Table of Contents

Foreword ................................................................. 1

Summary ................................................................. 3

1 Introduction ............................................................. 5
  1.1 Availability ...................................................... 5
  1.2 How to Use This Guide ......................................... 5
  1.3 Operating systems compatible with NCO ....................... 6
    1.3.1 Compiling NCO for Microsoft Windows OS ............... 7
  1.4 Symbolic Links ................................................... 8
  1.5 Libraries .......................................................... 9
  1.6 netCDF2/3/4 and HDF4/5 Support ............................. 9
  1.7 Help Requests and Bug Reports ............................... 13

2 Operator Strategies .................................................. 15
  2.1 Philosophy ....................................................... 15
  2.2 Climate Model Paradigm ........................................ 15
  2.3 Temporary Output Files ........................................ 15
  2.4 Appending Variables ............................................ 17
  2.5 Simple Arithmetic and Interpolation ........................ 17
  2.6 Statistics vs. Concatenation ................................... 18
    2.6.1 Concatenators ncrcat and ncecat ........................ 18
    2.6.2 Averagers nces, ncra, and ncwa ......................... 19
    2.6.3 Interpolator ncflint ...................................... 19
  2.7 Large Numbers of Files ........................................ 19
  2.8 Large Datasets .................................................. 21
  2.9 Memory Requirements .......................................... 22
    2.9.1 Single and Multi-file Operators .......................... 22
    2.9.2 Memory for ncap2 ......................................... 23
  2.10 Performance .................................................... 24

3 Shared Features ...................................................... 25
  3.1 Internationalization ............................................ 25
  3.2 Metadata Optimization ......................................... 25
  3.3 OpenMP Threading .............................................. 25
  3.4 Command Line Options ......................................... 27
  3.5 Specifying Input Files ........................................ 28
  3.6 Specifying Output Files ....................................... 30
  3.7 Accessing Remote Files ....................................... 30
    3.7.1 OPeNDAP .................................................... 32
  3.8 Retaining Retrieved Files .................................... 35
  3.9 File Formats and Conversion .................................. 35
3.9.1 File Formats .................................................. 36
3.9.2 Determining File Format ........................................ 37
3.9.3 File Conversion ............................................. 38
3.9.4 Autoconversion .............................................. 39
3.10 Large File Support ........................................... 40
3.11 Subsetting Files ............................................. 40
3.12 Subsetting Coordinate Variables .............................. 45
3.13 Group Path Editing ............................................ 46
   3.13.1 Deletion, Truncation, and Flattening of Groups ......... 46
   3.13.2 Moving Groups .......................................... 48
   3.13.3 Dismembering Files ..................................... 49
   3.13.4 Checking CF-compliance ................................ 51
3.14 C and Fortran Index conventions ............................ 54
3.15 Hyperslabs .................................................... 55
3.16 Stride ....................................................... 57
3.17 Record Appending ........................................... 58
3.18 Subcycle ..................................................... 59
3.19 Multislabs ................................................... 60
3.20 Wrapped Coordinates ......................................... 63
3.21 Auxiliary Coordinates ........................................ 64
3.22 UDUnits Support ............................................. 66
3.23 Rebasing Time Coordinate .................................. 69
3.24 Multiple Record Dimensions ................................ 70
3.25 Missing values ............................................... 71
3.26 Chunking ..................................................... 72
3.27 Compression .................................................. 78
3.28 Deflation ..................................................... 88
3.29 MD5 digests ................................................... 88
3.30 Buffer sizes .................................................. 90
3.31 RAM disks .................................................... 90
3.32 Packed data .................................................. 92
   Packing Algorithm ............................................ 92
   Unpacking Algorithm ........................................ 93
   Default Handling of Packed Data ............................. 93
   Default Handling of Packed Data ............................. 94
3.33 Operation Types ............................................. 94
3.34 Type Conversion ............................................. 99
   3.34.1 Automatic type conversion ............................. 100
   3.34.2 Promoting Single-precision to Double ................ 102
   3.34.3 Manual type conversion ............................... 108
3.35 Batch Mode .................................................. 108
3.36 History Attribute ........................................... 108
3.37 File List Attributes ......................................... 109
3.38 CF Conventions ............................................. 109
3.39 ARM Conventions ........................................... 112
3.40 Operator Version ........................................... 112
4 Reference Manual .................................. 115
  4.1 ncap2 netCDF Arithmetic Processor .......... 116
    4.1.1 Syntax of ncap2 statements ............... 117
    4.1.2 Expressions ................................ 117
    4.1.3 Dimensions ................................ 121
    4.1.4 Left hand casting .......................... 122
    4.1.5 Arrays and hyperslabs ....................... 122
    4.1.6 Attributes ................................ 125
    4.1.7 Number literals .............................. 126
    4.1.8 if statement ................................ 127
    4.1.9 print statement .............................. 129
    4.1.10 Missing values ncap2 ....................... 129
    4.1.11 Methods and functions ...................... 131
    4.1.12 RAM variables ............................... 134
    4.1.13 Where statement ............................. 134
    4.1.14 Loops ...................................... 136
    4.1.15 Include files ............................... 137
    4.1.16 sort methods ................................ 137
    4.1.17 Irregular Grids ............................. 142
    4.1.18 Bilinear interpolation ....................... 144
    4.1.19 GSL special functions ....................... 146
    4.1.20 GSL interpolation ........................... 154
    4.1.21 GSL least-squares fitting ................... 155
    4.1.22 GSL statistics ............................... 157
    4.1.23 GSL random number generation ............... 159
    4.1.24 Examples ncap2 ............................. 161
    4.1.25 Intrinsic mathematical methods ............ 163
    4.1.26 Operator precedence and associativity .... 165
    4.1.27 ID Quoting .................................. 166
  4.2 ncatted netCDF Attribute Editor ............... 168
  4.3 ncbo netCDF Binary Operator .................... 174
  4.4 ncens netCDF Ensemble Statistics ................ 179
  4.5 ncecat netCDF Ensemble Concatenator ............ 182
  4.6 ncflint netCDF File Interpolator ............... 185
  4.7 ncks netCDF Kitchen Sink ....................... 188
    Options specific to ncks ......................... 189
    4.7.2 Filters for ncks ............................. 196
  4.8 ncpdq netCDF Permute Dimensions Quickly ...... 202
    Packing and Unpacking Functions .................. 202
    Dimension Permutation ............................ 205
  4.9 ncra netCDF Record Averager .................... 210
  4.10 ncrcat netCDF Record Concatenator ............. 212
  4.11 ncrename netCDF Renamer ....................... 214
  4.12 ncwa netCDF Weighted Averager ................. 219
    4.12.1 Mask condition ............................. 220
    4.12.2 Normalization and Integration .......... 221
5 Contributing ............................................ 225
   5.1 Contributors ..................................... 225
   5.2 Citation ......................................... 227
   5.3 Proposals for Institutional Funding ............ 227

6 Quick Start ........................................... 229
   6.1 Daily data in one file ............................ 229
   6.2 Monthly data in one file ......................... 229
   6.3 One time point one file ......................... 230
   6.4 Multiple files with multiple time points ....... 230

7 CMIP5 Example ....................................... 231
   7.1 Combine Files .................................... 231
   7.2 Global Distribution of Long-term Average .... 237
   7.3 Annual Average over Regions ................... 240
   7.4 Monthly Cycle .................................... 247
   7.5 Regrid MODIS Data ............................... 250
   7.6 Add Coordinates to MODIS Data ................ 253
   7.7 Permute MODIS Coordinates ..................... 254

8 Parallel .............................................. 257

9 CCSM Example ....................................... 259

10 References .......................................... 267

General Index ......................................... 269
Foreword

NCO is the result of software needs that arose while I worked on projects funded by NCAR, NASA, and ARM. Thinking they might prove useful as tools or templates to others, it is my pleasure to provide them freely to the scientific community. Many users (most of whom I have never met) have encouraged the development of NCO. Thanks especially to Jan Polcher, Keith Lindsay, Arlindo da Silva, John Sheldon, and William Weibel for stimulating suggestions and correspondence. Your encouragement motivated me to complete the NCO User Guide. So if you like NCO, send me a note! I should mention that NCO is not connected to or officially endorsed by Unidata, ACD, ASP, CGD, or Nike.

Charlie Zender
May 1997
Boulder, Colorado

Major feature improvements entitle me to write another Foreword. In the last five years a lot of work has been done to refine NCO. NCO is now an open source project and appears to be much healthier for it. The list of illustrious institutions that do not endorse NCO continues to grow, and now includes UCI.

Charlie Zender
October 2000
Irvine, California

The most remarkable advances in NCO capabilities in the last few years are due to contributions from the Open Source community. Especially noteworthy are the contributions of Henry Butowsky and Rorik Peterson.

Charlie Zender
January 2003
Irvine, California

NCO was generously supported from 2004–2008 by US National Science Foundation (NSF) grant IIS-0431203. This support allowed me to maintain and extend core NCO code, and others to advance NCO in new directions: Gayathri Venkitachalam helped implement
MPI: Harry Mangalam improved regression testing and benchmarking; Daniel Wang developed the server-side capability, SWAMP; and Henry Butowsky, a long-time contributor, developed ncap2. This support also led NCO to debut in professional journals and meetings. The personal and professional contacts made during this evolution have been immensely rewarding.

Charlie Zender
March 2008
Grenoble, France

The end of the NSF SEI grant in August, 2008 curtailed NCO development. Fortunately we could justify supporting Henry Butowsky on other research grants until May, 2010 while he developed the key ncap2 features used in our climate research. And recently the NASA ACCESS program commenced funding us to support netCDF4 group functionality. Thus NCO will grow and evade bit-rot for the foreseeable future.

I continue to receive with gratitude the thanks of NCO users at nearly every scientific meeting I attend. People introduce themselves, shake my hand and extol NCO, often effusively, while I grin in stupid embarrassment. These exchanges lighten me like anti-gravity. Sometimes I daydream how many hours NCO has turned from grunt work to productive research for researchers world-wide, or from research into early happy-hours. It's a cool feeling.

Charlie Zender
April, 2012
Irvine, California
Summary

This manual describes NCO, which stands for netCDF Operators. NCO is a suite of programs known as operators. Each operator is a standalone, command line program executed at the shell-level like, e.g., ls or mkdir. The operators take netCDF files (including HDF5 files constructed using the netCDF API) as input, perform an operation (e.g., averaging or hyperslabbing), and produce a netCDF file as output. The operators are primarily designed to aid manipulation and analysis of data. The examples in this documentation are typical applications of the operators for processing climate model output. This stems from their origin, though the operators are as general as netCDF itself.
1 Introduction

1.1 Availability

The complete NCO source distribution is currently distributed as a compressed tarfile from http://sf.net/projects/nco and from http://dust.ess.uci.edu/nco/nco.tar.gz. The compressed tarfile must be uncompressed and untarred before building NCO. Uncompress the file with `gunzip nco.tar.gz`. Extract the source files from the resulting tarfile with `tar -xvf nco.tar`. GNU tar lets you perform both operations in one step with `tar -xvzf nco.tar.gz`.

The documentation for NCO is called the NCO User Guide. The User Guide is available in PDF, Postscript, HTML, DVI, TEXinfo, and Info formats. These formats are included in the source distribution in the files nco.pdf, nco.ps, nco.html, nco.dvi, nco.texi, and nco.info*, respectively. All the documentation descends from a single source file, nco.texi\(^1\). Hence the documentation in every format is very similar. However, some of the complex mathematical expressions needed to describe ncwa can only be displayed in DVI, Postscript, and PDF formats.

A complete list of papers and publications on/about NCO is available on the NCO homepage. Most of these are freely available. The primary refereed publications are ZeM06 and Zen08. These contain copyright restrictions which limit their redistribution, but they are freely available in preprint form from the NCO.

If you want to quickly see what the latest improvements in NCO are (without downloading the entire source distribution), visit the NCO homepage at http://nco.sf.net. The HTML version of the User Guide is also available online through the World Wide Web at URL http://nco.sf.net/nco.html. To build and use NCO, you must have netCDF installed. The netCDF homepage is http://www.unidata.ucar.edu/packages/netcdf.

New NCO releases are announced on the netCDF list and on the nco-announce mailing list http://lists.sf.net/mailman/listinfo/nco-announce.

1.2 How to Use This Guide

Detailed instructions about how to download the newest version, and how to compile source code, as well as a FAQ and descriptions of Known Problems etc. are on our homepage (http://nco.sf.net/)

There are twelve operators in the current version (4.4.9). The function of each is explained in Chapter 4 [Reference Manual], page 115. Many of the tasks that NCO can accomplish are described during the explanation of common NCO Features (see Chapter 3 [Shared features], page 25). More specific use examples for each operator can be seen by visiting the operator-specific examples in the Chapter 4 [Reference Manual], page 115. These can be found directly by prepending the operator name with the xmp_ tag, e.g., http://nco.sf.net.

\(^1\) To produce these formats, nco.texi was simply run through the freely available programs texi2dvi, dvips, texi2html, and makeinfo. Due to a bug in \TeX\X, the resulting Postscript file, nco.ps, contains the Table of Contents as the final pages. Thus if you print nco.ps, remember to insert the Table of Contents after the cover sheet before you staple the manual.
Also, users can type the operator name on the shell command line to see all the available options, or type, e.g., `man ncks` to see a help man-page.

NCO is a command-line language. You may either use an operator after the prompt (e.g., `$` here), like,

```
$ operator [options] input [output]
```

or write all commands lines into a shell script, as in the CMIP5 Example (see Chapter 7 [CMIP5 Example], page 231).

If you are new to NCO, the Quick Start (see Chapter 6 [Quick Start], page 229) shows simple examples about how to use NCO on different kinds of data files. More detailed “real-world” examples are in the Chapter 7 [CMIP5 Example], page 231. The [General Index], page 269 presents multiple keyword entries for the same subject. If these resources do not help enough, please see Section 1.7 [Help Requests and Bug Reports], page 13.

### 1.3 Operating systems compatible with NCO

In its time on Earth, NCO has been successfully ported and tested on so many 32- and 64-bit platforms that if we did not write them down here we would forget their names: IBM AIX 4.x, 5.x, FreeBSD 4.x, GNU/Linux 2.x, LinuxPPC, LinuxAlpha, LinuxARM, LinuxSparc64, LinuxAMD64, SGI IRISH 5.x and 6.x, MacOS X 10.x, DEC OSF, NEC Super-UX 10.x, Sun SunOS 4.1.x, Solaris 2.x, Cray UNICOS 8.x–10.x, and Microsoft Windows (95, 98, NT, 2000, XP, Vista, 7, 8, 10). If you port the code to a new operating system, please send me a note and any patches you required.

The major prerequisite for installing NCO on a particular platform is the successful, prior installation of the netCDF library (and, as of 2003, the UDUnits library). Unidata has shown a commitment to maintaining netCDF and UDUnits on all popular UNIX platforms, and is moving towards full support for the Microsoft Windows operating system (OS). Given this, the only difficulty in implementing NCO on a particular platform is standardization of various C-language API system calls. NCO code is tested for ANSI compliance by compiling with C99 compilers including those from GNU (`gcc -std=c99 -pedantic -D_BSD_SOURCE -D_POSIX_SOURCE' -Wall)\(^2\), Comeau Computing (`como --c99'), Cray (`cc'), HP/Compaq/DEC (`cc'), IBM (`xlc -c -qlanglvl=extc99'), Intel (`icc -std=c99'), LLVM (`clang'), NEC (`cc'), PathScale (QLogic) (`pathcc -std=c99'), PGI (`pgcc -c9x'), SGI (`cc -c99'), and Sun (`cc'). NCO (all commands and the libnco library) and the C++ interface to netCDF (called `libnco_c++') comply with the ISO C++ standards as implemented by Comeau Computing (`como'), Cray (`CC'), GNU (`g++ -Wall'), HP/Compaq/DEC (`cxx'), IBM (`xlc'), Intel (`icc'), Microsoft (`MVS'), NEC (`c++'), PathScale (Qlogic) (`pathCC'), PGI (`pgCC'), SGI (`CC -LANG:std'), and Sun (`CC -LANG:std'). See `nco/bld/Makefile' and `nco/src/nco_c++/Makefile.old' for more details and exact settings.

Until recently (and not even yet), ANSI-compliant has meant compliance with the 1989 ISO C-standard, usually called C89 (with minor revisions made in 1994 and 1995). C89 lacks variable-size arrays, restricted pointers, some useful `printf' formats, and many mathemat-

\(^2\) The `_BSD_SOURCE' token is required on some Linux platforms where gcc dislikes the network header files like `netinet/in.h').
ical special functions. These are valuable features of C99, the 1999 ISO C-standard. NCO is C99-compliant where possible and C89-compliant where necessary. Certain branches in the code are required to satisfy the native SGI and SunOS C compilers, which are strictly ANSI C89 compliant, and cannot benefit from C99 features. However, C99 features are fully supported by modern AIX, GNU, Intel, NEC, Solaris, and UNICOS compilers. NCO requires a C99-compliant compiler as of NCO version 2.9.8, released in August, 2004.

The most time-intensive portion of NCO execution is spent in arithmetic operations, e.g., multiplication, averaging, subtraction. These operations were performed in Fortran by default until August, 1999. This was a design decision based on the relative speed of Fortran-based object code vs. C-based object code in late 1994. C compiler vectorization capabilities have dramatically improved since 1994. We have accordingly replaced all Fortran subroutines with C functions. This greatly simplifies the task of building NCO on nominally unsupported platforms. As of August 1999, NCO built entirely in C by default. This allowed NCO to compile on any machine with an ANSI C compiler. In August 2004, the first C99 feature, the restrict type qualifier, entered NCO in version 2.9.8. C compilers can obtain better performance with C99 restricted pointers since they inform the compiler when it may make Fortran-like assumptions regarding pointer contents alteration. Subsequently, NCO requires a C99 compiler to build correctly\(^3\).

In January 2009, NCO version 3.9.6 was the first to link to the GNU Scientific Library (GSL). GSL must be version 1.4 or later. NCO, in particular ncap2, uses the GSL special function library to evaluate geoscience-relevant mathematics such as Bessel functions, Legendre polynomials, and incomplete gamma functions (see Section 4.1.19 [GSL special functions], page 146).

In June 2005, NCO version 3.0.1 began to take advantage of C99 mathematical special functions. These include the standarized gamma function (called tgamma() for “true gamma”). NCO automagically takes advantage of some GNU Compiler Collection (GCC) extensions to ANSI C.

As of July 2000 and NCO version 1.2, NCO no longer performs arithmetic operations in Fortran. We decided to sacrifice executable speed for code maintainability. Since no objective statistics were ever performed to quantify the difference in speed between the Fortran and C code, the performance penalty incurred by this decision is unknown. Supporting Fortran involves maintaining two sets of routines for every arithmetic operation. The USE_FORTRAN_ARITHMETIC flag is still retained in the Makefile. The file containing the Fortran code, nco_fortran.F, has been deprecated but a volunteer (Dr. Frankenstein?) could resurrect it. If you would like to volunteer to maintain nco_fortran.F please contact me.

### 1.3.1 Compiling NCO for Microsoft Windows OS

NCO has been successfully ported and tested on most Microsoft Windows operating systems including: XP SP2/Vista/7. Support is provided for compiling either native Windows executables, using the Microsoft Visual Studio 2010 Compiler, or with Cygwin, the UNIX-

---

\(^3\) NCO may still build with an ANSI or ISO C89 or C94/95-compliant compiler if the C pre-processor undefines the restrict type qualifier, e.g., by invoking the compiler with ‘-Drestrict=’.”
emulating compatibility layer with the GNU toolchain. The switches necessary to accomplish both are included in the standard distribution of NCO.

Using Microsoft Visual Studio (MVS), one must build NCO with the C++ compiler since MVS does not support C99. Qt, a convenient integrated development environment, was used to convert the project files to MVS format. The Qt files themselves are distributed in the nco/qt directory.

Using the freely available Cygwin (formerly gnu-win32) development environment\(^4\), the compilation process is very similar to installing NCO on a UNIX system. Set the `PVM_ARCH` preprocessor token to `WIN32`. Note that defining `WIN32` has the side effect of disabling Internet features of NCO (see below). NCO should now build like it does on UNIX.

The least portable section of the code is the use of standard UNIX and Internet protocols (e.g., `ftp`, `rcp`, `scp`, `sftp`, `getuid`, `gethostname`, and header files `<arpa/nameser.h>` and `<resolv.h>`). Fortunately, these UNIX-y calls are only invoked by the single NCO subroutine which is responsible for retrieving files stored on remote systems (see Section 3.7 [Remote storage], page 30). In order to support NCO on the Microsoft Windows platforms, this single feature was disabled (on Windows OS only). This was required by Cygwin 18.x—newer versions of Cygwin may support these protocols (let me know if this is the case). The NCO operators should behave identically on Windows and UNIX platforms in all other respects.

1.4 Symbolic Links

NCO relies on a common set of underlying algorithms. To minimize duplication of source code, multiple operators sometimes share the same underlying source. This is accomplished by symbolic links from a single underlying executable program to one or more invoked executable names. For example, `nces` and `ncrcat` are symbolically linked to the `ncra` executable. The `ncra` executable behaves slightly differently based on its invocation name (i.e., `argv[0]`), which can be `nces`, `ncra`, or `ncrcat`. Logically, these are three different operators that happen to share the same executable.

For historical reasons, and to be more user friendly, multiple synonyms (or pseudonyms) may refer to the same operator invoked with different switches. For example, `ncdiff` is the same as `ncbo` and `ncpack` is the same as `ncpdq`. We implement the symbolic links and synonyms by the executing the following UNIX commands in the directory where the NCO executables are installed.

\[
\begin{align*}
\text{ln } -s -f \text{ ncbo ncdiff} & \quad \# \text{ ncbo } --\text{op_typ=}'+'$
\text{ln } -s -f \text{ ncra nces} & \quad \# \text{ ncra } --\text{pseudonym=}’nces’
\text{ln } -s -f \text{ ncra ncrcat} & \quad \# \text{ ncra } --\text{pseudonym=}’ncrcat’
\text{ln } -s -f \text{ ncbo ncadd} & \quad \# \text{ ncbo } --\text{op_typ=}'+'$
\text{ln } -s -f \text{ ncbo ncssubtract} & \quad \# \text{ ncbo } --\text{op_typ=}’-’$
\text{ln } -s -f \text{ ncbo ncmultiply} & \quad \# \text{ ncbo } --\text{op_typ=}’*’$
\text{ln } -s -f \text{ ncbo ncdimdivide} & \quad \# \text{ ncbo } --\text{op_typ=}’/’$
\end{align*}
\]

\(^4\) The Cygwin package is available from [http://sourceware.redhat.com/cygwin](http://sourceware.redhat.com/cygwin)

Currently, Cygwin 20.x comes with the GNU C/C++ compilers (`gcc`, `g++`). These GNU compilers may be used to build the netCDF distribution itself.
The imputed command called by the link is given after the comment. As can be seen, some these links impute the passing of a command line argument to further modify the behavior of the underlying executable. For example, `ncdivide` is a pseudonym for `ncbo --op_typ='/'`.

### 1.5 Libraries

Like all executables, the NCO operators can be built using dynamic linking. This reduces the size of the executable and can result in significant performance enhancements on multiuser systems. Unfortunately, if your library search path (usually the `LD_LIBRARY_PATH` environment variable) is not set correctly, or if the system libraries have been moved, renamed, or deleted since NCO was installed, it is possible NCO operators will fail with a message that they cannot find a dynamically loaded (aka `shared object` or `*.so`) library. This will produce a distinctive error message, such as `ld.so.1: /usr/local/bin/nces: fatal: libsunmath.so.1: can't open file: errno=2`. If you received an error message like this, ask your system administrator to diagnose whether the library is truly missing, or whether you simply need to alter your library search path. As a final remedy, you may re-compile and install NCO with all operators statically linked.

