netcdf/software.html` If you know of any that should be added to this list, send email to support-netcdf@unidata.ucar.edu.

1.7 The NetCDF Programming APIs

Unidata supports netCDF APIs in C, C++, Fortran 77, Fortran 90, and Java.

The Java API is a complete implementation of netCDF in Java. It is distributed independently of the other APIs. For more information see the netCDF Java page: `http://www.unidata.ucar.edu/software/netcdf-java`. If you are writing web server software, you should certainly be doing so in Java.

The C, C++, Fortran 77 and Fortran 90 APIs are distributed and installed when the netCDF C library is built, if compilers exist to build them, and if they are not turned off when configuring the netCDF build.

The C++ and Fortran APIs depend on the C API. Due to the nature of C++ and Fortran 90, users of those languages can also use the C and Fortran 77 APIs (respectively) directly.

In the netCDF-4.0 beta release, only the C API is well-tested. The Fortran APIs include support for netCDF-4 advanced features, but need more testing, which will be added in a future release of netCDF.

The C++ API can handle netCDF-4.0/HDF5 files, but can not yet handle advanced netCDF-4 features. The successor to the current C++ API is under active development, and will include support for netCDF-4 advanced features.

Full documentation exists for each API (see Section 1.8 [Documentation], page 6).

In addition, many other language APIs exist, including Perl, Python, and Ruby. Most of these APIs were written and supported by netCDF users. Some of them are listed on the netCDF software page, see `http://www.unidata.ucar.edu/netcdf/software.html`. Since these generally use the C API, they should work well with netCDF-4/HDF5 files, but the maintainers of the APIs must add support for netCDF-4 advanced features.

In addition to the main netCDF-3 C API, there is an additional (older) C API, the netCDF-2 API. This API produces exactly the same files as the netCDF-3 API - only the API is different. (That is, users can create either classic format files, the default, or 64-bit offset files, or netCDF-4/HDF5 files.)

The version 2 API was the API before netCDF-3.0 came out. It is still fully supported, however. Programs written to the version 2 API will continue to work.

Users writing new programs should use the netCDF-3 API, which contains better type checking, better error handling, and better documentation.

The netCDF-2 API is provided for backward compatibility. Documentation for the netCDF-2 API can be found on the netCDF website, see `http://www.unidata.ucar.edu/netcdf/guide_toc.html`.

1.8 NetCDF Documentation

This tutorial is brief. A much more complete description of netCDF can be found in The NetCDF Users Guide. It fully describes the netCDF model and format. For more information see Section "Top" in *The NetCDF Users Guide*.

The netCDF distribution, in various forms, can be obtained from the netCDF web site: `http://www.unidata.ucar.edu/netcdf`.

A porting and installation guide for the C, C++, Fortran 77, and Fortran 90 APIs describes how to build these APIs on a variety of platforms. See Section "Top" in *The NetCDF Installation and Porting Guide*.

Language specific programming guides are available for netCDF for the C, C++, Fortran 77, Fortran 90, and Java APIs:

`C` Section "Top" in *The NetCDF C Interface Guide*.

`C++` Section "Top" in *The NetCDF C++ Interface Guide*.

`Fortran 77`
Section "Top" in *The NetCDF Fortran 77 Interface Guide*.

`Fortran 90`
Section "Top" in *The NetCDF Fortran 90 Interface Guide*.

`Java` `http://www.unidata.ucar.edu/software/netcdf-java/v2.1/NetcdfJavaUserManual.htm`.

Man pages for the C, F77, and F90 interfaces, and ncgen and ncdump, are available on the documentation page of the netCDF web site (`http://www.unidata.ucar.edu/netcdf/docs`), and are installed with the netCDF distribution.

The latest version of all netCDF documentation can always be found at the documentation page of the netCDF web site: `http://www.unidata.ucar.edu/netcdf/docs`

1.9 A Note on NetCDF Versions and Formats

NetCDF has changed (and improved) over its lifetime. That means the user must have some understanding of netCDF versions.

To add to the confusion, there are versions for the APIs, and also for the data files that they produce. The API version is the version number that appears in the tarball file that is downloaded from the netCDF website. For example this document applied to API version 4.1.1.

The good news is that all netCDF files ever written can always be read by the latest netCDF release. That is, we guarantee backward data compatibility.

1.9.1 Classic Format

The default format is classic format. This is the original netCDF binary format - the format that the netCDF library has been using for almost 20 years, since its introduction with the first release of netCDF. No special flag is needed to create a file in classic format; it is the default.

Classic format has some strict limitations for files larger than two gigabytes. (see Section "NetCDF Classic Format Limitations" in *The NetCDF Users Guide*).

1.9.2 64-bit Offset Format

In December, 2004, version 3.6.0 of the netCDF library was released. It allows users to use a new version of the netCDF file format which greatly expands the sizes of variables and files which may be written.

The format which was introduced in 3.6.0 is called "64-bit Offset Format."

Create files in this format by passing the 64-bit offset format flag to the create call (for example, in C, set the NC_64BIT_OFFSET flag when calling the function nc_create. (see Section "nc_create" in *The NetCDF C Interface Guide*).

64-bit offset is very useful for very large data files (over two gigabytes), however these files can only be shared with those who have upgraded to version 3.6.0 (or better) of netCDF. Earlier versions of netCDF will not be able to read these files.

1.9.3 NetCDF-4/HDF5 Format

With version 4.0 of netCDF, we introduce another new data format: netCDF-4/HDF5 format. This format is HDF5, with full use of the new dimension scales, creation ordering, and other features of HDF5 added in its version 1.8.0 release.

As with 64-bit offset, this format is turned on when the file is created. (For example, with the nf_netcdf4 flag in the nf_create function. see Section "nf_create" in *The NetCDF Fortran 77 Interface Guide*).

1.9.4 Sharing Data

The classic format is the most portable. Classic format files can be read correctly by any version of netCDF. A netCDF-4 user can create a classic file, and share it with a user who has not upgraded netCDF since the version 2.3 in 1994.

64-bit offset format files can be read by any user who has at least version 3.6.0 of the netCDF API (released in Dec., 2004).

Users must have netCDF 4.0 to read netCDF-4/HDF5 files. However, netCDF-4 does produce backward compatible classic and 64-bit offset format files. That is, a netCDF-4.0 user can create a classic format file, and share it with researchers who are still using a old version of netCDF. Similarly a netCDF-4.0 user can read any existing netCDF data file, whatever version of netCDF was used to create it.

1.9.5 Classic Model

The original netCDF API represents a data model as well as a programming API. That is, the idea of variables, attributes, and the six data types (char, byte, short, integer, float, and double), comprises a model of how data may be stored.

The netCDF-4 release expands this model with groups, user-defined types, and new base types. New functions have been added to the APIs to accommodate these extensions, but once they are used, the file can no longer be output as a classic format file.

That is, once you use groups in your code, you can only produce netCDF-4/HDF5 files. If you try to change the format to classic, you will get an error when you attempt to use any of the group functions.

Since it is convenient to be able to produce files of all formats from the same code (restricting oneself to the classic data model), a flag has been added which, when used in the creation of a netCDF-4/HDF5 file, will instruct the library to disallow the use of any advanced features in the file.

This is referred to as a “classic model” netCDF-4/HDF5 file.

To get a classic model file, use the classic model flag when creating the file. For example, in Fortran 77, use the nf_classic_model flag when calling nf_create (see Section “nf_create” in *The NetCDF Fortran 77 Interface Guide*).

For more information about format issues see Section “Format” in *The NetCDF Users Guide*.

2 Example Programs

The netCDF example programs show how to use netCDF.

In the netCDF distribution, the "examples" directory contains examples in C, Fortran 77, Fortran 90, C++, and CDL.

There are three sets of netCDF-3 example programs in each language. Each language has its own subdirectory under the "examples" directory (for example, the Fortran 77 examples are in "examples/F77").

There is also one example for netCDF-4, which is only provided in the C language. This example will only be run if the –enable-netcdf-4 option was used with configure.

The examples are built and run with the "make check" command. (For more information on building netCDF, see Section "Top" in *The NetCDF Installation and Porting Guide*).

The examples create, and then read, example data files of increasing complexity.

The corresponding examples in each language create identical netCDF data files. For example, the C program sfc_pres_temp_wr.c produces the same data file as the Fortran 77 program sfc_pres_temp_wr.f.

For convenience, the complete source code in each language can be found in this tutorial, as well as in the netCDF distribution.

2.1 The simple_xy Example

This example is an unrealistically simple netCDF file, to demonstrate the minimum operation of the netCDF APIs. Users should seek to make their netCDF files more self-describing than this primitive example.

As in all the netCDF tutorial examples, this example file is created by C, Fortran 77, Fortran 90, and C++ programs, and by ncgen, which creates it from a CDL script. All examples create identical files, "simple_xy.nc."

The programs that create this sample file all have the base name "simple_xy_wr", with different extensions depending on the language.

Therefore the example files that create simple_xy.nc can be found in: C/simple_xy_wr.c, F77/simple_xy_wr.f, F90/simple_xy_wr.f90, CXX/simple_xy_wr.cpp, and CDL/simple_xy_wr.cdl.

Corresponding read programs (C/simple_xy_rd.c, etc.) read the simple_xy.nc data file, and ensure that it contains the correct values.

The simple_xy.nc data file contains two dimensions, "x" and "y", and one netCDF variable, "data."

The utility ncdump can be used to show the contents of netCDF files. By default, ncdump shows the CDL description of the file. This CDL description can be fed into ncgen to create the data file.

The CDL for this example is shown below. For more information on ncdump and ncgen see Section "NetCDF Utilities" in *The NetCDF Users Guide*.

```
netcdf simple_xy {
dimensions:
x = 6 ;
```

```
y = 12 ;
variables:
int data(x, y) ;
data:

 data =
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
  12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
  24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
  36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
  48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
  60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71 ;
}
```

2.1.1 simple_xy_wr.c and simple_xy_rd.c

These example programs can be found in the netCDF distribution, under examples/C.

The example program simple_xy_wr.c creates the example data file simple_xy.nc. The example program simple_xy_rd.c reads the data file.

2.1.1.1 simple_xy_wr.c

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.

   This is a very simple example which writes a 2D array of
   sample data. To handle this in netCDF we create two shared
   dimensions, "x" and "y", and a netCDF variable, called "data".

   This example demonstrates the netCDF C API. This is part of the
   netCDF tutorial, which can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

   Full documentation of the netCDF C API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c

   $Id: simple_xy_wr.c,v 1.12 2007/02/14 20:59:21 ed Exp $
*/
#include <stdlib.h>
#include <stdio.h>
#include <netcdf.h>

/* This is the name of the data file we will create. */
#define FILE_NAME "simple_xy.nc"

/* We are writing 2D data, a 6 x 12 grid. */
#define NDIMS 2
```

```
#define NX 6
#define NY 12

/* Handle errors by printing an error message and exiting with a
 * non-zero status. */
#define ERRCODE 2
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); exit(ERRCODE);}

int
main()
{
   /* When we create netCDF variables and dimensions, we get back an
    * ID for each one. */
   int ncid, x_dimid, y_dimid, varid;
   int dimids[NDIMS];

   /* This is the data array we will write. It will be filled with a
    * progression of numbers for this example. */
   int data_out[NX][NY];

   /* Loop indexes, and error handling. */
   int x, y, retval;

   /* Create some pretend data. If this wasn't an example program, we
    * would have some real data to write, for example, model
    * output. */
   for (x = 0; x < NX; x++)
      for (y = 0; y < NY; y++)
 data_out[x][y] = x * NY + y;

   /* Always check the return code of every netCDF function call. In
    * this example program, any retval which is not equal to NC_NOERR
    * (0) will cause the program to print an error message and exit
    * with a non-zero return code. */

   /* Create the file. The NC_CLOBBER parameter tells netCDF to
    * overwrite this file, if it already exists.*/
   if ((retval = nc_create(FILE_NAME, NC_CLOBBER, &ncid)))
      ERR(retval);

   /* Define the dimensions. NetCDF will hand back an ID for each. */
   if ((retval = nc_def_dim(ncid, "x", NX, &x_dimid)))
      ERR(retval);
   if ((retval = nc_def_dim(ncid, "y", NY, &y_dimid)))
      ERR(retval);

   /* The dimids array is used to pass the IDs of the dimensions of
```

```
    * the variable. */
   dimids[0] = x_dimid;
   dimids[1] = y_dimid;

   /* Define the variable. The type of the variable in this case is
    * NC_INT (4-byte integer). */
   if ((retval = nc_def_var(ncid, "data", NC_INT, NDIMS,
    dimids, &varid)))
      ERR(retval);

   /* End define mode. This tells netCDF we are done defining
    * metadata. */
   if ((retval = nc_enddef(ncid)))
      ERR(retval);

   /* Write the pretend data to the file. Although netCDF supports
    * reading and writing subsets of data, in this case we write all
    * the data in one operation. */
   if ((retval = nc_put_var_int(ncid, varid, &data_out[0][0])))
      ERR(retval);

   /* Close the file. This frees up any internal netCDF resources
    * associated with the file, and flushes any buffers. */
   if ((retval = nc_close(ncid)))
      ERR(retval);

   printf("*** SUCCESS writing example file simple_xy.nc!\n");
   return 0;
}
```

2.1.1.2 simple_xy_rd.c

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.

   This is a simple example which reads a small dummy array, which was
   written by simple_xy_wr.c. This is intended to illustrate the use
   of the netCDF C API.

   This program is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

   Full documentation of the netCDF C API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c

   $Id: simple_xy_rd.c,v 1.9 2006/08/17 23:00:55 russ Exp $
```

```
*/
#include <stdlib.h>
#include <stdio.h>
#include <netcdf.h>

/* This is the name of the data file we will read. */
#define FILE_NAME "simple_xy.nc"

/* We are reading 2D data, a 6 x 12 grid. */
#define NX 6
#define NY 12

/* Handle errors by printing an error message and exiting with a
 * non-zero status. */
#define ERRCODE 2
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); exit(ERRCODE);}

int
main()
{
   /* This will be the netCDF ID for the file and data variable. */
   int ncid, varid;

   int data_in[NX][NY];

   /* Loop indexes, and error handling. */
   int x, y, retval;

   /* Open the file. NC_NOWRITE tells netCDF we want read-only access
    * to the file.*/
   if ((retval = nc_open(FILE_NAME, NC_NOWRITE, &ncid)))
      ERR(retval);

   /* Get the varid of the data variable, based on its name. */
   if ((retval = nc_inq_varid(ncid, "data", &varid)))
      ERR(retval);

   /* Read the data. */
   if ((retval = nc_get_var_int(ncid, varid, &data_in[0][0])))
      ERR(retval);

   /* Check the data. */
   for (x = 0; x < NX; x++)
      for (y = 0; y < NY; y++)
  if (data_in[x][y] != x * NY + y)
     return ERRCODE;
```

```
   /* Close the file, freeing all resources. */
   if ((retval = nc_close(ncid)))
      ERR(retval);

   printf("*** SUCCESS reading example file %s!\n", FILE_NAME);
   return 0;
}
```

2.1.2 simple_xy_wr.f and simple_xy_rd.f

These example programs can be found in the netCDF distribution, under examples/F77.

The example program simple_xy_wr.f creates the example data file simple_xy.nc. The example program simple_xy_rd.f reads the data file.

2.1.2.1 simple_xy_wr.f

```
C     This is part of the netCDF package.
C     Copyright 2006 University Corporation for Atmospheric Research/Unidata.
C     See COPYRIGHT file for conditions of use.

C     This is a very simple example which writes a 2D array of
C     sample data. To handle this in netCDF we create two shared
C     dimensions, "x" and "y", and a netCDF variable, called "data".

C     This example demonstrates the netCDF Fortran 77 API. This is part
C     of the netCDF tutorial, which can be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

C     Full documentation of the netCDF Fortran 77 API can be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f77

C     $Id: simple_xy_wr.f,v 1.11 2008/08/20 22:29:56 russ Exp $

      program simple_xy_wr
      implicit none
      include 'netcdf.inc'

C     This is the name of the data file we will create.
      character*(*) FILE_NAME
      parameter (FILE_NAME='simple_xy.nc')

C     We are writing 2D data, a 12 x 6 grid.
      integer NDIMS
      parameter (NDIMS=2)
      integer NX, NY
      parameter (NX = 6, NY = 12)

C     When we create netCDF files, variables and dimensions, we get back
```

```
C     an ID for each one.
      integer ncid, varid, dimids(NDIMS)
      integer x_dimid, y_dimid

C     This is the data array we will write. It will just be filled with
C     a progression of integers for this example.
      integer data_out(NY, NX)

C     Loop indexes, and error handling.
      integer x, y, retval

C     Create some pretend data. If this wasn't an example program, we
C     would have some real data to write, for example, model output.
      do x = 1, NX
         do y = 1, NY
            data_out(y, x) = (x - 1) * NY + (y - 1)
         end do
      end do

C     Always check the return code of every netCDF function call. In
C     this example program, any retval which is not equal to nf_noerr
C     (0) will call handle_err, which prints a netCDF error message, and
C     then exits with a non-zero return code.

C     Create the netCDF file. The nf_clobber parameter tells netCDF to
C     overwrite this file, if it already exists.
      retval = nf_create(FILE_NAME, NF_CLOBBER, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the dimensions. NetCDF will hand back an ID for each.
      retval = nf_def_dim(ncid, "x", NX, x_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_dim(ncid, "y", NY, y_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     The dimids array is used to pass the IDs of the dimensions of
C     the variables. Note that in fortran arrays are stored in
C     column-major format.
      dimids(2) = x_dimid
      dimids(1) = y_dimid

C     Define the variable. The type of the variable in this case is
C     NF_INT (4-byte integer).
      retval = nf_def_var(ncid, "data", NF_INT, NDIMS, dimids, varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     End define mode. This tells netCDF we are done defining metadata.
```

```
      retval = nf_enddef(ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Write the pretend data to the file. Although netCDF supports
C     reading and writing subsets of data, in this case we write all the
C     data in one operation.
      retval = nf_put_var_int(ncid, varid, data_out)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Close the file. This frees up any internal netCDF resources
C     associated with the file, and flushes any buffers.
      retval = nf_close(ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

      print *,'*** SUCCESS writing example file simple_xy.nc!'
      end

      subroutine handle_err(errcode)
      implicit none
      include 'netcdf.inc'
      integer errcode

      print *, 'Error: ', nf_strerror(errcode)
      stop 2
      end
```

2.1.2.2 simple_xy_rd.f

```
C     This is part of the netCDF package.
C     Copyright 2006 University Corporation for Atmospheric Research/Unidata.
C     See COPYRIGHT file for conditions of use.

C     This is a simple example which reads a small dummy array, from a
C     netCDF data file created by the companion program simple_xy_wr.f.

C     This is intended to illustrate the use of the netCDF fortran 77
C     API. This example program is part of the netCDF tutorial, which can
C     be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

C     Full documentation of the netCDF Fortran 77 API can be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f77

C     $Id: simple_xy_rd.f,v 1.8 2007/02/14 20:59:20 ed Exp $

      program simple_xy_rd
      implicit none
```

```
      include 'netcdf.inc'

C     This is the name of the data file we will read.
      character*(*) FILE_NAME
      parameter (FILE_NAME='simple_xy.nc')

C     We are reading 2D data, a 12 x 6 grid.
      integer NX, NY
      parameter (NX = 6, NY = 12)
      integer data_in(NY, NX)

C     This will be the netCDF ID for the file and data variable.
      integer ncid, varid

C     Loop indexes, and error handling.
      integer x, y, retval

C     Open the file. NF_NOWRITE tells netCDF we want read-only access to
C     the file.
      retval = nf_open(FILE_NAME, NF_NOWRITE, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Get the varid of the data variable, based on its name.
      retval = nf_inq_varid(ncid, 'data', varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Read the data.
      retval = nf_get_var_int(ncid, varid, data_in)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Check the data.
      do x = 1, NX
         do y = 1, NY
            if (data_in(y, x) .ne. (x - 1) * NY + (y - 1)) then
               print *, 'data_in(', y, ', ', x, ') = ', data_in(y, x)
               stop 2
            end if
         end do
      end do

C     Close the file, freeing all resources.
      retval = nf_close(ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

      print *,'*** SUCCESS reading example file ', FILE_NAME, '!'
      end
```

```
      subroutine handle_err(errcode)
      implicit none
      include 'netcdf.inc'
      integer errcode

      print *, 'Error: ', nf_strerror(errcode)
      stop 2
      end
```

2.1.3 simple_xy_wr.f90 and simple_xy_rd.f90

These example programs can be found in the netCDF distribution, under examples/F90.

The example program simple_xy_wr.f90 creates the example data file simple_xy.nc. The example program simple_xy_rd.f90 reads the data file.

2.1.3.1 simple_xy_wr.f90

```
!     This is part of the netCDF package.
!     Copyright 2006 University Corporation for Atmospheric Research/Unidata.
!     See COPYRIGHT file for conditions of use.

!     This is a very simple example which writes a 2D array of
!     sample data. To handle this in netCDF we create two shared
!     dimensions, "x" and "y", and a netCDF variable, called "data".

!     This example demonstrates the netCDF Fortran 90 API. This is part
!     of the netCDF tutorial, which can be found at:
!     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

!     Full documentation of the netCDF Fortran 90 API can be found at:
!     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90

!     $Id: simple_xy_wr.f90,v 1.11 2010/04/06 19:32:08 ed Exp $

program simple_xy_wr
  use netcdf
  implicit none

  ! This is the name of the data file we will create.
  character (len = *), parameter :: FILE_NAME = "simple_xy.nc"

  ! We are writing 2D data, a 12 x 6 grid.
  integer, parameter :: NDIMS = 2
  integer, parameter :: NX = 6, NY = 12

  ! When we create netCDF files, variables and dimensions, we get back
  ! an ID for each one.
  integer :: ncid, varid, dimids(NDIMS)
```

```
integer :: x_dimid, y_dimid

! This is the data array we will write. It will just be filled with
! a progression of integers for this example.
integer, dimension(:,:), allocatable :: data_out

! Loop indexes, and error handling.
integer :: x, y

! Allocate memory for data.
allocate(data_out(NY, NX))

! Create some pretend data. If this wasn't an example program, we
! would have some real data to write, for example, model output.
do x = 1, NX
   do y = 1, NY
      data_out(y, x) = (x - 1) * NY + (y - 1)
   end do
end do

! Always check the return code of every netCDF function call. In
! this example program, wrapping netCDF calls with "call check()"
! makes sure that any return which is not equal to nf90_noerr (0)
! will print a netCDF error message and exit.

! Create the netCDF file. The nf90_clobber parameter tells netCDF to
! overwrite this file, if it already exists.
call check( nf90_create(FILE_NAME, NF90_CLOBBER, ncid) )

! Define the dimensions. NetCDF will hand back an ID for each.
call check( nf90_def_dim(ncid, "x", NX, x_dimid) )
call check( nf90_def_dim(ncid, "y", NY, y_dimid) )

! The dimids array is used to pass the IDs of the dimensions of
! the variables. Note that in fortran arrays are stored in
! column-major format.
dimids =  (/ y_dimid, x_dimid /)

! Define the variable. The type of the variable in this case is
! NF90_INT (4-byte integer).
call check( nf90_def_var(ncid, "data", NF90_INT, dimids, varid) )

! End define mode. This tells netCDF we are done defining metadata.
call check( nf90_enddef(ncid) )

! Write the pretend data to the file. Although netCDF supports
! reading and writing subsets of data, in this case we write all the
```

```
  ! data in one operation.
  call check( nf90_put_var(ncid, varid, data_out) )

  ! Close the file. This frees up any internal netCDF resources
  ! associated with the file, and flushes any buffers.
  call check( nf90_close(ncid) )

  print *, "*** SUCCESS writing example file simple_xy.nc! "

contains
  subroutine check(status)
    integer, intent ( in) :: status

    if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status))
      stop 2
    end if
  end subroutine check
end program simple_xy_wr
```

2.1.3.2 simple_xy_rd.f90

```
! This is part of the netCDF package.
! Copyright 2006 University Corporation for Atmospheric Research/Unidata.
! See COPYRIGHT file for conditions of use.

! This is a simple example which reads a small dummy array, from a
! netCDF data file created by the companion program simple_xy_wr.f90.

! This is intended to illustrate the use of the netCDF fortran 90
! API. This example program is part of the netCDF tutorial, which can
! be found at:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

! Full documentation of the netCDF Fortran 90 API can be found at:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90

! $Id: simple_xy_rd.f90,v 1.11 2010/04/06 19:32:09 ed Exp $

program simple_xy_rd
  use netcdf
  implicit none

  ! This is the name of the data file we will read.
  character (len = *), parameter :: FILE_NAME = "simple_xy.nc"

  ! We are reading 2D data, a 12 x 6 grid.
```

```
  integer, parameter :: NX = 6, NY = 12
  integer, dimension(:,:), allocatable :: data_in

  ! This will be the netCDF ID for the file and data variable.
  integer :: ncid, varid

  ! Loop indexes, and error handling.
  integer :: x, y

  ! Allocate memory for data.
  allocate(data_in(NY, NX))

  ! Open the file. NF90_NOWRITE tells netCDF we want read-only access to
  ! the file.
  call check( nf90_open(FILE_NAME, NF90_NOWRITE, ncid) )

  ! Get the varid of the data variable, based on its name.
  call check( nf90_inq_varid(ncid, "data", varid) )

  ! Read the data.
  call check( nf90_get_var(ncid, varid, data_in) )

  ! Check the data.
  do x = 1, NX
     do y = 1, NY
        if (data_in(y, x) /= (x - 1) * NY + (y - 1)) then
           print *, "data_in(", y, ", ", x, ") = ", data_in(y, x)
           stop "Stopped"
        end if
     end do
  end do

  ! Close the file, freeing all resources.
  call check( nf90_close(ncid) )

  print *,"*** SUCCESS reading example file ", FILE_NAME, "! "

contains
  subroutine check(status)
    integer, intent ( in) :: status

    if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status))
      stop 2
    end if
  end subroutine check
end program simple_xy_rd
```

2.1.4 simple_xy_wr.cpp and simple_xy_rd.cpp

These example programs can be found in the netCDF distribution, under examples/CXX.