### 1.6 netCDF2/3/4 and HDF4/5 Support

netCDF version 2 was released in 1993. NCO (specifically `ncks`) began soon after this in 1994. netCDF 3.0 was released in 1996, and we were not exactly eager to convert all code to the newer, less tested netCDF implementation. One netCDF3 interface call (`nc_inq_libvers`) was added to NCO in January, 1998, to aid in maintainance and debugging. In March, 2001, the final NCO conversion to netCDF3 was completed (coincidentally on the same day netCDF 3.5 was released). NCO versions 2.0 and higher are built with the `-DNO_NETCDF_2` flag to ensure no netCDF2 interface calls are used.

However, the ability to compile NCO with only netCDF2 calls is worth maintaining because HDF version 4, aka HDF4 or simply HDF,\(^6\) (available from HDF) supports only the netCDF2 library calls (see [http://hdfgroup.org/UG41r3_html/SDS_SD.fm12.html#47784](http://hdfgroup.org/UG41r3_html/SDS_SD.fm12.html#47784)). There are two versions of HDF. Currently HDF version 4.x supports the full netCDF2 API and thus NCO version 1.2.x. If NCO version 1.2.x (or earlier) is built with only netCDF2 calls then all NCO operators should work with HDF4 files as well as netCDF

---

\(^5\) The `ldd` command, if it is available on your system, will tell you where the executable is looking for each dynamically loaded library. Use, e.g., `ldd `which nces`'.

\(^6\) The Hierarchical Data Format, or HDF, is another self-describing data format similar to, but more elaborate than, netCDF. HDF comes in two flavors, HDF4 and HDF5. Often people use the shorthand HDF to refer to the older format HDF4. People almost always use HDF5 to refer to HDF5.
files. The preprocessor token `NETCDF2_ONLY` exists in NCO version 1.2.x to eliminate all netCDF3 calls. Only versions of NCO numbered 1.2.x and earlier have this capability.

HDF version 5 became available in 1999, but did not support netCDF (or, for that matter, Fortran) as of December 1999. By early 2001, HDF5 did support Fortran90. Thanks to an NSF-funded “harmonization” partnership, HDF began to fully support the netCDF3 read interface (which is employed by NCO 2.x and later). In 2004, Unidata and THG began a project to implement the HDF5 features necessary to support the netCDF API. NCO version 3.0.3 added support for reading/writing netCDF4-formatted HDF5 files in October, 2005. See Section 3.9 [File Formats and Conversion], page 35 for more details.

HDF support for netCDF was completed with HDF5 version version 1.8 in 2007. The netCDF front-end that uses this HDF5 back-end was completed and released soon after as netCDF version 4. Download it from the netCDF4 website.

NCO version 3.9.0, released in May, 2007, added support for all netCDF4 atomic data types except `NC_STRING`. Support for `NC_STRING`, including ragged arrays of strings, was finally added in version 3.9.9, released in June, 2009. Support for additional netCDF4 features has been incremental. We add one netCDF4 feature at a time. You must build NCO with netCDF4 to obtain this support.

NCO supports many netCDF4 features including atomic data types, Lempel-Ziv compression (deflation), chunking, and groups. The new atomic data types are `NC_UBYTE`, `NC_USHORT`, `NC_UINT`, `NC_INT64`, and `NC_UINT64`. Eight-byte integer support is an especially useful improvement from netCDF3. All NCO operators support these types, e.g., `ncks` copies and prints them, `ncra` averages them, and `ncap2` processes algebraic scripts with them. `ncks` prints compression information, if any, to screen.

NCO version 3.9.1 (June, 2007) added support for netCDF4 Lempel-Ziv deflation. Lempel-Ziv deflation is a lossless compression technique. See Section 3.28 [Deflation], page 88 for more details.

NCO version 3.9.9 (June, 2009) added support for netCDF4 chunking in `ncks` and `ncecat`. NCO version 4.0.4 (September, 2010) completed support for netCDF4 chunking in the remaining operators. See Section 3.26 [Chunking], page 72 for more details.

NCO version 4.2.2 (October, 2012) added support for netCDF4 groups in `ncks` and `ncecat`. Group support for these operators was complete (e.g., regular expressions to select groups and Group Path Editing) as of NCO version 4.2.6 (March, 2013). See Section 3.13 [Group Path Editing], page 46 for more details. Group support for all other operators was finished in the NCO version 4.3.x series completed in December, 2013.

Support for netCDF4 in the first arithmetic operator, `ncbo`, was introduced in NCO version 4.3.0 (March, 2013). NCO version 4.3.1 (May, 2013) completed this support and introduced the first example of automatic group broadcasting. See Section 4.3 [ncbo netCDF Binary Operator], page 174 for more details.

---

7 One must link the NCO code to the HDF4 MFHDF library instead of the usual netCDF library. Apparently ‘MF’ stands for Multi-file not for Mike Folk. In any case, until about 2007 the MFHDF library only supported netCDF2 calls. Most people will never again install NCO 1.2.x and so will never use NCO to write HDF4 files. It is simply too much trouble.
netCDF4-enabled NCO handles netCDF3 files without change. In addition, it automatically handles netCDF4 (HDF5) files: If you feed NCO netCDF3 files, it produces netCDF3 output. If you feed NCO netCDF4 files, it produces netCDF4 output. Use the handy-dandy ‘-4’ switch to request netCDF4 output from netCDF3 input, i.e., to convert netCDF3 to netCDF4. See Section 3.9 [File Formats and Conversion], page 35 for more details.

When linked to a netCDF library that was built with HDF4 support, NCO automatically supports reading HDF4 files and writing them as netCDF3/netCDF4/HDF5 files. NCO can only write through the netCDF API, which can only write netCDF3/netCDF4/HDF5 files. So NCO can read HDF4 files, perform manipulations and calculations, and then it must write the results in netCDF format.

NCO support for HDF4 has been quite functional since December, 2013. For best results install NCO versions 4.4.0 or later on top of netCDF versions 4.3.1 or later. Getting to this point has been an iterative effort where Unidata improved netCDF library capabilities in response to our requests. NCO versions 4.3.6 and earlier do not explicitly support HDF4, yet should work with HDF4 if compiled with a version of netCDF (4.3.2 or later?) that does not unexpectedly die when probing HDF4 files with standard netCDF calls. NCO versions 4.3.7–4.3.9 (October–December, 2013) use a special flag to workaround netCDF HDF4 issues. The user must tell these versions of NCO that an input file is HDF4 format by using the ‘--hdf4’ switch.

When compiled with netCDF version 4.3.1 (20140116) or later, NCO versions 4.4.0 (January, 2014) and later more gracefully handle HDF4 files. In particular, the ‘--hdf4’ switch is obsolete. Current versions of NCO use netCDF to determine automatically whether the underlying file is HDF4, and then take appropriate precautions to avoid netCDF4 API calls that fail when applied to HDF4 files (e.g., nc_inq_var_chunking(), nc_inq_var_deflate()). When compiled with netCDF version 4.3.2 (20140423) or earlier, NCO will report that chunking and deflation properties of HDF4 files as HDF4_UNKNOWN, because determining those properties was impossible. When compiled with netCDF version 4.3.3-rc2 (20140925) or later, NCO versions 4.4.6 (October, 2014) and later fully support chunking and deflation features of HDF4 files. The ‘--hdf4’ switch is supported (for backwards compatibility) yet redundant (i.e., does no harm) with current versions of NCO and netCDF.

Converting HDF4 files to netCDF: Since NCO reads HDF4 files natively, it is now easy to convert HDF4 files to netCDF files directly, e.g.,

```bash
ncks fl.hdf fl.nc # Convert HDF4->netCDF4 (NCO 4.4.0+, netCDF 4.3.1+)
ncks --hdf4 fl.hdf fl.nc # Convert HDF4->netCDF4 (NCO 4.3.7~4.3.9)
```

The most efficient and accurate way to convert HDF4 data to netCDF format is to convert to netCDF4 using NCO as above. Many HDF4 producers (NASA!) love to use netCDF4 types, e.g., unsigned bytes, so this procedure is the most typical. Conversion of HDF4 to netCDF4 as above suffices when the data will only be processed by NCO and other netCDF4-aware tools.

However, many tools are not fully netCDF4-aware, and so conversion to netCDF3 may be desirable. Obtaining any netCDF file from an HDF4 is easy:

---

8 The procedure for doing this is documented at http://www.unidata.ucar.edu/software/netcdf/docs/build_hdf4.html.
ncks -3 fl.hdf fl.nc # HDF4->netCDF3 (NCO 4.4.0+, netCDF 4.3.1+)
ncks -4 fl.hdf fl.nc # HDF4->netCDF4 (NCO 4.4.0+, netCDF 4.3.1+)
ncks -6 fl.hdf fl.nc # HDF4->netCDF3 64-bit (NCO 4.4.0+, ...)
ncks -7 -L 1 fl.hdf fl.nc # HDF4->netCDF4 classic (NCO 4.4.0+, ...)
ncks --hdf4 -3 fl.hdf fl.nc # HDF4->netCDF3 (netCDF 4.3.0-)
ncks --hdf4 -4 fl.hdf fl.nc # HDF4->netCDF4 (netCDF 4.3.0-)
ncks --hdf4 -6 fl.hdf fl.nc # HDF4->netCDF3 64-bit (netCDF 4.3.0-)
ncks --hdf4 -7 fl.hdf fl.nc # HDF4->netCDF4 classic (netCDF 4.3.0-)

As of NCO version 4.4.0 (January, 2014), these commands work even when the HDF4 file contains netCDF4 atomic types (e.g., unsigned bytes, 64-bit integers) because NCO can autoconvert everything to atomic types supported by netCDF3.9

As of NCO version 4.4.4 (May, 2014) both ncl_convert2nc and NCO have built-in, automatic workarounds to handle element names that contain characters that are legal in HDF though are illegal in netCDF. For example, slashes and leading special characters are legal in HDF and illegal in netCDF element (i.e., group, variable, dimension, and attribute) names. NCO converts these forbidden characters to underscores, and retains the original names of variables in automatically produced attributes named hdf_name.10

Finally, in February 2014, we learned that the HDF group has a project called H4CF (described here) whose goal is to make HDF4 files accessible to CF tools and conventions. Their project includes a tool named h4tonccf that converts HDF4 files to netCDF3 or netCDF4 files. We are not yet sure what advantages or features h4tonccf has that are not in NCO, though we suspect both methods have their own advantages. Corrections welcome.

As of 2012, netCDF4 is relatively stable software. Problems with netCDF4 and HDF libraries have mainly been fixed. Binary NCO distributions shipped as RPMs and as debs have used the netCDF4 library since 2010 and 2011, respectively.

One must often build NCO from source to obtain netCDF4 support. Typically, one specifies the root of the netCDF4 installation directory. Do this with the NETCDF4_ROOT variable. Then use your preferred NCO build mechanism, e.g.,

```
export NETCDF4_ROOT=/usr/local/netcdf4 # Set netCDF4 location
cd ~/nco;./configure --enable-netcdf4 # Configure mechanism -or-
cd ~/nco/bld;./make NETCDF4=Y allinone # Old Makefile mechanism
```

9 Prior to NCO version 4.4.0 (January, 2014), we recommended the ncl_convert2nc tool to convert HDF to netCDF3 when both these are true: 1. You must have netCDF3 and 2. the HDF file contains netCDF4 atomic types. More recent versions of NCO handle this problem fine, and include other advantages so we no longer recommend ncl_convert2nc because ncks is faster and more space-efficient. Both automatically convert netCDF4 types to netCDF3 types, yet ncl_convert2nc cannot produce full netCDF4 files. In contrast, ncks will happily convert HDF straight to netCDF4 files with netCDF4 types. Hence ncks can and does preserve the variable types. Unsigned bytes stay unsigned bytes. 64-bit integers stay 64-bit integers. Strings stay strings. Hence, ncks conversions often result in smaller files than ncl_convert2nc conversions. A tool useful for converting netCDF3 to netCDF4 files is the Python script nc3tonc4 by Jeff Whitaker.

10 Two real-world examples: NCO translates the NASA CERES dimension (FOV) Footprints to _FOV_Footprints, and Cloud & Aerosol, Cloud Only, Clear Sky w/Aerosol, and Clear Sky (yes, the dimension name includes whitespace and special characters) to Cloud & Aerosol, Cloud Only, Clear Sky w_Aerosol, and Clear Sky ncl_convert2nc makes the element name netCDF-safe in a slightly different manner, and also stores the original name in the hdf_name attribute.
We carefully track the netCDF4 releases, and keep the netCDF4 atomic type support and other features working. Our long term goal is to utilize more of the extensive new netCDF4 feature set. The next major netCDF4 feature we are likely to utilize is parallel I/O. We will enable this in the MPI netCDF operators.

1.7 Help Requests and Bug Reports

We generally receive three categories of mail from users: help requests, bug reports, and feature requests. Notes saying the equivalent of “Hey, NCO continues to work great and it saves me more time everyday than it took to write this note” are a distant fourth.

There is a different protocol for each type of request. The preferred etiquette for all communications is via NCO Project Forums. Do not contact project members via personal e-mail unless your request comes with money or you have damaging information about our personal lives. Please use the Forums—they preserve a record of the questions and answers so that others can learn from our exchange. Also, since NCO is government-funded, this record helps us provide program officers with information they need to evaluate our project.

Before posting to the NCO forums described below, you might first register your name and email address with SourceForge.net or else all of your postings will be attributed to nobody. Once registered you may choose to monitor any forum and to receive (or not) email when there are any postings including responses to your questions. We usually reply to the forum message, not to the original poster.

If you want us to include a new feature in NCO, check first to see if that feature is already on the TODO list. If it is, why not implement that feature yourself and send us the patch? If the feature is not yet on the list, then send a note to the NCO Discussion forum. Read the manual before reporting a bug or posting a help request. Sending questions whose answers are not in the manual is the best way to motivate us to write more documentation. We would also like to accentuate the contrapositive of this statement. If you think you have found a real bug the most helpful thing you can do is simplify the problem to a manageable size and then report it. The first thing to do is to make sure you are running the latest publicly released version of NCO.

Once you have read the manual, if you are still unable to get NCO to perform a documented function, submit a help request. Follow the same procedure as described below for reporting bugs (after all, it might be a bug). That is, describe what you are trying to do, and include the complete commands (run with ‘-D 5’), error messages, and version of NCO (with ‘-r’). Post your help request to the NCO Help forum.

If you think you used the right command when NCO misbehaves, then you might have found a bug. Incorrect numerical answers are the highest priority. We usually fix those within one or two days. Core dumps and sementation violations receive lower priority. They are always fixed, eventually.

How do you simplify a problem that reveal a bug? Cut out extraneous variables, dimensions, and metadata from the offending files and re-run the command until it no longer breaks. Then back up one step and report the problem. Usually the file(s) will be very small, i.e., one variable with one or two small dimensions ought to suffice. Run the operator with ‘-r’ and then run the command with ‘-D 5’ to increase the verbosity of the debugging
output. It is very important that your report contain the exact error messages and compile-time environment. Include a copy of your sample input file, or place one on a publically accessible location, of the file(s). Post the full bug report to the NCO Project buglist.

Build failures count as bugs. Our limited machine access means we cannot fix all build failures. The information we need to diagnose, and often fix, build failures are the three files output by GNU build tools, `nco.config.log.${GNU_TRP}.foo`, `nco.configure.${GNU_TRP}.foo`, and `nco.make.${GNU_TRP}.foo`. The file `configure.eg` shows how to produce these files. Here `${GNU_TRP}` is the “GNU architecture triplet”, the chip-vendor-OS string returned by `config.guess`. Please send us your improvements to the examples supplied in `configure.eg`. The regressions archive at [http://dust.ess.uci.edu/nco/rgr](http://dust.ess.uci.edu/nco/rgr) contains the build output from our standard test systems. You may find you can solve the build problem yourself by examining the differences between these files and your own.
2 Operator Strategies

2.1 Philosophy

The main design goal is command line operators which perform useful, scriptable operations on netCDF files. Many scientists work with models and observations which produce too much data to analyze in tabular format. Thus, it is often natural to reduce and massage this raw or primary level data into summary, or second level data, e.g., temporal or spatial averages. These second level data may become the inputs to graphical and statistical packages, and are often more suitable for archival and dissemination to the scientific community.

NCO performs a suite of operations useful in manipulating data from the primary to the second level state. Higher level interpretive languages (e.g., IDL, Yorick, Matlab, NCL, Perl, Python), and lower level compiled languages (e.g., C, Fortran) can always perform any task performed by NCO, but often with more overhead. NCO, on the other hand, is limited to a much smaller set of arithmetic and metadata operations than these full blown languages.

Another goal has been to implement enough command line switches so that frequently used sequences of these operators can be executed from a shell script or batch file. Finally, NCO was written to consume the absolute minimum amount of system memory required to perform a given job. The arithmetic operators are extremely efficient; their exact memory usage is detailed in Section 2.9 [Memory Requirements], page 22.

2.2 Climate Model Paradigm

NCO was developed at NCAR to aid analysis and manipulation of datasets produced by General Circulation Models (GCMs). GCM datasets share many features with other gridded scientific datasets and so provide a useful paradigm for the explication of the NCO operator set. Examples in this manual use a GCM paradigm because latitude, longitude, time, temperature and other fields related to our natural environment are as easy to visualize for the layman as the expert.

2.3 Temporary Output Files

NCO operators are designed to be reasonably fault tolerant, so that a system failure or user-abort of the operation (e.g., with C-c) does not cause loss of data. The user-specified output-file is only created upon successful completion of the operation\(^1\). This is accomplished by performing all operations in a temporary copy of output-file. The name of the temporary output file is constructed by appending .pid<process ID>.<operator name>.tmp to the user-specified output-file name. When the operator completes its task with no fatal errors, the temporary output file is moved to the user-specified output-file. This imbues the process with fault-tolerance since fatal error (e.g., disk space fills up) affect only the temporary output file not created if it did not already exist. Note the construction of a temporary output file uses more disk space than just overwriting existing files “in place” (because there may be two copies of the same file on disk until the NCO operation successfully concludes and the temporary output file overwrites the existing output-file).

---

\(^1\) The ncrename and ncatted operators are exceptions to this rule. See Section 4.11 [ncrename netCDF Renamer], page 214.
Also, note this feature increases the execution time of the operator by approximately the
time it takes to copy the output-file\(^2\). Finally, note this fault-tolerant feature allows the
output-file to be the same as the input-file without any danger of “overlap”.

Over time many “power users” have requested a way to turn-off the fault-tolerance safety
feature of automatically creating a temporary file. Often these users build and execute
production data analysis scripts that are repeated frequently on large datasets. Obituating
an extra file write can then conserve significant disk space and time. For this purpose NCO
has, since version 4.2.1 in August, 2012, made configurable the controls over temporary
file creation. The ‘--wrt_tmp_fl’ and equivalent ‘--write_tmp_fl’ switches ensure NCO
writes output to an intermediate temporary file. This is and has always been the default
behavior so there is currently no need to specify these switches. However, the default
may change some day, especially since writing to RAM disks (see Section 3.31 [RAM disks],
page 90) may some day become the default. The ‘--no_tmp_fl’ switch causes NCO to write
directly to the final output file instead of to an intermediate temporary file. “Power users”
may wish to invoke this switch to increase performance (i.e., reduce wallclock time) when
manipulating large files. When eschewing temporary files, users may forsake the ability
to have the same name for both output-file and input-file since, as described above, the
temporary file prevented overlap issues. However, if the user creates the output file in RAM
(see Section 3.31 [RAM disks], page 90) then it is still possible to have the same name for
both output-file and input-file.

```
ncks in.nc out.nc # Default: create out.pid.tmp.nc then move to out.nc
ncks --wrt_tmp_fl in.nc out.nc # Same as default
ncks --no_tmp_fl in.nc out.nc # Create out.nc directly on disk
ncks --no_tmp_fl in.nc in.nc # ERROR-prone! Overwrite in.nc with itself
ncks --create_ram --no_tmp_fl in.nc in.nc # Create in RAM, write to disk
ncks --open_ram --no_tmp_fl in.nc in.nc # Read into RAM, write to disk
```

There is no reason to expect the fourth example to work. The behavior of overwriting a
file while reading from the same file is undefined, much as is the shell command ‘cat foo
> foo’. Although it may “work” in some cases, it is unreliable. One way around this is
to use ‘--create_ram’ so that the output file is not written to disk until the input file is
closed, See Section 3.31 [RAM disks], page 90. However, as of 20130328, the behavior of
the ‘--create_ram’ and ‘--open_ram’ examples has not been thoroughly tested.

The NCO authors have seen compelling use cases for utilizing the RAM switches, though
not (yet) for combining them with ‘--no_tmp_fl’. NCO implements both options because
they are largely independent of eachother. It is up to “power users” to discover which best
fit their needs. We welcome accounts of your experiences posted to the forums.

Other safeguards exist to protect the user from inadvertently overwriting data. If the
output-file specified for a command is a pre-existing file, then the operator will prompt
the user whether to overwrite (erase) the existing output-file, attempt to append to it, or
abort the operation. However, in processing large amounts of data, too many interactive
questions slows productivity. Therefore NCO also implements two ways to override its own
safety features, the ‘-O’ and ‘-A’ switches. Specifying ‘-O’ tells the operator to overwrite
any existing output-file without prompting the user interactively. Specifying ‘-A’ tells the

\(^2\) The OS-specific system move command is used. This is `mv` for UNIX, and `move` for Windows.
operator to attempt to append to any existing output-file without prompting the user interactively. These switches are useful in batch environments because they suppress interactive keyboard input.

2.4 Appending Variables
Adding variables from one file to another is often desirable. This is referred to as appending, although some prefer the terminology merging\(^3\) or pasting. Appending is often confused with what NCO calls concatenation. In NCO, concatenation refers to splicing a variable along the record dimension. The length along the record dimension of the output is the sum of the lengths of the input files. Appending, on the other hand, refers to copying a variable from one file to another file which may or may not already contain the variable\(^4\). NCO can append or concatenate just one variable, or all the variables in a file at the same time.

In this sense, ncks can append variables from one file to another file. This capability is invoked by naming two files on the command line, input-file and output-file. When output-file already exists, the user is prompted whether to overwrite, append/replace, or exit from the command. Selecting overwrite tells the operator to erase the existing output-file and replace it with the results of the operation. Selecting exit causes the operator to exit—the output-file will not be touched in this case. Selecting append/replace causes the operator to attempt to place the results of the operation in the existing output-file, See Section 4.7 [ncks netCDF Kitchen Sink], page 188.

The simplest way to create the union of two files is

\[ \text{ncks -A fl}_1\.nc \text{ fl}_2\.nc \]

This puts the contents of fl\_1.nc into fl\_2.nc. The ‘-A’ is optional. On output, fl\_2.nc is the union of the input files, regardless of whether they share dimensions and variables, or are completely disjoint. The append fails if the input files have differently named record dimensions (since netCDF supports only one), or have dimensions of the same name but different sizes.

2.5 Simple Arithmetic and Interpolation
Users comfortable with NCO semantics may find it easier to perform some simple mathematical operations in NCO rather than higher level languages. ncbo (see Section 4.3 [ncbo netCDF Binary Operator], page 174) does file addition, subtraction, multiplication, division, and broadcasting. It even does group broadcasting. ncflint (see Section 4.6 [ncflint netCDF File Interpolator], page 185) does file addition, subtraction, multiplication and interpolation. Sequences of these commands can accomplish simple yet powerful operations from the command line.

---

\(^3\) The terminology merging is reserved for an (unwritten) operator which replaces hyperslabs of a variable in one file with hyperslabs of the same variable from another file

\(^4\) Yes, the terminology is confusing. By all means mail me if you think of a better nomenclature. Should NCO use paste instead of append?
2.6 Statistics vs. Concatenation

The most frequently used operators of NCO are probably the statisticians (i.e., tools that do statistics) and concatenators. Because there are so many types of statistics like averaging (e.g., across files, within a file, over the record dimension, over other dimensions, with or without weights and masks) and of concatenating (across files, along the record dimension, along other dimensions), there are currently no fewer than five operators which tackle these two purposes: ncra, nces, ncwa, ncrcat, and ncecat. These operators do share many capabilities\(^5\), though each has its unique specialty. Two of these operators, ncrcat and ncecat, concatenate hyperslabs across files. The other two operators, ncra and nces, compute statistics across (and/or within) files\(^6\). First, let’s describe the concatenators, then the statistics tools.

2.6.1 Concatenators ncrcat and ncecat

Joining together independent files along a common record dimension is called concatenation. ncrcat is designed for concatenating record variables, while ncecat is designed for concatenating fixed length variables. Consider five files, 85.nc, 86.nc, ... 89.nc each containing a year’s worth of data. Say you wish to create from them a single file, 8589.nc containing all the data, i.e., spanning all five years. If the annual files make use of the same record variable, then ncrcat will do the job nicely with, e.g., ncrcat 8?.nc 8589.nc.

The number of records in the input files is arbitrary and can vary from file to file. See Section 4.10 [ncrcat netCDF Record Concatenator], page 212, for a complete description of ncrcat.

However, suppose the annual files have no record variable, and thus their data are all fixed length. For example, the files may not be conceptually sequential, but rather members of the same group, or ensemble. Members of an ensemble may have no reason to contain a record dimension. ncecat will create a new record dimension (named record by default) with which to glue together the individual files into the single ensemble file. If ncecat is used on files which contain an existing record dimension, that record dimension is converted to a fixed-length dimension of the same name and a new record dimension (named record) is created. Consider five realizations, 85a.nc, 85b.nc, ... 85e.nc of 1985 predictions from the same climate model. Then ncecat 85?.nc 85_ens.nc glues together the individual realizations into the single file, 85_ens.nc. If an input variable was dimensioned [lat,lon], it will have dimensions [record,lat,lon] in the output file. A restriction of ncecat is that the hyperslabs of the processed variables must be the same from file to file. Normally this means all the input files are the same size, and contain data on different realizations of the same variables. See Section 4.5 [ncecat netCDF Ensemble Concatenator], page 182, for a complete description of ncecat.

ncpdq makes it possible to concatenate files along any dimension, not just the record dimension. First, use ncpdq to convert the dimension to be concatenated (i.e., extended

---

\(^5\) Currently nces and ncrcat are symbolically linked to the ncra executable, which behaves slightly differently based on its invocation name (i.e., ‘argv[0]’). These three operators share the same source code, and merely have different inner loops.

\(^6\) The third averaging operator, ncwa, is the most sophisticated averager in NCO. However, ncwa is in a different class than ncra and nces because it operates on a single file per invocation (as opposed to multiple files). On that single file, however, ncwa provides a richer set of averaging options—including weighting, masking, and broadcasting.
with data from other files) into the record dimension. Second, use `ncrcat` to concatenate these files. Finally, if desirable, use `ncpdq` to revert to the original dimensionality. As a concrete example, say that files `x_01.nc`, `x_02.nc`, ..., `x_10.nc` contain time-evolving datasets from spatially adjacent regions. The time and spatial coordinates are `time` and `x`, respectively. Initially the record dimension is `time`. Our goal is to create a single file that contains joins all the spatially adjacent regions into one single time-evolving dataset.

```bash
for idx in 01 02 03 04 05 06 07 08 09 10; do # Bourne Shell
    ncpdq -a x,time x_${idx}.nc foo_${idx}.nc # Make x record dimension
done
ncrcat foo_??.nc out.nc # Concatenate along x
ncpdq -a time,x out.nc out.nc # Revert to time as record dimension
```

Note that `ncrcat` will not concatenate fixed-length variables, whereas `ncrcat` concatenates both fixed-length and record variables along a new record variable. To conserve system memory, use `ncrcat` where possible.

### 2.6.2 Averagers `nces`, `ncra`, and `ncwa`

The differences between the averagers `ncra` and `nces` are analogous to the differences between the concatenators. `ncra` is designed for averaging record variables from at least one file, while `nces` is designed for averaging fixed length variables from multiple files. `ncra` performs a simple arithmetic average over the record dimension of all the input files, with each record having an equal weight in the average. `nces` performs a simple arithmetic average of all the input files, with each file having an equal weight in the average. Note that `ncra` cannot average fixed-length variables, but `nces` can average both fixed-length and record variables. To conserve system memory, use `ncra` rather than `nces` where possible (e.g., if each input-file is one record long). The file output from `nces` will have the same dimensions (meaning dimension names as well as sizes) as the input hyperslabs (see Section 4.4 [nces netCDF Ensemble Statistics], page 179, for a complete description of `nces`). The file output from `ncra` will have the same dimensions as the input hyperslabs except for the record dimension, which will have a size of 1 (see Section 4.9 [ncra netCDF Record Averager], page 210, for a complete description of `ncra`).

### 2.6.3 Interpolator `ncflint`

`ncflint` can interpolate data between or two files. Since no other operators have this ability, the description of interpolation is given fully on the `ncflint` reference page (see Section 4.6 [ncflint netCDF File Interpolator], page 185). Note that this capability also allows `ncflint` to linearly rescale any data in a netCDF file, e.g., to convert between differing units.

### 2.7 Large Numbers of Files

Occasionally one desires to digest (i.e., concatenate or average) hundreds or thousands of input files. Unfortunately, data archives (e.g., NASA EOSDIS) may not name netCDF files in a format understood by the `-n loop` switch (see Section 3.5 [Specifying Input Files], page 28) that automagically generates arbitrary numbers of input filenames. The `-n loop` switch has the virtue of being concise, and of minimizing the command line. This helps keeps output file small since the command line is stored as metadata in the `history` attribute (see Section 3.36 [History Attribute], page 108). However, the `-n loop` switch is useless when
there is no simple, arithmetic pattern to the input filenames (e.g., h00001.nc, h00002.nc, ... h90210.nc). Moreover, filename globbing does not work when the input files are too numerous or their names are too lengthy (when strung together as a single argument) to be passed by the calling shell to the NCO operator. When this occurs, the ANSI C-standard argc-argv method of passing arguments from the calling shell to a C-program (i.e., an NCO operator) breaks down. There are (at least) three alternative methods of specifying the input filenames to NCO in environment-limited situations.

The recommended method for sending very large numbers (hundreds or more, typically) of input filenames to the multi-file operators is to pass the filenames with the UNIX standard input feature, aka stdin:

```
# Pipe large numbers of filenames to stdin
/bin/ls | grep ${CASEID}_"......".nc | ncecat -o foo.nc
```

This method avoids all constraints on command line size imposed by the operating system. A drawback to this method is that the history attribute (see Section 3.36 [History Attribute], page 108) does not record the name of any input files since the names were not passed on the command line. This makes determining the data provenance at a later date difficult. To remedy this situation, multi-file operators store the number of input files in the nco_input_file_number global attribute and the input file list itself in the nco_input_file_list global attribute (see Section 3.37 [File List Attributes], page 109). Although this does not preserve the exact command used to generate the file, it does retains all the information required to reconstruct the command and determine the data provenance.

A second option is to use the UNIX xargs command. This simple example selects as input to xargs all the filenames in the current directory that match a given pattern. For illustration, consider a user trying to average millions of files which each have a six character filename. If the shell buffer cannot hold the results of the corresponding globbing operator, ??????.nc, then the filename globbing technique will fail. Instead we express the filename pattern as an extended regular expression, ......\.nc (see Section 3.11 [Subsetting Files], page 40). We use grep to filter the directory listing for this pattern and to pipe the results to xargs which, in turn, passes the matching filenames to an NCO multi-file operator, e.g., ncecat.

```
# Use xargs to transfer filenames on the command line
/bin/ls | grep ${CASEID}_"......".nc | xargs -x ncecat -o foo.nc
```

The single quotes protect the only sensitive parts of the extended regular expression (the grep argument), and allow shell interpolation (the ${CASEID} variable substitution) to proceed unhindered on the rest of the command. xargs uses the UNIX pipe feature to append the suitably filtered input file list to the end of the ncecat command options. The -o foo.nc switch ensures that the input files supplied by xargs are not confused with the output file name. xargs does, unfortunately, have its own limit (usually about 20,000 characters) on the size of command lines it can pass. Give xargs the ‘-x’ switch to ensure it

---

7 The exact length which exceeds the operating system internal limit for command line lengths varies from OS to OS and from shell to shell. GNU bash may not have any arbitrary fixed limits to the size of command line arguments. Many OSs cannot handle command line arguments (including results of file globbing) exceeding 4096 characters.
dies if it reaches this internal limit. When this occurs, use either the `stdin` method above, or the symbolic link presented next.

Even when its internal limits have not been reached, the `xargs` technique may not be sophisticated enough to handle all situations. A full scripting language like Perl can handle any level of complexity of filtering input filenames, and any number of filenames. The technique of last resort is to write a script that creates symbolic links between the irregular input filenames and a set of regular, arithmetic filenames that the ‘-n loop’ switch understands. For example, the following Perl script creates a monotonically enumerated symbolic link to up to one million `.nc` files in a directory. If there are 999,999 netCDF files present, the links are named 000001.nc to 999999.nc:

```bash
# Create enumerated symbolic links
/bin/ls | grep \.nc | perl -e \n    "$idx=1;while(<STDIN>)\{chop;symlink \$_,sprintf("%06d.nc",$idx++);\}\nncescat -n 999999,6,1 000001.nc foo.nc
# Remove symbolic links when finished
/bin/rm ??????.nc
```

The ‘-n loop’ option tells the NCO operator to automatically generate the filenames of the symbolic links. This circumvents any OS and shell limits on command line size. The symbolic links are easily removed once NCO is finished. One drawback to this method is that the `history` attribute (see Section 3.36 [History Attribute], page 108) retains the filename list of the symbolic links, rather than the data files themselves. This makes it difficult to determine the data provenance at a later date.

### 2.8 Large Datasets

Large datasets are those files that are comparable in size to the amount of random access memory (RAM) in your computer. Many users of NCO work with files larger than 100 MB. Files this large not only push the current edge of storage technology, they present special problems for programs which attempt to access the entire file at once, such as `nces` and `ncecat`. If you work with a 300 MB files on a machine with only 32 MB of memory then you will need large amounts of swap space (virtual memory on disk) and NCO will work slowly, or even fail. There is no easy solution for this. The best strategy is to work on a machine with sufficient amounts of memory and swap space. Since about 2004, many users have begun to produce or analyze files exceeding 2 GB in size. These users should familiarize themselves with NCO’s Large File Support (LFS) capabilities (see Section 3.10 [Large File Support], page 40). The next section will increase your familiarity with NCO’s memory requirements. With this knowledge you may re-design your data reduction approach to divide the problem into pieces solvable in memory-limited situations.

If your local machine has problems working with large files, try running NCO from a more powerful machine, such as a network server. If you get a memory-related core dump (e.g., ‘Error exit (core dumped)’) on a GNU/Linux system, or the operation ends before the entire output file is written, try increasing the process-available memory with `ulimit`:

```
ulimit -f unlimited
```
This may solve constraints on clusters where sufficient hardware resources exist yet where system administrators felt it wise to prevent any individual user from consuming too much of resource. Certain machine architectures, e.g., Cray UNICOS, have special commands which allow one to increase the amount of interactive memory. On Cray systems, try to increase the available memory with the `ilimit` command.

The speed of the NCO operators also depends on file size. When processing large files the operators may appear to hang, or do nothing, for large periods of time. In order to see what the operator is actually doing, it is useful to activate a more verbose output mode. This is accomplished by supplying a number greater than 0 to the `-D debug-level` (or `--debug-level`, or `--dbg_lvl`) switch. When the `debug-level` is nonzero, the operators report their current status to the terminal through the `stderr` facility. Using `-D` does not slow the operators down. Choose a `debug-level` between 1 and 3 for most situations, e.g., `nces -D 2 85.nc 86.nc 8586.nc`. A full description of how to estimate the actual amount of memory the multi-file NCO operators consume is given in Section 2.9 [Memory Requirements], page 22.

### 2.9 Memory Requirements

Many people use NCO on gargantuan files which dwarf the memory available (free RAM plus swap space) even on today’s powerful machines. These users want NCO to consume the least memory possible so that their scripts do not have to tediously cut files into smaller pieces that fit into memory. We commend these greedy users for pushing NCO to its limits!

This section describes the memory NCO requires during operation. The required memory is based on the underlying algorithms. The description below is the memory usage per thread. Users with shared memory machines may use the threaded NCO operators (see Section 3.3 [OpenMP Threading], page 25). The peak and sustained memory usage will scale accordingly, i.e., by the number of threads. Memory consumption patterns of all operators are similar, with the exception of `ncap2`.

#### 2.9.1 Single and Multi-file Operators

The multi-file operators currently comprise the record operators, `ncra` and `ncrcat`, and the ensemble operators, `nces` and `ncecat`. The record operators require much less memory than the ensemble operators. This is because the record operators operate on one single record (i.e., time-slice) at a time, whereas the ensemble operators retrieve the entire variable into memory. Let $MS$ be the peak sustained memory demand of an operator, $FT$ be the memory required to store the entire contents of all the variables to be processed in an input file, $FR$ be the memory required to store the entire contents of a single record of each of the variables to be processed in an input file, $VR$ be the memory required to store a single record of the largest record variable to be processed in an input file, $VT$ be the memory required to store the largest variable to be processed in an input file, $VI$ be the memory required to store the largest variable which is not processed, but is copied from the initial file to the output file. All operators require $MI = VI$ during the initial copying of variables from the first input file to the output file. This is the initial (and transient) memory demand. The sustained memory demand is that memory required by the operators during the processing (i.e., averaging, concatenation) phase which lasts until all the input files have been processed. The operators have the following memory requirements: `ncrcat`
requires \( MS \leq VR \). \texttt{nccat} requires \( MS \leq VT \). \texttt{ncra} requires \( MS = 2FR + VR \). \texttt{nces} requires \( MS = 2FT + VT \). \texttt{ncbo} requires \( MS \leq 3VT \) (both input variables and the output variable). \texttt{ncflint} requires \( MS \leq 3VT \) (both input variables and the output variable). \texttt{ncpdq} requires \( MS \leq 2VT \) (one input variable and the output variable). \texttt{ncwa} requires \( MS \leq 8VT \) (see below). Note that only variables that are processed, e.g., averaged, concatenated, or differenced, contribute to \( MS \). Variables which do not appear in the output file (see Section 3.11 [Subsetting Files], page 40) are never read and contribute nothing to the memory requirements.

Further note that some operators perform internal type-promotion on some variables prior to arithmetic (see Section 3.34 [Type Conversion], page 99). For example, \texttt{ncra} and \texttt{nces} both promote integer types to double-precision floating point prior to arithmetic, then perform the arithmetic, then demote back to the original integer type after arithmetic. This preserves the on-disk storage type while obtaining the accuracy advantages of floating point arithmetic. Since version 4.3.6 (released in September, 2013), NCO also by default converts single-precision floating point to double-precision prior to arithmetic, which incurs the same RAM penalty. Hence, the sustained memory required for integer variables and single-precision floats are two or four-times their on-disk, uncompressed, unpacked sizes if they meet the rules for automatic internal promotion. Put another way, disabling auto-promotion of single-precision variables (with ‘\texttt{--flt}’) considerably reduces the RAM footprint of arithmetic operators.

The ‘\texttt{--open_ram}’ switch (and switches that invoke it like ‘\texttt{--ram_all}’ and ‘\texttt{--diskless_all}’) incurs a RAM penalty. These switches cause each input file to be copied to RAM upon opening. Hence any operator invoking these switches utilizes an additional \( FT \) of RAM (i.e., \( MS+ = FT \)). See Section 3.31 [RAM disks], page 90 for further details.

\texttt{ncwa} consumes between two and seven times the memory of a variable in order to process it. Peak consumption occurs when storing simultaneously in memory one input variable, one tally array, one input weight, one conformed/working weight, one weight tally, one input mask, one conformed/working mask, and one output variable. When invoked, the weighting and masking features contribute up to three-sevenths and two-sevenths of these requirements apiece. If weights and masks are not specified (i.e., no \texttt{--w} or \texttt{--a} options) then \texttt{ncwa} requirements drop to \( MS \leq 3VT \) (one input variable, one tally array, and the output variable).

The above memory requirements must be multiplied by the number of threads \( thr\_nbr \) (see Section 3.3 [OpenMP Threading], page 25). If this causes problems then reduce (with ‘\texttt{--t thr\_nbr}’) the number of threads.

### 2.9.2 Memory for \texttt{ncap2}

\texttt{ncap2} has unique memory requirements due its ability to process arbitrarily long scripts of any complexity. All scripts acceptable to \texttt{ncap2} are ultimately processed as a sequence of binary or unary operations. \texttt{ncap2} requires \( MS \leq 2VT \) under most conditions. An exception to this is when left hand casting (see Section 4.1.4 [Left hand casting], page 122) is used to stretch the size of derived variables beyond the size of any input variables. Let \( VC \) be the memory required to store the largest variable defined by left hand casting. In this case, \( MS \leq 2VC \).
ncap2 scripts are complete dynamic and may be of arbitrary length. A script that contains many thousands of operations, may uncover a slow memory leak even though each single operation consumes little additional memory. Memory leaks are usually identifiable by their memory usage signature. Leaks cause peak memory usage to increase monotonically with time regardless of script complexity. Slow leaks are very difficult to find. Sometimes a malloc() (or new[]) failure is the only noticeable clue to their existence. If you have good reasons to believe that a memory allocation failure is ultimately due to an NCO memory leak (rather than inadequate RAM on your system), then we would be very interested in receiving a detailed bug report.

2.10 Performance


It is helpful to be aware of the aspects of NCO design that can limit its performance:

1. No data buffering is performed during nc_get_var and nc_put_var operations. Hyperslabs too large too hold in core memory will suffer substantial performance penalties because of this.
2. Since coordinate variables are assumed to be monotonic, the search for bracketing the user-specified limits should employ a quicker algorithm, like bisection, than the two-sided incremental search currently implemented.
3. C_format, FORTRAN_format, signedness, scale_format and add_offset attributes are ignored by ncks when printing variables to screen.
4. In the late 1990s it was discovered that some random access operations on large files on certain architectures (e.g., UNICOS) were much slower with NCO than with similar operations performed using languages that bypass the netCDF interface (e.g., Yorick). This may have been a penalty of unnecessary byte-swapping in the netCDF interface. It is unclear whether such problems exist in present day (2007) netCDF/NCO environments, where unnecessary byte-swapping has been reduced or eliminated.
3 Shared Features

Many features have been implemented in more than one operator and are described here for brevity. The description of each feature is preceded by a box listing the operators for which the feature is implemented. Command line switches for a given feature are consistent across all operators wherever possible. If no “key switches” are listed for a feature, then that particular feature is automatic and cannot be controlled by the user.

3.1 Internationalization

Availability: All operators

NCO support for internationalization of textual input and output (e.g., Warning messages) is nascent. We introduced the first foreign language string catalogues (French and Spanish) in 2004, yet did not activate these in distributions because the catalogues were nearly empty. We seek volunteers to populate our templates with translations for their favorite languages.

3.2 Metadata Optimization

Availability: All operators
Short options: None
Long options: ‘--hdr_pad’, ‘--header_pad’

NCO supports padding headers to improve the speed of future metadata operations. Use the ‘--hdr_pad’ and ‘--header_pad’ switches to request that hdr_pad bytes be inserted into the metadata section of the output file. Future metadata expansions will not incur the netCDF3 performance penalty of copying the entire output file unless the expansion exceeds the amount of header padding exceeded. This can be beneficial when it is known that some metadata will be added at a future date. The operators which benefit most from judicious use of header padding are ncatfile and ncrename, since they only alter metadata.

This optimization exploits the netCDF library nc__enddef() function, which behaves differently with different versions of netCDF. It will improve speed of future metadata expansion with CLASSIC and 64bit netCDF files, though not necessarily with NETCDF4 files, i.e., those created by the netCDF interface to the HDF5 library (see Section 3.9 [File Formats and Conversion], page 35).

3.3 OpenMP Threading
NCO supports shared memory parallelism (SMP) when compiled with an OpenMP-enabled compiler. Threads requests and allocations occur in two stages. First, users may request a specific number of threads \texttt{thr_nbr} with the \texttt{-t} switch (or its long option equivalents, \texttt{--thr_nbr}, \texttt{--threads}, and \texttt{--omp_num_threads}). If not user-specified, OpenMP obtains \texttt{thr_nbr} from the \texttt{OMP_NUM_THREADS} environment variable, if present, or from the OS, if not.

Caveat: Unfortunately, threading does not improve NCO throughput (i.e., wallclock time) because nearly all NCO operations are I/O-bound. This means that NCO spends negligible time doing anything compared to reading and writing. We have seen some and can imagine other use cases where \texttt{ncwa}, \texttt{ncpdq}, and \texttt{ncap2} (with long scripts) will complete faster due to threading. The main benefits of threading so far have been to isolate the serial from parallel portions of code. This parallelism is now exploited by OpenMP but then runs into the I/O bottleneck during output. The bottleneck will be ameliorated for large files by the use of MPI-enabled calls in the netCDF4 library when the underlying filesystem is parallel (e.g., PVFS or JFS). Implementation of the parallel output calls in NCO is not a goal of our current funding and would require new volunteers or funding.

NCO may modify \texttt{thr_nbr} according to its own internal settings before it requests any threads from the system. Certain operators contain hard-code limits to the number of threads they request. We base these limits on our experience and common sense, and to reduce potentially wasteful system usage by inexperienced users. For example, \texttt{ncrcat} is extremely I/O-intensive so we restrict \texttt{thr_nbr} \leq 2 for \texttt{ncrcat}. This is based on the notion that the best performance that can be expected from an operator which does no arithmetic is to have one thread reading and one thread writing simultaneously. In the future (perhaps with netCDF4), we hope to demonstrate significant threading improvements with operators like \texttt{ncrcat} by performing multiple simultaneous writes.

Compute-intensive operators (\texttt{ncap2}, \texttt{ncwa} and \texttt{ncpdq}) benefit most from threading. The greatest increases in throughput due to threading occur on large datasets where each thread performs millions, at least, of floating point operations. Otherwise, the system overhead of setting up threads probably outweighs the speed enhancements due to SMP parallelism. However, we have not yet demonstrated that the SMP parallelism scales beyond four threads for these operators. Hence we restrict \texttt{thr_nbr} \leq 4 for all operators. We encourage users to play with these limits (edit file \texttt{nco_omp.c}) and send us their feedback.

Once the initial \texttt{thr_nbr} has been modified for any operator-specific limits, NCO requests the system to allocate a team of \texttt{thr_nbr} threads for the body of the code. The operating system then decides how many threads to allocate based on this request. Users may keep track of this information by running the operator with \texttt{dbg_lvl > 0}.