The example program simple_xy_wr.cpp creates the example data file simple_xy.nc. The example program simple_xy_rd.cpp reads the data file.

2.1.4.1 simple_xy_wr.cpp

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.

   This is a very simple example which writes a 2D array of
   sample data. To handle this in netCDF we create two shared
   dimensions, "x" and "y", and a netCDF variable, called "data".

   This example is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

   Full documentation of the netCDF C++ API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-cxx

   $Id: simple_xy_wr.cpp,v 1.15 2007/01/19 12:52:13 ed Exp $
*/

#include <iostream>
#include <netcdfcpp.h>

using namespace std;

// We are writing 2D data, a 6 x 12 grid.
static const int NX = 6;
static const int NY = 12;

// Return this in event of a problem.
static const int NC_ERR = 2;

int
main(void)
{
   // This is the data array we will write. It will just be filled
   // with a progression of numbers for this example.
   int dataOut[NX][NY];

   // Create some pretend data. If this wasn't an example program, we
   // would have some real data to write, for example, model output.
   for(int i = 0; i < NX; i++)
      for(int j = 0; j < NY; j++)
```

```
     dataOut[i][j] = i * NY + j;

   // Create the file. The Replace parameter tells netCDF to overwrite
   // this file, if it already exists.
   NcFile dataFile("simple_xy.nc", NcFile::Replace);

   // You should always check whether a netCDF file creation or open
   // constructor succeeded.
   if (!dataFile.is_valid())
   {
      cout << "Couldn't open file!\n";
      return NC_ERR;
   }

   // For other method calls, the default behavior of the C++ API is
   // to exit with a message if there is an error.  If that behavior
   // is OK, there is no need to check return values in simple cases
   // like the following.

   // When we create netCDF dimensions, we get back a pointer to an
   // NcDim for each one.
   NcDim* xDim = dataFile.add_dim("x", NX);
   NcDim* yDim = dataFile.add_dim("y", NY);

   // Define a netCDF variable. The type of the variable in this case
   // is ncInt (32-bit integer).
   NcVar *data = dataFile.add_var("data", ncInt, xDim, yDim);

   // Write the pretend data to the file. Although netCDF supports
   // reading and writing subsets of data, in this case we write all
   // the data in one operation.
   data->put(&dataOut[0][0], NX, NY);

   // The file will be automatically close when the NcFile object goes
   // out of scope. This frees up any internal netCDF resources
   // associated with the file, and flushes any buffers.
   cout << "*** SUCCESS writing example file simple_xy.nc!" << endl;

   return 0;
}
```

2.1.4.2 simple_xy_rd.cpp

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.
```

```
   This is a very simple example which reads a 2D array of
   sample data produced by simple_xy_wr.cpp.

   This example is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

   Full documentation of the netCDF C++ API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-cxx

   $Id: simple_xy_rd.cpp,v 1.13 2007/01/19 12:52:13 ed Exp $
*/

#include <iostream>
#include <netcdfcpp.h>

using namespace std;

// We are reading 2D data, a 6 x 12 grid.
static const int NX = 6;
static const int NY = 12;

// Return this in event of a problem.
static const int NC_ERR = 2;

int main(void)
{
   // This is the array we will read.
   int dataIn[NX][NY];

   // Open the file. The ReadOnly parameter tells netCDF we want
   // read-only access to the file.
   NcFile dataFile("simple_xy.nc", NcFile::ReadOnly);

   // You should always check whether a netCDF file open or creation
   // constructor succeeded.
   if (!dataFile.is_valid())
   {
      cout << "Couldn't open file!\n";
      return NC_ERR;
   }

   // For other method calls, the default behavior of the C++ API is
   // to exit with a message if there is an error.  If that behavior
   // is OK, there is no need to check return values in simple cases
   // like the following.

   // Retrieve the variable named "data"
```

```
   NcVar *data = dataFile.get_var("data");

   // Read all the values from the "data" variable into memory.
   data->get(&dataIn[0][0], NX, NY);

   // Check the values.
   for (int i = 0; i < NX; i++)
      for (int j = 0; j < NY; j++)
 if (dataIn[i][j] != i * NY + j)
    return NC_ERR;

   // The netCDF file is automatically closed by the NcFile destructor
   cout << "*** SUCCESS reading example file simple_xy.nc!" << endl;

   return 0;
}
```

2.2 The sfc_pres_temp Example

This example has been constructed for the meteorological mind.

Suppose you have some data you want to write to a netCDF file. For example, you have one time step of surface temperature and surface pressure, on a 6 x 12 latitude longitude grid.

To store this in netCDF, create a file, add two dimensions (latitude and longitude) and two variables (pressure and temperature).

In this example we add some netCDF attributes, as is typical in scientific applications, to further describe the data. In this case we add a units attribute to every netCDF variable.

In this example we also add additional netCDF variables to describe the coordinate system. These "coordinate variables" allow us to specify the latitudes and longitudes that describe the data grid.

The CDL version of the data file, generated by ncdump, is shown below.

For more information on ncdump and ncgen see Section "NetCDF Utilities" in *The NetCDF Users Guide*.

```
netcdf sfc_pres_temp {
dimensions:
latitude = 6 ;
longitude = 12 ;
variables:
float latitude(latitude) ;
latitude:units = "degrees_north" ;
float longitude(longitude) ;
longitude:units = "degrees_east" ;
float pressure(latitude, longitude) ;
pressure:units = "hPa" ;
float temperature(latitude, longitude) ;
temperature:units = "celsius" ;
```

```
data:

 latitude = 25, 30, 35, 40, 45, 50 ;

 longitude = -125, -120, -115, -110, -105, -100, -95, -90, -85, -80, -75, -70 ;

 pressure =
  900, 906, 912, 918, 924, 930, 936, 942, 948, 954, 960, 966,
  901, 907, 913, 919, 925, 931, 937, 943, 949, 955, 961, 967,
  902, 908, 914, 920, 926, 932, 938, 944, 950, 956, 962, 968,
  903, 909, 915, 921, 927, 933, 939, 945, 951, 957, 963, 969,
  904, 910, 916, 922, 928, 934, 940, 946, 952, 958, 964, 970,
  905, 911, 917, 923, 929, 935, 941, 947, 953, 959, 965, 971 ;

 temperature =
  9, 10.5, 12, 13.5, 15, 16.5, 18, 19.5, 21, 22.5, 24, 25.5,
  9.25, 10.75, 12.25, 13.75, 15.25, 16.75, 18.25, 19.75, 21.25, 22.75, 24.25,
    25.75,
  9.5, 11, 12.5, 14, 15.5, 17, 18.5, 20, 21.5, 23, 24.5, 26,
  9.75, 11.25, 12.75, 14.25, 15.75, 17.25, 18.75, 20.25, 21.75, 23.25, 24.75,
    26.25,
  10, 11.5, 13, 14.5, 16, 17.5, 19, 20.5, 22, 23.5, 25, 26.5,
  10.25, 11.75, 13.25, 14.75, 16.25, 17.75, 19.25, 20.75, 22.25, 23.75,
    25.25, 26.75 ;
}
```

2.2.1 sfc_pres_temp_wr.c and sfc_pres_temp_rd.c

These example programs can be found in the netCDF distribution, under examples/C.

The example program sfc_pres_temp_wr.c creates the example data file sfc_pres_temp.nc. The example program sfc_pres_temp_rd.c reads the data file.

2.2.1.1 sfc_pres_temp_wr.c

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.

   This example writes some surface pressure and temperatures. It is
   intended to illustrate the use of the netCDF C API. The companion
   program sfc_pres_temp_rd.c shows how to read the netCDF data file
   created by this program.

   This program is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

   Full documentation of the netCDF C API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c
```

```
   $Id: sfc_pres_temp_wr.c,v 1.3 2006/06/13 20:46:16 ed Exp $
*/

#include <stdio.h>
#include <string.h>
#include <netcdf.h>

/* This is the name of the data file we will create. */
#define FILE_NAME "sfc_pres_temp.nc"

/* We are writing 2D data, a 6 x 12 lat-lon grid. We will need two
 * netCDF dimensions. */
#define NDIMS 2
#define NLAT 6
#define NLON 12
#define LAT_NAME "latitude"
#define LON_NAME "longitude"

/* Names of things. */
#define PRES_NAME "pressure"
#define TEMP_NAME "temperature"
#define UNITS "units"
#define DEGREES_EAST "degrees_east"
#define DEGREES_NORTH "degrees_north"

/* These are used to construct some example data. */
#define SAMPLE_PRESSURE 900
#define SAMPLE_TEMP 9.0
#define START_LAT 25.0
#define START_LON -125.0

/* Handle errors by printing an error message and exiting with a
 * non-zero status. */
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); return 2;}

int
main()
{
   int ncid, lon_dimid, lat_dimid, pres_varid, temp_varid;

/* In addition to the latitude and longitude dimensions, we will also
   create latitude and longitude netCDF variables which will hold the
   actual latitudes and longitudes. Since they hold data about the
   coordinate system, the netCDF term for these is: "coordinate
   variables." */
   int lat_varid, lon_varid;
```

```
   int dimids[NDIMS];

   /* We will write surface temperature and pressure fields. */
   float pres_out[NLAT][NLON];
   float temp_out[NLAT][NLON];
   float lats[NLAT], lons[NLON];

   /* It's good practice for each netCDF variable to carry a "units"
    * attribute. */
   char pres_units[] = "hPa";
   char temp_units[] = "celsius";

   /* Loop indexes. */
   int lat, lon;

   /* Error handling. */
   int retval;

   /* Create some pretend data. If this wasn't an example program, we
    * would have some real data to write, for example, model
    * output. */
   for (lat = 0; lat < NLAT; lat++)
      lats[lat] = START_LAT + 5.*lat;
   for (lon = 0; lon < NLON; lon++)
      lons[lon] = START_LON + 5.*lon;

   for (lat = 0; lat < NLAT; lat++)
      for (lon = 0; lon < NLON; lon++)
      {
 pres_out[lat][lon] = SAMPLE_PRESSURE + (lon * NLAT + lat);
 temp_out[lat][lon] = SAMPLE_TEMP + .25 * (lon * NLAT + lat);
      }

   /* Create the file. */
   if ((retval = nc_create(FILE_NAME, NC_CLOBBER, &ncid)))
      ERR(retval);

   /* Define the dimensions. */
   if ((retval = nc_def_dim(ncid, LAT_NAME, NLAT, &lat_dimid)))
      ERR(retval);
   if ((retval = nc_def_dim(ncid, LON_NAME, NLON, &lon_dimid)))
      ERR(retval);

   /* Define coordinate netCDF variables. They will hold the
      coordinate information, that is, the latitudes and longitudes. A
      varid is returned for each.*/
```

```
  if ((retval = nc_def_var(ncid, LAT_NAME, NC_FLOAT, 1, &lat_dimid,
   &lat_varid)))
     ERR(retval);
  if ((retval = nc_def_var(ncid, LON_NAME, NC_FLOAT, 1, &lon_dimid,
   &lon_varid)))
     ERR(retval);

  /* Define units attributes for coordinate vars. This attaches a
     text attribute to each of the coordinate variables, containing
     the units. Note that we are not writing a trailing NULL, just
     "units", because the reading program may be fortran which does
     not use null-terminated strings. In general it is up to the
     reading C program to ensure that it puts null-terminators on
     strings where necessary.*/
  if ((retval = nc_put_att_text(ncid, lat_varid, UNITS,
strlen(DEGREES_NORTH), DEGREES_NORTH)))
     ERR(retval);
  if ((retval = nc_put_att_text(ncid, lon_varid, UNITS,
strlen(DEGREES_EAST), DEGREES_EAST)))
     ERR(retval);

  /* Define the netCDF variables. The dimids array is used to pass
     the dimids of the dimensions of the variables.*/
  dimids[0] = lat_dimid;
  dimids[1] = lon_dimid;
  if ((retval = nc_def_var(ncid, PRES_NAME, NC_FLOAT, NDIMS,
   dimids, &pres_varid)))
     ERR(retval);
  if ((retval = nc_def_var(ncid, TEMP_NAME, NC_FLOAT, NDIMS,
   dimids, &temp_varid)))
     ERR(retval);

  /* Define units attributes for vars. */
  if ((retval = nc_put_att_text(ncid, pres_varid, UNITS,
strlen(pres_units), pres_units)))
     ERR(retval);
  if ((retval = nc_put_att_text(ncid, temp_varid, UNITS,
strlen(temp_units), temp_units)))
     ERR(retval);

  /* End define mode. */
  if ((retval = nc_enddef(ncid)))
     ERR(retval);

  /* Write the coordinate variable data. This will put the latitudes
     and longitudes of our data grid into the netCDF file. */
  if ((retval = nc_put_var_float(ncid, lat_varid, &lats[0])))
```

```
      ERR(retval);
   if ((retval = nc_put_var_float(ncid, lon_varid, &lons[0])))
      ERR(retval);

   /* Write the pretend data. This will write our surface pressure and
      surface temperature data. The arrays of data are the same size
      as the netCDF variables we have defined. */
   if ((retval = nc_put_var_float(ncid, pres_varid, &pres_out[0][0])))
      ERR(retval);
   if ((retval = nc_put_var_float(ncid, temp_varid, &temp_out[0][0])))
      ERR(retval);

   /* Close the file. */
   if ((retval = nc_close(ncid)))
      ERR(retval);

   printf("*** SUCCESS writing example file sfc_pres_temp.nc!\n");
   return 0;
}
```

2.2.1.2 sfc_pres_temp_rd.c

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.

   This is an example which reads some surface pressure and
   temperatures. The data file read by this program is produced by the
   companion program sfc_pres_temp_wr.c. It is intended to illustrate
   the use of the netCDF C API.

   This program is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

   Full documentation of the netCDF C API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c

   $Id: sfc_pres_temp_rd.c,v 1.5 2007/02/14 20:59:21 ed Exp $
*/

#include <stdio.h>
#include <string.h>
#include <netcdf.h>

/* This is the name of the data file we will read. */
#define FILE_NAME "sfc_pres_temp.nc"
```

```
/* We are reading 2D data, a 6 x 12 lat-lon grid. */
#define NDIMS 2
#define NLAT 6
#define NLON 12

#define LAT_NAME "latitude"
#define LON_NAME "longitude"
#define PRES_NAME "pressure"
#define TEMP_NAME "temperature"

/* These are used to calculate the values we expect to find. */
#define SAMPLE_PRESSURE 900
#define SAMPLE_TEMP 9.0
#define START_LAT 25.0
#define START_LON -125.0

/* For the units attributes. */
#define UNITS "units"
#define PRES_UNITS "hPa"
#define TEMP_UNITS "celsius"
#define LAT_UNITS "degrees_north"
#define LON_UNITS "degrees_east"
#define MAX_ATT_LEN 80

/* Handle errors by printing an error message and exiting with a
 * non-zero status. */
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); return 2;}

int
main()
{
   int ncid, pres_varid, temp_varid;
   int lat_varid, lon_varid;

   /* We will read surface temperature and pressure fields. */
   float pres_in[NLAT][NLON];
   float temp_in[NLAT][NLON];

   /* For the lat lon coordinate variables. */
   float lats_in[NLAT], lons_in[NLON];

   /* To check the units attributes. */
   char pres_units_in[MAX_ATT_LEN], temp_units_in[MAX_ATT_LEN];
   char lat_units_in[MAX_ATT_LEN], lon_units_in[MAX_ATT_LEN];

   /* We will learn about the data file and store results in these
      program variables. */
```

```
   int ndims_in, nvars_in, ngatts_in, unlimdimid_in;

   /* Loop indexes. */
   int lat, lon;

   /* Error handling. */
   int retval;

   /* Open the file. */
   if ((retval = nc_open(FILE_NAME, NC_NOWRITE, &ncid)))
      ERR(retval);

   /* There are a number of inquiry functions in netCDF which can be
      used to learn about an unknown netCDF file. NC_INQ tells how
      many netCDF variables, dimensions, and global attributes are in
      the file; also the dimension id of the unlimited dimension, if
      there is one. */
   if ((retval = nc_inq(ncid, &ndims_in, &nvars_in, &ngatts_in,
&unlimdimid_in)))
      ERR(retval);

   /* In this case we know that there are 2 netCDF dimensions, 4
      netCDF variables, no global attributes, and no unlimited
      dimension. */
   if (ndims_in != 2 || nvars_in != 4 || ngatts_in != 0 ||
       unlimdimid_in != -1) return 2;

   /* Get the varids of the latitude and longitude coordinate
    * variables. */
   if ((retval = nc_inq_varid(ncid, LAT_NAME, &lat_varid)))
      ERR(retval);
   if ((retval = nc_inq_varid(ncid, LON_NAME, &lon_varid)))
      ERR(retval);

   /* Read the coordinate variable data. */
   if ((retval = nc_get_var_float(ncid, lat_varid, &lats_in[0])))
      ERR(retval);
   if ((retval = nc_get_var_float(ncid, lon_varid, &lons_in[0])))
      ERR(retval);

   /* Check the coordinate variable data. */
   for (lat = 0; lat < NLAT; lat++)
      if (lats_in[lat] != START_LAT + 5.*lat)
 return 2;
   for (lon = 0; lon < NLON; lon++)
      if (lons_in[lon] != START_LON + 5.*lon)
 return 2;
```

```
   /* Get the varids of the pressure and temperature netCDF
    * variables. */
   if ((retval = nc_inq_varid(ncid, PRES_NAME, &pres_varid)))
      ERR(retval);
   if ((retval = nc_inq_varid(ncid, TEMP_NAME, &temp_varid)))
      ERR(retval);

   /* Read the data. Since we know the contents of the file we know
    * that the data arrays in this program are the correct size to
    * hold all the data. */
   if ((retval = nc_get_var_float(ncid, pres_varid, &pres_in[0][0])))
      ERR(retval);
   if ((retval = nc_get_var_float(ncid, temp_varid, &temp_in[0][0])))
      ERR(retval);

   /* Check the data. */
   for (lat = 0; lat < NLAT; lat++)
      for (lon = 0; lon < NLON; lon++)
 if (pres_in[lat][lon] != SAMPLE_PRESSURE + (lon * NLAT + lat) ||
     temp_in[lat][lon] != SAMPLE_TEMP + .25 * (lon * NLAT + lat))
    return 2;

   /* Each of the netCDF variables has a "units" attribute. Let's read
      them and check them. */
   if ((retval = nc_get_att_text(ncid, lat_varid, UNITS, lat_units_in)))
      ERR(retval);
   if (strncmp(lat_units_in, LAT_UNITS, strlen(LAT_UNITS)))
      return 2;

   if ((retval = nc_get_att_text(ncid, lon_varid, UNITS, lon_units_in)))
      ERR(retval);
   if (strncmp(lon_units_in, LON_UNITS, strlen(LON_UNITS)))
      return 2;

   if ((retval = nc_get_att_text(ncid, pres_varid, UNITS, pres_units_in)))
      ERR(retval);
   if (strncmp(pres_units_in, PRES_UNITS, strlen(PRES_UNITS)))
      return 2;

   if ((retval = nc_get_att_text(ncid, temp_varid, UNITS, temp_units_in)))
      ERR(retval);
   if (strncmp(temp_units_in, TEMP_UNITS, strlen(TEMP_UNITS))) return 2;

   /* Close the file. */
   if ((retval = nc_close(ncid)))
      ERR(retval);
```

```
   printf("*** SUCCESS reading example file sfc_pres_temp.nc!\n");
   return 0;
}
```

2.2.2 sfc_pres_temp_wr.f and sfc_pres_temp_rd.f

These example programs can be found in the netCDF distribution, under examples/F77.

The example program sfc_pres_temp_wr.f creates the example data file sfc_pres_temp.nc. The example program sfc_pres_temp_rd.f reads the data file.