By default, threaded operators attach one global attribute, \texttt{nco_openmp_thread_number}, to any file they create or modify. This attribute contains the number of threads the op-
erator used to process the input files. This information helps to verify that the answers with threaded and non-threaded operators are equal to within machine precision. This information is also useful for benchmarking.

### 3.4 Command Line Options

NCO achieves flexibility by using *command line options*. These options are implemented in all traditional UNIX commands as single letter *switches*, e.g., `ls -l`. For many years NCO used only single letter option names. In late 2002, we implemented GNU/POSIX extended or long option names for all options. This was done in a backward compatible way such that the full functionality of NCO is still available through the familiar single letter options. In the future, however, some features of NCO may require the use of long options, simply because we have nearly run out of single letter options. More importantly, mnemonics for single letter options are often non-intuitive so that long options provide a more natural way of expressing intent.

Extended options, also called long options, are implemented using the system-supplied `getopt.h` header file, if possible. This provides the `getopt_long` function to NCO.\(^1\)

The syntax of *short options* (single letter options) is `-key value` (dash-key-space-value). Here, key is the single letter option name, e.g., `-D 2`.

The syntax of *long options* (multi-letter options) is `--long_name value` (dash-dash-key-space-value), e.g., `--dbg_lvl=3`. Thus the following are all valid for the `-D` (short version) or `--dbg_lvl` (long version) command line option.

```bash
ncks -D 3 in.nc # Short option
ncks --dbg_lvl=3 in.nc # Long option, preferred form
ncks --dbg_lvl=3 in.nc # Long option, alternate form
```

The last example is preferred for two reasons. First, `--dbg_lvl` is more specific and less ambiguous than `-D`. The long option form makes scripts more self documenting and less error prone. Often long options are named after the source code variable whose value they carry. Second, the equals sign `=` joins the key (i.e., `long_name`) to the value in an uninterruptible text block. Experience shows that users are less likely to mis-parse commands when restricted to this form.

GNU implements a superset of the POSIX standard which allows any unambiguous truncation of a valid option to be used.

```bash
ncks -D 3 in.nc # Short option
ncks --dbg_lvl=3 in.nc # Long option, full form
```

---

\(^1\) If a `getopt_long` function cannot be found on the system, NCO will use the `getopt_long` from the `my_getopt` package by Benjamin Sittler `bsittler@iname.com`. This is BSD-licensed software available from [http://www.geocities.com/ResearchTriangle/Node/9405/#my_getopt](http://www.geocities.com/ResearchTriangle/Node/9405/#my_getopt).
ncks --db=3 in.nc  # Long option, ambiguous truncation
ncks --d=3 in.nc   # Long option, ambiguous truncation
ncks --dbg=3 in.nc # Long option, unambiguous truncation
ncks --dbg_lvl=3 in.nc # Long option, unambiguous truncation

The first four examples are equivalent and will work as expected. The final example will exit with an error since ncks cannot disambiguate whether ‘--d’ is intended as a truncation of ‘--dbg_lvl’, of ‘--dimension’, or of some other long option.

NCO provides many long options for common switches. For example, the debugging level may be set in all operators with any of the switches ‘-D’, ‘--debug-level’, or ‘--dbg_lvl’. This flexibility allows users to choose their favorite mnemonic. For some, it will be ‘--debug’ (an unambiguous truncation of ‘--debug-level’, and other will prefer ‘--dbg’. Interactive users usually prefer the minimal amount of typing, i.e., ‘-D’. We recommend that scripts which are re-usable employ some form of the long options for future maintainability.

This manual generally uses the short option syntax in examples. This is for historical reasons and to conserve space in printed output. Users are expected to pick the unambiguous truncation of each option name that most suits their taste.

3.5 Specifying Input Files

<table>
<thead>
<tr>
<th>Availability (-n):</th>
<th>nces, ncecat, ncra, nrcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability (-p):</td>
<td>All operators</td>
</tr>
<tr>
<td>Short options:</td>
<td>‘-n’, ‘-p’</td>
</tr>
<tr>
<td>Long options:</td>
<td>‘--nintap’, ‘--pth’, ‘--path’</td>
</tr>
</tbody>
</table>

It is important that users be able to specify multiple input files without typing every filename in full, often a tedious task even by graduate student standards. There are four different ways of specifying input files to NCO: explicitly typing each, using UNIX shell wildcards, and using the NCO ‘-n’ and ‘-p’ switches (or their long option equivalents, ‘--nintap’ or ‘--pth’ and ‘--path’, respectively). Techniques to augment these methods to specify arbitrary numbers (e.g., thousands) and patterns of filenames are discussed separately (see Section 2.7 [Large Numbers of Files], page 19).

To illustrate these methods, consider the simple problem of using ncra to average five input files, 85.nc, 86.nc, ... 89.nc, and store the results in 8589.nc. Here are the four methods in order. They produce identical answers.

ncra 85.nc 86.nc 87.nc 88.nc 89.nc 8589.nc
ncra 8[56789].nc 8589.nc
ncra -p input-path 85.nc 86.nc 87.nc 88.nc 89.nc 8589.nc
ncra -n 5,2,1 85.nc 8589.nc

The first method (explicitly specifying all filenames) works by brute force. The second method relies on the operating system shell to glob (expand) the regular expression 8[56789].nc. The shell then passes the valid filenames (those which match the regular expansion) to ncra. In this case ncra never knows that a regular expression was used, because the shell intercepts and expands and matches the regular expression before ncra is actually
invoked. The third method uses the `-p input-path' argument to specify the directory where all the input files reside. NCO prepends `input-path' (e.g., `/data/username/model') to all `input-file' (though not to `output-file'). Thus, using `-p', the path to any number of input files need only be specified once. Note `input-path' need not end with `/'; the `/' is automatically generated if necessary.

The last method passes (with `-n') syntax concisely describing the entire set of filenames2. This option is only available with the multi-file operators: `ncra', `ncrcat', `nces', and `ncecat'. By definition, multi-file operators are able to process an arbitrary number of `input-files'. This option is very useful for abbreviating lists of filenames representable as `alphanumeric_prefix+numeric_suffix+filetype' where `alphanumeric_prefix' is a string of arbitrary length and composition, `numeric_suffix' is a fixed width field of digits, and `filetype' is a standard filetype indicator. For example, in the file `ccm3_h0001.nc', we have `alphanumeric_prefix' = `ccm3_h', `numeric_suffix' = `0001', and `filetype' = `nc'.

NCO decodes lists of such filenames encoded using the `-n' syntax. The simpler (3-argument) `-n' usage takes the form `-n file_number, digit_number, numeric_increment' where `file_number' is the number of files, `digit_number' is the fixed number of numeric digits comprising the `numeric_suffix', and `numeric_increment' is the constant, integer-valued difference between the `numeric_suffix' of any two consecutive files. The value of `alphanumeric_prefix' is taken from the input file, which serves as a template for decoding the filenames. In the example above, the encoding `-n 5,2,1' along with the input file name `85.nc' tells NCO to construct five (5) filenames identical to the template `85.nc' except that the final two (2) digits are a numeric suffix to be incremented by one (1) for each successive file. Currently `filetype' may be either be empty, `nc', `cdf', `hdf', or `hd5'. If present, these `filetype' suffixes (and the preceding .) are ignored by NCO as it uses the `-n' arguments to locate, evaluate, and compute the `numeric_suffix' component of filenames.

Recently the `-n' option has been extended to allow convenient specification of filenames with “circular” characteristics. This means it is now possible for NCO to automatically generate filenames which increment regularly until a specified maximum value, and then wrap back to begin again at a specified minimum value. The corresponding `-n' usage becomes more complex, taking one or two additional arguments for a total of four or five, respectively: `-n file_number, digit_number, numeric_increment[, numeric_max[, numeric_min]]' where `numeric_max', if present, is the maximum integer-value of `numeric_suffix' and `numeric_min', if present, is the minimum integer-value of `numeric_suffix'. Consider, for example, the problem of specifying non-consecutive input files where the filename suffixes end with the month index. In climate modeling it is common to create summertime and wintertime averages which contain the averages of the months June–July–August, and December–January–February, respectively:

```
ncra -n 3,2,1 85_06.nc 85_0608.nc
ncra -n 3,2,1,12 85_12.nc 85_1202.nc
ncra -n 3,2,1,12,1 85_12.nc 85_1202.nc
```

2 The `-n' option is a backward-compatible superset of the NINTAP option from the NCAR CCM Processor. The CCM Processor was custom-written Fortran code maintained for many years by Lawrence Buja at NCAR, and phased-out in the late 1990s. NCO stole some ideas, like NINTAP-functionality, from the CCM Processor capabilities.
The first example shows that three arguments to the ‘-n’ option suffice to specify consecutive months (06, 07, 08) which do not “wrap” back to a minimum value. The second example shows how to use the optional fourth and fifth elements of the ‘-n’ option to specify a wrap value. The fourth argument to ‘-n’, when present, specifies the maximum integer value of numeric_suffix. In the example the maximum value is 12, and will be formatted as 12 in the filename string. The fifth argument to ‘-n’, when present, specifies the minimum integer value of numeric_suffix. The default minimum filename suffix is 1, which is formatted as 01 in this case. Thus the second and third examples have the same effect, that is, they automatically generate, in order, the filenames 85_12.nc, 85_01.nc, and 85_02.nc as input to NCO.

3.6 Specifying Output Files

<table>
<thead>
<tr>
<th>Availability: All operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: ‘-o’</td>
</tr>
<tr>
<td>Long options: ‘--fl_out’, ‘--output’</td>
</tr>
</tbody>
</table>

NCO commands produce no more than one output file, fl_out. Traditionally, users specify fl_out as the final argument to the operator, following all input file names. This is the positional argument method of specifying input and output file names. The positional argument method works well in most applications. NCO also supports specifying fl_out using the command line switch argument method, ‘-o fl_out’.

Specifying fl_out with a switch, rather than as a positional argument, allows fl_out to precede input files in the argument list. This is particularly useful with multi-file operators for three reasons. Multi-file operators may be invoked with hundreds (or more) filenames. Visual or automatic location of fl_out in such a list is difficult when the only syntactic distinction between input and output files is their position. Second, specification of a long list of input files may be difficult (see Section 2.7 [Large Numbers of Files], page 19). Making the input file list the final argument to an operator facilitates using xargs for this purpose. Some alternatives to xargs are heinous and undesirable. Finally, many users are more comfortable specifying output files with ‘-o fl_out’ near the beginning of an argument list. Compilers and linkers are usually invoked this way.

Users should specify fl_out using either (not both) method. If fl_out is specified twice (once with the switch and once as the last positional argument), then the positional argument takes precedence.

3.7 Accessing Remote Files

<table>
<thead>
<tr>
<th>Availability: All operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: ‘-p’, ‘-l’</td>
</tr>
<tr>
<td>Long options: ‘--pth’, '--path', '--lcl', '--local'</td>
</tr>
</tbody>
</table>

NCO also supports accessing remote files through the ‘-p’ or ‘--pth’ option. This allows users to specify remote file locations in the argument list. The ‘-p’ option is particularly useful for invoking multi-file operators with remote file specifications. Users can specify remote filenames using either the positional or switch argument methods. The positional argument method is preferred for its ease of use and flexibility. Some alternatives to using ‘--pth’ are heinous and undesirable. Finally, many users are more comfortable specifying remote files with ‘-p fl_out’ near the beginning of an argument list. Compilers and linkers are usually invoked this way.

Users should specify remote files using either (not both) method. If remote files are specified twice (once with the switch and once as the last positional argument), then the positional argument takes precedence.
All NCO operators can retrieve files from remote sites as well as from the local file system. A remote site can be an anonymous FTP server, a machine on which the user has rcp, scp, or sftp privileges, NCAR’s Mass Storage System (MSS), or an OPeNDAP server. Examples of each are given below, following a brief description of the particular access protocol.

To access a file via an anonymous FTP server, simply supply the remote file’s URL. Anonymous FTP usually requires no further credentials, e.g., no .netrc file is necessary. FTP is an intrinsically insecure protocol because it transfers passwords in plain text format. Users should access sites using anonymous FTP, or better yet, secure FTP (SFTP, see below) when possible. Some FTP servers require a login/password combination for a valid user account. NCO allows transactions that require additional credentials so long as the required information is stored in the .netrc file. Usually this information is the remote machine name, login, and password, in plain text, separated by those very keywords, e.g.,

```
machine dust.ess.uci.edu login zender password bushlied
```

Eschew using valuable passwords for FTP transactions, since .netrc passwords are potentially exposed to eavesdropping software.

SFTP, i.e., secure FTP, uses SSH-based security protocols that solve the security issues associated with plain FTP. NCO supports SFTP protocol access to files specified with a homebrew syntax of the form

```
sftp://machine.domain.tld:/path/to/filename
```

Note the second colon following the top-level-domain, tld. This syntax is a hybrid between an FTP URL and standard remote file syntax.

To access a file using rcp or scp, specify the Internet address of the remote file. Of course in this case you must have rcp or scp privileges which allow transparent (no password entry required) access to the remote machine. This means that “~/.rhosts” or “/ssh/authorized_keys” must be set accordingly on both local and remote machines.

To access a file on a High Performance Storage System (HPSS) (such as that at NCAR, ECMWF, LANL, DKRZ, LLNL) specify the full HPSS pathname of the remote file. NCO will attempt to detect whether the local machine has direct (synchronous) HPSS access. In this case, NCO attempts to use the Hierarchical Storage Interface (HSI) command hsi get.

The following examples show how one might analyze files stored on remote systems.

```
ncks -l . ftp://dust.ess.uci.edu/pub/zender/nco/in.nc
ncks -l . sftp://dust.ess.uci.edu:/home/ftp/pub/zender/nco/in.nc
ncks -l . dust.ess.uci.edu:/home/zender/nco/data/in.nc
ncks -l . /ZENDER/nco/in.nc
ncks -l . /home/zender/nco/in.nc
```

---

3 NCO does not implement command line options to specify FTP logins and passwords because copying those data into the history global attribute in the output file (done by default) poses an unacceptable security risk.

4 The hsi command must be in the user’s path in one of the following directories: /usr/local/bin, /opt/hpss/bin, or /ncar/opt/hpss/hsi. Tell us if the HPSS installation at your site places the hsi command in a different location, and we will add that location to the list of acceptable paths to search for hsi.
The first example works verbatim if your system is connected to the Internet and is not behind a firewall. The second example works if you have `sftp` access to the machine `dust.ess.uci.edu`. The third example works if you have `rcp` or `scp` access to the machine `dust.ess.uci.edu`. The fourth and fifth examples work on NCAR computers with local access to the HPSS `hsi get` command. The sixth command works if your local version of NCO is OPeNDAP-enabled (this is fully described in Section 3.7.1 [OPeNDAP], page 32), or if the remote file is accessible via `wget`. The above commands can be rewritten using the `'-p input-path'` option as follows:

```bash
ncks -p ftp://dust.ess.uci.edu/pub/zender/nco -l . in.nc
ncks -p sftp://dust.ess.uci.edu:/home/ftp/pub/zender/nco -l . in.nc
ncks -p dust.ess.uci.edu:/home/zender/nco -l . in.nc
ncks -p /ZENDER/nco -l . in.nc
ncks -p /home/zender/nco -l . in.nc # HPSS
ncks -p http://thredds-test.ucar.edu/thredds/dodsC/testdods \n -l . in.nc
```

Using `'-p'` is recommended because it clearly separates the `input-path` from the filename itself, sometimes called the `stub`. When `input-path` is not explicitly specified using `'-p'`, NCO internally generates an `input-path` from the first input filename. The automatically generated `input-path` is constructed by stripping the input filename of everything following the final `'/` character (i.e., removing the stub). The `'-l output-path'` option tells NCO where to store the remotely retrieved file. It has no effect on locally-retrieved files, or on the output file. Often the path to a remotely retrieved file is quite different than the path on the local machine where you would like to store the file. If `'-l'` is not specified then NCO internally generates an `output-path` by simply setting `output-path` equal to `input-path` stripped of any machine names. If `'-l'` is not specified and the remote file resides on the NCAR HPSS system, then the leading character of `input-path`, `'/`, is also stripped from `output-path`. Specifying `output-path` as `'-l ./'` tells NCO to store the remotely retrieved file and the output file in the current directory. Note that `'-l .'` is equivalent to `'-l ./'` though the latter is syntactically more clear.

### 3.7.1 OPeNDAP

The Distributed Oceanographic Data System (DODS) provides useful replacements for common data interface libraries like netCDF. The DODS versions of these libraries implement network transparent access to data via a client-server data access protocol that uses the HTTP protocol for communication. Although DODS-technology originated with oceanography data, it applies to virtually all scientific data. In recognition of this, the data access protocol underlying DODS (which is what NCO cares about) has been renamed the Open-source Project for a Network Data Access Protocol, OPeNDAP. We use the terms DODS
Chapter 3: Shared Features

and OPeNDAP interchangeably, and often write OPeNDAP/DODS for now. In the future we will deprecate DODS in favor of DAP or OPeNDAP, as appropriate.

NCO may be DAP-enabled by linking NCO to the OPeNDAP libraries. This is described in the OPeNDAP documentation and automatically implemented in NCO build mechanisms. The ./configure mechanism automatically enables NCO as OPeNDAP clients if it can find the required OPeNDAP libraries, in the usual locations. The $DODS_ROOT environment variable may be used to override the default OPeNDAP library location at NCO compile-time. Building NCO with bld/Makefile and the command make DODS=Y adds the (non-intuitive) commands to link to the OPeNDAP libraries installed in the $DODS_ROOT directory. The file doc/opendap.sh contains a generic script intended to help users install OPeNDAP before building NCO. The documentation at the OPeNDAP Homepage is voluminous. Check there and on the DODS mail lists to learn more about the extensive capabilities of OPeNDAP.

Once NCO is DAP-enabled the operators are OPeNDAP clients. All OPeNDAP clients have network transparent access to any files controlled by a OPeNDAP server. Simply specify the input file path(s) in URL notation and all NCO operations may be performed on remote files made accessible by a OPeNDAP server. This command tests the basic functionality of OPeNDAP-enabled NCO clients:

```
% ncks -O -o ~/foo.nc -C -H -v one -l /tmp \
   -p http://thredds-test.ucar.edu/thredds/dodsC/testdods in.nc
% ncks -H -v one ~/foo.nc
one = 1
```

The one = 1 outputs confirm (first) that ncks correctly retrieved data via the OPeNDAP protocol and (second) that ncks created a valid local copy of the subsetted remote file. With minor changes to the above command, netCDF4 can be used as both the input and output file format:

```
% ncks -4 -O -o ~/foo.nc -C -H -v one -l /tmp \
   -p http://thredds-test.ucar.edu/thredds/dodsC/testdods in_4.nc
% ncks -H -v one ~/foo.nc
one = 1
```

---

6 DODS is being deprecated because it is ambiguous, referring both to a protocol and to a collection of (oceanography) data. It is superceded by two terms. DAP is the discipline-neutral Data Access Protocol at the heart of DODS. The National Virtual Ocean Data System (NVODS) refers to the collection of oceanography data and oceanographic extensions to DAP. In other words, NVODS is implemented with OPeNDAP. OPeNDAP is also the open source project which maintains, develops, and promulgates the DAP standard. OPeNDAP and DAP really are interchangeable. Got it yet?

7 Automatic support for DODS version 3.2.x was deprecated in December, 2003 after NCO version 2.8.4. NCO support for OPeNDAP versions 3.4.x commenced in December, 2003, with NCO version 2.8.5. NCO support for OPeNDAP versions 3.5.x commenced in June, 2005, with NCO version 3.0.1. NCO support for OPeNDAP versions 3.6.x commenced in June, 2006, with NCO version 3.1.3. NCO support for OPeNDAP versions 3.7.x commenced in January, 2007, with NCO version 3.1.9.

8 The minimal set of libraries required to build NCO as OPeNDAP clients, where OPeNDAP is supplied as a separate library apart from libnetcdf.a, are, in link order, libnc-dap.a, libdap.a, and libxml2 and libcurl.a.

9 We are most familiar with the OPeNDAP ability to enable network-transparent data access. OPeNDAP has many other features, including sophisticated hyperslabbing and server-side processing via constraint expressions. If you know more about this, please consider writing a section on “OPeNDAP Capabilities of Interest to NCO Users” for incorporation in the NCO User Guide.
And, of course, OPeNDAP-enabled NCO clients continue to support orthogonal features such as UDUnits (see Section 3.22 [UDUnits Support], page 66):

\[
\% \text{ncks -u -C -H -v wvl -d wvl,'0.4 micron','0.7 micron'} \\
\quad -p http://thredds-test.ucar.edu/thredds/dodsC/testdods in_4.nc \\
\% \text{wvl[0]=5e-07 meter}
\]

The next command is a more advanced example which demonstrates the real power of OPeNDAP-enabled NCO clients. The ncwa client requests an equatorial hyperslab from remotely stored NCEP reanalyses data of the year 1969. The NOAA OPeNDAP server (hopefully!) serves these data. The local ncwa client then computes and stores (locally) the regional mean surface pressure (in Pa).

\[
\text{ncwa -C -a lat,lon,time -d lon,-10.,10. -d lat,-10.,10. -l /tmp -p} \\
\quad \text{http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets/ncep.reanalysis.dailyavgs/surface} \\
\quad \text{pres.sfc.1969.nc ~/foo.nc}
\]

All with one command! The data in this particular input file also happen to be packed (see Section 4.1.11 [Methods and functions], page 131), although this complication is transparent to the user since NCO automatically unpacks data before attempting arithmetic.

NCO obtains remote files from the OPeNDAP server (e.g., \text{www.cdc.noaa.gov}) rather than the local machine. Input files are first copied to the local machine, then processed. The OPeNDAP server performs data access, hyperslabbing, and transfer to the local machine. This allows the I/O to appear to NCO as if the input files were local. The local machine performs all arithmetic operations. Only the hyperslabbed output data are transferred over the network (to the local machine) for the number-crunching to begin. The advantages of this are obvious if you are examining small parts of large files stored at remote locations.

Naturally there are many versions of OPeNDAP servers supplying data and bugs in the server can appear to be bugs in NCO. However, with very few exceptions\(^{10}\) an NCO command that works on a local file must work across an OPeNDAP connection or else there is a bug in the server. This is because NCO does nothing special to handle files served by OPeNDAP, the whole process is (supposed to be) completely transparent to the client NCO software. Therefore it is often useful to try NCO commands on various OPeNDAP servers in order to isolate whether a problem may be due to a bug in the OPeNDAP server on a particular machine. For this purpose, one might try variations of the following commands that access files on public OPeNDAP servers:

\[
\text{# Strided access to HDF5 file} \\
\text{ncks -v Time -d Time,0,10,2 http://eosdap.hdfgroup.uiuc.edu:8080/opendap/data/NASAFILES/hdf5/BUV-Nimbus04_L3zm_v01-00-2012m0203t144121.h5}
\]

\[
\text{# Strided access to netCDF3 file} \\
\text{ncks -O -D 1 -d time,1 -d lev,0 -d lat,0,100,10 -d lon,0,100,10 -v u_velocity http://nomads.ncep.noaa.gov:9090/dods/rtofs/rtofs_global20140303/rtofs_glo_2ds_forecast_daily_prog ~/foo.nc}
\]

These servers were operational at the time of writing, March 2014. Unfortunately, administrators often move or rename path directories. Recommendations for additional public OPeNDAP servers on which to test NCO are welcome.