2.2.2.1 sfc_pres_temp_wr.f

```
C     This is part of the netCDF package.
C     Copyright 2006 University Corporation for Atmospheric Research/Unidata.
C     See COPYRIGHT file for conditions of use.

C     This example writes some surface pressure and temperatures. It is
C     intended to illustrate the use of the netCDF fortran 77 API. The
C     companion program sfc_pres_temp_rd.f shows how to read the netCDF
C     data file created by this program.

C     This program is part of the netCDF tutorial:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

C     Full documentation of the netCDF Fortran 77 API can be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f77

C     $Id: sfc_pres_temp_wr.f,v 1.10 2007/02/14 20:59:20 ed Exp $

      program sfc_pres_temp_wr
      implicit none
      include 'netcdf.inc'

C     This is the name of the data file we will create.
      character*(*) FILE_NAME
      parameter (FILE_NAME='sfc_pres_temp.nc')
      integer ncid

C     We are writing 2D data, a 12 x 6 lon-lat grid. We will need two
C     netCDF dimensions.
      integer NDIMS
      parameter (NDIMS=2)
      integer NLATS, NLONS
      parameter (NLATS = 6, NLONS = 12)
      character*(*) LAT_NAME, LON_NAME
      parameter (LAT_NAME='latitude', LON_NAME='longitude')
      integer lon_dimid, lat_dimid
```

```
C     In addition to the latitude and longitude dimensions, we will also
C     create latitude and longitude netCDF variables which will hold the
C     actual latitudes and longitudes. Since they hold data about the
C     coordinate system, the netCDF term for these is: "coordinate
C     variables."
      real lats(NLATS), lons(NLONS)
      integer lat_varid, lon_varid
      real START_LAT, START_LON
      parameter (START_LAT = 25.0, START_LON = -125.0)

C     We will write surface temperature and pressure fields.
      character*(*) PRES_NAME, TEMP_NAME
      parameter (PRES_NAME='pressure')
      parameter (TEMP_NAME='temperature')
      integer pres_varid, temp_varid
      integer dimids(NDIMS)

C     It's good practice for each variable to carry a "units" attribute.
      character*(*) UNITS
      parameter (UNITS = 'units')
      character*(*) PRES_UNITS, TEMP_UNITS, LAT_UNITS, LON_UNITS
      parameter (PRES_UNITS = 'hPa', TEMP_UNITS = 'celsius')
      parameter (LAT_UNITS = 'degrees_north')
      parameter (LON_UNITS = 'degrees_east')

C     We will create some pressure and temperature data to write out.
      real pres_out(NLONS, NLATS), temp_out(NLONS, NLATS)
      real SAMPLE_PRESSURE
      parameter (SAMPLE_PRESSURE = 900.0)
      real SAMPLE_TEMP
      parameter (SAMPLE_TEMP = 9.0)

C     Loop indices.
      integer lat, lon

C     Error handling.
      integer retval

C     Create pretend data. If this were not an example program, we would
C     have some real data to write, for example, model output.
      do lat = 1, NLATS
         lats(lat) = START_LAT + (lat - 1) * 5.0
      end do
      do lon = 1, NLONS
         lons(lon) = START_LON + (lon - 1) * 5.0
      end do
```

```
      do lon = 1, NLONS
         do lat = 1, NLATS
            pres_out(lon, lat) = SAMPLE_PRESSURE +
     +           (lon - 1) * NLATS + (lat - 1)
            temp_out(lon, lat) = SAMPLE_TEMP +
     +           .25 * ((lon - 1) * NLATS + (lat - 1))
         end do
      end do

C     Create the file.
      retval = nf_create(FILE_NAME, nf_clobber, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the dimensions.
      retval = nf_def_dim(ncid, LAT_NAME, NLATS, lat_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_dim(ncid, LON_NAME, NLONS, lon_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the coordinate variables. They will hold the coordinate
C     information, that is, the latitudes and longitudes. A varid is
C     returned for each.
      retval = nf_def_var(ncid, LAT_NAME, NF_REAL, 1, lat_dimid,
     +     lat_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_var(ncid, LON_NAME, NF_REAL, 1, lon_dimid,
     +     lon_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Assign units attributes to coordinate var data. This attaches a
C     text attribute to each of the coordinate variables, containing the
C     units.
      retval = nf_put_att_text(ncid, lat_varid, UNITS, len(LAT_UNITS),
     +     LAT_UNITS)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_put_att_text(ncid, lon_varid, UNITS, len(LON_UNITS),
     +     LON_UNITS)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the netCDF variables. The dimids array is used to pass the
C     dimids of the dimensions of the netCDF variables.
      dimids(1) = lon_dimid
      dimids(2) = lat_dimid
      retval = nf_def_var(ncid, PRES_NAME, NF_REAL, NDIMS, dimids,
     +     pres_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_var(ncid, TEMP_NAME, NF_REAL, NDIMS, dimids,
```

```
     +     temp_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Assign units attributes to the pressure and temperature netCDF
C     variables.
      retval = nf_put_att_text(ncid, pres_varid, UNITS, len(PRES_UNITS),
     +     PRES_UNITS)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_put_att_text(ncid, temp_varid, UNITS, len(TEMP_UNITS),
     +     TEMP_UNITS)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     End define mode.
      retval = nf_enddef(ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Write the coordinate variable data. This will put the latitudes
C     and longitudes of our data grid into the netCDF file.
      retval = nf_put_var_real(ncid, lat_varid, lats)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_put_var_real(ncid, lon_varid, lons)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Write the pretend data. This will write our surface pressure and
C     surface temperature data. The arrays of data are the same size as
C     the netCDF variables we have defined.
      retval = nf_put_var_real(ncid, pres_varid, pres_out)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_put_var_real(ncid, temp_varid, temp_out)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Close the file.
      retval = nf_close(ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     If we got this far, everything worked as expected. Yipee!
      print *,'*** SUCCESS writing example file sfc_pres_temp.nc!'
      end

      subroutine handle_err(errcode)
      implicit none
      include 'netcdf.inc'
      integer errcode

      print *, 'Error: ', nf_strerror(errcode)
      stop 2
      end
```

2.2.2.2 sfc_pres_temp_rd.f

```
C     This is part of the netCDF package.
C     Copyright 2006 University Corporation for Atmospheric Research/Unidata.
C     See COPYRIGHT file for conditions of use.

C     This is an example which reads some surface pressure and
C     temperatures. The data file read by this program is produced
C     comapnion program sfc_pres_temp_wr.f. It is intended to illustrate
C     the use of the netCDF fortran 77 API.

C     This program is part of the netCDF tutorial:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

C     Full documentation of the netCDF Fortran 77 API can be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f77

C     $Id: sfc_pres_temp_rd.f,v 1.9 2007/02/14 20:59:20 ed Exp $

      program sfc_pres_temp_rd
      implicit none
      include 'netcdf.inc'

C     This is the name of the data file we will read.
      character*(*) FILE_NAME
      parameter (FILE_NAME='sfc_pres_temp.nc')
      integer ncid

C     We are reading 2D data, a 12 x 6 lon-lat grid.
      integer NDIMS
      parameter (NDIMS=2)
      integer NLATS, NLONS
      parameter (NLATS = 6, NLONS = 12)
      character*(*) LAT_NAME, LON_NAME
      parameter (LAT_NAME='latitude', LON_NAME='longitude')
      integer lat_dimid, lon_dimid

C     For the lat lon coordinate netCDF variables.
      real lats(NLATS), lons(NLONS)
      integer lat_varid, lon_varid

C     We will read surface temperature and pressure fields.
      character*(*) PRES_NAME, TEMP_NAME
      parameter (PRES_NAME='pressure')
      parameter (TEMP_NAME='temperature')
      integer pres_varid, temp_varid
      integer dimids(NDIMS)
```

```
C     To check the units attributes.
      character*(*) UNITS
      parameter (UNITS = 'units')
      character*(*) PRES_UNITS, TEMP_UNITS, LAT_UNITS, LON_UNITS
      parameter (PRES_UNITS = 'hPa', TEMP_UNITS = 'celsius')
      parameter (LAT_UNITS = 'degrees_north')
      parameter (LON_UNITS = 'degrees_east')
      integer MAX_ATT_LEN
      parameter (MAX_ATT_LEN = 80)
      character*(MAX_ATT_LEN) pres_units_in, temp_units_in
      character*(MAX_ATT_LEN) lat_units_in, lon_units_in
      integer att_len

C     Read the data into these arrays.
      real pres_in(NLONS, NLATS), temp_in(NLONS, NLATS)

C     These are used to calculate the values we expect to find.
      real START_LAT, START_LON
      parameter (START_LAT = 25.0, START_LON = -125.0)
      real SAMPLE_PRESSURE
      parameter (SAMPLE_PRESSURE = 900.0)
      real SAMPLE_TEMP
      parameter (SAMPLE_TEMP = 9.0)

C     We will learn about the data file and store results in these
C     program variables.
      integer ndims_in, nvars_in, ngatts_in, unlimdimid_in

C     Loop indices
      integer lat, lon

C     Error handling
      integer retval

C     Open the file.
      retval = nf_open(FILE_NAME, nf_nowrite, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     There are a number of inquiry functions in netCDF which can be
C     used to learn about an unknown netCDF file. NF_INQ tells how many
C     netCDF variables, dimensions, and global attributes are in the
C     file; also the dimension id of the unlimited dimension, if there
C     is one.
      retval = nf_inq(ncid, ndims_in, nvars_in, ngatts_in,
     +     unlimdimid_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

```
C     In this case we know that there are 2 netCDF dimensions, 4 netCDF
C     variables, no global attributes, and no unlimited dimension.
      if (ndims_in .ne. 2 .or. nvars_in .ne. 4 .or. ngatts_in .ne. 0
     +     .or. unlimdimid_in .ne. -1) stop 2

C     Get the varids of the latitude and longitude coordinate variables.
      retval = nf_inq_varid(ncid, LAT_NAME, lat_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_inq_varid(ncid, LON_NAME, lon_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Read the latitude and longitude data.
      retval = nf_get_var_real(ncid, lat_varid, lats)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_get_var_real(ncid, lon_varid, lons)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Check to make sure we got what we expected.
      do lat = 1, NLATS
         if (lats(lat) .ne. START_LAT + (lat - 1) * 5.0) stop 2
      end do
      do lon = 1, NLONS
         if (lons(lon) .ne. START_LON + (lon - 1) * 5.0) stop 2
      end do

C     Get the varids of the pressure and temperature netCDF variables.
      retval = nf_inq_varid(ncid, PRES_NAME, pres_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_inq_varid(ncid, TEMP_NAME, temp_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Read the surface pressure and temperature data from the file.
C     Since we know the contents of the file we know that the data
C     arrays in this program are the correct size to hold all the data.
      retval = nf_get_var_real(ncid, pres_varid, pres_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_get_var_real(ncid, temp_varid, temp_in)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Check the data. It should be the same as the data we wrote.
      do lon = 1, NLONS
         do lat = 1, NLATS
             if (pres_in(lon, lat) .ne. SAMPLE_PRESSURE +
     +           (lon - 1) * NLATS + (lat - 1)) stop 2
             if (temp_in(lon, lat) .ne. SAMPLE_TEMP +
     +           .25 * ((lon - 1) * NLATS + (lat - 1))) stop 2
```

```
         end do
      end do

C     Each of the netCDF variables has a "units" attribute. Let's read
C     them and check them.

      retval = nf_get_att_text(ncid, lat_varid, UNITS, lat_units_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_inq_attlen(ncid, lat_varid, UNITS, att_len)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (lat_units_in(1:att_len) .ne. LAT_UNITS) stop 2

      retval = nf_get_att_text(ncid, lon_varid, UNITS, lon_units_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_inq_attlen(ncid, lon_varid, UNITS, att_len)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (lon_units_in(1:att_len) .ne. LON_UNITS) stop 2

      retval = nf_get_att_text(ncid, pres_varid, UNITS, pres_units_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_inq_attlen(ncid, pres_varid, UNITS, att_len)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (pres_units_in(1:att_len) .ne. PRES_UNITS) stop 2

      retval = nf_get_att_text(ncid, temp_varid, UNITS, temp_units_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_inq_attlen(ncid, temp_varid, UNITS, att_len)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (temp_units_in(1:att_len) .ne. TEMP_UNITS) stop 2

C     Close the file. This frees up any internal netCDF resources
C     associated with the file.
      retval = nf_close(ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     If we got this far, everything worked as expected. Yipee!
      print *,'*** SUCCESS reading example file sfc_pres_temp.nc!'
      end

      subroutine handle_err(errcode)
      implicit none
      include 'netcdf.inc'
      integer errcode

      print *, 'Error: ', nf_strerror(errcode)
      stop 2
      end
```

2.2.3 sfc_pres_temp_wr.f90 and sfc_pres_temp_rd.f90

These example programs can be found in the netCDF distribution, under examples/F90.

The example program sfc_pres_temp_wr.f90 creates the example data file sfc_pres_temp.nc. The example program sfc_pres_temp_rd.f90 reads the data file.

2.2.3.1 sfc_pres_temp_wr.f90

```
! This is part of the netCDF package.
! Copyright 2006 University Corporation for Atmospheric Research/Unidata.
! See COPYRIGHT file for conditions of use.

! This example writes some surface pressure and temperatures. It is
! intended to illustrate the use of the netCDF fortran 90 API. The
! companion program sfc_pres_temp_rd.f90 shows how to read the netCDF
! data file created by this program.

! This program is part of the netCDF tutorial:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

! Full documentation of the netCDF Fortran 90 API can be found at:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90

! $Id: sfc_pres_temp_wr.f90,v 1.12 2010/04/06 19:32:09 ed Exp $

program sfc_pres_temp_wr
  use netcdf
  implicit none

  ! This is the name of the data file we will create.
  character (len = *), parameter :: FILE_NAME = "sfc_pres_temp.nc"
  integer :: ncid

  ! We are writing 2D data, a 12 x 6 lon-lat grid. We will need two
  ! netCDF dimensions.
  integer, parameter :: NDIMS = 2
  integer, parameter :: NLATS = 6, NLONS = 12
  character (len = *), parameter :: LAT_NAME = "latitude"
  character (len = *), parameter :: LON_NAME = "longitude"
  integer :: lat_dimid, lon_dimid

  ! In addition to the latitude and longitude dimensions, we will also
  ! create latitude and longitude netCDF variables which will hold the
  ! actual latitudes and longitudes. Since they hold data about the
  ! coordinate system, the netCDF term for these is: "coordinate
  ! variables."
  real :: lats(NLATS), lons(NLONS)
  integer :: lat_varid, lon_varid
```

```
real, parameter :: START_LAT = 25.0, START_LON = -125.0

! We will write surface temperature and pressure fields.
character (len = *), parameter :: PRES_NAME="pressure"
character (len = *), parameter :: TEMP_NAME="temperature"
integer :: pres_varid, temp_varid
integer :: dimids(NDIMS)

! It's good practice for each variable to carry a "units" attribute.
character (len = *), parameter :: UNITS = "units"
character (len = *), parameter :: PRES_UNITS = "hPa"
character (len = *), parameter :: TEMP_UNITS = "celsius"
character (len = *), parameter :: LAT_UNITS = "degrees_north"
character (len = *), parameter :: LON_UNITS = "degrees_east"

! We will create some pressure and temperature data to write out.
real, dimension(:,:), allocatable :: temp_out
real, dimension(:,:), allocatable :: pres_out
real, parameter :: SAMPLE_PRESSURE = 900.0
real, parameter :: SAMPLE_TEMP = 9.0

! Loop indices
integer :: lat, lon

! Allocate memory.
allocate(pres_out(NLONS, NLATS))
allocate(temp_out(NLONS, NLATS))

! Create pretend data. If this were not an example program, we would
! have some real data to write, for example, model output.
do lat = 1, NLATS
   lats(lat) = START_LAT + (lat - 1) * 5.0
end do
do lon = 1, NLONS
   lons(lon) = START_LON + (lon - 1) * 5.0
end do
do lon = 1, NLONS
   do lat = 1, NLATS
      pres_out(lon, lat) = SAMPLE_PRESSURE + (lon - 1) * NLATS + (lat - 1)
      temp_out(lon, lat) = SAMPLE_TEMP + .25 * ((lon - 1) * NLATS + (lat - 1))
   end do
end do

! Create the file.
call check( nf90_create(FILE_NAME, nf90_clobber, ncid) )

! Define the dimensions.
```

```
  call check( nf90_def_dim(ncid, LAT_NAME, NLATS, lat_dimid) )
  call check( nf90_def_dim(ncid, LON_NAME, NLONS, lon_dimid) )

  ! Define the coordinate variables. They will hold the coordinate
  ! information, that is, the latitudes and longitudes. A varid is
  ! returned for each.
  call check( nf90_def_var(ncid, LAT_NAME, NF90_REAL, lat_dimid, lat_varid) )
  call check( nf90_def_var(ncid, LON_NAME, NF90_REAL, lon_dimid, lon_varid) )

  ! Assign units attributes to coordinate var data. This attaches a
  ! text attribute to each of the coordinate variables, containing the
  ! units.
  call check( nf90_put_att(ncid, lat_varid, UNITS, LAT_UNITS) )
  call check( nf90_put_att(ncid, lon_varid, UNITS, LON_UNITS) )

  ! Define the netCDF variables. The dimids array is used to pass the
  ! dimids of the dimensions of the netCDF variables.
  dimids = (/ lon_dimid, lat_dimid /)
  call check( nf90_def_var(ncid, PRES_NAME, NF90_REAL, dimids, pres_varid) )
  call check( nf90_def_var(ncid, TEMP_NAME, NF90_REAL, dimids, temp_varid) )

  ! Assign units attributes to the pressure and temperature netCDF
  ! variables.
  call check( nf90_put_att(ncid, pres_varid, UNITS, PRES_UNITS) )
  call check( nf90_put_att(ncid, temp_varid, UNITS, TEMP_UNITS) )

  ! End define mode.
  call check( nf90_enddef(ncid) )

  ! Write the coordinate variable data. This will put the latitudes
  ! and longitudes of our data grid into the netCDF file.
  call check( nf90_put_var(ncid, lat_varid, lats) )
  call check( nf90_put_var(ncid, lon_varid, lons) )

  ! Write the pretend data. This will write our surface pressure and
  ! surface temperature data. The arrays of data are the same size as
  ! the netCDF variables we have defined.
  call check( nf90_put_var(ncid, pres_varid, pres_out) )
  call check( nf90_put_var(ncid, temp_varid, temp_out) )

  ! Close the file.
  call check( nf90_close(ncid) )

  ! If we got this far, everything worked as expected. Yipee!
  print *,"*** SUCCESS writing example file sfc_pres_temp.nc!"

contains
```

```
  subroutine check(status)
    integer, intent ( in) :: status

    if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status))
      stop 2
    end if
  end subroutine check
end program sfc_pres_temp_wr
```

2.2.3.2 sfc_pres_temp_rd.f90

```
! This is part of the netCDF package.
! Copyright 2006 University Corporation for Atmospheric Research/Unidata.
! See COPYRIGHT file for conditions of use.

! This is an example which reads some surface pressure and
! temperatures. The data file read by this program is produced
! comapnion program sfc_pres_temp_wr.f90. It is intended to illustrate
! the use of the netCDF fortran 90 API.

! This program is part of the netCDF tutorial:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

! Full documentation of the netCDF Fortran 90 API can be found at:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90

! $Id: sfc_pres_temp_rd.f90,v 1.10 2010/04/06 19:32:09 ed Exp $

program sfc_pres_temp_rd
  use netcdf
  implicit none

  ! This is the name of the data file we will read.
  character (len = *), parameter :: FILE_NAME = "sfc_pres_temp.nc"
  integer :: ncid

  ! We are reading 2D data, a 12 x 6 lon-lat grid.
  integer, parameter :: NDIMS = 2
  integer, parameter :: NLATS = 6, NLONS = 12
  character (len = *), parameter :: LAT_NAME = "latitude"
  character (len = *), parameter :: LON_NAME = "longitude"

  ! For the lat lon coordinate netCDF variables.
  real :: lats(NLATS), lons(NLONS)
  integer :: lat_varid, lon_varid
```

```
! We will read surface temperature and pressure fields.
character (len = *), parameter :: PRES_NAME = "pressure"
character (len = *), parameter :: TEMP_NAME = "temperature"
integer :: pres_varid, temp_varid

! To check the units attributes.
character (len = *), parameter :: UNITS = "units"
character (len = *), parameter :: PRES_UNITS = "hPa"
character (len = *), parameter :: TEMP_UNITS = "celsius"
character (len = *), parameter :: LAT_UNITS = "degrees_north"
character (len = *), parameter :: LON_UNITS = "degrees_east"
integer, parameter :: MAX_ATT_LEN = 80
integer :: att_len
character*(MAX_ATT_LEN) :: pres_units_in, temp_units_in
character*(MAX_ATT_LEN) :: lat_units_in, lon_units_in

! Read the data into these arrays.
real, dimension(:,:), allocatable :: pres_in
real, dimension(:,:), allocatable :: temp_in

! These are used to calculate the values we expect to find.
real, parameter :: START_LAT = 25.0, START_LON = -125.0
real, parameter :: SAMPLE_PRESSURE = 900.0
real, parameter :: SAMPLE_TEMP = 9.0

! We will learn about the data file and store results in these
! program variables.
integer :: ndims_in, nvars_in, ngatts_in, unlimdimid_in

! Loop indices
integer :: lat, lon

! Allocate memory.
allocate(pres_in(NLONS, NLATS))
allocate(temp_in(NLONS, NLATS))

! Open the file.
call check( nf90_open(FILE_NAME, nf90_nowrite, ncid) )

! There are a number of inquiry functions in netCDF which can be
! used to learn about an unknown netCDF file. NF90_INQ tells how many
! netCDF variables, dimensions, and global attributes are in the
! file; also the dimension id of the unlimited dimension, if there
! is one.
call check( nf90_inquire(ncid, ndims_in, nvars_in, ngatts_in, unlimdimid_in) )
```

```
! In this case we know that there are 2 netCDF dimensions, 4 netCDF
! variables, no global attributes, and no unlimited dimension.
if (ndims_in /= 2 .or. nvars_in /= 4 .or. ngatts_in /= 0 &
     .or. unlimdimid_in /= -1) stop 2

! Get the varids of the latitude and longitude coordinate variables.
call check( nf90_inq_varid(ncid, LAT_NAME, lat_varid) )
call check( nf90_inq_varid(ncid, LON_NAME, lon_varid) )

! Read the latitude and longitude data.
call check( nf90_get_var(ncid, lat_varid, lats) )
call check( nf90_get_var(ncid, lon_varid, lons) )

! Check to make sure we got what we expected.
do lat = 1, NLATS
   if (lats(lat) /= START_LAT + (lat - 1) * 5.0) stop 2
end do
do lon = 1, NLONS
   if (lons(lon) /= START_LON + (lon - 1) * 5.0) stop 2
end do

! Get the varids of the pressure and temperature netCDF variables.
call check( nf90_inq_varid(ncid, PRES_NAME, pres_varid) )
call check( nf90_inq_varid(ncid, TEMP_NAME, temp_varid) )

! Read the surface pressure and temperature data from the file.
! Since we know the contents of the file we know that the data
! arrays in this program are the correct size to hold all the data.
call check( nf90_get_var(ncid, pres_varid, pres_in) )
call check( nf90_get_var(ncid, temp_varid, temp_in) )

! Check the data. It should be the same as the data we wrote.
do lon = 1, NLONS
   do lat = 1, NLATS
      if (pres_in(lon, lat) /= SAMPLE_PRESSURE + &
           (lon - 1) * NLATS + (lat - 1)) stop 2
      if (temp_in(lon, lat) /= SAMPLE_TEMP + &
           .25 * ((lon - 1) * NLATS + (lat - 1))) stop 2
   end do
end do

! Each of the netCDF variables has a "units" attribute. Let's read
! them and check them.
call check( nf90_get_att(ncid, lat_varid, UNITS, lat_units_in) )
call check( nf90_inquire_attribute(ncid, lat_varid, UNITS, len = att_len) )
if (lat_units_in(1:att_len) /= LAT_UNITS) stop 2
```

```
  call check( nf90_get_att(ncid, lon_varid, UNITS, lon_units_in) )
  call check( nf90_inquire_attribute(ncid, lon_varid, UNITS, len = att_len) )
  if (lon_units_in(1:att_len) /= LON_UNITS) stop 2

  call check( nf90_get_att(ncid, pres_varid, UNITS, pres_units_in) )
  call check( nf90_inquire_attribute(ncid, pres_varid, UNITS, len = att_len) )
  if (pres_units_in(1:att_len) /= PRES_UNITS) stop 2

  call check( nf90_get_att(ncid, temp_varid, UNITS, temp_units_in) )
  call check( nf90_inquire_attribute(ncid, temp_varid, UNITS, len = att_len) )
  if (temp_units_in(1:att_len) /= TEMP_UNITS) stop 2

  ! Close the file. This frees up any internal netCDF resources
  ! associated with the file.
  call check( nf90_close(ncid) )

  ! If we got this far, everything worked as expected. Yipee!
  print *,"*** SUCCESS reading example file sfc_pres_temp.nc!"

contains
  subroutine check(status)
    integer, intent ( in) :: status

    if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status))
      stop 2
    end if
  end subroutine check
end program sfc_pres_temp_rd
```

2.2.4 sfc_pres_temp_wr.cpp and sfc_pres_temp_rd.cpp

These example programs can be found in the netCDF distribution, under examples/CXX.

The example program sfc_pres_temp_wr.cpp creates the example data file sfc_pres_temp.nc. The example program sfc_pres_temp_rd.cpp reads the data file.

2.2.4.1 sfc_pres_temp_wr.cpp

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.

   This example writes some surface pressure and temperatures. It is
   intended to illustrate the use of the netCDF C++ API. The companion
   program sfc_pres_temp_rd.cpp shows how to read the netCDF data file
   created by this program.
```

```
   This program is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

   Full documentation of the netCDF C++ API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-cxx

   $Id: sfc_pres_temp_wr.cpp,v 1.12 2007/01/19 12:52:13 ed Exp $
*/

#include <iostream>
#include <netcdfcpp.h>

using namespace std;

// We are writing 2D data, a 6 x 12 lat-lon grid. We will need two
// netCDF dimensions.
static const int NLAT = 6;
static const int NLON = 12;

// These are used to construct some example data.
static const float SAMPLE_PRESSURE = 900;
static const float SAMPLE_TEMP = 9.0;
static const float START_LAT = 25.0;
static const float START_LON = -125.0;

// Return this to OS if there is a failure.
static const int NC_ERR = 2;

int main(void)
{
   // These will hold our pressure and temperature data.
   float presOut[NLAT][NLON];
   float tempOut[NLAT][NLON];

   // These will hold our latitudes and longitudes.
   float lats[NLAT];
   float lons[NLON];

   // Create some pretend data. If this wasn't an example program, we
   // would have some real data to write, for example, model
   // output.
   for(int lat = 0; lat < NLAT; lat++)
      lats[lat] = START_LAT + 5. * lat;

   for(int lon = 0; lon < NLON; lon++)
      lons[lon] = START_LON + 5. * lon;
```

```
  for (int lat = 0; lat < NLAT; lat++)
     for(int lon = 0; lon < NLON; lon++)
     {
presOut[lat][lon] = SAMPLE_PRESSURE + (lon * NLAT + lat);
tempOut[lat][lon] = SAMPLE_TEMP + .25 * (lon * NLAT + lat);
     }

  // Change the error behavior of the netCDF C++ API by creating an
  // NcError object. Until it is destroyed, this NcError object will
  // ensure that the netCDF C++ API silently returns error codes
  // on any failure, and leaves any other error handling to the
  // calling program. In the case of this example, we just exit with
  // an NC_ERR error code.
  NcError err(NcError::silent_nonfatal);

  // Create the file. The Replace parameter tells netCDF to overwrite
  // this file, if it already exists.
  NcFile dataFile("sfc_pres_temp.nc", NcFile::Replace);

  // Check to see if the file was created.
  if(!dataFile.is_valid())
     return NC_ERR;

  // Define the dimensions. NetCDF will hand back an ncDim object for
  // each.
  NcDim *latDim, *lonDim;
  if (!(latDim = dataFile.add_dim("latitude", NLAT)))
     return NC_ERR;
  if (!(lonDim = dataFile.add_dim("longitude", NLON)))
     return NC_ERR;

  // In addition to the latitude and longitude dimensions, we will
  // also create latitude and longitude netCDF variables which will
  // hold the actual latitudes and longitudes. Since they hold data
  // about the coordinate system, the netCDF term for these is:
  // "coordinate variables."
  NcVar *latVar, *lonVar;
  if (!(latVar = dataFile.add_var("latitude", ncFloat, latDim)))
     return NC_ERR;
  if (!(lonVar = dataFile.add_var("longitude", ncFloat, lonDim)))
     return NC_ERR;

  // Define units attributes for coordinate vars. This attaches a
  // text attribute to each of the coordinate variables, containing
  // the units.
  if (!lonVar->add_att("units", "degrees_east"))
     return NC_ERR;
```

```
   if (!latVar->add_att("units", "degrees_north"))
      return NC_ERR;

   // Define the netCDF data variables.
   NcVar *presVar, *tempVar;
   if (!(presVar = dataFile.add_var("pressure", ncFloat, latDim, lonDim)))
      return NC_ERR;
   if (!(tempVar = dataFile.add_var("temperature", ncFloat, latDim, lonDim)))
      return NC_ERR;

   // Define units attributes for variables.
   if (!presVar->add_att("units", "hPa"))
      return NC_ERR;
   if (!tempVar->add_att("units", "celsius"))
      return NC_ERR;

   // Write the coordinate variable data. This will put the latitudes
   // and longitudes of our data grid into the netCDF file.
   if (!latVar->put(lats, NLAT))
      return NC_ERR;
   if (!lonVar->put(lons, NLON))
      return NC_ERR;

   // Write the pretend data. This will write our surface pressure and
   // surface temperature data. The arrays of data are the same size
   // as the netCDF variables we have defined, and below we write them
   // each in one step.
   if (!presVar->put(&presOut[0][0], NLAT, NLON))
      return NC_ERR;
   if (!tempVar->put(&tempOut[0][0], NLAT, NLON))
      return NC_ERR;

   // The file is automatically closed by the destructor. This frees
   // up any internal netCDF resources associated with the file, and
   // flushes any buffers.
   cout << "*** SUCCESS writing example file sfc_pres_temp.nc!" << endl;

   return 0;
}
```

2.2.4.2 sfc_pres_temp_rd.cpp

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.

   This is an example which reads some surface pressure and
```

```
   temperatures. The data file read by this program is produced
   companion program sfc_pres_temp_wr.cxx. It is intended to
   illustrate the use of the netCDF C++ API.

   This program is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

   Full documentation of the netCDF C++ API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-cxx

   $Id: sfc_pres_temp_rd.cpp,v 1.17 2008/05/16 13:42:28 ed Exp $
*/

#include <iostream>
#include <cstring>
#include <netcdfcpp.h>

using namespace std;

// We are reading 2D data, a 6 x 12 lat-lon grid.
static const int NLAT = 6;
static const int NLON = 12;

// These are used to calculate the values we expect to find.
static const float SAMPLE_PRESSURE = 900;
static const float SAMPLE_TEMP = 9.0;
static const float START_LAT = 25.0;
static const float START_LON = -125.0;

// Return this code to the OS in case of failure.
static const int NC_ERR = 2;

int main(void)
{
   // These will hold our pressure and temperature data.
   float presIn[NLAT][NLON];
   float tempIn[NLAT][NLON];

   // These will hold our latitudes and longitudes.
   float latsIn[NLAT];
   float lonsIn[NLON];

   // Change the error behavior of the netCDF C++ API by creating an
   // NcError object. Until it is destroyed, this NcError object will
   // ensure that the netCDF C++ API silently returns error codes on
   // any failure, and leaves any other error handling to the calling
   // program. In the case of this example, we just exit with an
```

```
    // NC_ERR error code.
    NcError err(NcError::silent_nonfatal);

    // Open the file and check to make sure it's valid.
    NcFile dataFile("sfc_pres_temp.nc", NcFile::ReadOnly);
    if(!dataFile.is_valid())
       return NC_ERR;

    // There are a number of inquiry functions in netCDF which can be
    // used to learn about an unknown netCDF file. In this case we know
    // that there are 2 netCDF dimensions, 4 netCDF variables, no
    // global attributes, and no unlimited dimension.
    if (dataFile.num_dims() != 2 || dataFile.num_vars() != 4 ||
        dataFile.num_atts() != 0 || dataFile.rec_dim() != 0)
       return NC_ERR;

    // We get back a pointer to each NcVar we request. Get the
    // latitude and longitude coordinate variables.
    NcVar *latVar, *lonVar;
    if (!(latVar = dataFile.get_var("latitude")))
       return NC_ERR;
    if (!(lonVar = dataFile.get_var("longitude")))
       return NC_ERR;

    // Read the latitude and longitude coordinate variables into arrays
    // latsIn and lonsIn.
    if (!latVar->get(latsIn, NLAT))
       return NC_ERR;
    if (!lonVar->get(lonsIn, NLON))
       return NC_ERR;

    // Check the coordinate variable data.
    for(int lat = 0; lat < NLAT; lat++)
       if (latsIn[lat] != START_LAT + 5. * lat)
  return NC_ERR;

    // Check longitude values.
    for (int lon = 0; lon < NLON; lon++)
       if (lonsIn[lon] != START_LON + 5. * lon)
  return NC_ERR;

    // We get back a pointer to each NcVar we request.
    NcVar *presVar, *tempVar;
    if (!(presVar = dataFile.get_var("pressure")))
       return NC_ERR;
    if (!(tempVar = dataFile.get_var("temperature")))
       return NC_ERR;
```

```
   // Read the data. Since we know the contents of the file we know
   // that the data arrays in this program are the correct size to
   // hold all the data.
   if (!presVar->get(&presIn[0][0], NLAT, NLON))
      return NC_ERR;
   if (!tempVar->get(&tempIn[0][0], NLAT, NLON))
      return NC_ERR;

   // Check the data.
   for (int lat = 0; lat < NLAT; lat++)
      for (int lon = 0; lon < NLON; lon++)
 if (presIn[lat][lon] != SAMPLE_PRESSURE + (lon * NLAT + lat)
     || tempIn[lat][lon] != SAMPLE_TEMP + .25 * (lon * NLAT + lat))
    return NC_ERR;

   // Each of the netCDF variables has a "units" attribute. Let's read
   // them and check them.
   NcAtt *att;
   char *units;

   if (!(att = latVar->get_att("units")))
      return NC_ERR;
   units = att->as_string(0);
   if (strncmp(units, "degrees_north", strlen("degrees_north")))
      return NC_ERR;
   // Attributes and attribute values should be deleted by the caller
   // when no longer needed, to prevent memory leaks.
   delete units;
   delete att;

   if (!(att = lonVar->get_att("units")))
      return NC_ERR;
   units = att->as_string(0);
   if (strncmp(units, "degrees_east", strlen("degrees_east")))
      return NC_ERR;
   delete units;
   delete att;

   if (!(att = presVar->get_att("units")))
      return NC_ERR;
   units = att->as_string(0);
   if (strncmp(units, "hPa", strlen("hPa")))
      return NC_ERR;
   delete units;
   delete att;
```

```
   if (!(att = tempVar->get_att("units")))
      return NC_ERR;
   units = att->as_string(0);
   if (strncmp(units, "celsius", strlen("celsius")))
      return NC_ERR;
   delete units;
   delete att;

   // The file will be automatically closed by the destructor. This
   // frees up any internal netCDF resources associated with the file,
   // and flushes any buffers.
   cout << "*** SUCCESS reading example file sfc_pres_temp.nc!" << endl;

   return 0;
}
```

2.3 The pres_temp_4D Example

This example expands on the previous example by making our two-dimensional data into four-dimensional data, adding a vertical level axis and an unlimited time step axis.

Additionally, in this example the data are written and read one time step at a time, as is typical in scientific applications that use the unlimited dimension.

The sample data file created by pres_temp_4D_wr can be examined with the utility ncdump. The output is shown below. For more information on ncdump see Section "NetCDF Utilities" in *The NetCDF Users Guide*.

```
netcdf pres_temp_4D {
dimensions:
level = 2 ;
latitude = 6 ;
longitude = 12 ;
time = UNLIMITED ; // (2 currently)
variables:
float latitude(latitude) ;
latitude:units = "degrees_north" ;
float longitude(longitude) ;
longitude:units = "degrees_east" ;
float pressure(time, level, latitude, longitude) ;
pressure:units = "hPa" ;
float temperature(time, level, latitude, longitude) ;
temperature:units = "celsius" ;
data:

 latitude = 25, 30, 35, 40, 45, 50 ;

 longitude = -125, -120, -115, -110, -105, -100, -95, -90, -85, -80, -75, -70 ;

 pressure =
```

```
     900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911,
     912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923,
     924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935,
     936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947,
     948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959,
     960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971,
     972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983,
     984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995,
     996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007,
     1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019,
     1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031,
     1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043,
     900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911,
     912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923,
     924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935,
     936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947,
     948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959,
     960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971,
     972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983,
     984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995,
     996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007,
     1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019,
     1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031,
     1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043 ;

    temperature =
     9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
     21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
     33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
     45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
     57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
     69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
     81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
     93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104,
     105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
     117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128,
     129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140,
     141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152,
     9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
     21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
     33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
     45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
     57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
     69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
     81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
     93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104,
     105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
```

```
  117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128,
  129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140,
  141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152 ;
}
```

2.3.1 pres_temp_4D_wr.c and pres_temp_4D_rd.c

These example programs can be found in the netCDF distribution, under examples/C.

The example program pres_temp_4D_wr.c creates the example data file pres_temp_4D.nc. The example program pres_temp_4D_rd.c reads the data file.

2.3.1.1 pres_temp_4D_wr.c

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.

   This is an example program which writes some 4D pressure and
   temperatures. It is intended to illustrate the use of the netCDF
   C API. The companion program pres_temp_4D_rd.c shows how
   to read the netCDF data file created by this program.

   This program is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

   Full documentation of the netCDF C API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c

   $Id: pres_temp_4D_wr.c,v 1.5 2006/11/04 21:13:04 russ Exp $
*/

#include <stdio.h>
#include <string.h>
#include <netcdf.h>

/* This is the name of the data file we will create. */
#define FILE_NAME "pres_temp_4D.nc"

/* We are writing 4D data, a 2 x 6 x 12 lvl-lat-lon grid, with 2
   timesteps of data. */
#define NDIMS 4
#define NLAT 6
#define NLON 12
#define LAT_NAME "latitude"
#define LON_NAME "longitude"
#define NREC 2
#define REC_NAME "time"
#define LVL_NAME "level"
```

```
#define NLVL 2

/* Names of things. */
#define PRES_NAME "pressure"
#define TEMP_NAME "temperature"
#define UNITS "units"
#define DEGREES_EAST "degrees_east"
#define DEGREES_NORTH "degrees_north"

/* These are used to construct some example data. */
#define SAMPLE_PRESSURE 900
#define SAMPLE_TEMP 9.0
#define START_LAT 25.0
#define START_LON -125.0

/* For the units attributes. */
#define UNITS "units"
#define PRES_UNITS "hPa"
#define TEMP_UNITS "celsius"
#define LAT_UNITS "degrees_north"
#define LON_UNITS "degrees_east"
#define MAX_ATT_LEN 80

/* Handle errors by printing an error message and exiting with a
 * non-zero status. */
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); return 2;}

int
main()
{
   /* IDs for the netCDF file, dimensions, and variables. */
   int ncid, lon_dimid, lat_dimid, lvl_dimid, rec_dimid;
   int lat_varid, lon_varid, pres_varid, temp_varid;
   int dimids[NDIMS];

   /* The start and count arrays will tell the netCDF library where to
      write our data. */
   size_t start[NDIMS], count[NDIMS];

   /* Program variables to hold the data we will write out. We will only
      need enough space to hold one timestep of data; one record. */
   float pres_out[NLVL][NLAT][NLON];
   float temp_out[NLVL][NLAT][NLON];

   /* These program variables hold the latitudes and longitudes. */
   float lats[NLAT], lons[NLON];
```

```
   /* Loop indexes. */
   int lvl, lat, lon, rec, i = 0;

   /* Error handling. */
   int retval;

   /* Create some pretend data. If this wasn't an example program, we
    * would have some real data to write, for example, model
    * output. */
   for (lat = 0; lat < NLAT; lat++)
      lats[lat] = START_LAT + 5.*lat;
   for (lon = 0; lon < NLON; lon++)
      lons[lon] = START_LON + 5.*lon;

   for (lvl = 0; lvl < NLVL; lvl++)
      for (lat = 0; lat < NLAT; lat++)
 for (lon = 0; lon < NLON; lon++)
 {
    pres_out[lvl][lat][lon] = SAMPLE_PRESSURE + i;
    temp_out[lvl][lat][lon] = SAMPLE_TEMP + i++;
 }

   /* Create the file. */
   if ((retval = nc_create(FILE_NAME, NC_CLOBBER, &ncid)))
      ERR(retval);

   /* Define the dimensions. The record dimension is defined to have
    * unlimited length - it can grow as needed. In this example it is
    * the time dimension.*/
   if ((retval = nc_def_dim(ncid, LVL_NAME, NLVL, &lvl_dimid)))
      ERR(retval);
   if ((retval = nc_def_dim(ncid, LAT_NAME, NLAT, &lat_dimid)))
      ERR(retval);
   if ((retval = nc_def_dim(ncid, LON_NAME, NLON, &lon_dimid)))
      ERR(retval);
   if ((retval = nc_def_dim(ncid, REC_NAME, NC_UNLIMITED, &rec_dimid)))
      ERR(retval);

   /* Define the coordinate variables. We will only define coordinate
      variables for lat and lon.  Ordinarily we would need to provide
      an array of dimension IDs for each variable's dimensions, but
      since coordinate variables only have one dimension, we can
      simply provide the address of that dimension ID (&lat_dimid) and
      similarly for (&lon_dimid). */
   if ((retval = nc_def_var(ncid, LAT_NAME, NC_FLOAT, 1, &lat_dimid,
    &lat_varid)))
      ERR(retval);
```

```
  if ((retval = nc_def_var(ncid, LON_NAME, NC_FLOAT, 1, &lon_dimid,
   &lon_varid)))
     ERR(retval);

  /* Assign units attributes to coordinate variables. */
  if ((retval = nc_put_att_text(ncid, lat_varid, UNITS,
strlen(DEGREES_NORTH), DEGREES_NORTH)))
     ERR(retval);
  if ((retval = nc_put_att_text(ncid, lon_varid, UNITS,
strlen(DEGREES_EAST), DEGREES_EAST)))
     ERR(retval);

  /* The dimids array is used to pass the dimids of the dimensions of
     the netCDF variables. Both of the netCDF variables we are
     creating share the same four dimensions. In C, the
     unlimited dimension must come first on the list of dimids. */
  dimids[0] = rec_dimid;
  dimids[1] = lvl_dimid;
  dimids[2] = lat_dimid;
  dimids[3] = lon_dimid;

  /* Define the netCDF variables for the pressure and temperature
   * data. */
  if ((retval = nc_def_var(ncid, PRES_NAME, NC_FLOAT, NDIMS,
   dimids, &pres_varid)))
     ERR(retval);
  if ((retval = nc_def_var(ncid, TEMP_NAME, NC_FLOAT, NDIMS,
   dimids, &temp_varid)))
     ERR(retval);

  /* Assign units attributes to the netCDF variables. */
  if ((retval = nc_put_att_text(ncid, pres_varid, UNITS,
strlen(PRES_UNITS), PRES_UNITS)))
     ERR(retval);
  if ((retval = nc_put_att_text(ncid, temp_varid, UNITS,
strlen(TEMP_UNITS), TEMP_UNITS)))
     ERR(retval);

  /* End define mode. */
  if ((retval = nc_enddef(ncid)))
     ERR(retval);

  /* Write the coordinate variable data. This will put the latitudes
     and longitudes of our data grid into the netCDF file. */
  if ((retval = nc_put_var_float(ncid, lat_varid, &lats[0])))
     ERR(retval);
  if ((retval = nc_put_var_float(ncid, lon_varid, &lons[0])))
```

```
      ERR(retval);

   /* These settings tell netcdf to write one timestep of data. (The
     setting of start[0] inside the loop below tells netCDF which
     timestep to write.) */
   count[0] = 1;
   count[1] = NLVL;
   count[2] = NLAT;
   count[3] = NLON;
   start[1] = 0;
   start[2] = 0;
   start[3] = 0;

   /* Write the pretend data. This will write our surface pressure and
      surface temperature data. The arrays only hold one timestep worth
      of data. We will just rewrite the same data for each timestep. In
      a real application, the data would change between timesteps. */
   for (rec = 0; rec < NREC; rec++)
   {
      start[0] = rec;
      if ((retval = nc_put_vara_float(ncid, pres_varid, start, count,
      &pres_out[0][0][0])))
 ERR(retval);
      if ((retval = nc_put_vara_float(ncid, temp_varid, start, count,
      &temp_out[0][0][0])))
 ERR(retval);
   }

   /* Close the file. */
   if ((retval = nc_close(ncid)))
      ERR(retval);

   printf("*** SUCCESS writing example file %s!\n", FILE_NAME);
   return 0;
}
```

2.3.1.2 pres_temp_4D_rd.c

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.

   This is an example which reads some 4D pressure and
   temperatures. The data file read by this program is produced by the
   companion program pres_temp_4D_wr.c. It is intended to illustrate
   the use of the netCDF C API.
```

```
   This program is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

   Full documentation of the netCDF C API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c

   $Id: pres_temp_4D_rd.c,v 1.5 2006/06/26 20:37:31 russ Exp $
*/

#include <stdio.h>
#include <string.h>
#include <netcdf.h>

/* This is the name of the data file we will read. */
#define FILE_NAME "pres_temp_4D.nc"

/* We are reading 4D data, a 2 x 6 x 12 lvl-lat-lon grid, with 2
   timesteps of data. */
#define NDIMS 4
#define NLAT 6
#define NLON 12
#define LAT_NAME "latitude"
#define LON_NAME "longitude"
#define NREC 2
#define REC_NAME "time"
#define LVL_NAME "level"
#define NLVL 2

/* Names of things. */
#define PRES_NAME "pressure"
#define TEMP_NAME "temperature"
#define UNITS "units"
#define DEGREES_EAST "degrees_east"
#define DEGREES_NORTH "degrees_north"

/* These are used to calculate the values we expect to find. */
#define SAMPLE_PRESSURE 900
#define SAMPLE_TEMP 9.0
#define START_LAT 25.0
#define START_LON -125.0

/* For the units attributes. */
#define UNITS "units"
#define PRES_UNITS "hPa"
#define TEMP_UNITS "celsius"
#define LAT_UNITS "degrees_north"
#define LON_UNITS "degrees_east"
```

```
#define MAX_ATT_LEN 80

/* Handle errors by printing an error message and exiting with a
 * non-zero status. */
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); return 2;}

int
main()
{
   int ncid, pres_varid, temp_varid;
   int lat_varid, lon_varid;

   /* The start and count arrays will tell the netCDF library where to
      read our data. */
   size_t start[NDIMS], count[NDIMS];

   /* Program variables to hold the data we will read. We will only
      need enough space to hold one timestep of data; one record. */
   float pres_in[NLVL][NLAT][NLON];
   float temp_in[NLVL][NLAT][NLON];

   /* These program variables hold the latitudes and longitudes. */
   float lats[NLAT], lons[NLON];

   /* Loop indexes. */
   int lvl, lat, lon, rec, i = 0;

   /* Error handling. */
   int retval;

   /* Open the file. */
   if ((retval = nc_open(FILE_NAME, NC_NOWRITE, &ncid)))
      ERR(retval);

   /* Get the varids of the latitude and longitude coordinate
    * variables. */
   if ((retval = nc_inq_varid(ncid, LAT_NAME, &lat_varid)))
      ERR(retval);
   if ((retval = nc_inq_varid(ncid, LON_NAME, &lon_varid)))
      ERR(retval);

   /* Read the coordinate variable data. */
   if ((retval = nc_get_var_float(ncid, lat_varid, &lats[0])))
      ERR(retval);
   if ((retval = nc_get_var_float(ncid, lon_varid, &lons[0])))
      ERR(retval);
```

```
   /* Check the coordinate variable data. */
   for (lat = 0; lat < NLAT; lat++)
      if (lats[lat] != START_LAT + 5.*lat)
 return 2;
   for (lon = 0; lon < NLON; lon++)
      if (lons[lon] != START_LON + 5.*lon)
 return 2;

   /* Get the varids of the pressure and temperature netCDF
    * variables. */
   if ((retval = nc_inq_varid(ncid, PRES_NAME, &pres_varid)))
      ERR(retval);
   if ((retval = nc_inq_varid(ncid, TEMP_NAME, &temp_varid)))
      ERR(retval);

   /* Read the data. Since we know the contents of the file we know
    * that the data arrays in this program are the correct size to
    * hold one timestep. */
   count[0] = 1;
   count[1] = NLVL;
   count[2] = NLAT;
   count[3] = NLON;
   start[1] = 0;
   start[2] = 0;
   start[3] = 0;

   /* Read and check one record at a time. */
   for (rec = 0; rec < NREC; rec++)
   {
      start[0] = rec;
      if ((retval = nc_get_vara_float(ncid, pres_varid, start,
      count, &pres_in[0][0][0])))
 ERR(retval);
      if ((retval = nc_get_vara_float(ncid, temp_varid, start,
      count, &temp_in[0][0][0])))
 ERR(retval);

      /* Check the data. */
      i = 0;
      for (lvl = 0; lvl < NLVL; lvl++)
 for (lat = 0; lat < NLAT; lat++)
    for (lon = 0; lon < NLON; lon++)
    {
       if (pres_in[lvl][lat][lon] != SAMPLE_PRESSURE + i)
  return 2;
       if (temp_in[lvl][lat][lon] != SAMPLE_TEMP + i)
  return 2;
```

```
        i++;
     }

   } /* next record */

   /* Close the file. */
   if ((retval = nc_close(ncid)))
      ERR(retval);

   printf("*** SUCCESS reading example file pres_temp_4D.nc!\n");
   return 0;
}
```

2.3.2 pres_temp_4D_wr.f and pres_temp_4D_rd.f

These example programs can be found in the netCDF distribution, under examples/F77.

The example program pres_temp_4D_wr.f creates the example data file pres_temp_4D.nc. The example program pres_temp_4D_rd.f reads the data file.

2.3.2.1 pres_temp_4D_wr.f

```
C     This is part of the netCDF package.
C     Copyright 2006 University Corporation for Atmospheric Research/Unidata.
C     See COPYRIGHT file for conditions of use.

C     This is an example program which writes some 4D pressure and
C     temperatures. It is intended to illustrate the use of the netCDF
C     fortran 77 API. The companion program pres_temp_4D_rd.f shows how
C     to read the netCDF data file created by this program.