\(^{10}\) For example, DAP servers do not like variables with periods (".") in their names even though this is perfectly legal with netCDF. Such names may cause the DAP service to fail because DAP interprets the period as structure delimiter in an HTTP query string.
3.8 Retaining Retrieved Files

<table>
<thead>
<tr>
<th>Availability: All operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: ‘-R’</td>
</tr>
<tr>
<td>Long options: ‘--rtn’, ‘--retain’</td>
</tr>
</tbody>
</table>

In order to conserve local file system space, files retrieved from remote locations are automatically deleted from the local file system once they have been processed. Many NCO operators were constructed to work with numerous large (e.g., 200 MB) files. Retrieval of multiple files from remote locations is done serially. Each file is retrieved, processed, then deleted before the cycle repeats. In cases where it is useful to keep the remotely-retrieved files on the local file system after processing, the automatic removal feature may be disabled by specifying ‘-R’ on the command line.

Invoking -R disables the default printing behavior of ncks. This allows ncks to retrieve remote files without automatically trying to print them. See Section 4.7 [ncks netCDF Kitchen Sink], page 188, for more details.

Note that the remote retrieval features of NCO can always be used to retrieve any file, including non-netCDF files, via SSH, anonymous FTP, or msr cp. Often this method is quicker than using a browser, or running an FTP session from a shell window yourself. For example, say you want to obtain a JPEG file from a weather server.

```bash
ncks -R -p ftp://weather.edu/pub/pix/jpeg -l . storm.jpg
```

In this example, ncks automatically performs an anonymous FTP login to the remote machine and retrieves the specified file. When ncks attempts to read the local copy of storm.jpg as a netCDF file, it fails and exits, leaving storm.jpg in the current directory.

If your NCO is DAP-enabled (see Section 3.7.1 [OPeNDAP], page 32), then you may use NCO to retrieve any files (including netCDF, HDF, etc.) served by an OPeNDAP server to your local machine. For example,

```bash
```

It may occasionally be useful to use NCO to transfer files when your other preferred methods are not available locally.

3.9 File Formats and Conversion

<table>
<thead>
<tr>
<th>Availability: ncap2, nces, ncecat, ncflint, ncks, ncpdq, ncra, ncrcat, ncwa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: ‘-3’, ‘-4’, ‘-6’, ‘-7’</td>
</tr>
</tbody>
</table>
All NCO operators support (read and write) all three (or four, depending on how one counts) file formats supported by netCDF4. The default output file format for all operators is the input file format. The operators listed under “Availability” above allow the user to specify the output file format independent of the input file format. These operators allow the user to convert between the various file formats. (The operators ncatted and ncrename do not support these switches so they always write the output netCDF file in the same format as the input netCDF file.)

3.9.1 File Formats
netCDF supports four types of files: CLASSIC, 64BIT, NETCDF4, and NETCDF4_CLASSIC. The CLASSIC format is the traditional 32-bit offset written by netCDF2 and netCDF3. As of 2005, nearly all netCDF datasets were in CLASSIC format. The 64BIT format was added in Fall, 2004. As of 2010, many netCDF datasets were in 64BIT format. As of 2013, an increasing number of netCDF datasets were in NETCDF4_CLASSIC format.

The NETCDF4 format uses HDF5 as the file storage layer. The files are (usually) created, accessed, and manipulated using the traditional netCDF3 API (with numerous extensions). The NETCDF4_CLASSIC format refers to netCDF4 files created with the NC_CLASSIC_MODEL mask. Such files use HDF5 as the back-end storage format (unlike netCDF3), though they incorporate only netCDF3 features. Hence NETCDF4_CLASSIC files are entirely readable by applications that use only the netCDF3 API (though the applications must be linked with the netCDF4 library). NCO must be built with netCDF4 to write files in the new NETCDF4 and NETCDF4_CLASSIC formats, and to read files in these formats. Datasets in the default CLASSIC or the newer 64BIT formats have maximum backwards- compatibility with older applications. NCO has deep support for NETCDF4 formats. If performance or disk-space as important as backwards compatibility, then use NETCDF4_CLASSIC instead of CLASSIC format files. As of 2014, NCO support for the NETCDF4 format is nearly complete and the most powerful and disk/RAM efficient workflows will utilize this format.

As mentioned above, all operators write use the input file format for output files unless told otherwise. Toggling the short option ‘-6’ or the long option ‘--6’ or ‘--64bit’ (or their key-value equivalent ‘--fl_fmt=64bit’) produces the netCDF3 64-bit offset format named 64BIT. NCO must be built with netCDF 3.6 or higher to produce a 64BIT file. Using the ‘-4’ switch (or its long option equivalents ‘--4’ or ‘--netcdf4’), or setting its key-value equivalent ‘--fl_fmt=netcdf4’ produces a NETCDF4 file (i.e., with all supported HDF5 features). Using the ‘-7’ switch (or its long option equivalent ‘--7’11, or setting its key-value equivalent ‘--fl_fmt=netcdf4_classic’ produces a NETCDF4_CLASSIC file (i.e., with all supported HDF5 features like compression and chunking but without groups or new atomic types). Operators given the ‘-3’ (or ‘--3’) switch without arguments will (attempt to) produce netCDF3 CLASSIC output, even from netCDF4 input files.

Note that NETCDF4 and NETCDF4_CLASSIC are the same binary format. The latter simply causes a writing application to fail if it attempts to write a NETCDF4 file that cannot be completely read by the netCDF3 library. Conversely, NETCDF4_CLASSIC indicates to a reading application that all of the file contents are readable with the netCDF3 library. NCO

11 The reason (and mnemonic) for ‘-7’ is that NETCDF4_CLASSIC files include great features of both netCDF3 (compatibility) and netCDF4 (compression, chunking) and, well, 3 + 4 = 7.
has supported reading/writing basic NETCDF4 and NETCDF4_CLASSIC files since October, 2005.

3.9.2 Determining File Format

Input files often end with the generic .nc suffix that leaves (perhaps by intention) the internal file format ambiguous. There are at least three ways to discover the internal format of a netCDF-supported file. These methods determine whether it is a classic (32-bit offset) or newer 64-bit offset netCDF3 format, or is a netCDF4 format. Each method returns the information using slightly different terminology that becomes easier to understand with practice.

First, examine the first line of global metadata output by `ncks -M`:

```
% ncks -M foo_3c.nc
Summary of foo_3c.nc: filetype = NC_FORMAT_CLASSIC, 0 groups ...
% ncks -M foo_364.nc
Summary of foo_364.nc: filetype = NC_FORMAT_64BIT, 0 groups ...
% ncks -M foo_4c.nc
Summary of foo_4c.nc: filetype = NC_FORMAT_NETCDF4_CLASSIC, 0 groups ...
% ncks -M foo_4.nc
Summary of foo_4.nc: filetype = NC_FORMAT_NETCDF4, 0 groups ...
```

This method requires a netCDF4-enabled NCO version 3.9.0+ (i.e., from 2007 or later). As of NCO version 4.4.0 (January, 2014), ncks will also print the extended or underlying format of the input file. The extended filetype will be one of the six underlying formats that are accessible through the netCDF API. These formats are NC_FORMAT_NC3 (classic and 64-bit versions of netCDF3 formats), NC_FORMAT_NC_HDF5 (classic and extended versions of netCDF4, and “pure” HDF5 format), NC_FORMAT_NC_HDF4 (HDF4 format), NC_FORMAT_PNETCDF (PnetCDF format), NC_FORMAT_DAP2 (accessed via DAP2 protocol), and NC_FORMAT_DAP4 (accessed via DAP2 protocol). For example,

```
% ncks -D 2 -M hdf.hdf
Summary of hdf.hdf: filetype = NC_FORMAT_NETCDF4 (representation of 
extended/underlying filetype NC_FORMAT_HDF4), 0 groups ...
% ncks -D 2 -M http://thredds-test.uncar.edu/thredds/dodsC/testdods/in.nc
Summary of http://thredds-test.uncar.edu/thredds/dodsC/testdods/in.nc: 
  filetype = NC_FORMAT_CLASSIC (representation of extended/underlying 
  filetype NC_FORMAT_DAP2), 0 groups
% ncks -D 2 -M foo_4.nc
Summary of foo_4.nc: filetype = NC_FORMAT_NETCDF4 (representation of 
extended/underlying filetype NC_FORMAT_HDF5), 0 groups
```

The extended filetype determines some of the capabilities that netCDF has to alter the file.

Second, query the file with `ncdump -k`:

```
% ncdump -k foo_3c.nc
classic
% ncdump -k foo_364.nc
```
64-bit-offset
% ncdump -k foo_4c.nc
netCDF-4 classic model
% ncdump -k foo_4.nc
netCDF-4

This method requires a netCDF4-enabled netCDF 3.6.2+ (i.e., from 2007 or later).

The third option uses the POSIX-standard od (octal dump) command:

% od -An -c -N4 foo_3c.nc
  C  D  F 001
% od -An -c -N4 foo_364.nc
  C  D  F 002
% od -An -c -N4 foo_4c.nc
  211  H  D  F
% od -An -c -N4 foo_4.nc
  211  H  D  F

This option works without NCO and ncdump. Values of ‘C D F 001’ and ‘C D F 002’ indicate 32-bit (classic) and 64-bit netCDF3 formats, respectively, while values of ‘211 H D F’ indicate either of the newer netCDF4 file formats.

3.9.3 File Conversion

Let us demonstrate converting a file from any netCDF-supported input format into any netCDF output format (subject to limits of the output format). Here the input file in.nc may be in any of these formats: netCDF3 (classic and 64bit), netCDF4 (classic and extended), HDF4, HDF5, HDF-EOS (version 2 or 5), and DAP. The switch determines the output format written in the comment:12

ncks --fl_fmt=classic in.nc foo_3c.nc # netCDF3 classic
ncks --fl_fmt=64bit in.nc foo_364.nc # netCDF3 64bit
ncks --fl_fmt=netcdf4_classic in.nc foo_4c.nc # netCDF4 classic
ncks --fl_fmt=netcdf4 in.nc foo_4c.nc # netCDF4
ncks -3 in.nc foo_3c.nc # netCDF3 classic
ncks --3 in.nc foo_3c.nc # netCDF3 classic
ncks -6 in.nc foo_364.nc # netCDF3 64bit
ncks --64 in.nc foo_364.nc # netCDF3 64bit
ncks -4 in.nc foo_4.nc # netCDF4
ncks --4 in.nc foo_4.nc # netCDF4
ncks -7 in.nc foo_4c.nc # netCDF4 classic
ncks --7 in.nc foo_4c.nc # netCDF4 classic

Of course since most operators support these switches, the “conversions” can be done at the output stage of arithmetic or metadata processing rather than requiring a separate step. Producing (netCDF3) CLASSIC or 64BIT files from NETCDF4_CLASSIC files will always work.

12 The switches ‘-5’, ‘--5’, and ‘pnetcdf’ are reserved for PnetCDF files, i.e., NC_FORMAT_PNETCDF. Such files are similar to netCDF3 classic files, yet also support 64-bit offsets and the additional netCDF4 atomic types.
3.9.4 Autoconversion

Because of the dearth of support for netCDF4 amongst tools and user communities (including the CF conventions), it is often useful to convert netCDF4 to netCDF3 for certain applications. Until NCO version 4.4.0 (January, 2014), producing netCDF3 files from netCDF4 files only worked if the input files contained no netCDF4-specific features (e.g., atomic types, multiple record dimensions, or groups). As of NCO version 4.4.0, ncks supports autoconversion of many netCDF4 features to their closest netCDF3-compatible representations. Since converting netCDF4 to netCDF3 results in loss of features, “automatic down-conversion” may be a more precise description of what we term autoconversion.

NCO employs three algorithms to downconvert netCDF4 to netCDF3:

1. Autoconversion of atomic types: Autoconversion automatically promotes NC_UBYTE to NC_SHORT, and NC_USHORT to NC_INT. It automatically demotes the three types NC_UINT, NC_UINT64, and NC_INT64 to NC_INT. And it converts NC_STRING to NC_CHAR. All numeric conversions work for attributes and variables of any rank. Two numeric types (NC_UBYTE and NC_USHORT) are promoted to types with greater range (and greater storage). This extra range is often not used so promotion perhaps conveys the wrong impression. However, promotion never truncates values or loses data (this perhaps justifies the extra storage). Three numeric types (NC_UINT, NC_UINT64 and NC_INT64) are demoted. Since the input range is larger than the output range, demotion can result in numeric truncation and thus loss of data. In such cases, it would possible to convert the data to floating point values instead. If this feature interests you, please be the squeaky wheel and let us know.

String conversions (to NC_CHAR) work for all attributes, but not for variables. This is because attributes are at most one-dimensional and may be of any size whereas variables require gridded dimensions that usually do not fit the ragged sizes of text strings. Hence scalar NC_STRING attributes are correctly converted to and stored as NC_CHAR attributes in the netCDF3 output file, but NC_STRING variables are not correctly converted. If this limitation annoys or enrages you, please let us know by being the squeaky wheel.

2. Convert multiple record dimensions to fixed-size dimensions. Many netCDF4 and HDF5 datasets have multiple unlimited dimensions. Since a netCDF3 file may have at most one unlimited dimension, all but possibly one unlimited dimension from the input file must be converted to fixed-length dimensions prior to storing netCDF4 input as netCDF3 output. By invoking --fix_rec_dmn all the user ensures the output file will adhere to netCDF3 conventions and the user need not know the names of the specific record dimensions to fix. See Section 4.7 [ncks netCDF Kitchen Sink], page 188 for a description of the ‘--fix_rec_dmn’ option.

3. Flattening (removal) of groups. Many netCDF4 and HDF5 datasets have group hierarchies. Since a netCDF3 file may not have any groups, groups in the input file must be removed. This is also called “flattening” the hierarchical file. See Section 3.13 [Group Path Editing], page 46 for a description of the GPE option ‘--G :’ to flatten files.

Putting the three algorithms together, one sees that the recipe to convert netCDF4 to netCDF4 becomes increasingly complex as the netCDF4 features in the input file become more elaborate:

```bash
# Convert file with netCDF4 atomic types
```
Future versions of NCO may automatically invoke the record dimension fixation and group flattening when converting to netCDF3 (rather than requiring it be specified manually). If this feature would interest you, please let us know.

### 3.10 Large File Support

<table>
<thead>
<tr>
<th>Availability: All operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: none</td>
</tr>
<tr>
<td>Long options: none</td>
</tr>
</tbody>
</table>

NCO has Large File Support (LFS), meaning that NCO can write files larger than 2 GB on some 32-bit operating systems with netCDF libraries earlier than version 3.6. If desired, LFS support must be configured when both netCDF and NCO are installed. netCDF versions 3.6 and higher support 64-bit file addresses as part of the netCDF standard. We recommend that users ignore LFS support which is difficult to configure and is implemented in NCO only to support netCDF versions prior to 3.6. This obviates the need for configuring explicit LFS support in applications (such as NCO) that now support 64-bit files directly through the netCDF interface. See Section 3.9 [File Formats and Conversion], page 35 for instructions on accessing the different file formats, including 64-bit files, supported by the modern netCDF interface.

If you are still interested in explicit LFS support for netCDF versions prior to 3.6, know that LFS support depends on a complex, interlocking set of operating system and netCDF support issues. The netCDF LFS FAQ describes the various file size limitations imposed by different versions of the netCDF standard. NCO and netCDF automatically attempt to configure LFS at build time.

### 3.11 Subsetting Files

---

13 Linux and AIX do support LFS.
Subsetting variables refers to explicitly specifying variables and groups to be included or excluded from operator actions. Subsetting is controlled by the ‘-v var[, ...]’ and ‘-x’ options for directly specifying variables. Specifying groups, whether in addition to or instead of variables, is quite similar and is controlled by the ‘-g grp[, ...]’ and ‘-x’ options. A list of variables or groups to extract is specified following the ‘-v’ and ‘-g’ options, e.g., ‘-v time, lat, lon’ or ‘-g grp1, grp2’. Both options may be specified simultaneously and NCO will extract the intersection of the lists, i.e., only variables of the specified names found in groups of the specified names. The ‘--unn’ option causes NCO to extract the union, rather than the intersection, of the specified groups and variables. Not using the ‘-v’ or ‘-g’ option is equivalent to specifying all variables or groups, respectively. The ‘-x’ option causes the list of variables specified with ‘-v’ to be excluded rather than extracted. Thus ‘-x’ saves typing when you only want to extract fewer than half of the variables in a file.

Variables or groups explicitly specified for extraction with ‘-v var[, ...]’ or ‘-g grp[, ...]’ must be present in the input file or an error will result. Variables explicitly specified for exclusion with ‘-x -v var[, ...]’ need not be present in the input file. To accord with the sophistication of the underlying hierarchy, group subsetting is controlled by a few powerful yet subtle syntactical distinctions. When learning this syntax it is helpful to keep in mind the similarity between group hierarchies and directory structures.

As of NCO 4.4.4 (June, 2014), ncks (alone) supports an option to include specified groups yet exclude specified variables. The ‘--grp_xtr_var_xcl’ switch (with long option equivalent ‘--gxvx’) extracts all contents of groups given as arguments to ‘-g grp[, ...]’, except for variables given as arguments to ‘-v var[, ...]’. Use this when one or a few variables in hierarchical files are not to be extracted, and all other variables are. This is useful when coercing netCDF4 files into netCDF3 files such as with converting, flattening, or dismembering files (see Section 3.13.1 [Flattening Groups], page 46).

ncks --grp_xtr_var_xcl -g g1 -v v1 # Extract all of group g1 except v1
Two properties of subsetting, recursion and anchoring, are best illustrated by reminding
the user of their UNIX equivalents. The UNIX command \texttt{mv src dst} moves \texttt{src} and all its
subdirectories (and all their subdirectories etc.) to \texttt{dst}. In other words \texttt{mv} is, by default,
recursive. In contrast, the UNIX command \texttt{cp src dst} moves \texttt{src}, and only \texttt{src}, to \texttt{dst}. If
\texttt{src} is a directory, not a file, then that command fails. One must explicitly request to copy
directories recursively, i.e., with \texttt{cp -r src dst}. In NCO recursive extraction (and copying)
of groups is the default (like with \texttt{mv}, not with \texttt{cp}). Recursion is turned off by appending a
trailing slash to the path.

These UNIX commands also illustrate a property we call anchoring. The command \texttt{mv src dst}
moves (recursively) the source directory \texttt{src} to the destination directory \texttt{dst}. If
\texttt{src} begins with the slash character then the specified path is relative to the root directory,
otherwise the path is relative to the current working directory. In other words, an initial
slash character anchors the subsequent path to the root directory. In NCO an initial slash
anchors the path at the root group. Paths that begin and end with slash characters (e.g.,
\texttt{//}, \texttt{/g1/}, and \texttt{/g1/g2/}) are both anchored and non-recursive.

Consider the following commands, all of which may be assumed to end with \texttt{'in.nc out.nc'}:

\begin{verbatim}
  ncks -g g1   # Extract, recursively, all groups with a g1 component
  ncks -g g1/  # Extract, non-recursively, all groups terminating in g1
  ncks -g /g1  # Extract, recursively, root group g1
  ncks -g /g1/ # Extract, non-recursively root group g1
  ncks -g //   # Extract, non-recursively the root group
\end{verbatim}

The first command is probably the most useful and common. It would extract these
groups, if present, and all their direct ancestors and children: \texttt{/g1}, \texttt{/g2/g1}, and \texttt{/g3/g1/g2}. In
other words, the simplest form of \texttt{'-g grp'} grabs all groups that (and their direct ancestors
and children, recursively) that have \texttt{grp} as a complete component of their path. A simple
string match is insufficient, \texttt{grp} must be a complete component (i.e., group name) in the
path. The option \texttt{'-g g1'} would not extract these groups because \texttt{g1} is not a complete
component of the path: \texttt{/g12}, \texttt{/fg1}, and \texttt{/g1g1}. The second command above shows how
a terminating slash character / cancels the recursive copying of groups. An argument to
\texttt{'-g'} which terminates with a slash character extracts the group and its direct ancestors,
but none of its children. The third command above shows how an initial slash character /
anchors the argument to the root group. The third command would not extract the group
\texttt{/g2/g1} because the \texttt{g1} group is not at the root level, but it would extract, any group \texttt{/g1}
at the root level and all its children, recursively. The fourth command is the non-recursive
version of the third command. The fifth command is a special case of the fourth command.

As mentioned above, both \texttt{','-v'} and \texttt{','-g'} options may be specified simultaneously and
NCO will, by default, extract the intersection of the lists, i.e., the specified variables found
in the specified groups\footnote{Intersection-mode can also be explicitly invoked with the \texttt{'--nx'} or \texttt{('--intersection')} switches. These
switches are supplied for clarity and consistency and do absolutely nothing since intersection-mode is
the default.}. The \texttt{'--unn'} option causes NCO to extract the union, rather than
the intersection, of the specified groups and variables. Consider the following commands
(which may be assumed to end with \texttt{'in.nc out.nc'}):

\begin{verbatim}
  ncks -v   # Extract, recursively the intersection of the groups
  ncks -v -g g1   # Extract, recursively the intersection of the groups
                # and their g1 component
  ncks -v -g g1/  # Extract, recursively the intersection of the groups
                # and their g1 component
  ncks -v -g /g1  # Extract, recursively the intersection of the groups
                 # and their root group g1
  ncks -v -g /g1/ # Extract, recursively the intersection of the groups
                 # and their root group g1
  ncks -v -g //   # Extract, recursively the intersection of the groups
                 # and the root group
\end{verbatim}
Chapter 3: Shared Features

# Intersection-mode subsetting (default)

ncks -g g1 -v v1 # Yes: /g1/v1, /g2/g1/v1. No: /v1, /g2/v1
ncks -g /g1 -v v1 # Yes: /g1/v1, /g2/g1/v1. No: /v1, /g2/v1, /g2/g1/v1
ncks -g g1/ -v v1 # Yes: /g1/v1, /g2/g1/v1. No: /v1, /g2/v1, /g1/g2/v1
ncks -v g1/v1 # Yes: /g1/v1, /g2/g1/v1. No: /v1, /g2/v1, /g1/g2/v1
ncks -g /g1/ -v v1 # Yes: /g1/v1. No: /g2/g1/v1, /v1, /g2/v1 ...

ncks -v /g1/v1 # Yes: /g1/v1. No: /g2/g1/v1, /v1, /g2/v1 ...

# Union-mode subsetting (invoke with --unn or --union)

ncks -g g1 -v v1 --unn # All variables in g1 or progeny, or named v1
ncks -g /g1 -v v1 --unn # All variables in /g1 or progeny, or named v1
ncks -g g1/ -v v1 --unn # All variables in g1 or named v1
ncks -g /g1/ -v v1 --unn # All variables in /g1 or named v1

The first command (‘-g g1 -v v1’) extracts the variable v1 from any group named g1 or descendent g1. The second command extracts v1 from any root group named g1 and any descendent groups as well. The third and fourth commands are equivalent ways of extracting v1 only from the root group named g1 (not its descendents). The fifth and sixth commands are equivalent ways of extracting the variable v1 only from the root group named g1. Subsetting in union-mode (with ‘--unn’) causes all variables to be extracted which meet either one or both of the specifications of the variable and group specifications. Union-mode subsetting is simply the logical “OR” of intersection-mode subsetting. As discussed below, the group and variable specifications may be comma separated lists of regular expressions for added control over subsetting.