C     This program is part of the netCDF tutorial:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

C     Full documentation of the netCDF Fortran 77 API can be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f77

C     $Id: pres_temp_4D_wr.f,v 1.12 2007/02/14 20:59:20 ed Exp $

      program pres_temp_4D_wr
      implicit none
      include 'netcdf.inc'

C     This is the name of the data file we will create.
      character*(*) FILE_NAME
      parameter (FILE_NAME = 'pres_temp_4D.nc')
      integer ncid

C     We are writing 4D data, a 12 x 6 x 2 lon-lat-lvl grid, with 2
```

```
C     timesteps of data.
      integer NDIMS, NRECS
      parameter (NDIMS = 4, NRECS = 2)
      integer NLVLS, NLATS, NLONS
      parameter (NLVLS = 2, NLATS = 6, NLONS = 12)
      character*(*) LVL_NAME, LAT_NAME, LON_NAME, REC_NAME
      parameter (LVL_NAME = 'level')
      parameter (LAT_NAME = 'latitude', LON_NAME = 'longitude')
      parameter (REC_NAME = 'time')
      integer lvl_dimid, lon_dimid, lat_dimid, rec_dimid

C     The start and count arrays will tell the netCDF library where to
C     write our data.
      integer start(NDIMS), count(NDIMS)

C     These program variables hold the latitudes and longitudes.
      real lats(NLATS), lons(NLONS)
      integer lon_varid, lat_varid

C     We will create two netCDF variables, one each for temperature and
C     pressure fields.
      character*(*) PRES_NAME, TEMP_NAME
      parameter (PRES_NAME='pressure')
      parameter (TEMP_NAME='temperature')
      integer pres_varid, temp_varid
      integer dimids(NDIMS)

C     We recommend that each variable carry a "units" attribute.
      character*(*) UNITS
      parameter (UNITS = 'units')
      character*(*) PRES_UNITS, TEMP_UNITS, LAT_UNITS, LON_UNITS
      parameter (PRES_UNITS = 'hPa', TEMP_UNITS = 'celsius')
      parameter (LAT_UNITS = 'degrees_north')
      parameter (LON_UNITS = 'degrees_east')

C     Program variables to hold the data we will write out. We will only
C     need enough space to hold one timestep of data; one record.
      real pres_out(NLONS, NLATS, NLVLS)
      real temp_out(NLONS, NLATS, NLVLS)
      real SAMPLE_PRESSURE
      parameter (SAMPLE_PRESSURE = 900.0)
      real SAMPLE_TEMP
      parameter (SAMPLE_TEMP = 9.0)

C     Use these to construct some latitude and longitude data for this
C     example.
      integer START_LAT, START_LON
```

```
      parameter (START_LAT = 25.0, START_LON = -125.0)

C     Loop indices.
      integer lvl, lat, lon, rec, i

C     Error handling.
      integer retval

C     Create pretend data. If this wasn't an example program, we would
C     have some real data to write, for example, model output.
      do lat = 1, NLATS
         lats(lat) = START_LAT + (lat - 1) * 5.0
      end do
      do lon = 1, NLONS
         lons(lon) = START_LON + (lon - 1) * 5.0
      end do
      i = 0
      do lvl = 1, NLVLS
         do lat = 1, NLATS
            do lon = 1, NLONS
               pres_out(lon, lat, lvl) = SAMPLE_PRESSURE + i
               temp_out(lon, lat, lvl) = SAMPLE_TEMP + i
               i = i + 1
            end do
         end do
      end do

C     Create the file.
      retval = nf_create(FILE_NAME, nf_clobber, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the dimensions. The record dimension is defined to have
C     unlimited length - it can grow as needed. In this example it is
C     the time dimension.
      retval = nf_def_dim(ncid, LVL_NAME, NLVLS, lvl_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_dim(ncid, LAT_NAME, NLATS, lat_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_dim(ncid, LON_NAME, NLONS, lon_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_dim(ncid, REC_NAME, NF_UNLIMITED, rec_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the coordinate variables. We will only define coordinate
C     variables for lat and lon.  Ordinarily we would need to provide
C     an array of dimension IDs for each variable's dimensions, but
C     since coordinate variables only have one dimension, we can
```

```
C     simply provide the address of that dimension ID (lat_dimid) and
C     similarly for (lon_dimid).
      retval = nf_def_var(ncid, LAT_NAME, NF_REAL, 1, lat_dimid,
     +     lat_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_var(ncid, LON_NAME, NF_REAL, 1, lon_dimid,
     +     lon_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Assign units attributes to coordinate variables.
      retval = nf_put_att_text(ncid, lat_varid, UNITS, len(LAT_UNITS),
     +     LAT_UNITS)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_put_att_text(ncid, lon_varid, UNITS, len(LON_UNITS),
     +     LON_UNITS)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     The dimids array is used to pass the dimids of the dimensions of
C     the netCDF variables. Both of the netCDF variables we are creating
C     share the same four dimensions. In Fortran, the unlimited
C     dimension must come last on the list of dimids.
      dimids(1) = lon_dimid
      dimids(2) = lat_dimid
      dimids(3) = lvl_dimid
      dimids(4) = rec_dimid

C     Define the netCDF variables for the pressure and temperature data.
      retval = nf_def_var(ncid, PRES_NAME, NF_REAL, NDIMS, dimids,
     +     pres_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_var(ncid, TEMP_NAME, NF_REAL, NDIMS, dimids,
     +     temp_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Assign units attributes to the netCDF variables.
      retval = nf_put_att_text(ncid, pres_varid, UNITS, len(PRES_UNITS),
     +     PRES_UNITS)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_put_att_text(ncid, temp_varid, UNITS, len(TEMP_UNITS),
     +     TEMP_UNITS)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     End define mode.
      retval = nf_enddef(ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Write the coordinate variable data. This will put the latitudes
```

```
C     and longitudes of our data grid into the netCDF file.
      retval = nf_put_var_real(ncid, lat_varid, lats)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_put_var_real(ncid, lon_varid, lons)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     These settings tell netcdf to write one timestep of data. (The
C     setting of start(4) inside the loop below tells netCDF which
C     timestep to write.)
      count(1) = NLONS
      count(2) = NLATS
      count(3) = NLVLS
      count(4) = 1
      start(1) = 1
      start(2) = 1
      start(3) = 1

C     Write the pretend data. This will write our surface pressure and
C     surface temperature data. The arrays only hold one timestep worth
C     of data. We will just rewrite the same data for each timestep. In
C     a real application, the data would change between timesteps.
      do rec = 1, NRECS
         start(4) = rec
         retval = nf_put_vara_real(ncid, pres_varid, start, count,
     +        pres_out)
         if (retval .ne. nf_noerr) call handle_err(retval)
         retval = nf_put_vara_real(ncid, temp_varid, start, count,
     +        temp_out)
         if (retval .ne. nf_noerr) call handle_err(retval)
      end do

C     Close the file. This causes netCDF to flush all buffers and make
C     sure your data are really written to disk.
      retval = nf_close(ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

      print *,'*** SUCCESS writing example file', FILE_NAME, '!'
      end

      subroutine handle_err(errcode)
      implicit none
      include 'netcdf.inc'
      integer errcode

      print *, 'Error: ', nf_strerror(errcode)
      stop 2
      end
```

2.3.2.2 pres_temp_4D_rd.f

```
C     This is part of the netCDF package.
C     Copyright 2006 University Corporation for Atmospheric Research/Unidata.
C     See COPYRIGHT file for conditions of use.

C     This is an example which reads some 4D pressure and
C     temperatures. The data file read by this program is produced by
C     the companion program pres_temp_4D_wr.f. It is intended to
C     illustrate the use of the netCDF Fortran 77 API.

C     This program is part of the netCDF tutorial:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

C     Full documentation of the netCDF Fortran 77 API can be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f77

C     $Id: pres_temp_4D_rd.f,v 1.12 2007/02/14 20:59:20 ed Exp $

      program pres_temp_4D_rd
      implicit none
      include 'netcdf.inc'

C     This is the name of the data file we will read.
      character*(*) FILE_NAME
      parameter (FILE_NAME='pres_temp_4D.nc')
      integer ncid

C     We are reading 4D data, a 12 x 6 x 2 lon-lat-lvl grid, with 2
C     timesteps of data.
      integer NDIMS, NRECS
      parameter (NDIMS = 4, NRECS = 2)
      integer NLVLS, NLATS, NLONS
      parameter (NLVLS = 2, NLATS = 6, NLONS = 12)
      character*(*) LVL_NAME, LAT_NAME, LON_NAME, REC_NAME
      parameter (LVL_NAME = 'level')
      parameter (LAT_NAME = 'latitude', LON_NAME = 'longitude')
      parameter (REC_NAME = 'time')
      integer lvl_dimid, lon_dimid, lat_dimid, rec_dimid

C     The start and count arrays will tell the netCDF library where to
C     read our data.
      integer start(NDIMS), count(NDIMS)

C     In addition to the latitude and longitude dimensions, we will also
C     create latitude and longitude variables which will hold the actual
C     latitudes and longitudes. Since they hold data about the
```

```
C     coordinate system, the netCDF term for these is: "coordinate
C     variables."
      real lats(NLATS), lons(NLONS)
      integer lon_varid, lat_varid

C     We will read surface temperature and pressure fields. In netCDF
C     terminology these are called "variables."
      character*(*) PRES_NAME, TEMP_NAME
      parameter (PRES_NAME='pressure')
      parameter (TEMP_NAME='temperature')
      integer pres_varid, temp_varid
      integer dimids(NDIMS)

C     We recommend that each variable carry a "units" attribute.
      character*(*) UNITS
      parameter (UNITS = 'units')
      character*(*) PRES_UNITS, TEMP_UNITS, LAT_UNITS, LON_UNITS
      parameter (PRES_UNITS = 'hPa', TEMP_UNITS = 'celsius')
      parameter (LAT_UNITS = 'degrees_north')
      parameter (LON_UNITS = 'degrees_east')

C     Program variables to hold the data we will read in. We will only
C     need enough space to hold one timestep of data; one record.
      real pres_in(NLONS, NLATS, NLVLS)
      real temp_in(NLONS, NLATS, NLVLS)
      real SAMPLE_PRESSURE
      parameter (SAMPLE_PRESSURE = 900.0)
      real SAMPLE_TEMP
      parameter (SAMPLE_TEMP = 9.0)

C     Use these to calculate the values we expect to find.
      integer START_LAT, START_LON
      parameter (START_LAT = 25.0, START_LON = -125.0)

C     Loop indices.
      integer lvl, lat, lon, rec, i

C     Error handling.
      integer retval

C     Open the file.
      retval = nf_open(FILE_NAME, nf_nowrite, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Get the varids of the latitude and longitude coordinate variables.
      retval = nf_inq_varid(ncid, LAT_NAME, lat_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

```
      retval = nf_inq_varid(ncid, LON_NAME, lon_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Read the latitude and longitude data.
      retval = nf_get_var_real(ncid, lat_varid, lats)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_get_var_real(ncid, lon_varid, lons)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Check to make sure we got what we expected.
      do lat = 1, NLATS
         if (lats(lat) .ne. START_LAT + (lat - 1) * 5.0) stop 2
      end do
      do lon = 1, NLONS
         if (lons(lon) .ne. START_LON + (lon - 1) * 5.0) stop 2
      end do

C     Get the varids of the pressure and temperature netCDF variables.
      retval = nf_inq_varid(ncid, PRES_NAME, pres_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_inq_varid(ncid, TEMP_NAME, temp_varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Read 1 record of NLONS*NLATS*NLVLS values, starting at the
C     beginning of the record (the (1, 1, 1, rec) element in the netCDF
C     file).
      count(1) = NLONS
      count(2) = NLATS
      count(3) = NLVLS
      count(4) = 1
      start(1) = 1
      start(2) = 1
      start(3) = 1

C     Read the surface pressure and temperature data from the file, one
C     record at a time.
      do rec = 1, NRECS
         start(4) = rec
         retval = nf_get_vara_real(ncid, pres_varid, start, count,
     $        pres_in)
         if (retval .ne. nf_noerr) call handle_err(retval)
         retval = nf_get_vara_real(ncid, temp_varid, start, count,
     $        temp_in)
         if (retval .ne. nf_noerr) call handle_err(retval)

         i = 0
         do lvl = 1, NLVLS
```

```
               do lat = 1, NLATS
                  do lon = 1, NLONS
                     if (pres_in(lon, lat, lvl) .ne. SAMPLE_PRESSURE + i)
     $                    stop 2
                     if (temp_in(lon, lat, lvl) .ne. SAMPLE_TEMP + i)
     $                    stop 2
                     i = i + 1
                  end do
               end do
            end do
C     next record
      end do

C     Close the file. This frees up any internal netCDF resources
C     associated with the file.
      retval = nf_close(ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     If we got this far, everything worked as expected. Yipee!
      print *,'*** SUCCESS reading example file pres_temp_4D.nc!'
      end

      subroutine handle_err(errcode)
      implicit none
      include 'netcdf.inc'
      integer errcode

      print *, 'Error: ', nf_strerror(errcode)
      stop 2
      end
```

2.3.3 pres_temp_4D_wr.f90 and pres_temp_4D_rd.f90

These example programs can be found in the netCDF distribution, under examples/F90.

The example program pres_temp_4D_wr.f90 creates the example data file pres_temp_4D.nc. The example program pres_temp_4D_rd.f90 reads the data file.

2.3.3.1 pres_temp_4D_wr.f90

```
! This is part of the netCDF package.
! Copyright 2006 University Corporation for Atmospheric Research/Unidata.
! See COPYRIGHT file for conditions of use.

! This is an example program which writes some 4D pressure and
! temperatures. It is intended to illustrate the use of the netCDF
! fortran 90 API. The companion program pres_temp_4D_rd.f shows how
! to read the netCDF data file created by this program.
```

```
! This program is part of the netCDF tutorial:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

! Full documentation of the netCDF Fortran 90 API can be found at:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90

! $Id: pres_temp_4D_wr.f90,v 1.10 2010/04/06 19:32:09 ed Exp $

program pres_temp_4D_wr
  use netcdf
  implicit none

  ! This is the name of the data file we will create.
  character (len = *), parameter :: FILE_NAME = "pres_temp_4D.nc"
  integer :: ncid

  ! We are writing 4D data, a 12 x 6 x 2 lon-lat-lvl grid, with 2
  ! timesteps of data.
  integer, parameter :: NDIMS = 4, NRECS = 2
  integer, parameter :: NLVLS = 2, NLATS = 6, NLONS = 12
  character (len = *), parameter :: LVL_NAME = "level"
  character (len = *), parameter :: LAT_NAME = "latitude"
  character (len = *), parameter :: LON_NAME = "longitude"
  character (len = *), parameter :: REC_NAME = "time"
  integer :: lvl_dimid, lon_dimid, lat_dimid, rec_dimid

  ! The start and count arrays will tell the netCDF library where to
  ! write our data.
  integer :: start(NDIMS), count(NDIMS)

  ! These program variables hold the latitudes and longitudes.
  real :: lats(NLATS), lons(NLONS)
  integer :: lon_varid, lat_varid

  ! We will create two netCDF variables, one each for temperature and
  ! pressure fields.
  character (len = *), parameter :: PRES_NAME="pressure"
  character (len = *), parameter :: TEMP_NAME="temperature"
  integer :: pres_varid, temp_varid
  integer :: dimids(NDIMS)

  ! We recommend that each variable carry a "units" attribute.
  character (len = *), parameter :: UNITS = "units"
  character (len = *), parameter :: PRES_UNITS = "hPa"
  character (len = *), parameter :: TEMP_UNITS = "celsius"
  character (len = *), parameter :: LAT_UNITS = "degrees_north"
  character (len = *), parameter :: LON_UNITS = "degrees_east"
```

```
! Program variables to hold the data we will write out. We will only
! need enough space to hold one timestep of data; one record.
real, dimension(:,:,:), allocatable :: pres_out
real, dimension(:,:,:), allocatable :: temp_out
real, parameter :: SAMPLE_PRESSURE = 900.0
real, parameter :: SAMPLE_TEMP = 9.0

! Use these to construct some latitude and longitude data for this
! example.
real, parameter :: START_LAT = 25.0, START_LON = -125.0

! Loop indices
integer :: lvl, lat, lon, rec, i

! Allocate memory.
allocate(pres_out(NLONS, NLATS, NLVLS))
allocate(temp_out(NLONS, NLATS, NLVLS))

! Create pretend data. If this were not an example program, we would
! have some real data to write, for example, model output.
do lat = 1, NLATS
   lats(lat) = START_LAT + (lat - 1) * 5.0
end do
do lon = 1, NLONS
   lons(lon) = START_LON + (lon - 1) * 5.0
end do
i = 0
do lvl = 1, NLVLS
   do lat = 1, NLATS
      do lon = 1, NLONS
         pres_out(lon, lat, lvl) = SAMPLE_PRESSURE + i
         temp_out(lon, lat, lvl) = SAMPLE_TEMP + i
         i = i + 1
      end do
   end do
end do

! Create the file.
call check( nf90_create(FILE_NAME, nf90_clobber, ncid) )

! Define the dimensions. The record dimension is defined to have
! unlimited length - it can grow as needed. In this example it is
! the time dimension.
call check( nf90_def_dim(ncid, LVL_NAME, NLVLS, lvl_dimid) )
call check( nf90_def_dim(ncid, LAT_NAME, NLATS, lat_dimid) )
call check( nf90_def_dim(ncid, LON_NAME, NLONS, lon_dimid) )
```

```
call check( nf90_def_dim(ncid, REC_NAME, NF90_UNLIMITED, rec_dimid) )

! Define the coordinate variables. We will only define coordinate
! variables for lat and lon.  Ordinarily we would need to provide
! an array of dimension IDs for each variable's dimensions, but
! since coordinate variables only have one dimension, we can
! simply provide the address of that dimension ID (lat_dimid) and
! similarly for (lon_dimid).
call check( nf90_def_var(ncid, LAT_NAME, NF90_REAL, lat_dimid, lat_varid) )
call check( nf90_def_var(ncid, LON_NAME, NF90_REAL, lon_dimid, lon_varid) )

! Assign units attributes to coordinate variables.
call check( nf90_put_att(ncid, lat_varid, UNITS, LAT_UNITS) )
call check( nf90_put_att(ncid, lon_varid, UNITS, LON_UNITS) )

! The dimids array is used to pass the dimids of the dimensions of
! the netCDF variables. Both of the netCDF variables we are creating
! share the same four dimensions. In Fortran, the unlimited
! dimension must come last on the list of dimids.
dimids = (/ lon_dimid, lat_dimid, lvl_dimid, rec_dimid /)

! Define the netCDF variables for the pressure and temperature data.
call check( nf90_def_var(ncid, PRES_NAME, NF90_REAL, dimids, pres_varid) )
call check( nf90_def_var(ncid, TEMP_NAME, NF90_REAL, dimids, temp_varid) )

! Assign units attributes to the netCDF variables.
call check( nf90_put_att(ncid, pres_varid, UNITS, PRES_UNITS) )
call check( nf90_put_att(ncid, temp_varid, UNITS, TEMP_UNITS) )

! End define mode.
call check( nf90_enddef(ncid) )

! Write the coordinate variable data. This will put the latitudes
! and longitudes of our data grid into the netCDF file.
call check( nf90_put_var(ncid, lat_varid, lats) )
call check( nf90_put_var(ncid, lon_varid, lons) )

! These settings tell netcdf to write one timestep of data. (The
! setting of start(4) inside the loop below tells netCDF which
! timestep to write.)
count = (/ NLONS, NLATS, NLVLS, 1 /)
start = (/ 1, 1, 1, 1 /)

! Write the pretend data. This will write our surface pressure and
! surface temperature data. The arrays only hold one timestep worth
! of data. We will just rewrite the same data for each timestep. In
! a real :: application, the data would change between timesteps.
```

```
  do rec = 1, NRECS
     start(4) = rec
     call check( nf90_put_var(ncid, pres_varid, pres_out, start = start, &
                              count = count) )
     call check( nf90_put_var(ncid, temp_varid, temp_out, start = start, &
                              count = count) )
  end do

  ! Close the file. This causes netCDF to flush all buffers and make
  ! sure your data are really written to disk.
  call check( nf90_close(ncid) )

  print *,"*** SUCCESS writing example file ", FILE_NAME, "!"

contains
  subroutine check(status)
    integer, intent ( in) :: status

    if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status))
      stop 2
    end if
  end subroutine check
end program pres_temp_4D_wr
```

2.3.3.2 pres_temp_4D_rd.f90

```
! This is part of the netCDF package.
! Copyright 2006 University Corporation for Atmospheric Research/Unidata.
! See COPYRIGHT file for conditions of use.

! This is an example which reads some 4D pressure and
! temperatures. The data file read by this program is produced by
! the companion program pres_temp_4D_wr.f90. It is intended to
! illustrate the use of the netCDF Fortran 90 API.

! This program is part of the netCDF tutorial:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

! Full documentation of the netCDF Fortran 90 API can be found at:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90

! $Id: pres_temp_4D_rd.f90,v 1.9 2010/04/06 19:32:09 ed Exp $

program pres_temp_4D_rd
  use netcdf
```

```
implicit none

! This is the name of the data file we will read.
character (len = *), parameter :: FILE_NAME = "pres_temp_4D.nc"
integer :: ncid

! We are reading 4D data, a 12 x 6 x 2 lon-lat-lvl grid, with 2
! timesteps of data.
integer, parameter :: NDIMS = 4, NRECS = 2
integer, parameter :: NLVLS = 2, NLATS = 6, NLONS = 12
character (len = *), parameter :: LVL_NAME = "level"
character (len = *), parameter :: LAT_NAME = "latitude"
character (len = *), parameter :: LON_NAME = "longitude"
character (len = *), parameter :: REC_NAME = "time"

! The start and count arrays will tell the netCDF library where to
! read our data.
integer :: start(NDIMS), count(NDIMS)

! In addition to the latitude and longitude dimensions, we will also
! create latitude and longitude variables which will hold the actual
! latitudes and longitudes. Since they hold data about the
! coordinate system, the netCDF term for these is: "coordinate
! variables."
real :: lats(NLATS), lons(NLONS)
integer :: lon_varid, lat_varid

! We will read surface temperature and pressure fields. In netCDF
! terminology these are called "variables."
character (len = *), parameter :: PRES_NAME="pressure"
character (len = *), parameter :: TEMP_NAME="temperature"
integer :: pres_varid, temp_varid

! We recommend that each variable carry a "units" attribute.
character (len = *), parameter :: UNITS = "units"
character (len = *), parameter :: PRES_UNITS = "hPa", TEMP_UNITS = "celsius"
character (len = *), parameter :: LAT_UNITS = "degrees_north"
character (len = *), parameter :: LON_UNITS = "degrees_east"

! Program variables to hold the data we will read in. We will only
! need enough space to hold one timestep of data; one record.
! Allocate memory for data.
real, dimension(:,:,:), allocatable :: pres_in
real, dimension(:,:,:), allocatable :: temp_in
real, parameter :: SAMPLE_PRESSURE = 900.0
real, parameter :: SAMPLE_TEMP = 9.0
```

```
! Use these to calculate the values we expect to find.
real, parameter :: START_LAT = 25.0, START_LON = -125.0

! Loop indices
integer :: lvl, lat, lon, rec, i

! Allocate memory.
allocate(pres_in(NLONS, NLATS, NLVLS))
allocate(temp_in(NLONS, NLATS, NLVLS))

! Open the file.
call check( nf90_open(FILE_NAME, nf90_nowrite, ncid) )

! Get the varids of the latitude and longitude coordinate variables.
call check( nf90_inq_varid(ncid, LAT_NAME, lat_varid) )
call check( nf90_inq_varid(ncid, LON_NAME, lon_varid) )

! Read the latitude and longitude data.
call check( nf90_get_var(ncid, lat_varid, lats) )
call check( nf90_get_var(ncid, lon_varid, lons) )

! Check to make sure we got what we expected.
do lat = 1, NLATS
   if (lats(lat) /= START_LAT + (lat - 1) * 5.0) stop 2
end do
do lon = 1, NLONS
   if (lons(lon) /= START_LON + (lon - 1) * 5.0) stop 2
end do

! Get the varids of the pressure and temperature netCDF variables.
call check( nf90_inq_varid(ncid, PRES_NAME, pres_varid) )
call check( nf90_inq_varid(ncid, TEMP_NAME, temp_varid) )

! Read 1 record of NLONS*NLATS*NLVLS values, starting at the beginning
! of the record (the (1, 1, 1, rec) element in the netCDF file).
count = (/ NLONS, NLATS, NLVLS, 1 /)
start = (/ 1, 1, 1, 1 /)

! Read the surface pressure and temperature data from the file, one
! record at a time.
do rec = 1, NRECS
   start(4) = rec
   call check( nf90_get_var(ncid, pres_varid, pres_in, start = start, &
                            count = count) )
   call check( nf90_get_var(ncid, temp_varid, temp_in, start, count) )

   i = 0
```

```
      do lvl = 1, NLVLS
         do lat = 1, NLATS
            do lon = 1, NLONS
               if (pres_in(lon, lat, lvl) /= SAMPLE_PRESSURE + i) stop 2
               if (temp_in(lon, lat, lvl) /= SAMPLE_TEMP + i) stop 2
               i = i + 1
            end do
         end do
      end do
      ! next record
   end do

   ! Close the file. This frees up any internal netCDF resources
   ! associated with the file.
   call check( nf90_close(ncid) )

   ! If we got this far, everything worked as expected. Yipee!
   print *,"*** SUCCESS reading example file ", FILE_NAME, "!"

contains
  subroutine check(status)
    integer, intent ( in) :: status

    if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status))
      stop 2
    end if
  end subroutine check
end program pres_temp_4D_rd
```

2.3.4 pres_temp_4D_wr.cpp and pres_temp_4D_rd.cpp

These example programs can be found in the netCDF distribution, under examples/CXX.

The example program pres_temp_4D_wr.cpp creates the example data file pres_temp_4D.nc. The example program pres_temp_4D_rd.cpp reads the data file.

2.3.4.1 pres_temp_4D_wr.cpp

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.

   This is an example program which writes some 4D pressure and
   temperatures. This example demonstrates the netCDF C++ API.

   This is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial
```

```
   Full documentation of the netCDF C++ API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-cxx

   $Id: pres_temp_4D_wr.cpp,v 1.11 2007/01/19 12:52:13 ed Exp $
*/

#include <iostream>
#include <netcdfcpp.h>

using namespace std;

// We are writing 4D data, a 2 x 6 x 12 lvl-lat-lon grid, with 2
// timesteps of data.
static const int NLVL = 2;
static const int NLAT = 6;
static const int NLON = 12;
static const int NREC = 2;

// These are used to construct some example data.
static const float SAMPLE_PRESSURE = 900.0;
static const float SAMPLE_TEMP = 9.0;
static const float START_LAT = 25.0;
static const float START_LON = -125.0;

// Return this code to the OS in case of failure.
static const int NC_ERR = 2;

int main()
{
   // These arrays will store the latitude and longitude values.
   float lats[NLAT],lons[NLON];

   // These arrays will hold the data we will write out. We will
   // only need enough space to hold one timestep of data; one record.
   float pres_out[NLVL][NLAT][NLON];
   float temp_out[NLVL][NLAT][NLON];

   int i = 0;

   // Create some pretend data. If this wasn't an example program, we
   // would have some real data to write for example, model output.
   for (int lat = 0; lat < NLAT; lat++)
      lats[lat] = START_LAT + 5. * lat;
   for (int lon = 0; lon < NLON; lon++)
      lons[lon] = START_LON + 5. * lon;
```

```
  for (int lvl = 0; lvl < NLVL; lvl++)
     for (int lat = 0; lat < NLAT; lat++)
for (int lon = 0; lon < NLON; lon++)
{
   pres_out[lvl][lat][lon] = SAMPLE_PRESSURE + i;
   temp_out[lvl][lat][lon]  = SAMPLE_TEMP + i++;
}

  // Change the error behavior of the netCDF C++ API by creating an
  // NcError object. Until it is destroyed, this NcError object will
  // ensure that the netCDF C++ API returns error codes on any
  // failure, prints an error message, and leaves any other error
  // handling to the calling program. In the case of this example, we
  // just exit with an NC_ERR error code.
  NcError err(NcError::verbose_nonfatal);

  // Create the file.
  NcFile dataFile("pres_temp_4D.nc", NcFile::Replace);

  // Check to see if the file was created.
  if(!dataFile.is_valid())
     return NC_ERR;

  // Define the dimensions. NetCDF will hand back an ncDim object for
  // each.
  NcDim *lvlDim, *latDim, *lonDim, *recDim;
  if (!(lvlDim = dataFile.add_dim("level", NLVL)))
     return NC_ERR;
  if (!(latDim = dataFile.add_dim("latitude", NLAT)))
     return NC_ERR;
  if (!(lonDim = dataFile.add_dim("longitude", NLON)))
     return NC_ERR;
  // Add an unlimited dimension...
  if (!(recDim = dataFile.add_dim("time")))
     return NC_ERR;

  // Define the coordinate variables.
  NcVar *latVar, *lonVar;
  if (!(latVar = dataFile.add_var("latitude", ncFloat, latDim)))
     return NC_ERR;
  if (!(lonVar = dataFile.add_var("longitude", ncFloat, lonDim)))
     return NC_ERR;

  // Define units attributes for coordinate vars. This attaches a
  // text attribute to each of the coordinate variables, containing
  // the units.
  if (!latVar->add_att("units", "degrees_north"))
```

```
      return NC_ERR;
   if (!lonVar->add_att("units", "degrees_east"))
      return NC_ERR;

   // Define the netCDF variables for the pressure and temperature
   // data.
   NcVar *presVar, *tempVar;
   if (!(presVar = dataFile.add_var("pressure", ncFloat, recDim,
    lvlDim, latDim, lonDim)))
      return NC_ERR;
   if (!(tempVar = dataFile.add_var("temperature", ncFloat, recDim,
    lvlDim, latDim, lonDim)))
      return NC_ERR;

   // Define units attributes for data variables.
   if (!presVar->add_att("units", "hPa"))
      return NC_ERR;
   if (!tempVar->add_att("units", "celsius"))
      return NC_ERR;

   // Write the coordinate variable data to the file.
   if (!latVar->put(lats, NLAT))
      return NC_ERR;
   if (!lonVar->put(lons, NLON))
      return NC_ERR;

   // Write the pretend data. This will write our surface pressure and
   // surface temperature data. The arrays only hold one timestep
   // worth of data. We will just rewrite the same data for each
   // timestep. In a real application, the data would change between
   // timesteps.
   for (int rec = 0; rec < NREC; rec++)
   {
      if (!presVar->put_rec(&pres_out[0][0][0], rec))
 return NC_ERR;
      if (!tempVar->put_rec(&temp_out[0][0][0], rec))
 return NC_ERR;
   }

   // The file is automatically closed by the destructor. This frees
   // up any internal netCDF resources associated with the file, and
   // flushes any buffers.

   cout << "*** SUCCESS writing example file pres_temp_4D.nc!" << endl;
   return 0;
}
```

2.3.4.2 pres_temp_4D_rd.cpp

```
/* This is part of the netCDF package.
   Copyright 2006 University Corporation for Atmospheric Research/Unidata.
   See COPYRIGHT file for conditions of use.

   This is an example which reads some 4D pressure and temperature
   values. The data file read by this program is produced by the
   companion program pres_temp_4D_wr.cpp. It is intended to illustrate
   the use of the netCDF C++ API.

   This program is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

   Full documentation of the netCDF C++ API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-cxx

   $Id: pres_temp_4D_rd.cpp,v 1.13 2007/02/14 20:59:21 ed Exp $
*/

#include <iostream>
#include <netcdfcpp.h>

using namespace std;

// We are writing 4D data, a 2 x 6 x 12 lvl-lat-lon grid, with 2
// timesteps of data.
static const int NLVL = 2;
static const int NLAT = 6;
static const int NLON = 12;
static const int NREC = 2;

// These are used to construct some example data.
static const float SAMPLE_PRESSURE = 900.0;
static const float SAMPLE_TEMP = 9.0;
static const float START_LAT = 25.0;
static const float START_LON = -125.0;

// Return this code to the OS in case of failure.
static const int NC_ERR = 2;

int main()
{
   // These arrays will store the latitude and longitude values.
   float lats[NLAT], lons[NLON];

   // These arrays will hold the data we will read in. We will only
```

```
   // need enough space to hold one timestep of data; one record.
   float pres_in[NLVL][NLAT][NLON];
   float temp_in[NLVL][NLAT][NLON];

   // Change the error behavior of the netCDF C++ API by creating an
   // NcError object. Until it is destroyed, this NcError object will
   // ensure that the netCDF C++ API returns error codes on any
   // failure, prints an error message, and leaves any other error
   // handling to the calling program. In the case of this example, we
   // just exit with an NC_ERR error code.
   NcError err(NcError::verbose_nonfatal);

   // Open the file.
   NcFile dataFile("pres_temp_4D.nc", NcFile::ReadOnly);

   // Check to see if the file was opened.
   if(!dataFile.is_valid())
      return NC_ERR;

   // Get pointers to the latitude and longitude variables.
   NcVar *latVar, *lonVar;
   if (!(latVar = dataFile.get_var("latitude")))
      return NC_ERR;
   if (!(lonVar = dataFile.get_var("longitude")))
      return NC_ERR;

   // Get the lat/lon data from the file.
   if (!latVar->get(lats, NLAT))
      return NC_ERR;
   if (!lonVar->get(lons, NLON))
      return NC_ERR;

   // Check the coordinate variable data.
   for (int lat = 0; lat < NLAT; lat++)
      if (lats[lat] != START_LAT + 5. * lat)
 return NC_ERR;
   for (int lon = 0; lon < NLON; lon++)
      if (lons[lon] != START_LON + 5. * lon)
 return NC_ERR;

   // Get pointers to the pressure and temperature variables.
   NcVar *presVar, *tempVar;
   if (!(presVar = dataFile.get_var("pressure")))
      return NC_ERR;
   if (!(tempVar  = dataFile.get_var("temperature")))
      return NC_ERR;
```

```
   // Read the data. Since we know the contents of the file we know
   // that the data arrays in this program are the correct size to
   // hold one timestep.
   for (int rec = 0; rec < NREC; rec++)
   {
      // Read the data one record at a time.
      if (!presVar->set_cur(rec, 0, 0, 0))
 return NC_ERR;
      if (!tempVar->set_cur(rec, 0, 0, 0))
 return NC_ERR;

      // Get 1 record of NLVL by NLAT by NLON values for each variable.
      if (!presVar->get(&pres_in[0][0][0], 1, NLVL, NLAT, NLON))
 return NC_ERR;
      if (!tempVar->get(&temp_in[0][0][0], 1, NLVL, NLAT, NLON))
 return NC_ERR;

      // Check the data.
      int  i = 0;
      for (int lvl = 0; lvl < NLVL; lvl++)
 for (int lat = 0; lat < NLAT; lat++)
    for (int lon = 0; lon < NLON; lon++)
       if (pres_in[lvl][lat][lon] != SAMPLE_PRESSURE + i ||
   temp_in[lvl][lat][lon] != SAMPLE_TEMP + i++)
  return NC_ERR;
   } // next record

   // The file is automatically closed by the destructor. This frees
   // up any internal netCDF resources associated with the file, and
   // flushes any buffers.

   cout << "*** SUCCESS reading example file pres_temp_4D.nc!" << endl;
   return 0;
}
```

3 The Functions You Need in NetCDF-3

The netCDF-3 C and Fortran APIs each have over 100 functions, but most users need only a handful. Listed below are the essential netCDF functions for four important tasks in netCDF: creating new files, reading existing files, learning about a netCDF file of unknown structure, and reading and writing subsets of data.

In each case the functions are presented for each of the four language APIs: C, Fortran 77, Fortran 90, and C++, with hyper-links to the detailed documentation of each function.

3.1 Creating New Files and Metadata, an Overview

To construct a netCDF file you need to:

`create the file`
: Specify the name, optionally the format: classic (the default) or 64bit-offset.

`define metadata`
: Specify the names and types of dimensions, data variables, and attributes.

`write data`
: Write arrays of data from program variables to the netCDF file. Arrays of data may be written all at once, or in subsets.

`close the file`
: Close the file to flush all buffers to the disk and free all resources allocated for this file.

3.1.1 Creating a NetCDF File in C

Use nc_create to create a file. Then use nc_def_dim to define each shared dimension. The data variables are then specified with nc_def_var. Any attributes are added with nc_put_att. Finally, call nc_enddef to tell the library that you are done defining the metadata, and ready to start writing the data.

After all data are written to the file, call nc_close to ensure that all buffers are flushed, and any resources associated with the open file are returned to the operating system.

For a very simple example, See Section 2.1.1 [simple_xy in C], page 10.

For a typical sequence of calls to the C versions of these functions, see See Section "Creating a NetCDF Dataset" in *The NetCDF C Interface Guide*.

Section "nc_create" in *The NetCDF C Interface Guide*	create a new netCDF file
Section "nc_def_dim" in *The NetCDF C Interface Guide*	define a dimension
Section "nc_def_var" in *The NetCDF C Interface Guide*	define a variable
Section "nc_put_att_ type" in *The NetCDF C Interface Guide*	write attributes
Section "nc_enddef" in *The NetCDF C Interface Guide*	leave define mode

Section "nc_put_vara_ type" in *The NetCDF C Interface Guide*	write arrays of data
Section "nc_close" in *The NetCDF C Interface Guide*	close a file

3.1.2 Creating a NetCDF File in Fortran 77

Use NF_CREATE to create a file. Then use NF_DEF_DIM to define each shared dimension. The data variables are then specified with NF_DEF_VAR. Any attributes are added with NF_PUT_ATT. Finally, call NF_ENDDEF to tell the library that you are done defining the metadata, and ready to start writing the data.

After all data are written to the file, call NF_CLOSE to ensure that all buffers are flushed, and any resources associated with the open file are returned to the operating system.

For a typical sequence of calls see Section "Creating a NetCDF Dataset" in *The NetCDF Fortran 77 Interface Guide*.

Fortran users take note: the netCDF Fortran 77 API consists of wrappers around the functions of the netCDF C library. There is no Fortran 77 code in netCDF except for these wrappers, and tests to ensure that the wrappers work.

The name of each Fortran function shows the outline of the C function it wraps (for example, NF_CREATE is a wrapper around nc_create).

Section "NF_CREATE" in *The NetCDF Fortran 77 Interface Guide*	create a new netCDF file
Section "NF_DEF_DIM" in *The NetCDF Fortran 77 Interface Guide*	define a dimension
Section "NF_DEF_VAR" in *The NetCDF Fortran 77 Interface Guide*	define a variable
Section "NF_PUT_ATT_ type" in *The NetCDF Fortran 77 Interface Guide*	write an attribute
Section "NF_ENDDEF" in *The NetCDF Fortran 77 Interface Guide*	end define mode
Section "NF_PUT_VARA_ type" in *The NetCDF Fortran 77 Interface Guide*	write arrays of data
Section "NF_CLOSE" in *The NetCDF Fortran 77 Interface Guide*	close the netCDF file

3.1.3 Creating a NetCDF File in Fortran 90

Use NF90_CREATE to create a file. Then use NF90_DEF_DIM to define each shared dimension. The data variables are then specified with NF90_DEF_VAR. Any attributes are added with NF90_PUT_ATT. Finally, call NF90_ENDDEF to tell the library that you are done defining the metadata, and ready to start writing the data.

After all data are written to the file, call NF90_CLOSE to ensure that all buffers are flushed, and any resources associated with the open file are returned to the operating system.

For a typical sequence of calls see Section "Creating a NetCDF Dataset" in *The NetCDF Fortran 90 Interface Guide*.

The netCDF Fortran 90 API calls the Fortran 77 API, which in turn calls the netCDF C library.

The name of each Fortran function shows the outline of the F77 function it wraps (for example, NF90_CREATE is a wrapper around NF_CREATE). The F77 functions are, in turn, wrappers around the C functions.

Section "NF90_CREATE" in *The NetCDF Fortran 90 Interface Guide*	create a netCDF file
Section "NF90_DEF_DIM" in *The NetCDF Fortran 90 Interface Guide*	define a dimension
Section "NF90_DEF_VAR" in *The NetCDF Fortran 90 Interface Guide*	define a variable
Section "NF90_PUT_ATT_ type" in *The NetCDF Fortran 90 Interface Guide*	write an attribute
Section "NF90_ENDDEF" in *The NetCDF Fortran 90 Interface Guide*	end define mode
Section "NF90_PUT_VARA_ type" in *The NetCDF Fortran 90 Interface Guide*	write arrays of data
Section "NF90_CLOSE" in *The NetCDF Fortran 90 Interface Guide*	close the netCDF file

3.1.4 Creating a NetCDF File in C++

Create an instance of the NcFile class to create a netCDF file. Use its add_dim and add_var methods to add dimensions and variables. The add_att method is available for both NcFile and NcVar.

Use the NcError class to specify error handling behavior.

For an example creating a simple file see Section 2.1.4.1 [simple_xy_wr.cpp], page 22. For a more complex example see Section 2.3.4.1 [pres_temp_4D_wr.cpp], page 80.

Section "Class NcFile" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF files
Section "Class NcDim" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF dimensions
Section "Class NcVar" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF variables
Section "Class NcAtt" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF attributes
Section "Class NcError" in *The NetCDF C++ Interface Guide*	a C++ class to control netCDF error handling

3.2 Reading NetCDF Files of Known Structure

To read a netCDF file of known structure, you need to:

`open the file`
: Specify the file name and whether you want read-write or read-only access.

`read variable or attribute data`
: Read the data or attributes of interest.

`close the file`
: Release all resources associated with this file.

Use ncdump to learn the structure of a file (use the -h option). For more information about ncdump see Section "NetCDF Utilities" in *The NetCDF Users Guide*.

3.2.1 Numbering of NetCDF IDs

In C, Fortran 77, and Fortran 90, netCDF objects are identified by an integer: the ID. NetCDF functions use this ID to identify the object. It's helpful for the programmer to understand these IDs.

Open data files, dimensions, variables, and attributes are each numbered independently, and are always numbered in the order in which they were defined. (They also appear in this order in ncdump output.) Numbering starts with 0 in C, and 1 in Fortran 77/90.

For example, the first variable defined in a file will have an ID of 0 in C programs, and 1 in Fortran programs, and functions that apply to a variable will need to know the ID of the variable you mean.

(The numbering of files is an exception: file IDs are assigned by the operating system when a file is opened, and are not permanently associated with the file. IDs for netCDF dimensions and variables are persistent, but deleting an attribute changes subsequent attribute numbers.)

Although netCDF refers to everything by an integer id (varid, dimid, attnum), there are inquiry functions which, given a name, will return an ID. For example, in the C API, nc_inq_varid will take a character string (the name), and give back the ID of the variable of that name. The variable ID is then used in subsequent calls (to read the data, for example).

Other inquiry functions exist to further describe the file. (see Section 3.3 [Inquiry Functions], page 92).

3.2.2 Reading a Known NetCDF File in C

For a typical sequence of calls to these C functions see Section "Reading a NetCDF Dataset with Known Names" in *The NetCDF C Interface Guide*.

Section "nc_open" in *The NetCDF C Interface Guide*	open a netCDF file

Section "nc_get_att" in *The NetCDF C Interface Guide*	read an attribute
Section "nc_get_vara_ type" in *The NetCDF C Interface Guide*	read arrays of data
Section "nc_close" in *The NetCDF C Interface Guide*	close the file

3.2.3 Reading a Known NetCDF File in Fortran 77

For a typical sequence of calls to these functions see Section "Reading a NetCDF Dataset with Known Names" in *The NetCDF Fortran 77 Interface Guide*.

Section "NF_OPEN" in *The NetCDF Fortran 77 Interface Guide*	open a netCDF file
Section "NF_GET_ATT" in *The NetCDF Fortran 77 Interface Guide*	read an attribute
Section "NF_GET_VARA_ type" in *The NetCDF Fortran 77 Interface Guide*	read arrays of data
Section "NF_CLOSE" in *The NetCDF Fortran 77 Interface Guide*	close the file

3.2.4 Reading a Known NetCDF File in Fortran 90

For a typical sequence of calls to these functions see Section "Reading a NetCDF Dataset with Known Names" in *The NetCDF Fortran 90 Interface Guide*.

Section "NF90_OPEN" in *The NetCDF Fortran 90 Interface Guide*	open a netCDF file
Section "NF90_GET_ATT" in *The NetCDF Fortran 90 Interface Guide*	read an attribute
Section "NF90_GET_VARA" in *The NetCDF Fortran 90 Interface Guide*	read arrays of data
Section "NF90_CLOSE" in *The NetCDF Fortran 90 Interface Guide*	close the file

3.2.5 Reading a Known NetCDF File in C++

Section "Class NcFile" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF files
Section "Class NcDim" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF dimensions
Section "Class NcVar" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF variables
Section "Class NcAtt" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF attributes

3.3 Reading NetCDF Files of Unknown Structure

Perhaps you would like to write your software to handle more general cases, so that you don't have to adjust your source every time the grid size changes, or a variable is added to the file.

There are inquiry functions that let you find out everything you need to know about a file. These functions contain "inq" or "INQ" in their names.

Using the inquiry functions, it is possible to write code that will read and understand any netCDF file, whatever its contents. (For example, ncdump does just that.)

3.3.1 Inquiry in C

First use nc_inq, which will tell you how many variables and global attributes there are in the file.

Start with global attribute 0, and proceed to natts - 1, the number of global attributes minus one. The nc_inq_att function will tell you the name, type, and length of each global attribute.

Then start with dimid 0, and proceed to dimid ndims - 1, calling nc_inq_dim. This will tell you the name and length of each dimension, and whether it is unlimited.

Then start with varid 0, and proceed to varid nvars - 1, calling nc_inq_var. This will tell you the number of dimensions of this variable, and their associated IDs. It will also get the name and type of this variable, and whether there are any attributes attached. If there are attributes attached, use the nc_inq_att function to get their names, types, and lengths.

(To read an attribute, use the appropriate nc_get_att_<TYPE> function, like nc_get_att_int() to get the data from an attribute that is an array of integers.)

There are also functions that return an item's ID, given its name. To find IDs from the names, use functions nc_inq_dimid, nc_inq_attnum, and nc_inq_varid.

For a typical sequence of calls to these functions see Section "Reading a netCDF Dataset with Unknown Names" in *The NetCDF C Interface Guide*.

3.3.1.1 NULL Parameters in Inquiry Functions

With any of the C inquiry functions, a NULL pointer can be used to ignore a return parameter. Consider the nc_inq function:

```
EXTERNL int
nc_inq(int ncid, int *ndimsp, int *nvarsp, int *nattsp, int *unlimdimidp);
```

If you call this with NULL for the last three parameters, you can learn the number of dimensions without bothering about the number of variables, number of global attributes, and the ID of the unlimited dimension.

For further convenience, we provide functions like nc_inq_ndims, which only finds the number of dimensions, exactly as if you had called nc_inq, with NULLs in all parameters except ndimsp. (In fact, this is just what the nc_inq_ndims functions does).

Section "nc_inq" in *The NetCDF C Interface Guide*	Find number of dimensions, variables, and global attributes, and the unlimited dimid.
Section "nc_inq_att" in *The NetCDF C Interface Guide*	Find attribute name, type, and length.
Section "nc_inq_dim Family" in *The NetCDF C Interface Guide*	Find dimension name and length.

Section "nc_inq_var" in *The NetCDF C Interface Guide*	Find variable name, type, num dimensions, dim IDs, and num attributes.
Section "nc_inq_dimid" in *The NetCDF C Interface Guide*	Find dimension ID from its name.
Section "nc_inq_varid" in *The NetCDF C Interface Guide*	Find variable ID from its name.
Section "nc_inq_format" in *The NetCDF C Interface Guide*	Find file format: classic or 64-bit offset
Section "nc_inq_libvers" in *The NetCDF C Interface Guide*	Find the netCDF version. (Currently 4.1.1).

3.3.2 Inquiry in Fortran 77

First use NF_INQ, which will tell you how many variables and global attributes there are in the file. Then start with varid 1, and proceed to varid nvars, calling NF_INQ_VAR.

For a typical sequence of calls to these functions see Section "Reading a netCDF Dataset with Unknown Names" in *The NetCDF Fortran 77 Interface Guide*.

Section "NF_INQ" in *The NetCDF Fortran 77 Interface Guide*.	Find number of dimensions, variables, and global attributes, and the unlimited dimid.

Section "NF_INQ_DIM" in *The NetCDF Fortran 77 Interface Guide*.	Find dimension name and length.
Section "NF_INQ_VARID" in *The NetCDF Fortran 77 Interface Guide*.	Find variable ID from its name.
Section "NF_INQ_VAR" in *The NetCDF Fortran 77 Interface Guide*.	Find variable name, type, num dimensions, dim IDs, and num attributes.
Section "NF_INQ_DIMID" in *The NetCDF Fortran 77 Interface Guide*.	Find dimension ID from its name.
Section "NF_INQ_DIM" in *The NetCDF Fortran 77 Interface Guide*.	Find dimension name and length.
Section "NF_INQ_ATT" in *The NetCDF Fortran 77 Interface Guide*.	Find attribute name, type, and length.
Section "NF_INQ_FORMAT" in *The NetCDF Fortran 77 Interface Guide*.	Find file format: classic or 64-bit offset
Section "NF_INQ_LIBVERS" in *The NetCDF Fortran 77 Interface Guide*.	Find the netCDF version. (Currently 4.1.1).

3.3.3 Inquiry in Fortran 90

First use NF90_INQ, which will tell you how many variables and global attributes there are in the file. Then start with varid 1, and proceed to varid nvars, calling NF90_INQ_VAR.

For a typical sequence of calls to these functions, see See Section "Reading a netCDF Dataset with Unknown Names" in *The NetCDF Fortran 90 Interface Guide*.

Section "NF90_INQ" in *The NetCDF Fortran 90 Interface Guide*.	Find number of dimensions, variables, and global attributes, and the unlimited dimid.

Section "NF90_INQ_DIM" in *The NetCDF Fortran 90 Interface Guide*.	Find dimension name and length.
Section "NF90_INQ_VARID" in *The NetCDF Fortran 90 Interface Guide*.	Find variable ID from its name.
Section "NF90_INQ_VAR" in *The NetCDF Fortran 90 Interface Guide*.	Find variable name, type, num dimensions, dim IDs, and num attributes.
Section "NF90_INQ_DIMID" in *The NetCDF Fortran 90 Interface Guide*.	Find dimension ID from its name.
Section "NF90_INQ_DIM" in *The NetCDF Fortran 90 Interface Guide*.	Find dimension name and length.
Section "NF90_INQ_ATT" in *The NetCDF Fortran 90 Interface Guide*.	Find attribute name, type, and length.
Section "NF90_INQ_FORMAT" in *The NetCDF Fortran 90 Interface Guide*.	Find file format: classic or 64-bit offset
Section "NF90_INQ_LIBVERS" in *The NetCDF Fortran 90 Interface Guide*.	Find the netCDF version. (Currently 4.1.1).

3.3.4 Inquiry Functions in the C++ API

Section "Class NcFile" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF files

Section "Class NcDim" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF dimensions
Section "Class NcVar" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF variables
Section "Class NcAtt" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF attributes

3.4 Reading and Writing Subsets of Data

Usually users are interested in reading or writing subsets of variables in a netCDF data file. The netCDF APIs provide a variety of functions for this purpose.

In the simplest case, you will use the same type for both file and in-memory storage, but in some cases you may wish to use different types. For example, you might have a netCDF file that contains integer data, and you wish to read it into floating-point storage, converting the data as it is read. The same sort of type conversion can be done when writing the data.

To convert to a type while reading data, use the appropriate nc_get_vara_<TYPE> or NF_GET_VARA_<TYPE> function. For example, the C function nc_get_vara_float(), and the Fortran function NF_GET_VARA_REAL will read netCDF data of any numeric type into a floating-point array, automatically converting each element to the desired type.

To convert from a type while writing data, use the appropriate nc_put_vara_<TYPE> or NF_PUT_VARA_<TYPE> function. For example, the C function nc_put_vara_float(), and the Fortran function NC_PUT_VARA_REAL will write floating-point data into netCDF arrays, automatically converting each element of the array to the type of the netCDF variable.

The <TYPE> in the function name refers to the type of the in-memory data, in both cases. They type of the file data is determined when the netCDF variable is defined.

3.4.1 Reading and Writing Subsets of Data in C

The type of the data may be automatically converted on read or write. For more information about type conversion see Section "Type Conversion" in *The NetCDF C Interface Guide*.

Read the entire variable at once	Section "nc_get_var_ type" in *The NetCDF C Interface Guide*
Write the entire variable at once	Section "nc_put_var_ type" in *The NetCDF C Interface Guide*
Read just one value	Section "nc_get_var1_ type" in *The NetCDF C Interface Guide*
Write just one value	Section "nc_put_var1_ type" in *The NetCDF C Interface Guide*
Read an array subset	Section "nc_get_vara_ type" in *The NetCDF C Interface Guide*
Write an array subset	Section "nc_put_vara_ type" in *The NetCDF C Interface Guide*
Read an array with strides	Section "nc_get_vars_ type" in *The NetCDF C Interface Guide*
Write an array with strides	Section "nc_put_vars_ type" in *The NetCDF C Interface Guide*

3.4.2 Reading and Writing Subsets of Data in Fortran 77

The type of the data may be automatically converted on read or write. For more information about type conversion see Section "Type Conversion" in *The NetCDF Fortran 77 Interface Guide*.

Read the entire variable at once	Section "NF_GET_VAR_type" in *The NetCDF Fortran 77 Interface Guide*
Write the entire variable at once	Section "NF_PUT_VAR_type" in *The NetCDF Fortran 77 Interface Guide*
Read just one value	Section "NF_GET_VAR1_type" in *The NetCDF Fortran 77 Interface Guide*
Write just one value	Section "NF_PUT_VAR1_type" in *The NetCDF Fortran 77 Interface Guide*
Read an array subset	Section "NF_GET_VARA_type" in *The NetCDF Fortran 77 Interface Guide*
Write an array subset	Section "NF_PUT_VARA_type" in *The NetCDF Fortran 77 Interface Guide*
Read an array with strides	Section "NF_GET_VARS_type" in *The NetCDF Fortran 77 Interface Guide*

Write an array with strides	Section "NF_PUT_VARS_type" in *The NetCDF Fortran 77 Interface Guide*

3.4.3 Reading and Writing Subsets of Data in Fortran 90

The type of the data may be automatically converted on read or write. For more information about type conversion see Section "Type Conversion" in *The NetCDF Fortran 90 Interface Guide*.

Read the entire variable at once	Section "NF90_GET_VAR_type" in *The NetCDF Fortran 90 Interface Guide*
Write the entire variable at once	Section "NF90_PUT_VAR_type" in *The NetCDF Fortran 90 Interface Guide*
Read just one value	Section "NF90_GET_VAR1_type" in *The NetCDF Fortran 90 Interface Guide*
Write just one value	Section "NF90_PUT_VAR1_type" in *The NetCDF Fortran 90 Interface Guide*
Read an array subset	Section "NF90_GET_VARA_type" in *The NetCDF Fortran 90 Interface Guide*

Write an array subset	Section "NF90_PUT_VARA_type" in *The NetCDF Fortran 90 Interface Guide*
Read an array with strides	Section "NF90_GET_VARS_type" in *The NetCDF Fortran 90 Interface Guide*
Write an array with strides	Section "NF90_PUT_VARS_type" in *The NetCDF Fortran 90 Interface Guide*

3.4.4 Reading and Writing Subsets of Data in C++

To read a record of data at a time, use the set_cur method of the NcVar class to set the number of the record of interest, and then use the get method to read the record.

Section "Class NcVar" in *The NetCDF C++ Interface Guide*	a C++ class to manipulate netCDF variables, use the set_cur and get methods to read records from a file.

4 API Extensions Introduced with NetCDF-4

NetCDF-4 includes many advanced features. These features are only available when working with files created in the netCDF format. (That is, HDF5 files, created by netCDF, or simple-model HDF5 files).

4.1 Interoperability with HDF5

NetCDF-4 allows some interoperability with HDF5.

4.1.1 Reading and Editing NetCDF-4 Files with HDF5

The HDF5 Files produced by netCDF-4 are perfectly respectable HDF5 files, and can be read by any HDF5 application.

NetCDF-4 relies on several new features of HDF5, including dimension scales. The HDF5 dimension scales feature adds a bunch of attributes to the HDF5 file to keep track of the dimension information.

It is not just wrong, but wrong-headed, to modify these attributes except with the HDF5 dimension scale API. If you do so, then you will deserve what you get, which will be a mess.

Additionally, netCDF stores some extra information for dimensions without dimension scale information. (That is, a dimension without an associated coordinate variable). So HDF5 users should not write data to a netCDF-4 file which extends any unlimited dimension.

Also there are some types allowed in HDF5, but not allowed in netCDF-4 (for example the time type). Using any such type in a netCDF-4 file will cause the file to become unreadable to netCDF-4. So don't do it.

NetCDF-4 ignores all HDF5 references. Can't make head nor tail of them. Also netCDF-4 assumes a strictly hierarchical group structure. No looping, you weirdo!

Attributes can be added (they must be one of the netCDF-4 types), modified, or even deleted, in HDF5.

4.1.2 Reading and Editing HDF5 Files with NetCDF-4

Assuming a HDF5 file is written in accordance with the netCDF-4 rules (i.e. no strange types, no looping groups), and assuming that *every* dataset has a dimension scale attached to each dimension, the netCDF-4 API can be used to read and edit the file.

In HDF5 (version 1.8.0 and later), dimension scales are (generally) 1D datasets, that hold dimension data. A multi-dimensional dataset can then attach a dimension scale to any or all of its dimensions. For example, a user might have 1D dimension scales for lat and lon, and a 2D dataset which has lat attached to the first dimension, and lon to the second.

Dimension scales are vital to netCDF-4, which uses shared dimensions. If you want to read a HDF5 file with netCDF-4, it must use dimension scales, and one dimension scale must be attached to each dimension of every dataset in the file.

4.2 Multiple Unlimited Dimensions

With classic and 64-bit offset netCDF files, each variable may use at most one unlimited dimension. With netCDF-4 format files, this restriction is lifted.

Simply define as many unlimited dimensions as you wish, and use them in a variable. When data are written to that variable, the dimensions will be expanded as needed.

4.3 Groups

NetCDF-4 files can store attributes, variables, and dimensions in hierarchical groups.

This allows the user to create a structure much like a Unix file system. In netCDF, each group gets an ncid. Opening or creating a file returns the ncid for the root group (which is named "/"). Groups can be added with the nc_def_grp function. Get the number of groups, and their ncids, with the nc_inq_grps function.

Dimensions are scoped such that they are visible to all child groups. For example, you can define a dimension in the root group, and use its dimension id when defining a variable in a sub-group.

Attributes defined as NC_GLOBAL apply to the group, not the entire file.

The degenerate case, in which only the root group is used, corresponds exactly with the classic data mode, before groups were introduced.

4.4 Compound Types

In netCDF-4 files it's possible to create a data type which corresponds to a C struct. These are known as "compound" types (following HDF5 nomenclature).

That is, a netCDF compound type is a data structure which contains an arbitrary collection of other data types, including other compound types.

To define a new compound type, use nc_def_compound. Then call nc_insert_compound for each type within the compound type.

Read and write arrays of compound data with the nc_get_vara and nc_put_vara functions. These functions were actually part of the netCDF-2 API, brought out of semi-retirement to handle user-defined types in netCDF-4.

4.5 Opaque Types

Store blobs of bits in opaque types. Create an opaque type with nc_def_opaque. Read and write them with nc_get_vara/nc_put_vara.

4.6 Variable Length Arrays (VLEN)

Create a VLEN type to store variable length arrays of a known base type. Use nc_def_vlen to define a VLEN type, read and write them with nc_get_vara/nc_put_vara.

4.7 Strings

Use the NC_STRING type to store arrays of strings. Read and write them with nc_get_vara/nc_put_vara.

4.8 New Inquiry Functions

There are many new inquiry functions to allow a program to navigate a completely unknown netCDF file.

To find the number To find all the dimensions visible from a group, use nc_inq_dimids.

4.9 Parallel I/O with NetCDF

Parallel I/O allows many processes to read/write netCDF data at the same time. Used properly, it allows users to overcome I/O bottlenecks in high performance computing environments.

4.9.1 Parallel I/O Choices for NetCDF Users

Parallel read-only access can be achieved netCDF files using the netCDF C/Fortran library. Each process can run a copy of the netCDF library and open and read any subsets of the data in the file. This sort of "fseek parallelism" will break down dramatically for any kind of writing.

There are two methods available to users for read/write parallel I/O netCDF-4 or the parallel netCDF package from Argonne/Northwestern. Unfortunately the two methods involve different APIs, and different binary formats.

For parallel read/write access to classic and 64-bit offset data users must use the parallel-netcdf library from Argonne/Northwestern University. This is not a Unidata software package, but was developed using the Unidata netCDF C library as a starting point. For more information see the parallel netcdf web site: http://www.mcs.anl.gov/parallel-netcdf.

For parallel read/write access to netCDF-4/HDF5 files users must use the netCDF-4 API. The Argonne/Northwestern parallel netcdf package cannot read netCDF-4/HDF5 files.

4.9.2 Parallel I/O with NetCDF-4

NetCDF-4 provides access to HDF5 parallel I/O features for netCDF-4/HDF5 files. NetCDF classic and 64-bit offset format may not be opened or created for use with parallel I/O. (They may be opened and created, but parallel I/O is not available.)

A few functions have been added to the netCDF C API to handle parallel I/O. These functions are also available in the Fortran 90 and Fortran 77 APIs.

4.9.2.1 Building NetCDF-4 for Parallel I/O

You must build netCDF-4 properly to take advantage of parallel features.

For parallel I/O HDF5 must be built with –enable-parallel. Typically the CC environment variable is set to mpicc. You must build HDF5 and netCDF-4 with the same compiler and compiler options.

The netCDF configure script will detect the parallel capability of HDF5 and build the netCDF-4 parallel I/O features automatically. No configure options to the netcdf configure are required. If the Fortran APIs are desired set environmental variable FC to mpif90 (or some local variant.)

4.9.2.2 Opening/Creating Files for Parallel I/O

The nc_open_par and nc_create_par functions are used to create/open a netCDF file with the C API. (Or use nf_open_par/nf_create_par from Fortran 77).

For Fortran 90 users the nf90_open and nf90_create calls have been modified to permit parallel I/O files to be opened/created using optional parameters comm and info.

The parallel access associated with these functions is not a characteristic of the data file, but the way it was opened.

4.9.2.3 Collective/Independent Access

Parallel file access is either collective (all processors must participate) or independent (any processor may access the data without waiting for others).

All netCDF metadata writing operations are collective. That is, all creation of groups, types, variables, dimensions, or attributes.

Data reads and writes (ex. calls to nc_put_vara_int and nc_get_vara_int) may be independent (the default) or collective. To make writes to a variable collective, call the nc_var_par_access function (or nf_var_par_access for Fortran 77 users, or nf90_var_par_access for Fortran 90 users).

The example program below demonstrates simple parallel writing and reading of a netCDF file.

4.9.3 simple_xy_par_wr.c and simple_xy_par_rd.c

For this release, only a Fortran 90 language version of this example is provided. Other APIs will be demonstrated in examples in future releases.

In the simple_xy_par_wr example program an num_procs x num_procs array is written to the disk, where num_proc is the number of processors on which this program is run. Each processor writes one row of length num_proc.

In the simple_xy_par_rd program the file is read in, and each processor expects to read in a row with its own MPI rank stored. (The read program must be run on no more processors than were used to create the file.)

4.9.3.1 simple_xy_par_wr.f90

```
!     This is part of the netCDF package.
!     Copyright 2006 University Corporation for Atmospheric Research/Unidata.
!     See COPYRIGHT file for conditions of use.

!     This is a very simple example which writes a 2D array of sample
!     data. To handle this in netCDF we create two shared dimensions,
!     "x" and "y", and a netCDF variable, called "data". It uses
!     parallel I/O to write the file from all processors at the same
!     time.

!     This example demonstrates the netCDF Fortran 90 API. This is part
!     of the netCDF tutorial, which can be found at:
!     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial
```

```
!     Full documentation of the netCDF Fortran 90 API can be found at:
!     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90

!     $Id: simple_xy_par_wr.f90,v 1.2 2009/03/12 18:29:48 ed Exp $

program simple_xy_par_wr
  use netcdf
  implicit none
  include 'mpif.h'

  ! This is the name of the data file we will create.
  character (len = *), parameter :: FILE_NAME = "simple_xy_par.nc"

  ! We are writing 2D data.
  integer, parameter :: NDIMS = 2

  ! When we create netCDF files, variables and dimensions, we get back
  ! an ID for each one.
  integer :: ncid, varid, dimids(NDIMS)
  integer :: x_dimid, y_dimid

  ! These will tell where in the data file this processor should
  ! write.
  integer :: start(NDIMS), count(NDIMS)

  ! This is the data array we will write. It will just be filled with
  ! the rank of this processor.
  integer, allocatable :: data_out(:)

  ! MPI stuff: number of processors, rank of this processor, and error
  ! code.
  integer :: p, my_rank, ierr

  ! Loop indexes, and error handling.
  integer :: x, y, stat

  ! Initialize MPI, learn local rank and total number of processors.
  call MPI_Init(ierr)
  call MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)
  call MPI_Comm_size(MPI_COMM_WORLD, p, ierr)

  ! Create some pretend data. We just need one row.
  allocate(data_out(p), stat = stat)
  if (stat .ne. 0) stop 3
  do x = 1, p
     data_out(x) = my_rank
```

```
  end do

  ! Create the netCDF file. The NF90_NETCDF4 flag causes a
  ! HDF5/netCDF-4 file to be created. The comm and info parameters
  ! cause parallel I/O to be enabled.
  call check( nf90_create(FILE_NAME, NF90_NETCDF4, ncid, comm = MPI_COMM_WORLD, &
       info = MPI_INFO_NULL) )

  ! Define the dimensions. NetCDF will hand back an ID for
  ! each. Metadata operations must take place on all processors.
  call check( nf90_def_dim(ncid, "x", p, x_dimid) )
  call check( nf90_def_dim(ncid, "y", p, y_dimid) )

  ! The dimids array is used to pass the IDs of the dimensions of
  ! the variables. Note that in fortran arrays are stored in
  ! column-major format.
  dimids =  (/ y_dimid, x_dimid /)

  ! Define the variable. The type of the variable in this case is
  ! NF90_INT (4-byte integer).
  call check( nf90_def_var(ncid, "data", NF90_INT, dimids, varid) )

  ! End define mode. This tells netCDF we are done defining
  ! metadata. This operation is collective and all processors will
  ! write their metadata to disk.
  call check( nf90_enddef(ncid) )

  ! Write the pretend data to the file. Each processor writes one row.
  start = (/ 1, my_rank + 1/)
  count = (/ p, 1 /)
  call check( nf90_put_var(ncid, varid, data_out, start = start, &
       count = count) )

  ! Close the file. This frees up any internal netCDF resources
  ! associated with the file, and flushes any buffers.
  call check( nf90_close(ncid) )

  ! Free my local memory.
  deallocate(data_out)

  ! MPI library must be shut down.
  call MPI_Finalize(ierr)

  if (my_rank .eq. 0) print *, "*** SUCCESS writing example file ", FILE_NAME, "! "

contains
  subroutine check(status)
```

```
    integer, intent ( in) :: status

    if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status))
      stop 2
    end if
  end subroutine check
end program simple_xy_par_wr
```

4.9.3.2 simple_xy_par_rd.f90

```
! This is part of the netCDF package.
! Copyright 2008 University Corporation for Atmospheric Research/Unidata.
! See COPYRIGHT file for conditions of use.

! This is a simple example which reads a small dummy array, from a
! netCDF data file created by the companion program
! simple_xy_par_wr.f90. The data are read using parallel I/O.

! This is intended to illustrate the use of the netCDF fortran 90
! API. This example program is part of the netCDF tutorial, which can
! be found at:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

! Full documentation of the netCDF Fortran 90 API can be found at:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90

! $Id: simple_xy_par_rd.f90,v 1.2 2009/03/12 18:30:41 ed Exp $

program simple_xy_par_rd
  use netcdf
  implicit none
  include 'mpif.h'

  ! This is the name of the data file we will read.
  character (len = *), parameter :: FILE_NAME = "simple_xy_par.nc"

  ! These will tell where in the data file this processor should
  ! write.
  integer, parameter :: NDIMS = 2
  integer :: start(NDIMS), count(NDIMS)

  ! We will read data into this array.
  integer, allocatable :: data_in(:)

  ! This will be the netCDF ID for the file and data variable.
  integer :: ncid, varid
```

```
! MPI stuff: number of processors, rank of this processor, and error
! code.
integer :: p, my_rank, ierr

! Loop indexes, and error handling.
integer :: x, y, stat

! Initialize MPI, learn local rank and total number of processors.
call MPI_Init(ierr)
call MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)
call MPI_Comm_size(MPI_COMM_WORLD, p, ierr)

! Allocate space to read in data.
allocate(data_in(p), stat = stat)
if (stat .ne. 0) stop 3

! Open the file. NF90_NOWRITE tells netCDF we want read-only access to
! the file.
call check( nf90_open(FILE_NAME, NF90_NOWRITE, ncid, comm = MPI_COMM_WORLD, &
     info = MPI_INFO_NULL) )

! Get the varid of the data variable, based on its name.
call check( nf90_inq_varid(ncid, "data", varid) )

! Read the data.
start = (/ 1, my_rank + 1/)
count = (/ p, 1 /)
call check( nf90_get_var(ncid, varid, data_in, &
     start = start, count = count) )

! Check the data.
do x = 1, p
   if (data_in(x) .ne. my_rank) then
      print *, "data_in(", x, ") = ", data_in(x)
      stop "Stopped"
   endif
end do

! Close the file, freeing all resources.
call check( nf90_close(ncid) )

! Free my local memory.
deallocate(data_in)

! MPI library must be shut down.
call MPI_Finalize(ierr)
```

```
  if (my_rank .eq. 0) print *,"*** SUCCESS reading example file ", FILE_NAME, "! "

contains
  subroutine check(status)
    integer, intent ( in) :: status

    if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status))
      stop 2
    end if
  end subroutine check
end program simple_xy_par_rd
```

4.10 The Future of NetCDF

NetCDF continues under active development at Unidata (see `http://www.unidata.ucar.edu`).

The next few releases of netCDF will include:

1. A new C++ API which has better error handling and handles netCDF-4 advanced features, such as groups and compound types.
2. Remote access to files stored on a DAP server.
3. Bundled packaging with udunits and other useful tools.
4. More documentation, more examples, more tests, and more fun!

5 NetCDF-4 Examples

Any existing netCDF applications can be converted to generate netCDF-4/HDF5 files. Simply change the file creation call to include the correct mode flag.

For example, in one of the C examples which write a data file, change the nc_create call so that NC_NETCDF4 is one of the flags set on the create.

The corresponding read example will work without modification; netCDF will notice that the file is a NetCDF-4/HDF5 file, and will read it automatically, just as if it were a netCDF classic format file.

In the example in this section we show some of the advanced features of netCDF-4. More examples will be added in future releases.

5.1 The simple_nc4 Example

This example, like the simple_xy netCDF-3 example above, is an overly simplified example which demonstrates how to use groups in a netCDF-4 file.

This example is only available in C for this version of netCDF-4. The example creates and then reads the file "simple_nc4.nc."

The simple_xy.nc data file contains two dimensions, "x" and "y", two groups, "grp1" and "grp2", and two data variables, one in each group, both named: "data." One data variable is an unsigned 64-bit integer, the other a user-defined compound type.

The example program simple_nc4_wr.c creates the example data file simple_nc4.nc. The example program simple_nc4_rd.c reads the data file.

5.1.1 simple_nc4_wr.c and simple_nc4_rd.c

For this release, only a C language version of this example is provided. Other APIs will be demonstrated in examples in future releases.

5.1.1.1 simple_nc4_wr.c

```
/* This is part of the netCDF-4 package. Copyright 2007 University
   Corporation for Atmospheric Research/Unidata.  See COPYRIGHT file
   for conditions of use.

   This is a very simple example which demonstrates some of the
   new features of netCDF-4.0.

   We create two shared dimensions, "x" and "y", in a parent group,
   and some netCDF variables in different subgroups. The variables
   will include a compound and an enum type, as well as some of the
   new atomic types, like the unsigned 64-bit integer.

   This example demonstrates the netCDF-4 C API. This is part of the
   netCDF tutorial, which can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial.html

   To understand this example program, users should have a good
```

```
   understanding of the netCDF-3 API. See the example program
   simple_xy_wr.c for a netCDF-3 example.