Remember, if averaging or concatenating large files stresses your systems memory or disk resources, then the easiest solution is often to subset (with ‘-g’ and/or ‘-v’) to retain only the most important variables (see Section 2.9 [Memory Requirements], page 22).

ncks in.nc out.nc # Extract all groups and variables
ncks -v scl # Extract variable scl from all groups
ncks -g g1 # Extract group g1 and descendents
ncks -x -g g1 # Extract all groups except g1 and descendents
ncks -g g2,g3 -v scl # Extract scl from groups g2 and g3

Overwriting and appending work as expected:

# Replace scl in group g2 in out.nc with scl from group g2 from in.nc
ncks -A -g g2 -v scl in.nc out.nc

Due to its special capabilities, ncap2 interprets the ‘-v’ switch differently (see Section 4.1 [ncap2 netCDF Arithmetic Processor], page 116). For ncap2, the ‘-v’ switch takes no arguments and indicates that only user-defined variables should be output. ncap2 neither accepts nor understands the -x and -g switches.

Regular expressions the syntax that NCO use pattern-match object names in netCDF file against user requests. The user can select all variables beginning with the string ‘DST’ from an input file by supplying the regular expression ‘^DST’ to the ‘-v’ switch, i.e., ‘-v ‘^DST’’. The meta-characters used to express pattern matching operations are ‘[^$+.*[]{}|]’. If the regular expression pattern matches any part of a variable name then that variable is
selected. This capability is also called wildcarding, and is very useful for sub-setting large data files.

Extended regular expressions are defined by the POSIX `grep -E` (aka `egrep`) command. As of NCO 2.8.1 (August, 2003), variable name arguments to the ‘-v’ switch may contain extended regular expressions. As of NCO 3.9.6 (January, 2009), variable names arguments to `ncatted` may contain extended regular expressions. As of NCO 4.2.4 (November, 2012), group name arguments to the ‘-g’ switch may contain extended regular expressions.

Because of its wide availability, NCO uses the POSIX regular expression library `regex`. Regular expressions of arbitrary complexity may be used. Since netCDF variable names are relatively simple constructs, only a few varieties of variable wildcards are likely to be useful. For convenience, we define the most useful pattern matching operators here:

- `^` Matches the beginning of a string
- `$` Matches the end of a string
- `.` Matches any single character

The most useful repetition and combination operators are

- `?` The preceding regular expression is optional and matched at most once
- `*` The preceding regular expression will be matched zero or more times
- `+` The preceding regular expression will be matched one or more times
- `|` The preceding regular expression will be joined to the following regular expression. The resulting regular expression matches any string matching either subexpression.

To illustrate the use of these operators in extracting variables and groups, consider file `in_grp.nc` with groups `g0–g9`, and subgroups `s0–s9`, in each of those groups, and file `in.nc` with variables `Q`, `Q01–Q99`, `Q100`, `QAA–QZZ`, `Q_H2O`, `X_H2O`, `Q_CO2`, `X_CO2`.

```bash
ncks -v '.+' in.nc  # All variables (default)
ncks -v 'Q.?' in.nc  # Variables that contain Q
ncks -v '^Q.?' in.nc  # Variables that start with Q
ncks -v '^Q+.?.' in.nc  # Q, Q0--Q9, Q01--Q99, QAA--QZZ, etc.
ncks -v '^Q...' in.nc  # Q01--Q99, QAA--QZZ, etc.
ncks -v '^Q[0-9][0-9]' in.nc  # Q01--Q99
ncks -v 'H2O$' in.nc  # Q_H2O, X_H2O
ncks -v '^Q[0-9][0-9]$' in.nc  # Q01--Q99
ncks -v '^[a-z]_[a-z]{3}$' in.nc  # Q_H2O, X_H2O, Q_CO2, X_CO2
ncks -g 'g.' in_grp.nc  # 10 Groups g0–g9
ncks -g 's.' in_grp.nc  # 100 sub-groups g0/s0, g0/s1, ..., g9/s9
ncks -g 'g.' -v 'v.' in_grp.nc  # All variables 'v.' in groups 'g.'
```
Beware—two of the most frequently used repetition pattern matching operators, ‘*’ and ‘?’, are also valid pattern matching operators for filename expansion (globbing) at the shell-level. Confusingly, their meanings in extended regular expressions and in shell-level filename expansion are significantly different. In an extended regular expression, ‘*’ matches zero or more occurrences of the preceding regular expression. Thus ‘Q*’ selects all variables, and ‘Q+.∗’ selects all variables containing ‘Q’ (the ‘+’ ensures the preceding item matches at least once). To match zero or one occurrence of the preceding regular expression, use ‘?’.

Documentation for the UNIX `egrep` command details the extended regular expressions which NCO supports.

One must be careful to protect any special characters in the regular expression specification from being interpreted (globbed) by the shell. This is accomplish by enclosing special characters within single or double quotes

```
ncra -v Q?? in.nc out.nc  # Error: Shell attempts to glob wildcards
ncra -v `Q+..' in.nc out.nc # Correct: NCO interprets wildcards
ncra -v `Q+..' in*.nc out.nc # Correct: NCO interprets, Shell globs
```

The final example shows that commands may use a combination of variable wildcarding and shell filename expansion (globbing). For globbing, ‘*’ and ‘?’ have nothing to do with the preceding regular expression! In shell-level filename expansion, ‘*’ matches any string, including the null string and ‘?’ matches any single character. Documentation for `bash` and `csh` describe the rules of filename expansion (globbing).

### 3.12 Subsetting Coordinate Variables

By default, coordinates variables associated with any variable appearing in the `input-file` will be placed in the `output-file`, even if they are not explicitly specified, e.g., with the ‘-v’ switch. Thus variables with a latitude coordinate `lat` always carry the values of `lat` with them into the `output-file`. This feature can be disabled with ‘-C’, which causes NCO to not automatically add coordinates to the variables appearing in the `output-file`. However, using ‘-C’ does not preclude the user from including some coordinates in the output files simply by explicitly selecting the coordinates with the `-v` option. The ‘-c’ option, on the other hand, is a shorthand way of automatically specifying that all coordinate variables in the `input-files` should appear in the `output-file`. Thus ‘-c’ allows the user to select all the coordinate variables without having to know their names. As of NCO version 4.4.5 (July, 2014) both ‘-c’ and ‘-C’ honor the CF `ancillary_variables` convention described in Section 3.38 [CF Conventions], page 109. As of NCO version 4.0.8 (April, 2011) both ‘-c’ and ‘-C’ honor the CF `bounds` convention described in Section 3.38 [CF Conventions], page 109. As of NCO version 3.9.6 (January, 2009) both ‘-c’ and ‘-C’ honor the CF `coordinates` convention described in Section 3.38 [CF Conventions], page 109.
3.13 Group Path Editing

**Options** `-G gpe_dsc`

Availability: ncbo, ncecat, nces, ncflint, ncks, ncpdq, ncra, ncrcat, ncwa

Short options: ‘-G’
Long options: ‘---gpe’

**Group Path Editing**, or GPE, allows the user to restructure (i.e., add, remove, and rename groups) in the output file relative to the input file based on the instructions they provide. As of NCO 4.2.3 (November, 2012), all operators that accept netCDF4 files with groups accept the ‘-G’ switch, or its long-option equivalent ‘--gpe’. To master GPE one must understand the meaning of the required `gpe_dsc` structure/argument that specifies the transformation of input-to-output group paths.

Each `gpe_dsc` contains up to three elements (two are optional) in the following order:

\[
gpe_dsc = grp_pth lvl_nbr \text{ or } grp_pth@lvl_nbr
\]

- **grp_pth**: Group Path. This (optional) component specifies the output group path that should be appended after any editing (i.e., deletion or truncation) of the input path is performed.
- **lvl_nbr**: The number of levels to delete (from the head) or truncate (from the tail) of the input path.

If both components of the argument are present, then a single character, either the colon or at-sign (: or @), must separate them. If only `grp_pth` is specified, the separator character may be omitted, e.g., ‘-G g1’. If only `lvl_nbr` is specified, the separator character is still required to indicate it is a `lvl_nbr` argument and not a `grp_pth`, e.g., ‘-G :-1’ or ‘-G @1’.

If the at-sign separator character @ is used instead of the colon separator character :, then the following `lvl_nbr` argument must be positive and it will be assumed to refer to Truncation-Mode. Hence, ‘-G :-1’ is the same as ‘-G @1’. This is simply a way of making the `lvl_nbr` argument positive-definite.

### 3.13.1 Deletion, Truncation, and Flattening of Groups

GPE has three editing modes: Delete, Truncate, and Flatten. Select one of GPE’s three editing modes by supplying a `lvl_nbr` that is positive, negative, or zero for Delete-, Truncate- and Flatten-mode, respectively.

In Delete-mode, `lvl_nbr` is a positive integer which specifies the maximum number of group path components (i.e., groups) that GPE will try to delete from the head of `grp_pth`. For example `lvl_nbr = 3` changes the input path `/g1/g2/g3/g4/g5` to the output path `/g4/g5`. Input paths with `lvl_nbr` or fewer components (groups) are completely erased and the output path commences from the root level.

In other words, GPE is tolerant of specifying too many group components to delete. It deletes as many as possible, without complaint, and then begins to flatten the file (which will fail if namespace conflicts arise).
In Truncate-mode, \texttt{lvl\_nbr} is a negative integer which specifies the maximum number of group path components (i.e., groups) that GPE will try to truncate from the tail of \texttt{grp\_pth}. For example \texttt{lvl\_nbr = -3} changes the input path /g1/g2/g3/g4/g5 to the output path /g1/g2. Input paths with \texttt{lvl\_nbr} or fewer components (groups) are completely erased and the output path commences from the root level.

In Flatten-mode, indicated by the separator character alone or with \texttt{lvl\_nbr = 0}, GPE removes the entire group path from the input file and constructs the output path beginning at the root level. For example \texttt{-G :0} and \texttt{-G :} are identical and change the input path /g1/g2/g3/g4/g5 to the output path / whereas \texttt{-G g1:0} and \texttt{-G g1:} are identical and result in the output path /g1 for all variables.

Subsequent to the alteration of the input path by the specified editing mode, if any, GPE prepends (in Delete Mode) or Appends (in Truncate-mode) any specified \texttt{grp\_pth} to the output path. For example \texttt{-G g2} changes the input paths / and /g1 to /g2 and /g1, respectively. Likewise, \texttt{-G g2/g3} changes the input paths / and /g1 to /g2/g3 and /g1, respectively. When \texttt{grp\_pth} and \texttt{lvl\_nbr} are both specified, the editing actions are taken in sequence so that, e.g., \texttt{-G g1/g2:2} changes the input paths / and /h1/h2/h3/h4 to /g1/g2 and /g1/g2/h3/h4, respectively. Likewise, \texttt{-G g1/g2:-2} changes the input paths / and /h1/h2/h3/h4 to /g1/g2 and /h1/h2/g1/g2, respectively.

Combining GPE with subsetting (see Section 3.11 [Subsetting Files], page 40) yields powerful control over the extracted (or excluded) variables and groups and their placement in the output file as shown by the following commands. All commands below may be assumed to end with \texttt{.in.nc out.nc}.

```plaintext
# Prepending paths without editing:
ncks # /g?/v?  -> /g?/v?
ncks -v v1 # /g?v1  -> /g?v1
ncks -G o1 # /g?v?  -> /o1/g?v?
ncks -G o1 -g g1 # /g?v?  -> /o1/g1/v?

# Delete-mode: Delete from and Prepend to path head
ncks -G :1 -g g1 -v v1 # /g1/v1  -> /g1/v1
ncks -G :1 -g g1/g1 -v v1 # /g1/g1/v1  -> /g1/v1

# Flatten-mode: Remove all input path components
ncks -G g2:1 -g g1 -v v1 # /g1/v1  -> /g2/v1
ncks -G g2:1 -g g1/v1 -v v1 # /g1/g1/v1  -> /g2/v1
```

Combining GPE with subsetting (see Section 3.11 [Subsetting Files], page 40) yields powerful control over the extracted (or excluded) variables and groups and their placement in the output file as shown by the following commands. All commands below may be assumed to end with `in.nc out.nc`.

```plaintext
# Prepending paths without editing:
ncks # /g?/v?  -> /g?/v?
ncks -v v1 # /g?v1  -> /g?v1
ncks -G o1 # /g?v?  -> /o1/g?v?
ncks -G o1 -g g1 # /g?v?  -> /o1/g1/v?

# Delete-mode: Delete from and Prepend to path head
ncks -G :1 -g g1 -v v1 # /g1/v1  -> /g1/v1
ncks -G :1 -g g1/g1 -v v1 # /g1/g1/v1  -> /g1/v1
ncks -G :2 -g g1/g1 -v v1 # /g1/g1/v1  -> /v1
ncks -G :2 -g g1 -v v1 # /g1/v1  -> /v1
ncks -G g2:1 -g g1 -v v1 # /g1/v1  -> /g2/v1
ncks -G g2:1 -g g1/v1 -v v1 # /g1/g1/v1  -> /g2/v1
ncks -G g2:1 -g / -v v1 # /v1  -> /g2/v1
ncks -G g2:1 -v v1 # /v1  -> /g2/v1
ncks -G g2:1 -g g1/g1 -v v1 # /g1/g1/v1  -> /g2/g1/v1
```

Combining GPE with subsetting (see Section 3.11 [Subsetting Files], page 40) yields powerful control over the extracted (or excluded) variables and groups and their placement in the output file as shown by the following commands. All commands below may be assumed to end with `in.nc out.nc`.

```plaintext
# Prepending paths without editing:
ncks # /g?/v?  -> /g?/v?
nucks -v v1 # /g?v1  -> /g?v1
ncks -G o1 # /g?v?  -> /o1/g?v?
ncks -G o1 -g g1 # /g?v?  -> /o1/g1/v?

# Delete-mode: Delete from and Prepend to path head
ncks -G :1 -g g1 -v v1 # /g1/v1  -> /g1/v1
ncks -G :1 -g g1/g1 -v v1 # /g1/g1/v1  -> /g1/v1
ncks -G :2 -g g1/g1 -v v1 # /g1/g1/v1  -> /v1
ncks -G :2 -g g1 -v v1 # /g1/v1  -> /v1
ncks -G g2:1 -g g1 -v v1 # /g1/v1  -> /g2/v1
ncks -G g2:1 -g g1/v1 -v v1 # /g1/g1/v1  -> /g2/v1
ncks -G g2:1 -g / -v v1 # /v1  -> /g2/v1
ncks -G g2:1 -v v1 # /v1  -> /g2/v1
ncks -G g2:1 -g g1/g1 -v v1 # /g1/g1/v1  -> /g2/g1/v1
```

Combining GPE with subsetting (see Section 3.11 [Subsetting Files], page 40) yields powerful control over the extracted (or excluded) variables and groups and their placement in the output file as shown by the following commands. All commands below may be assumed to end with `in.nc out.nc`.

```plaintext
# Prepending paths without editing:
ncks # /g?/v?  -> /g?/v?
ncks -v v1 # /g?v1  -> /g?v1
ncks -G o1 # /g?v?  -> /o1/g?v?
ncks -G o1 -g g1 # /g?v?  -> /o1/g1/v?

# Delete-mode: Delete from and Prepend to path head
ncks -G :1 -g g1 -v v1 # /g1/v1  -> /g1/v1
ncks -G :1 -g g1/g1 -v v1 # /g1/g1/v1  -> /g1/v1
ncks -G :2 -g g1/g1 -v v1 # /g1/g1/v1  -> /v1
ncks -G :2 -g g1 -v v1 # /g1/v1  -> /v1
ncks -G g2:1 -g g1 -v v1 # /g1/v1  -> /g2/v1
ncks -G g2:1 -g g1/v1 -v v1 # /g1/g1/v1  -> /g2/v1
ncks -G g2:1 -g / -v v1 # /v1  -> /g2/v1
ncks -G g2:1 -v v1 # /v1  -> /g2/v1
ncks -G g2:1 -g g1/g1 -v v1 # /g1/g1/v1  -> /g2/g1/v1
```
3.13.2 Moving Groups

Until fall 2013 (netCDF version 4.3.1-pre1), netCDF contained no library function for renaming groups, and therefore ncrename cannot rename groups. However, NCO built on earlier versions of netCDF than 4.3.1 can use a GPE-based workaround mechanism to “rename” groups. The GPE mechanism actually moves (i.e., copies to a new location) groups, a more arduous procedure than simply renaming them. GPE applies to all selected groups, so, in the general case, one must move only the desired group to a new file, and then merge that new file with the original to obtain a file where the desired group has been “renamed” and all else is unchanged. Here is how to “rename” group /g4 to group /f4 with GPE instead of ncrename

```
ncks -O -G : -g cesm -3 ~/nco/data/cmip5.nc ~/cesm.nc # Extract /cesm to /
ncks -O -G f4:1 -g g4 ~/nco/data/in_grp.nc ~/tmp.nc # Move /g4 to /f4
ncks -O -x -g g4 ~/nco/data/in_grp.nc ~/out.nc # Excise /g4
ncks -A ~/tmp.nc ~/out.nc # Add /f4 to new file
```

If the original group g4 is not excised from out.nc (step two above), then the final output file would contain both g4 and a copy named f4. Thus GPE can be used to both “rename” and copy groups. The recommended way to rename groups when when netCDF version 4.3.1 is available is to use ncrename (see Section 4.11 [ncrename netCDF Renamer], page 214).

One may wish to flatten hierarchical group files for many reasons. These include
1. Obtaining flat netCDF3 files for use with tools that do not work with netCDF4 files,
2. Splitting apart hierarchies to re-assemble into different hierarchies, and
3. Providing a subset of a hierarchical file with the simplest possible storage structure.

```
ncks -O -G : -g cesm -3 ~/nco/data/cmip5.nc ~/cesm.nc # Extract /cesm to /
```
The -3 switch\textsuperscript{15} specifies the output dataset should be in netCDF3 format, the -G option flattens all extracted groups, and the -g cesm option extracts only the cesm group and leaves all other groups (e.g., ecmwf, giss).

3.13.3 Dismembering Files

Let us show how to completely disaggregate (or, more memorably) dismember a hierarchical dataset. For now we take this to mean: store each group as a standalone flat dataset in netCDF3 format. This can be accomplished by looping the previous example over all groups. This script \texttt{ncdismember} dismembers the input file \texttt{fl.in} specified in the first argument and places the resulting files in the directory \texttt{drc.out} specified by the second argument:

\begin{verbatim}
cat > ~/ncdismember << 'EOF'
# Purpose: Dismember netCDF4/HDF5 hierarchical files. CF-check them.
# Place each input file group in separate netCDF3 output file
# Described in NCO User Guide at http://nco.sf.net/nco.html#dismember
# Requirements: NCO 4.3.x+, UNIX shell utilities awk, grep, sed
# Optional: CFchecker command https://bitbucket.org/mde_/cfchecker

# Usage:
# ncdismember <fl_in> <drc_out> [flg_cf] [cf_vrs] [opt]
# where fl_in is input file/URL to dismember, drc_out is output directory,
# CF-compliance check is performed when optional third argument is 'cf',
# Optional fourth argument cf_vrs is CF version to check
# Optional fifth argument opt passes straight through to ncks
# chmod a+x ~/sh/ncdismember
# ncdismember ~/nco/data/mdl_1.nc /data/zender/nco/tmp
# ncdismember ~/nco/data/mdl_1.nc /data/zender/nco/tmp
# ncdismember http://dust.ess.uci.edu/nco/mdl_1.nc /data/zender/nco/tmp
# ncdismember http://thredds-test.uci.edu/thredds/dodsC/testdods/foo.nc /data/zender/tmp
# ncdismember ~/nco/data/mdl_1.nc /data/zender/nco/tmp cf
# ncdismember ~/nco/data/mdl_1.nc /data/zender/nco/tmp cf 1.3
# ncdismember ~/nco/data/mdl_1.nc /data/zender/nco/tmp cf 1.5 --fix_rec_dmn=all

# Command line argument defaults
fl_in="${HOME}/nco/data/mdl_1.nc" # [sng] Input file to dismember/check
drc_out="${DATA}/nco/tmp" # [sng] Output directory
flg_cf='0' # [flg] Perform CF-compliance check
cf_vrs='1.5' # [sng] Compliance-check this CF version (e.g., '1.5')
opt=' ' # [flg] Additional ncks options (e.g., '--fix_rec_dmn=all')
# Use single quotes to pass multiple arguments to opt=${5}
# Otherwise arguments would be seen as ${5}, ${6}, ${7} ...

# Command line argument option parsing
if [ -n "${1}" ]; then fl_in=${1}; fi
if [ -n "${2}" ]; then drc_out=${2}; fi

\end{verbatim}

\textsuperscript{15}Note that the -3 switch should appear after the -G and -g switches. This is due to an artifact of the GPE implementation which we wish to remove in the future.
if [ -n "${3}" ]; then flg_cf=${3}; fi
if [ -n "${4}" ]; then cf_vrs=${4}; fi
if [ -n "${5}" ]; then opt=${5}; fi

# Prepare output directory
echo "NCO dismembering file ${fl_in}"
f1_stb=$(basename ${fl_in})
drc_out=${drc_out}/${f1_stb}
mkdir -p ${drc_out}
cd ${drc_out}

# Obtain group list
grp_lst='ncks --cdl -m ${fl_in} | grep '// group' | awk '{$1=$2=$3="";sub(/\*\*\*","",$0);print}'
IFS=$'\n' # Change Internal-Field-Separator from <Space><Tab><Newline> to <Newline>
for grp_in in $grp_lst; do
    # Replace slashes by dots for output group filenames
    grp_out='echo ${grp_in} | sed 's/\///\/' | sed 's/\//./g''
    if [ "${grp_out}" = ''; then
        grp_out='root';
    fi
    # Tell older NCO/netCDF if HDF4 with --hdf4 switch (signified by .hdf/.HDF suffix)
    hdf4='echo ${fl_in} | awk '{if(match(tolower($1),".hdf$")) hdf4="--hdf4"; print hdf4}'
    # Flatten to netCDF3, anchor, no history, no temporary file, padding, HDF4 flag, options
    cmd="ncks -O -3 -G : -g ${grp_in}/ -h --no_tmp_fl --hdr_pad=40 ${hdf4} ${opt} ${fl_in} ${drc_out}/${grp_out}.nc"
    # Use eval in case ${opt} contains multiple arguments separated by whitespace
    eval ${cmd}
    if [ ${flg_cf} = 'cf' ]; then
        # cfchecker needs Conventions <= 1.5
        no_bck_sls='echo ${drc_out}/${grp_out} | sed 's/\ / /g''
        ncatted -h -a Conventions,global,o,c,CF=${cf_vrs} ${no_bck_sls}.nc
    else # !flg_cf
        echo ${drc_out}/${grp_out}.nc
    fi # !flg_cf
done

if [ ${flg_cf} = 'cf' ]; then
    echo "CFchecker reports CF-compliance of each group in flat netCDF3 format"
    cfchecker -c ${cf_vrs} *.nc
fi # !flg_cf

EOF
chmod 755 ~/ncdismember # Make command executable
/bin/mv -f ~/ncdismember ~/sh # Store in location on $PATH, e.g., /usr/local/bin

zender@roulee:~$ ncdismember ~/nco/data/mdl_1.nc ${DATA}/nco/tmp
NCO dismembering file /home/zender/nco/data/mdl_1.nc
/data/zender/nco/tmp/mdl_1.nc/cesm.cesm_01.nc
/data/zender/nco/tmp/mdl_1.nc/cesm.cesm_02.nc
/zender/nco/tmp/mdl_1.nc/ecmwf.ecmwf_01.nc
/data/zender/nco/tmp/mdl_1.nc/ecmwf.ecmwf_02.nc
/data/zender/nco/tmp/mdl_1.nc/ecmwf.nc
A (potentially more portable) binary executable could be written to dismember all groups with a single invocation, yet dismembering without loss of information is possible now with this simple script on all platforms with UNIXy utilities. Note that all dimensions inherited by groups in the input file are correctly placed by ncdismember into the flat files. Moreover, each output file preserves the group metadata of all ancestor groups, including the global metadata from the input file. As written, the script could fail on groups that contain advanced netCDF4 features because the user requests (with the ‘-3’ switch) that output be netCDF3 classic format. However, ncks detects many format incompatibilities in advance and works around them. For example, ncks autoconverts netCDF4-only atomic-types (such as NC_STRING and NC_UBYTE) to corresponding netCDF3 atomic types (NC_CHAR and NC_SHORT) when the output format is netCDF3.

3.13.4 Checking CF-compliance

One application of dismembering is to check the CF-compliance of each group in a file. When invoked with the optional third argument ‘cf’, ncdismember passes each file it generates to the freely available\textsuperscript{16} cfchecker command.

```
$ ncdismember ~/nco/data/mdl_1.nc /data/zender/nco/tmp cf
NCO dismembering file /home/zender/nco/data/mdl_1.nc
CFchecker reports CF-compliance of each group in flat netCDF3 format
WARNING: Using the default (non-CF) Udunits database
cecm.cesm_01.nc:
INFO: INIT: running CFchecker version 1.5.15
INFO: INIT: checking compliance with convention CF-1.5
INFO: INIT: using standard name table version: 25, last modified: 2013-07-05T05:40:30Z
INFO: INIT: using area type table version: 2, date: 10 July 2013
INFO: 2.4: no axis information found in dimension variables, not checking dimension order
WARNING: 3: variable "tas1" contains neither long_name nor standard_name attribute
WARNING: 3: variable "tas2" contains neither long_name nor standard_name attribute
INFO: 3.1: variable "tas1" does not contain units attribute
INFO: 3.1: variable "tas2" does not contain units attribute
```

```
cecm.cesm_02.nc:
```

By default the CF version checked is determined automatically by cfchecker. The user can override this default by supplying a supported CF version, e.g., ‘1.3’, as an optional fourth argument to ncdismember. Current valid CF options are ‘1.0’, ‘1.1’, ‘1.2’, ‘1.3’, ‘1.4’, and ‘1.5’.

Our development and testing of ncdismember is funded by our involvement in NASA’s Dataset Interoperability Working Group (DIWG), though our interest extends beyond NASA datasets. Taken together, NCO’s features (autoconversion to netCDF3 atomic types, fixing multiple record dimensions, autosensing HDF4 input, scoping rules for CF conven-

\textsuperscript{16} CFchecker is developed by Michael Decker and Martin Schultz at Forschungszentrum Jülich and distributed at \url{https://bitbucket.org/mde_/cfchecker}. 
tions) make ncdismember reliable and friendly for both dismembering hierarchical files and for CF-compliance checks. Most HDF4 and HDF5 datasets can be checked for CF-compliance with a one-line command. Example compliance checks of common NASA datasets are at http://dust.ess.uci.edu/diwg. Our long-term goal is to enrich the hierarchical data model with the expressivity and syntactic power of CF conventions.

NASA asked the DIWG to prepare a one-page summary of the procedure necessary to check HDF files for CF-compliance:

```
cat > ~/ncdismember.txt << 'EOF'
Preparing an RPM-based OS to Test HDF & netCDF Files for CF-Compliance

By Charlie Zender, UCI & NASA Dataset Interoperability Working Group (DIWG)

Installation Summary:
1. HDF4 [with internal netCDF support _disabled_]
2. HDF5
3. netCDF [with external HDF4 support _enabled_]
4. NCO
5. numpy
6. netcdf4-python
7. python-lxml
8. CFunits-python
9. CFChecker
10. ncdismember

All 10 packages can use default installs _except_ HDF4 and netCDF.
Following instructions for Fedora Core 20 (FC20), an RPM-based Linux OS
Feedback and changes for other Linux-based OS’s welcome to zender at uci.edu
${H4DIR}, ${H5DIR}, ${NETCDFDIR}, ${NCODIR}, may all be different
For simplicity CZ sets them all to /usr/local

# 1. HDF4. Build in non-default manner. Turn-off its own netCDF support.
# Per http://www.unidata.ucar.edu/software/netcdf/docs/build_hdf4.html
# HDF4 support not necessary though it makes ncdismember more comprehensive
wget -c http://www.hdfgroup.org/ftp/HDF/HDF_Current/src/hdf-4.2.9.tar.gz
tar xvzf hdf-4.2.9.tar.gz
cd hdf-4.2.9
./configure --enable-shared --disable-netcdf --disable-fortran --prefix=${H4DIR}
make & make check & make install

# 2. HDF5. Build normally. RPM may work too. Please let me know if so.
# HDF5 is a necessary pre-requisite for netCDF4
tar xvzf hdf5-1.8.11.tar.gz
cd hdf5-1.8.11
./configure --enable-shared --prefix=${H5DIR}
make & make check & make install
```
# 3. netCDF version 4.3.1 or later. Build in non-default manner with HDF4.
# Per http://www.unidata.ucar.edu/software/netcdf/docs/build_hdf4.html
# Earlier versions of netCDF may fail checking some HDF4 files
wget -c ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4.3.2.tar.gz
tar xvzf netcdf-4.3.2.tar.gz
cd netcdf-4.3.2
CPPFLAGS="-I${H5DIR}/include -I${H4DIR}/include" \
LDFLAGS="-L${H5DIR}/lib -L${H4DIR}/lib" \
./configure --enable-hdf4 --enable-hdf4-file-tests
make && make check && make install

# 4. NCO version 4.4.0 or later. Some RPMs available. Or install by hand.
# Later versions of NCO have much better support for ncdismember
wget http://nco.sourceforge.net/src/nco-4.4.4.tar.gz .
tar xvzf nco-4.4.4.tar.gz
cd nco-4.4.4
./configure --prefix=${NCODIR}
make && make install

# 5. numpy
sudo yum install numpy -y

# 6. netcdf4-python
sudo yum install netcdf4-python -y

# 7. python-lxml
sudo yum install python-lxml -y

# 8. CFunits-python. No RPM available. Must install by hand.
# http://code.google.com/p/cfunits-python/
tar xvzf cfunits-0.9.6.tar.gz
cd cfunits-0.9.6
sudo python setup.py install

# 9. CFChecker. No RPM available. Must install by hand.
# https://bitbucket.org/mde_/cfchecker
wget https://bitbucket.org/mde_/cfchecker/downloads/CFchecker-1.5.15.tar.bz2 .
tar xvjf CFchecker-1.5.15.tar.bz2
cd CFchecker
sudo python setup.py install

# 10. ncdismember. Copy script from http://nco.sf.net/nco.html#ncdismember
# Store dismembered files somewhere, e.g., ${DATA}/nco/tmp/hdf
mkdir -p ${DATA}/nco/tmp/hdf
# Many datasets work with a simpler command...
ncdismember \~{/nco/data/in.nc} ${\text{DATA}}$/nco/tmp/hdf cf 1.5
ncdismember \~{/nco/data/mdl_1.nc} ${\text{DATA}}$/nco/tmp/hdf cf 1.5
ncdismember ${\text{DATA}}$/hdf/AMSR_E_L2_Rain_V10_200905312326_A.hdf \ ${\text{DATA}}$/nco/tmp/hdf cf 1.5
ncdismember ${\text{DATA}}$/hdf/BUV-Nimbus04_L3zm_v01-00-00-2012m0203t144121.h5 \ ${\text{DATA}}$/nco/tmp/hdf cf 1.5
ncdismember ${\text{DATA}}$/hdf/HIRDLS-Aura_L3ZAD_v06-00-00-c02_2005d022-2008d077.he5 ${\text{DATA}}$/
# Some datasets, typically .h5, require the --fix_rec_dmn=all argument
ncdismember_${\text{DATA}}$/hdf/GATMO_npp_d20100906_t1935191_e1935505_b00012_c2011070715593206_h5 \ ${\text{DATA}}$/nco/tmp/hdf cf 1.5 --fix_rec_dmn=all
ncdismember ${\text{DATA}}$/hdf/mabel_l2_20130927t201800_008_1.h5 \ ${\text{DATA}}$/nco/tmp/hdf cf 1.5
EOF

A PDF version of these instructions is available here.

### 3.14 C and Fortran Index conventions

| Availability: ncbo, ncps, ncecat, ncflint, ncks, ncflint, ncra, ncrcat, ncwa |
|-------------------|-------------------|-------------------|-------------------|-------------------|
| Short options: `-F` |
| Long options: `'--fortran'` |

The ‘-F’ switch changes NCO to read and write with the Fortran index convention. By default, NCO uses C-style (0-based) indices for all I/O. In C, indices count from 0 (rather than 1), and dimensions are ordered from slowest (inner-most) to fastest (outer-most) varying. In Fortran, indices count from 1 (rather than 0), and dimensions are ordered from fastest (inner-most) to slowest (outer-most) varying. Hence C and Fortran data storage conventions represent mathematical transposes of each other. Note that record variables contain the record dimension as the most slowly varying dimension. See Section 4.8 [ncpdq netCDF Permute Dimensions Quickly], page 202 for techniques to re-order (including transpose) dimensions and to reverse data storage order.

Consider a file 85.nc containing 12 months of data in the record dimension time. The following hyperslab operations produce identical results, a June-July-August average of the data:

```bash
ncra -d time,5,7 85.nc 85_JJA.nc
ncra -F -d time,6,8 85.nc 85_JJA.nc
```

Printing variable `three_dmn_var` in file `in.nc` first with the C indexing convention, then with Fortran indexing convention results in the following output formats:

```bash
% ncks -v three_dmn_var in.nc
lat[0]=-90 lev[0]=1000 lon[0]=-180 three_dmn_var[0]=0
...
% ncks -F -v three_dmn_var in.nc
lon(1)=0 lev(1)=100 lat(1)=-90 three_dmn_var(1)=0
...
```
### 3.15 Hyperslabs

A **hyperslab** is a subset of a variable’s data. The coordinates of a hyperslab are specified with the `-d dim,[min],[max],[stride]]` short option (or with the same arguments to the ‘--dimension’ or ‘--dmn’ long options). At least one hyperslab argument (`min`, `max`, or `stride`) must be present. The bounds of the hyperslab to be extracted are specified by the associated `min` and `max` values. A half-open range is specified by omitting either the `min` or `max` parameter. The separating comma must be present to indicate the omission of one of these arguments. The unspecified limit is interpreted as the maximum or minimum value in the unspecified direction. A cross-section at a specific coordinate is extracted by specifying only the `min` limit and omitting a trailing comma. Dimensions not mentioned are passed with no reduction in range. The dimensionality of variables is not reduced (in the case of a cross-section, the size of the constant dimension will be one).

```
# First and second longitudes
ncks -d lon,1,2 in.nc out.nc

# Second and third longitudes
ncks -d lon,1,2 in.nc out.nc
```

As of version 4.2.1 (August, 2012), NCO allows one to extract the last \( N \) elements of a hyperslab. Negative integers as `min` or `max` elements of a hyperslab specification indicate offsets from the end (Python also uses this convention). Consistent with this convention, the value ‘\(-1\)’ (negative one) indicates the last element of a dimension, and negative zero is algebraically equivalent to zero and so indicates the first element of a dimension. Previously, for example, `-d time,-2,-1` caused a domain error. Now it means select the penultimate and last timesteps, independent of the size of the `time` dimension. Select only the first and last timesteps, respectively, with ‘-d time,0’ and ‘-d time,-1’. Negative integers work for `min` and `max` indices, though not for `stride`.

```
# Second through penultimate longitudes
ncks -d lon,1,-2 in.nc out.nc

# Second through last longitude
ncks -d lon,1,-1 in.nc out.nc

# Second-to-last to last longitude
ncks -d lon,-3,-1 in.nc out.nc

# Second-to-last to last longitude
ncks -d lon,-3, in.nc out.nc
```

The ‘-F’ argument, if any, applies the Fortran index convention only to indices specified as positive integers:

```
# First through penultimate longitudes
ncks -F -d lon,1,-2 in.nc out.nc (-F affects only start index)
```
# First through last longitude
ncks -F -d lon,1,-1 in.nc out.nc
# Second-to-last to penultimate longitude (-F has no effect)
ncks -F -d lon,-3,-1 in.nc out.nc
# Second-to-last to last longitude (-F has no effect)
ncks -F -d lon,-3, in.nc out.nc

Coordinate values should be specified using real notation with a decimal point required in the value, whereas dimension indices are specified using integer notation without a decimal point. This convention serves only to differentiate coordinate values from dimension indices. It is independent of the type of any netCDF coordinate variables. For a given dimension, the specified limits must both be coordinate values (with decimal points) or dimension indices (no decimal points).

If values of a coordinate-variable are used to specify a range or cross-section, then the coordinate variable must be monotonic (values either increasing or decreasing). In this case, command-line values need not exactly match coordinate values for the specified dimension. Ranges are determined by seeking the first coordinate value to occur in the closed range \([\text{min}, \text{max}]\) and including all subsequent values until one falls outside the range. The coordinate value for a cross-section is the coordinate-variable value closest to the specified value and must lie within the range or coordinate-variable values. The \textit{stride} argument, if any, must be a dimension index, not a coordinate value. See Section 3.16 [Stride], page 57, for more information on the \textit{stride} option.

# All longitude values between 1 and 2 degrees
ncks -d lon,1.0,2.0 in.nc out.nc
# All longitude values between 1 and 2 degrees
ncks -F -d lon,1.0,2.0 in.nc out.nc
# Every other longitude value between 0 and 90 degrees
ncks -F -d lon,0.0,90.0,2 in.nc out.nc

As shown, we recommend using a full floating point suffix of \texttt{.0} instead of simply \texttt{.} in order to make obvious the selection of hyperslab elements based on coordinate value rather than index.

User-specified coordinate limits are promoted to double-precision values while searching for the indices which bracket the range. Thus, hyperslabs on coordinates of type \texttt{NC\_CHAR} are computed numerically rather than lexically, so the results are unpredictable.

The relative magnitude of \texttt{min} and \texttt{max} indicate to the operator whether to expect a \textit{wrapped coordinate} (see Section 3.20 [Wrapped Coordinates], page 63), such as longitude. If \texttt{min} > \texttt{max}, the NCO expects the coordinate to be wrapped, and a warning message will be printed. When this occurs, NCO selects all values outside the domain \([\text{max} < \text{min}]\), i.e., all the values exclusive of the values which would have been selected if \texttt{min} and \texttt{max} were swapped. If this seems confusing, test your command on just the coordinate variables with \texttt{ncks}, and then examine the output to ensure NCO selected the hyperslab you expected (coordinate wrapping is currently only supported by \texttt{ncks}).

Because of the way wrapped coordinates are interpreted, it is very important to make sure you always specify hyperslabs in the monotonically increasing sense, i.e., \texttt{min} < \texttt{max}
(even if the underlying coordinate variable is monotonically decreasing). The only exception to this is when you are indeed specifying a wrapped coordinate. The distinction is crucial to understand because the points selected by, e.g., `-d longitude,50.,340.` are exactly the complement of the points selected by `-d longitude,340.,50.`.

Not specifying any hyperslab option is equivalent to specifying full ranges of all dimensions. This option may be specified more than once in a single command (each hyperslabbed dimension requires its own `-d` option).

### 3.16 Stride

<table>
<thead>
<tr>
<th>Availability: ncbo, nces, ncecat, ncflint, ncks, ncpdq, ncra, ncrcat, ncwa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: <code>'-d dim,[min][,][max][,[stride]]'</code></td>
</tr>
<tr>
<td>Long options: <code>'--dimension dim,[min][,][max][,[stride]]'</code>, <code>'--dmn dim,[min][,][max][,[stride]]'</code></td>
</tr>
</tbody>
</table>

All data operators support specifying a *stride* for any and all dimensions at the same time. The *stride* is the spacing between consecutive points in a hyperslab. A stride of 1 picks all the elements of the hyperslab, and a stride of 2 skips every other element, etc.. *ncks* multislabs support strides, and are more powerful than the regular hyperslabs supported by the other operators (see Section 3.19 [Multislabs], page 60). Using the *stride* option for the record dimension with *ncra* and *ncrcat* makes it possible, for instance, to average or concatenate regular intervals across multi-file input data sets.

The *stride* is specified as the optional fourth argument to the `'-d'` hyperslab specification: `'-d dim,[min][,][max][,[stride]]'`. Specify *stride* as an integer (i.e., no decimal point) following the third comma in the `'-d'` argument. There is no default value for *stride*. Thus using `'-d time,,,2'` is valid but `'-d time,,,2.0'` and `'-d time,,,'` are not. When *stride* is specified but *min* is not, there is an ambiguity as to whether the extracted hyperslab should begin with (using C-style, 0-based indexes) element 0 or element `*stride-1*`. NCO must resolve this ambiguity and it chooses element 0 as the first element of the hyperslab when *min* is not specified. Thus `'-d time,,,stride'` is syntactically equivalent to `'-d time,0,,,stride'.` This means, for example, that specifying the operation `'-d time,,,2'` on the array `1,2,3,4,5'` selects the hyperslab `1,3,5'.` To obtain the hyperslab `2,4'` instead, simply explicitly specify the starting index as 1, i.e., `'-d time,1,,,2'`.

For example, consider a file `8501_8912.nc` which contains 60 consecutive months of data. Say you wish to obtain just the March data from this file. Using 0-based subscripts (see Section 3.14 [C and Fortran Index Conventions], page 54) these data are stored in records `2, 14, . . . 50` so the desired *stride* is 12. Without the *stride* option, the procedure is very awkward. One could use *ncks* five times and then use *ncrcat* to concatenate the resulting files together:

```
for idx in 02 14 26 38 50; do # Bourne Shell
  ncks -d time,${idx} 8501_8912.nc foo.${idx}
done
`for each` idx (02 14 26 38 50) # C Shell
```
ncks -d time,${idx} 8501_8912.nc foo.${idx}
end
ncrcat foo.?? 8589_03.nc
rm foo.??

With the *stride* option, *ncks* performs this hyperslab extraction in one operation:

```
ncks -d time,2,,12 8501_8912.nc 8589_03.nc
```

See Section 4.7 [ncks netCDF Kitchen Sink], page 188, for more information on *ncks*.

Applying the *stride* option to the record dimension in *ncra* and *ncrcat* makes it possible, for instance, to average or concatenate regular intervals across multi-file input data sets.

```
ncra -F -d time,3,,12 85.nc 86.nc 87.nc 88.nc 89.nc 8589_03.nc
ncrcat -F -d time,3,,12 85.nc 86.nc 87.nc 88.nc 89.nc 8503_8903.nc
```

### 3.17 Record Appending

<table>
<thead>
<tr>
<th>Availability: ncra, ncrcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: None</td>
</tr>
<tr>
<td>Long options: <code>--rec_apn</code>, <code>--record_append</code></td>
</tr>
</tbody>
</table>

As of version 4.2.6 (March, 2013), NCO allows both Multi-File, Multi-Record operators (*ncra* and *ncrcat*) to append their output directly to the end of an existing file. This feature may be used to augment a target file, rather than construct it from scratch. This helps, for example, when a timeseries is concatenated from input data that becomes available in stages rather than all at once. In such cases this switch significantly speeds writing.

Consider the use case where one wishes to preserve the contents of *fl_1.nc*, and add to them new records contained in *fl_2.nc*. Previously the output had to be placed in a third file, *fl_3.nc* (which could also safely be named *fl_2.nc*), via

```
ncrcat -O fl_1.nc fl_2.nc fl_3.nc
```

Under the hood this operation copies all information in *fl_1.nc* and *fl_2.nc* not once but twice. The first copy is performed through the netCDF interface, as all data from *fl_1.nc* and *fl_2.nc* are extracted and placed in the output file. The second copy occurs (usually much) more quickly as the (by default) temporary output file is copied (sometimes a quick re-link suffices) to the final output file (see Section 2.3 [Temporary Output Files], page 15). All this copying is expensive for large files.

The `--record_append` switch appends all records in *fl_2.nc* to the end (after the last record) of *fl_1.nc*:

```
ncrcat --rec_apn fl_2.nc fl_1.nc
```

The ordering of the filename arguments may seem non-intuitive. If the record variable represents time in these files, then the values in *fl_1.nc* precede those in *fl_2.nc*, so why do the files appear in the reverse order on the command line? *fl_1.nc* is the last file named because it is the pre-existing output file to which we will append all the other input files listed
in this case only fl_2.nc). The contents of fl_1.nc are completely preserved, and only values in fl_2.nc (and any other input files) are copied. This switch avoids the necessity of copying all of fl_1.nc through the netCDF interface to a new output file. The ‘--rec_apn’ switch automatically puts NCO into append mode (see Section 2.4 [Appending Variables], page 17), so specifying ‘-A’ is redundant, and simultaneously specifying overwrite mode with ‘-0’ causes an error. By default, NCO works in an intermediate temporary file. Power users may combine ‘--rec_apn’ with the ‘--no_tmp_fl’ switch (see Section 2.3 [Temporary Output Files], page 15):

```
ncrcat --rec_apn --no_tmp_fl fl_2.nc fl_1.nc
```

This avoids creating an intermediate file, and copies only the minimal amount of data (i.e., all of fl_2.nc). Hence, it is fast. We recommend users try to understand the safety trade-offs involved.

### 3.18 Subcycle

<table>
<thead>
<tr>
<th>Availability: ncra, ncrcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: ‘-d dim,[min][,[max]][,[stride]][,[subcycle]]’</td>
</tr>
<tr>
<td>Long options: ‘--mro’ ‘--dimension dim,[min][,[max]][,[stride]][,[subcycle]]’</td>
</tr>
<tr>
<td>‘--dmn dim,[min][,[max]][,[stride]][,[subcycle]]’</td>
</tr>
</tbody>
</table>

As of version 4.2.1 (August, 2012), NCO allows both Multi-File, Multi-Record operators, ncra and ncrcat, to extract and operate on multiple groups of records. These groups may be connected to physical sub-cycles of a periodic nature, e.g., months of a year, or hours of a day. Or they may be thought of as groups of a specified duration. The feature and the terminology to describe it are new. For now, we call this the subcycle feature, sometimes abbreviated SSC.

The subcycle feature allows processing of groups of records separated by regular intervals of records. It is perhaps best illustrated by an extended example which describes how to solve the same problem both with and without the SSC feature.

The first task in climate data processing is often creating seasonal cycles. Suppose a 150-year climate simulation produces 150 output files, each comprising 12 records, each record a monthly mean: 1850.nc, 1851.nc, ... 1999.nc. Our goal is to create a single file containing the summertime (June, July, and August, aka JJA) mean. Traditionally, we would first compute the climatological monthly mean for each month of summer. Each of these is a 150-year mean, i.e.,

```
# Step 1: Create climatological monthly files clm06.nc..clm08.nc
for mth in {6..8}; do
    mm='printf "%02d" $mth'
    ncra -O -F -d time,${mm},,12 -n 150,4,1 1850.nc clm${mm}.nc
done
# Step 2: Average climatological monthly files into summertime mean
```

When originally released in 2012 this was called the duration feature, and was abbreviated DRN.
ncra -O clm06 clm07.nc clm08.nc clm_JJA.nc

So far, nothing is unusual and this task can be performed by any NCO version. The SSC feature makes obsolete the need for the shell loop used in Step 1 above.