   Full documentation of the netCDF-4 C API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c.html

   $Id: simple_nc4_wr.c,v 1.4 2008/06/19 13:29:18 ed Exp $
*/

#include <stdlib.h>
#include <stdio.h>
#include <netcdf.h>

/* This is the name of the data file we will create. */
#define FILE_NAME "simple_nc4.nc"

/* We are writing 2D data, a 6 x 12 grid. */
#define NDIMS 2
#define NX 6
#define NY 12

/* Handle errors by printing an error message and exiting with a
 * non-zero status. */
#define ERRCODE 2
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); exit(ERRCODE);}

int
main()
{
   /* When we create netCDF variables, groups, dimensions, or types,
    * we get back an ID for each one. */
   int ncid, x_dimid, y_dimid, varid1, varid2, grp1id, grp2id, typeid;
   int dimids[NDIMS];

   /* This is the data array we will write. It will be filled with a
    * progression of numbers for this example. */
   unsigned long long data_out[NX][NY];

   /* Loop indexes, and error handling. */
   int x, y, retval;

   /* The following struct is written as a compound type. */
   struct s1
   {
         int i1;
         int i2;
   };
```

```
struct s1 compound_data[NX][NY];

/* Create some pretend data. */
for (x = 0; x < NX; x++)
   for (y = 0; y < NY; y++)
   {
      data_out[x][y] = x * NY + y;
      compound_data[x][y].i1 = 42;
      compound_data[x][y].i2 = -42;
   }

/* Create the file. The NC_NETCDF4 flag tells netCDF to
 * create a netCDF-4/HDF5 file.*/
if ((retval = nc_create(FILE_NAME, NC_NETCDF4|NC_CLOBBER, &ncid)))
   ERR(retval);

/* Define the dimensions in the root group. Dimensions are visible
 * in all subgroups. */
if ((retval = nc_def_dim(ncid, "x", NX, &x_dimid)))
   ERR(retval);
if ((retval = nc_def_dim(ncid, "y", NY, &y_dimid)))
   ERR(retval);

/* The dimids passes the IDs of the dimensions of the variable. */
dimids[0] = x_dimid;
dimids[1] = y_dimid;

/* Define two groups, "grp1" and "grp2." */
if ((retval = nc_def_grp(ncid, "grp1", &grp1id)))
   ERR (retval);
if ((retval = nc_def_grp(ncid, "grp2", &grp2id)))
   ERR (retval);

/* Define an unsigned 64bit integer variable in grp1, using dimensions
 * in the root group. */
if ((retval = nc_def_var(grp1id, "data", NC_UINT64, NDIMS,
                         dimids, &varid1)))
   ERR(retval);

/* Write unsigned long long data to the file. For netCDF-4 files,
 * nc_enddef will be called automatically. */
if ((retval = nc_put_var_ulonglong(grp1id, varid1, &data_out[0][0])))
   ERR(retval);

/* Create a compound type. This will cause nc_reddef to be called. */
if (nc_def_compound(grp2id, sizeof(struct s1), "sample_compound_type",
                    &typeid))
```

```
      ERR(retval);
   if (nc_insert_compound(grp2id, typeid, "i1",
                          offsetof(struct s1, i1), NC_INT))
      ERR(retval);
   if (nc_insert_compound(grp2id, typeid, "i2",
                          offsetof(struct s1, i2), NC_INT))
      ERR(retval);

   /* Define a compound type variable in grp2, using dimensions
    * in the root group. */
   if ((retval = nc_def_var(grp2id, "data", typeid, NDIMS,
                            dimids, &varid2)))
      ERR(retval);

   /* Write the array of struct to the file. This will cause nc_endef
    * to be called. */
   if ((retval = nc_put_var(grp2id, varid2, &compound_data[0][0])))
      ERR(retval);

   /* Close the file. */
   if ((retval = nc_close(ncid)))
      ERR(retval);

   printf("*** SUCCESS writing example file simple_nc4.nc!\n");
   return 0;
}
```

5.1.1.2 simple_nc4_rd.c

```
/* This is part of the netCDF package. Copyright 2006 University
   Corporation for Atmospheric Research/Unidata.  See COPYRIGHT file
   for conditions of use.

   This is a very simple example which demonstrates some of the
   new features of netCDF-4.0.

   This example reads a simple file created by simple_nc4_wr.c. This
   is intended to illustrate the use of the netCDF-4 C API.

   This program is part of the netCDF tutorial:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial.html

   Full documentation of the netCDF C API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c.html

   $Id: simple_nc4_rd.c,v 1.3 2008/06/19 13:29:18 ed Exp $
*/
```

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

/* This is the name of the data file we will read. */
#define FILE_NAME "simple_nc4.nc"

/* We are reading 2D data, a 6 x 12 grid. */
#define NX 6
#define NY 12

/* Handle errors by printing an error message and exiting with a
 * non-zero status. */
#define ERRCODE 2
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); exit(ERRCODE);}

int
main()
{
   /* There will be netCDF IDs for the file, each group, and each
    * variable. */
   int ncid, varid1, varid2, grp1id, grp2id;

   unsigned long long data_in[NX][NY];

   /* Loop indexes, and error handling. */
   int x, y, retval;

   /* The following struct is written as a compound type. */
   struct s1
   {
         int i1;
         int i2;
   };
   struct s1 compound_data[NX][NY];

   /* Open the file. NC_NOWRITE tells netCDF we want read-only access
    * to the file.*/
   if ((retval = nc_open(FILE_NAME, NC_NOWRITE, &ncid)))
      ERR(retval);

   /* Get the group ids of our two groups. */
   if ((retval = nc_inq_ncid(ncid, "grp1", &grp1id)))
      ERR(retval);
   if ((retval = nc_inq_ncid(ncid, "grp2", &grp2id)))
      ERR(retval);
```

```
   /* Get the varid of the uint64 data variable, based on its name, in
    * grp1. */
   if ((retval = nc_inq_varid(grp1id, "data", &varid1)))
      ERR(retval);

   /* Read the data. */
   if ((retval = nc_get_var_ulonglong(grp1id, varid1, &data_in[0][0])))
      ERR(retval);

   /* Get the varid of the compound data variable, based on its name,
    * in grp2. */
   if ((retval = nc_inq_varid(grp2id, "data", &varid2)))
      ERR(retval);

   /* Read the data. */
   if ((retval = nc_get_var(grp2id, varid2, &compound_data[0][0])))
      ERR(retval);

   /* Check the data. */
   for (x = 0; x < NX; x++)
      for (y = 0; y < NY; y++)
      {
         if (data_in[x][y] != x * NY + y ||
             compound_data[x][y].i1 != 42 ||
             compound_data[x][y].i2 != -42)
            return ERRCODE;
      }

   /* Close the file, freeing all resources. */
   if ((retval = nc_close(ncid)))
      ERR(retval);

   printf("*** SUCCESS reading example file %s!\n", FILE_NAME);
   return 0;
}
```

5.2 The simple_xy_nc4 Example

This example, like the simple_xy netCDF-3 example above, is an overly simplified example. It is based on the simple_xy example, but used data chunking, compression, and the fletcher32 filter.

(These are all HDF5 features. For more information see http://hdfgroup.org/HDF5/).

This example is not yet available in C++. We hope to have the C++ example in a future release of netCDF.

The example creates and then reads the file "simple_xy_nc4.nc."

The example program simple_xy_nc4_wr.c creates the example data file simple_xy_nc4.nc. The example program simple_xy_nc4_rd.c reads the data file.

5.2.1 simple_xy_nc4_wr.c and simple_xy_nc4_rd.c

This is just like the simple_xy example, but with chunking and variable compression.

5.2.1.1 simple_xy_nc4_wr.c

```
/* This is part of the netCDF package. Copyright 2007 University
   Corporation for Atmospheric Research/Unidata.  See COPYRIGHT file
   for conditions of use.

   This is a very simple example which is based on the simple_xy
   example, but whch uses netCDF-4 features, such as
   compression. Please see the simple_xy example to learn more about
   the netCDF-3 API.

   Like simple_xy_wr.c, this program writes a 2D netCDF variable
   (called "data") and fills it with sample data.  It has two
   dimensions, "x" and "y".

   This example demonstrates the netCDF C API, and netCDF-4
   extensions. This is part of the netCDF-4 tutorial, which can be
   found at:
   http://www.unidata.ucar.edu/software/netcdf/netcdf-4/newdocs/netcdf-tutorial

   Full documentation of the netCDF C API, including netCDF-4
   extensions, can be found at:
   http://www.unidata.ucar.edu/software/netcdf/netcdf-4/newdocs/netcdf-c

   $Id: simple_xy_nc4_wr.c,v 1.6 2008/12/11 16:40:05 russ Exp $
*/
#include <stdlib.h>
#include <stdio.h>
#include <netcdf.h>

/* This is the name of the data file we will create. */
#define FILE_NAME "simple_xy_nc4.nc"

/* We are writing 2D data, a 6 x 12 grid. */
#define NDIMS 2
#define NX 60
#define NY 120

/* Handle errors by printing an error message and exiting with a
 * non-zero status. */
#define ERRCODE 2
```

```
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); exit(ERRCODE);}

int
main()
{
   int ncid, x_dimid, y_dimid, varid;
   int dimids[NDIMS];
   size_t chunks[NDIMS];
   int shuffle, deflate, deflate_level;
   int data_out[NX][NY];
   int x, y, retval;

   /* Set chunking, shuffle, and deflate. */
   shuffle = NC_SHUFFLE;
   deflate = 1;
   deflate_level = 1;

   /* Create some pretend data. If this wasn't an example program, we
    * would have some real data to write, for example, model output. */
   for (x = 0; x < NX; x++)
      for (y = 0; y < NY; y++)
         data_out[x][y] = x * NY + y;

   /* Create the file. The NC_NETCDF4 parameter tells netCDF to create
    * a file in netCDF-4/HDF5 standard. */
   if ((retval = nc_create(FILE_NAME, NC_NETCDF4, &ncid)))
      ERR(retval);

   /* Define the dimensions. */
   if ((retval = nc_def_dim(ncid, "x", NX, &x_dimid)))
      ERR(retval);
   if ((retval = nc_def_dim(ncid, "y", NY, &y_dimid)))
      ERR(retval);

   /* Set up variabe data. */
   dimids[0] = x_dimid;
   dimids[1] = y_dimid;
   chunks[0] = NX/4;
   chunks[1] = NY/4;

   /* Define the variable. */
   if ((retval = nc_def_var(ncid, "data", NC_INT, NDIMS,
                            dimids, &varid)))
      ERR(retval);
   if ((retval = nc_def_var_chunking(ncid, varid, 0, &chunks[0])))
      ERR(retval);
   if ((retval = nc_def_var_deflate(ncid, varid, shuffle, deflate,
```

```
                                  deflate_level)))
      ERR(retval);

   /* No need to explicitly end define mode for netCDF-4 files. Write
    * the pretend data to the file. */
   if ((retval = nc_put_var_int(ncid, varid, &data_out[0][0])))
      ERR(retval);

   /* Close the file. */
   if ((retval = nc_close(ncid)))
      ERR(retval);

   printf("*** SUCCESS writing example file simple_xy_nc4.nc!\n");
   return 0;
}
```

5.2.1.2 simple_xy_nc4_rd.c

```
/* This is part of the netCDF package. Copyright 2007 University
   Corporation for Atmospheric Research/Unidata.  See COPYRIGHT file
   for conditions of use.

   This is a very simple example which is based on the simple_xy
   example, but whch uses netCDF-4 features, such as
   compression. Please see the simple_xy example to learn more about
   the netCDF-3 API.

   Like simple_xy_rd.c, this example reads a small dummy array, which
   was written by simple_xy_wr.c, and is compressed. This is intended
   to illustrate the use of the netCDF C API.

   This program is part of the netCDF-4 tutorial:
   http://www.unidata.ucar.edu/software/netcdf/netcdf-4/newdocs/netcdf-tutorial

   Full documentation of the netCDF-4 C API can be found at:
   http://www.unidata.ucar.edu/software/netcdf/netcdf-4/newdocs/netcdf-c

   $Id: simple_xy_nc4_rd.c,v 1.2 2007/06/05 21:47:50 ed Exp $
*/

#include <config.h>
#include <stdlib.h>
#include <stdio.h>
#include <netcdf.h>

/* This is the name of the data file we will read. */
```

```
#define FILE_NAME "simple_xy_nc4.nc"

/* We are reading 2D data, a 6 x 12 grid. */
#define NX 60
#define NY 120

/* Handle errors by printing an error message and exiting with a
 * non-zero status. */
#define ERRCODE 2
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); exit(ERRCODE);}

int
main()
{
   /* This will be the netCDF ID for the file and data variable. */
   int ncid, varid;

   int data_in[NX][NY];

   /* Loop indexes, and error handling. */
   int x, y, retval;

   /* Open the file. NC_NOWRITE tells netCDF we want read-only access
    * to the file.*/
   if ((retval = nc_open(FILE_NAME, NC_NOWRITE, &ncid)))
      ERR(retval);

   /* Get the varid of the data variable, based on its name. */
   if ((retval = nc_inq_varid(ncid, "data", &varid)))
      ERR(retval);

   /* Read the data. */
   if ((retval = nc_get_var_int(ncid, varid, &data_in[0][0])))
      ERR(retval);

   /* Check the data. */
   for (x = 0; x < NX; x++)
      for (y = 0; y < NY; y++)
 if (data_in[x][y] != x * NY + y)
    return ERRCODE;

   /* Close the file, freeing all resources. */
   if ((retval = nc_close(ncid)))
      ERR(retval);

   printf("*** SUCCESS reading example file %s!\n", FILE_NAME);
   return 0;
```

```
}
```

5.2.2 simple_xy_nc4_wr.f and simple_xy_nc4_rd.f

This is just like the simple_xy example, but with chunking and variable compression.

5.2.2.1 simple_xy_nc4_wr.f

```
C     This is part of the netCDF package.
C     Copyright 2006 University Corporation for Atmospheric Research/Unidata.
C     See COPYRIGHT file for conditions of use.

C     This is a very simple example which writes a 2D array of
C     sample data. To handle this in netCDF we create two shared
C     dimensions, "x" and "y", and a netCDF variable, called "data".

C     This example demonstrates the netCDF Fortran 77 API. This is part
C     of the netCDF tutorial, which can be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

C     Full documentation of the netCDF Fortran 77 API can be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f77

C     $Id: simple_xy_nc4_wr.f,v 1.5 2008/02/19 22:01:49 ed Exp $

      program simple_xy_wr
      implicit none
      include 'netcdf.inc'

      character*(*) FILE_NAME
      parameter (FILE_NAME='simple_xy_nc4.nc')
      integer NDIMS
      parameter (NDIMS = 2)
      integer NX, NY
      parameter (NX = 60, NY = 120)
      integer ncid, varid, dimids(NDIMS)
      integer x_dimid, y_dimid
      integer data_out(NY, NX)
      integer x, y, retval
      integer chunks(2)
      integer contiguous, shuffle, deflate, deflate_level

C     Create some pretend data. If this wasn't an example program, we
C     would have some real data to write, for example, model output.
      do x = 1, NX
         do y = 1, NY
            data_out(y, x) = (x - 1) * NY + (y - 1)
         end do
```

```
      end do

C     Create the netCDF file. The nf_netcdf4  tells netCDF to
C     create a netCDF-4/HDF5 file.
      retval = nf_create(FILE_NAME, NF_NETCDF4, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the dimensions.
      retval = nf_def_dim(ncid, "x", NX, x_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_dim(ncid, "y", NY, y_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     The dimids array is used to pass the IDs of the dimensions of
C     the variables. Note that in fortran arrays are stored in
C     column-major format.
      dimids(2) = x_dimid
      dimids(1) = y_dimid

C     Define the variable.
      retval = nf_def_var(ncid, "data", NF_INT, NDIMS, dimids, varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Set up chunking and compression.
      contiguous = 0
      chunks(1) = NY
      chunks(2) = NX
      shuffle = 1
      deflate = 1
      deflate_level = 4

      retval = nf_def_var_chunking(ncid, varid, contiguous, chunks)
      if (retval .ne. nf_noerr) call handle_err(retval)

      retval = nf_def_var_deflate(ncid, varid, shuffle, deflate,
     &     deflate_level)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Write the pretend data to the file.
      retval = nf_put_var_int(ncid, varid, data_out)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Close the file.
      retval = nf_close(ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

      print *,'*** SUCCESS writing example file simple_xy_nc4.nc!'
```

```
      end

      subroutine handle_err(errcode)
      implicit none
      include 'netcdf.inc'
      integer errcode

      print *, 'Error: ', nf_strerror(errcode)
      stop 2
      end
```

5.2.2.2 simple_xy_nc4_rd.f

```
C     This is part of the netCDF package.
C     Copyright 2006 University Corporation for Atmospheric Research/Unidata.
C     See COPYRIGHT file for conditions of use.

C     This is a simple example which reads a small dummy array, from a
C     netCDF data file created by the companion program simple_xy_wr.f.

C     This is intended to illustrate the use of the netCDF fortran 77
C     API. This example program is part of the netCDF tutorial, which can
C     be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

C     Full documentation of the netCDF Fortran 77 API can be found at:
C     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f77

C     $Id: simple_xy_nc4_rd.f,v 1.1 2007/05/04 13:33:20 ed Exp $

      program simple_xy_rd
      implicit none
      include 'netcdf.inc'

C     This is the name of the data file we will read.
      character*(*) FILE_NAME
      parameter (FILE_NAME='simple_xy_nc4.nc')

C     We are reading 2D data, a 12 x 6 grid.
      integer NX, NY
      parameter (NX = 60, NY = 120)
      integer data_in(NY, NX)

C     This will be the netCDF ID for the file and data variable.
      integer ncid, varid

C     Loop indexes, and error handling.
```

```
      integer x, y, retval

C     Open the file. NF_NOWRITE tells netCDF we want read-only access to
C     the file.
      retval = nf_open(FILE_NAME, NF_NOWRITE, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Get the varid of the data variable, based on its name.
      retval = nf_inq_varid(ncid, 'data', varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Read the data.
      retval = nf_get_var_int(ncid, varid, data_in)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Check the data.
      do x = 1, NX
         do y = 1, NY
            if (data_in(y, x) .ne. (x - 1) * NY + (y - 1)) then
               print *, 'data_in(', y, ', ', x, ') = ', data_in(y, x)
               stop 2
            end if
         end do
      end do

C     Close the file, freeing all resources.
      retval = nf_close(ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

      print *,'*** SUCCESS reading example file ', FILE_NAME, '!'
      end

      subroutine handle_err(errcode)
      implicit none
      include 'netcdf.inc'
      integer errcode

      print *, 'Error: ', nf_strerror(errcode)
      stop 2
      end
```

5.2.3 simple_xy_nc4_wr.f90 and simple_xy_nc4_rd.f90

This is just like the simple_xy example, but with chunking and variable compression.

5.2.3.1 simple_xy_nc4_wr.f90

```
!     This is part of the netCDF package.  Copyright 2006 University
```

```
!     Corporation for Atmospheric Research/Unidata.  See COPYRIGHT
!     file for conditions of use.

!     This is a very simple example which writes a 2D array of sample
!     data. To handle this in netCDF we create two shared dimensions,
!     "x" and "y", and a netCDF variable, called "data".

!     This example demonstrates the netCDF Fortran 90 API. This is
!     part of the netCDF tutorial, which can be found at:
!     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

!     Full documentation of the netCDF Fortran 90 API can be found at:
!     http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90

!     $Id: simple_xy_nc4_wr.f90,v 1.6 2010/04/06 19:32:09 ed Exp $

program simple_xy_wr
  use netcdf
  implicit none

  character (len = *), parameter :: FILE_NAME = "simple_xy_nc4.nc"
  integer, parameter :: NDIMS = 2
  integer, parameter :: NX = 6, NY = 12
  integer :: ncid, varid, dimids(NDIMS)
  integer :: x_dimid, y_dimid
  integer :: data_out(NY, NX)
  integer :: chunks(2)
  integer :: deflate_level
  integer :: x, y

  ! Create some pretend data. If this wasn't an example program, we
  ! would have some real data to write, for example, model output.
  do x = 1, NX
     do y = 1, NY
        data_out(y, x) = (x - 1) * NY + (y - 1)
     end do
  end do

  ! Always check the return code of every netCDF function call. In
  ! this example program, wrapping netCDF calls with "call check()"
  ! makes sure that any return which is not equal to nf90_noerr (0)
  ! will print a netCDF error message and exit.

  ! Create the netCDF file. The nf90_clobber parameter tells netCDF to
  ! overwrite this file, if it already exists.
  call check( nf90_create(FILE_NAME, nf90_hdf5, ncid) )
```

```
  ! Define the dimensions. NetCDF will hand back an ID for each.
  call check( nf90_def_dim(ncid, "x", NX, x_dimid) )
  call check( nf90_def_dim(ncid, "y", NY, y_dimid) )

  ! The dimids array is used to pass the IDs of the dimensions of
  ! the variables. Note that in fortran arrays are stored in
  ! column-major format.
  dimids =  (/ y_dimid, x_dimid /)

  ! Define the variable. The type of the variable in this case is
  ! NF90_INT (4-byte integer). Optional parameters chunking, shuffle,
  ! and deflate_level are used.
  chunks(1) = NY
  chunks(2) = NX
  deflate_level = 1
  call check( nf90_def_var(ncid, "data", NF90_INT, dimids, varid, &
       chunksizes = chunks, shuffle = .TRUE., deflate_level = deflate_level) )

  ! End define mode. This tells netCDF we are done defining metadata.
  call check( nf90_enddef(ncid) )

  ! Write the pretend data to the file. Although netCDF supports
  ! reading and writing subsets of data, in this case we write all the
  ! data in one operation.
  call check( nf90_put_var(ncid, varid, data_out) )

  ! Close the file. This frees up any internal netCDF resources
  ! associated with the file, and flushes any buffers.
  call check( nf90_close(ncid) )

  print *, '*** SUCCESS writing example file ', FILE_NAME, '!'

contains
  subroutine check(status)
    integer, intent ( in) :: status

    if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status))
      stop 2
    end if
  end subroutine check
end program simple_xy_wr
```

5.2.3.2 simple_xy_nc4_rd.f90

```
! This is part of the netCDF package.
! Copyright 2006 University Corporation for Atmospheric Research/Unidata.
```

```
! See COPYRIGHT file for conditions of use.

! This is a simple example which reads a small dummy array, from a
! netCDF data file created by the companion program simple_xy_wr.f90.

! This is intended to illustrate the use of the netCDF fortran 77
! API. This example program is part of the netCDF tutorial, which can
! be found at:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial

! Full documentation of the netCDF Fortran 90 API can be found at:
! http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90

! $Id: simple_xy_nc4_rd.f90,v 1.3 2009/02/25 12:23:45 ed Exp $

program simple_xy_rd
  use netcdf
  implicit none

  ! This is the name of the data file and variable.
  character (len = *), parameter :: FILE_NAME = "simple_xy_nc4.nc"
  character (len = *), parameter :: VAR_NAME = "data"

  ! We are reading 2D data, a 12 x 6 grid.
  integer, parameter :: NX = 6, NY = 12, MAX_DIMS = 2
  integer :: data_in(NY, NX)

  ! This will be the netCDF ID for the file and data variable.
  integer :: ncid, varid

  ! Information we will read about the variable.
  character (len = nf90_max_name) :: name
  integer :: xtype, ndims, dimids(MAX_DIMS), natts
  integer :: chunksizes(MAX_DIMS), deflate_level, endianness
  logical :: contiguous, shuffle, fletcher32

  ! Loop indexes, and error handling.
  integer :: x, y

  ! Open the file. NF90_NOWRITE tells netCDF we want read-only access
  ! to the file.
  call check( nf90_open(FILE_NAME, NF90_NOWRITE, ncid) )

  ! Get the varid of the data variable, based on its name.
  call check( nf90_inq_varid(ncid, VAR_NAME, varid) )

  ! Learn about the variable. This uses all optional parameters.
```

```
  call check( nf90_inquire_variable(ncid, varid, name, xtype, ndims, &
       dimids, natts, contiguous = contiguous, chunksizes = chunksizes, &
       deflate_level = deflate_level, shuffle = shuffle, &
       fletcher32 = fletcher32, endianness = endianness) )

  ! Make sure we got the correct answers. These depend on what was set
  ! when creating the file in simple_xy_nc4_wr.f90. Endianness will be
  ! whatever is native for the machine that's building this example.
  if (name .ne. VAR_NAME .or. xtype .ne. NF90_INT .or. ndims .ne. 2 .or. &
       natts .ne. 0 .or. contiguous .or. .not. shuffle .or. &
       deflate_level .ne. 1 .or. fletcher32) stop 3

  ! Read the data.
  call check( nf90_get_var(ncid, varid, data_in) )

  ! Check the data.
  do x = 1, NX
     do y = 1, NY
        if (data_in(y, x) /= (x - 1) * NY + (y - 1)) then
           print *, "data_in(", y, ", ", x, ") = ", data_in(y, x)
           stop "Stopped"
        end if
     end do
  end do

  ! Close the file, freeing all resources.
  call check( nf90_close(ncid) )

  print *,"*** SUCCESS reading example file ", FILE_NAME, "! "

contains
  subroutine check(status)
    integer, intent ( in) :: status

    if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status))
      stop 2
    end if
  end subroutine check
end program simple_xy_rd
```

Index

R

S

T

U

V