The new SSC option aggregates more than one input record at a time before performing arithmetic operations, and, with an additional switch, allows us to archive those results in multiple record output (MRO) files. This reduces the task of producing the climatological summertime mean to one step:

```
# Step 1: Compute climatological summertime mean
ncra -O -F -d time,6,,12,3 -n 150,4,1 1850.nc clm_JJA.nc
```

The SSC option instructs `ncra` (or `ncrcat`) to process files in groups of three records. To better understand the meaning of each argument to the `-d` hyperslab option, read it this way: “for the time dimension start with the sixth record, continue without end, repeat the process every twelfth record, and define a sub-cycle as three consecutive records”.

A separate option, `--mro`, instructs `ncra` to output its results from each sub-group, and to produce a Multi-Record Output (MRO) file rather than a Single-Record Output (SRO) file. Unless `--mro` is specified, `ncra` collects together all the sub-groups, operates on their ensemble, and produces a single output record. The addition of `--mro` to the above example causes `ncra` to archive all (150) annual summertime means to one file:

```
# Step 1: Archive all 150 summertime means in one file
ncra --mro -O -F -d time,6,,12,3 -n 150,4,1 1850.nc 1850_2009_JJA.nc
# ...or all (150) annual means...
ncra --mro -O -F -d time,,,12,12 -n 150,4,1 1850.nc 1850_2009.nc
```

These operations generate and require no intermediate files. This contrasts to previous NCO methods, which require generating, averaging, then concatenating 150 files. The `--mro` option has no effect on, or rather is redundant for, `ncrcat` since `ncrcat` always outputs all selected records.

### 3.19 Multislabs

| Availability: ncbo, ncex, ncecat, ncflint, ncks, ncpdq, ncra, ncrcat |
| Short options: ‘-d dim,[min][,[max]][,[stride]]’ |
| Long options: ‘--dimension dim,[min][,[max]][,[stride]]’, ‘--dmn dim,[min][,[max]][,[stride]]’ |
| ‘--msa_usr_rdr’, ‘--msa_user_order’ |

A multislab is a union of one or more hyperslabs. One defines multislabs by chaining together hyperslab commands, i.e., `-d` options (see Section 3.15 [Hyperslabs], page 55). Support for specifying a multi-hyperslab or multislab for any variable was first added to `ncks` in late 2002. The other operators received these capabilities in April 2008. Multi-slabbing is often referred to by the acronym MSA, which stands for “Multi-Slabbing Algorithm”. As explained below, the user may additionally request that the multislabs be returned in the user-specified order, rather than the on-disk storage order. Although MSA user-ordering
has been available in all operators since 2008, most users were unaware of it since the documentation (below, and in the man pages) was not written until July 2013.

Multislabs overcome many restraints that limit simple hyperslabs. A single \(-d\) option can only specify a contiguous and/or a regularly spaced multi-dimensional data array. Multislabs are constructed from multiple \(-d\) options and may therefore have non-regularly spaced arrays. For example, suppose it is desired to operate on all longitudes from 10.0 to 20.0 and from 80.0 to 90.0 degrees. The combined range of longitudes is not selectable in a single hyperslab specification of the form ‘\(-d\) dimension, min, max’ or \(-d\) dimension, min, max, stride’ because its elements are irregularly spaced in coordinate space (and presumably in index space too). The multislab specification for obtaining these values is simply the union of the hyperslabs specifications that comprise the multislab, i.e.,

\[
\text{ncks -d lon,10.,20. -d lon,80.,90. in.nc out.nc}
\]

\[
\text{ncks -d lon,10.,15. -d lon,15.,20. -d lon,80.,90. in.nc out.nc}
\]

Any number of hyperslabs specifications may be chained together to specify the multislab. MSA creates an output dimension equal in size to the sum of the sizes of the multislabs. This can be used to extend and or pad coordinate grids.

Users may specify redundant ranges of indices in a multislab, e.g.,

\[
\text{ncks -d lon,0,4 -d lon,2,9,2 in.nc out.nc}
\]

This command retrieves the first five longitudes, and then every other longitude value up to the tenth. Elements 0, 2, and 4 are specified by both hyperslab arguments (hence this is redundant) but will count only once if an arithmetic operation is being performed. This example uses index-based (not coordinate-based) multislabs because the \textit{stride} option only supports index-based hyper-slabbing. See Section 3.16 [Stride], page 57, for more information on the \textit{stride} option.

Multislabs are more efficient than the alternative of sequentially performing hyperslab operations and concatenating the results. This is because NCO employs a novel multislab algorithm to minimize the number of I/O operations when retrieving irregularly spaced data from disk. The NCO multislab algorithm retrieves each element from disk once and only once. Thus users may take some shortcuts in specifying multislabs and the algorithm will obtain the intended values. Specifying redundant ranges is not encouraged, but may be useful on occasion and will not result in unintended consequences.

Suppose the \(Q\) variable contains three dimensional arrays of distinct chemical constituents in no particular order. We are interested in the NOy species in a certain geographic range. Say that NO, NO2, and N2O5 are elements 0, 1, and 5 of the \textit{species} dimension of \(Q\). The multislab specification might look something like

\[
\text{ncks -d species,0,1 -d species,5 -d lon,0,4 -d lon,2,9,2 in.nc out.nc}
\]

Multislabs are powerful because they may be specified for every dimension at the same time. Thus multislabs obsolete the need to execute multiple \texttt{ncks} commands to gather the desired range of data.

The MSA user-order switch ‘\texttt{--msa_usr_rdr}’ (or ‘\texttt{--msa_user_order}’, both of which shorten to ‘\texttt{--msa}’) requests that the multislabs be output in the user-specified order from the command-line, rather than in the input-file on-disk storage order. This allows the
user to perform complex data re-ordering in one operation that would otherwise require
cumbersome steps of hyperslabbing, concatenating, and permuting. Consider the recent
example of a user who needed to convert datasets stored with the longitude coordinate \texttt{Lon}
ranging from \([-180,180)\) to datasets that follow the \([0,360)\) convention.

\begin{verbatim}
% ncks -H -v Lon in.nc
Lon[0]=-180
Lon[1]=-90
Lon[2]=0
Lon[3]=90
\end{verbatim}

Although simple in theory, this task requires both mathematics to change the numerical
value of the longitude coordinate, data hyperslabbing to split the input on-disk arrays at
Greenwich, and data re-ordering within to stitch the western hemisphere onto the eastern
hemisphere at the date-line. The '\texttt{--msa}' user-order switch overrides the default that data
are output in the same order in which they are stored on-disk in the input file, and instead
stores them in the same order as the multi-slabs are given to the command line. This
default is intuitive and is not important in most uses. However, the MSA user-order switch
allows users to meet their output order needs by specifying multi-slabs in a certain order.
Compare the results of default ordering to user-ordering for longitude:

\begin{verbatim}
% ncks -O -H -v Lon -d Lon,0.,180. -d Lon,-180.,-1.0 in.nc
Lon[0]=-180
Lon[1]=-90
Lon[2]=0
Lon[3]=90
% ncks -O -H --msa -v Lon -d Lon,0.,180. -d Lon,-180.,-1.0 in.nc
Lon[0]=0
Lon[1]=90
Lon[2]=-180
Lon[3]=-90
\end{verbatim}

The two multi-slabs are the same but they can be presented to screen, or to an output file,
in either order. The second example shows how to place the western hemisphere after the
eastern hemisphere, although they are stored in the opposite order in the input file.

With this background, one sees that the following commands suffice to rotate the input
file by 180 degrees longitude:

\begin{verbatim}
% ncks -O -v LatLon --msa -d Lon,0.,180. -d Lon,-180.,-1.0 in.nc out.nc
% ncap2 -O -s 'where(Lon < 0) Lon=Lon+360' out.nc out.nc
% ncks -C -H -v LatLon ~/nco/data/in.nc
Lat[0]=-45 Lon[0]=-180 LatLon[0]=0
\end{verbatim}
% ncks -C -H -v LatLon ~/out.nc
Lat[0]=-45 Lon[0]=0 LatLon[0]=2

There are other workable, valid methods to accomplish this rotation, yet none are simpler nor more efficient than utilizing MSA user-ordering. Some final comments on applying this algorithm: Be careful to specify hemispheres that do not overlap, e.g., by inadvertently specifying coordinate ranges that both include Greenwich. Some users will find using index-based rather than coordinate-based hyperslabs makes this clearer.

3.20 Wrapped Coordinates

A wrapped coordinate is a coordinate whose values increase or decrease monotonically (nothing unusual so far), but which represents a dimension that ends where it begins (i.e., wraps around on itself). Longitude (i.e., degrees on a circle) is a familiar example of a wrapped coordinate. Longitude increases to the East of Greenwich, England, where it is defined to be zero. Halfway around the globe, the longitude is 180 degrees East (or West). Continuing eastward, longitude increases to 360 degrees East at Greenwich. The longitude values of most geophysical data are either in the range \([0,360)\), or \([-180,180)\). In either case, the Westernmost and Easternmost longitudes are numerically separated by 360 degrees, but represent contiguous regions on the globe. For example, the Saharan desert stretches from roughly 340 to 50 degrees East. Extracting the hyperslab of data representing the Sahara from a global dataset presents special problems when the global dataset is stored consecutively in longitude from 0 to 360 degrees. This is because the data for the Sahara will not be contiguous in the input-file but is expected by the user to be contiguous in the output-file. In this case, ncks must invoke special software routines to assemble the desired output hyperslab from multiple reads of the input-file.

Assume the domain of the monotonically increasing longitude coordinate \(\text{lon}\) is \(0 < \text{lon} < 360\). ncks will extract a hyperslab which crosses the Greenwich meridian simply by specifying the westernmost longitude as \(\text{min}\) and the easternmost longitude as \(\text{max}\). The following commands extract a hyperslab containing the Saharan desert:

\[
\text{ncks -d lon,340.,50. in.nc out.nc}
\]

\[
\text{ncks -d lon,340.,50. -d lat,10.,35. in.nc out.nc}
\]
The first example selects data in the same longitude range as the Sahara. The second example further constrains the data to having the same latitude as the Sahara. The coordinate $\text{lon}$ in the output-file, out.nc, will no longer be monotonic! The values of $\text{lon}$ will be, e.g., ‘340, 350, 0, 10, 20, 30, 40, 50’. This can have serious implications should you run out.nc through another operation which expects the $\text{lon}$ coordinate to be monotonically increasing. Fortunately, the chances of this happening are slim, since $\text{lon}$ has already been hyperslabbed, there should be no reason to hyperslab $\text{lon}$ again. Should you need to hyperslab $\text{lon}$ again, be sure to give dimensional indices as the hyperslab arguments, rather than coordinate values (see Section 3.15 [Hyperslabs], page 55).

3.21 Auxiliary Coordinates

<table>
<thead>
<tr>
<th>Availability: ncbo, nces, ncecat, ncflint, ncks, ncpdq, ncra, ncrcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: ‘-X lon_min,lon_max,lat_min,lat_max’</td>
</tr>
<tr>
<td>Long options: ‘--auxiliary lon_min,lon_max,lat_min,lat_max’</td>
</tr>
</tbody>
</table>

Utilize auxiliary coordinates specified in values of the coordinate variable’s standard_name attributes, if any, when interpreting hyperslab and multi-slab options. Also ‘--auxiliary’. This switch supports hyperslabbing cell-based grids over coordinate ranges. This works on datasets that associate coordinate variables to grid-mappings using the CF-convention (see Section 3.38 [CF Conventions], page 109) coordinates and standard_name attributes described here. Currently, NCO understands auxiliary coordinate variables pointed to by the standard_name attributes for latitude and longitude. Cells that contain a value within the user-specified range [$\text{lon}$ min, $\text{lon}$ max, $\text{lat}$ min, $\text{lat}$ max] are included in the output hyperslab.

A cell-based grid collapses the horizontal spatial information (latitude and longitude) and stores it along a one-dimensional coordinate that has a one-to-one mapping to both latitude and longitude coordinates. Rectangular (in longitude and latitude) horizontal hyperslabs cannot be selected using the typical procedure (see Section 3.15 [Hyperslabs], page 55) of separately specifying ‘-d’ arguments for longitude and latitude. Instead, when the ‘-X’ is used, NCO learns the names of the latitude and longitude coordinates by searching the standard_name attribute of all variables until it finds the two variables whose standard_name’s are “latitude” and “longitude”, respectively. This standard_name attribute for latitude and longitude coordinates follows the CF-convention (see Section 3.38 [CF Conventions], page 109).

Putting it all together, consider a variable gds_3dvar output from simulations on a cell-based geodesic grid. Although the variable contains three dimensions of data (time, latitude, and longitude), it is stored in the netCDF file with only two dimensions, time and gds_crd.

```
% ncks -m -C -v gds_3dvar ~/nco/data/in.nc
```
```
gds_3dvar: type NC_FLOAT, 2 dimensions, 4 attributes, chunked? no, \n compressed? no, packed? no, ID = 41

gds_3dvar RAM size is 10*8*sizeof(NC_FLOAT) = 80*4 = 320 bytes

gds_3dvar dimension 0: time, size = 10 NC_DOUBLE, dim. ID = 20 \n```
The coordinates attribute lists the names of the latitude and longitude coordinates, `lat_gds` and `lon_gds`, respectively. The coordinates attribute is recommended though optional. With it, the user can immediately identify which variables contain the latitude and longitude coordinates. Without a coordinates attribute it would be unclear at first glance whether a variable resides on a cell-based grid. In this example, `time` is a normal record dimension and `gds_crd` is the cell-based dimension.

The cell-based grid file must contain two variables whose standard_name attributes are “latitude”, and “longitude”:

```bash
% ncks -m -C -v lat_gds,lon_gds ~/nco/data/in.nc
lat_gds: type NC_DOUBLE, 1 dimensions, 4 attributes, \n    chunked? no, compressed? no, packed? no, ID = 37
lat_gds RAM size is 8*sizeof(NC_DOUBLE) = 8*8 = 64 bytes
lat_gds dimension 0: gds_crd, size = 8 NC_FLOAT, dim. ID = 17 (CRD)
lat_gds attribute 0: long_name, size = 8 NC_CHAR, value = Latitude
lat_gds attribute 1: standard_name, size = 8 NC_CHAR, value = latitude
lat_gds attribute 2: units, size = 6 NC_CHAR, value = degree
lat_gds attribute 3: purpose, size = 62 NC_CHAR, value = \n    1-D latitude coordinate referred to by geodesic grid variables

lon_gds: type NC_DOUBLE, 1 dimensions, 4 attributes, \n    chunked? no, compressed? no, packed? no, ID = 38
lon_gds RAM size is 8*sizeof(NC_DOUBLE) = 8*8 = 64 bytes
lon_gds dimension 0: gds_crd, size = 8 NC_FLOAT, dim. ID = 17 (CRD)
lon_gds attribute 0: long_name, size = 9 NC_CHAR, value = Longitude
lon_gds attribute 1: standard_name, size = 9 NC_CHAR, value = longitude
lon_gds attribute 2: units, size = 6 NC_CHAR, value = degree
lon_gds attribute 3: purpose, size = 63 NC_CHAR, value = \n    1-D longitude coordinate referred to by geodesic grid variables
```

In this example `lat_gds` and `lon_gds` represent the latitude or longitude, respectively, of cell-based variables. These coordinates (must) have the same single dimension (`gds_crd`, in this case) as the cell-based variables. And the coordinates must be one-dimensional—multidimensional coordinates will not work.

This infrastructure allows NCO to identify, interpret, and process (e.g., hyperslab) the variables on cell-based grids as easily as it works with regular grids. To time-average all
the values between zero and 180 degrees longitude and between plus and minus 30 degrees latitude, we use

```
ncra -O -X 0.,180.,-30.,30. -v gds_3dvar in.nc out.nc
```

NCO accepts multiple ‘-X’ arguments for cell-based grid multi-slabs, just as it accepts multiple ‘-d’ arguments for multi-slabs of regular coordinates.

```
ncra -O -X 0.,180.,-30.,30. -X 270.,315.,45.,90. in.nc out.nc
```

The arguments to ‘-X’ are always interpreted as floating point numbers, i.e., as coordinate values rather than dimension indices so that these two commands produce identical results

```
ncra -X 0.,180.,-30.,30. in.nc out.nc
ncra -X 0,180,-30,30 in.nc out.nc
```

In contrast, arguments to ‘-d’ require decimal places to be recognized as coordinates not indices (see Section 3.15 [Hyperslabs], page 55). We recommend always using decimal points with ‘-X’ arguments to avoid confusion.

### 3.22 UDUnits Support

<table>
<thead>
<tr>
<th>Availability: ncbo, nces, ncecat, ncflint, ncks, ncpdq, ncra, ncrcat, ncwa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: ‘-d dim,[min][,max][,stride]’</td>
</tr>
<tr>
<td>Long options: ‘--dimension dim,[min][,max][,stride]’, ‘--dmn dim,[min][,max][,stride]’</td>
</tr>
</tbody>
</table>

There is more than one way to hyperskin a cat. The UDUnits package provides a library which, if present, NCO uses to translate user-specified physical dimensions into the physical dimensions of data stored in netCDF files. Unidata provides UDUnits under the same terms as netCDF, so sites should install both. Compiling NCO with UDUnits support is currently optional but may become required in a future version of NCO.

Two examples suffice to demonstrate the power and convenience of UDUnits support. First, consider extraction of a variable containing non-record coordinates with physical dimensions stored in MKS units. In the following example, the user extracts all wavelengths in the visible portion of the spectrum in terms of the units very frequently used in visible spectroscopy, microns:

```
% ncks -C -H -v wvl -d wvl,"0.4 micron","0.7 micron" in.nc
wvl[0]=5e-07 meter
```

The hyperslab returns the correct values because the wvl variable is stored on disk with a length dimension that UDUnits recognizes in the units attribute. The automagical algorithm that implements this functionality is worth describing since understanding it helps one avoid some potential pitfalls. First, the user includes the physical units of the hyperslab dimensions she supplies, separated by a simple space from the numerical values of the hyperslab limits. She encloses each coordinate specifications in quotes so that the shell does not break the value-space-unit string into separate arguments before passing them to NCO. Double quotes ("foo") or single quotes (’foo’) are equally valid for this purpose.
Second, NCO recognizes that units translation is requested because each hyperslab argument contains text characters and non-initial spaces. Third, NCO determines whether the wvl is dimensioned with a coordinate variable that has a units attribute. In this case, wvl itself is a coordinate variable. The value of its units attribute is meter. Thus wvl passes this test so UDUnits conversion is attempted. If the coordinate associated with the variable does not contain a units attribute, then NCO aborts. Fourth, NCO passes the specified and desired dimension strings (microns are specified by the user, meters are required by NCO) to the UDUnits library. Fifth, the UDUnits library that these dimension are commensurate and it returns the appropriate linear scaling factors to convert from microns to meters to NCO. If the units are incommensurate (i.e., not expressible in the same fundamental MKS units), or are not listed in the UDUnits database, then NCO aborts since it cannot determine the user’s intent. Finally, NCO uses the scaling information to convert the user-specified hyperslab limits into the same physical dimensions as those of the corresponding coordinate variable on disk. At this point, NCO can perform a coordinate hyperslab using the same algorithm as if the user had specified the hyperslab without requesting units conversion.

The translation and dimensional interpretation of time coordinates shows a more powerful, and probably more common, UDUnits application. In this example, the user prints all data between 4 PM and 7 PM on December 8, 1999, from a variable whose time dimension is hours since the year 1900:

```
% ncks -u -H -C -v time_udunits -d time_udunits,"1999-12-08 16:00:0.0","1999-12-08 19:00:0.0" in.nc
time_udunits[1]=876018 hours since 1900-01-01 00:00:0.0
```

Here, the user invokes the stride (see Section 3.16 [Stride], page 57) capability to obtain every other timeslice. This is possible because the UDUnits feature is additive, not exclusive—it works in conjunction with all other hyperslabbing (see Section 3.15 [Hyperslabs], page 55) options and in all operators which support hyperslabbing. The following example shows how one might average data in a time period spread across multiple input files

```
ncra -d time,"1939-09-09 12:00:0.0","1945-05-08 00:00:0.0" \n   in1.nc in2.nc in3.nc out.nc
```

Note that there is no excess whitespace before or after the individual elements of the ‘-d’ argument. This is important since, as far as the shell knows, ‘-d’ takes only one command-line argument. Parsing this argument into its component \texttt{dim,[min],[max],[stride]} elements (see Section 3.15 [Hyperslabs], page 55) is the job of NCO. When unquoted whitespace is present between these elements, the shell passes NCO argument fragments which will not parse as intended.

NCO implemented support for the UDUnits2 library with version 3.9.2 (August, 2007). The UDUnits2 package supports non-ASCII characters and logarithmic units. We are interested in user-feedback on these features.

One aspect that deserves mention is that UDUnits, and thus NCO, supports run-time definition of the location of the relevant UDUnits databases. With UDUnits version 1, users may specify the directory which contains the UDUnits database, udunits.dat, via the \texttt{UDUNITS\_PATH} environment variable. With UDUnits version 2, users may specify the
UDUnits database file itself, \texttt{udunits2.xml}, via the \texttt{UDUNITSD2.XML_PATH} environment variable.

```bash
# UDUnits1
export UDUNITSD2_PATH='/unusual/location/share/udunits'

# UDUnits2
export UDUNITSD2.XML_PATH='/unusual/location/share/udunits/udunits2.xml'
```

This run-time flexibility can enable the full functionality of pre-built binaries on machines with libraries in different locations.

The \texttt{UDUnits} package documentation describes the supported formats of time dimensions. Among the metadata conventions that adhere to these formats are the Climate and Forecast (CF) Conventions and the Cooperative Ocean/Atmosphere Research Data Service (COARDS) Conventions. The following \texttt{-d arguments} extract the same data using commonly encountered time dimension formats:

- \texttt{-d time,‘1918-11-11 00:00:0.0’}, ‘1939-09-09 00:00:0.0’
- \texttt{-d time,‘1918-11-11 00:00:0.0’}, ‘1939-09-09 00:00:0.0’
- \texttt{-d time,‘1918-11-11T00:00:0.0Z’}, ‘1939-09-09T00:00:0.0Z’
- \texttt{-d time,‘1918-11-11’, ‘1939-09-09’}
- \texttt{-d time,‘1918-11-11’, ‘1939-9-9’}

All of these formats include at least one dash - in a non-leading character position (a dash in a leading character position is a negative sign). NCO assumes that a space, colon, or non-leading dash in a limit string indicates that a UDUnits units conversion is requested. Some date formats like YYYYMMDD that are valid in UDUnits are ambiguous to NCO because it cannot distinguish a purely numerical date (i.e., no dashes or text characters in it) from a coordinate or index value:

- \texttt{-d time,1918-11-11} # Interpreted as the date November 11, 1918
- \texttt{-d time,19181111} # Interpreted as time-dimension index 19181111
- \texttt{-d time,19181111.} # Interpreted as time-coordinate value 19181111.0

Hence, use the YYYY-MM-DD format rather than YYYYMMDD for dates.

As of version 4.0.0 (January, 2010), NCO supports some calendar attributes specified by the CF conventions.

**Supported types:**

- "365\_day"/"noleap", "360\_day", "gregorian", "standard"

**Unsupported types:**

- "366\_day"/"all\_leap","proleptic\_gregorian","julian","none"

Unsupported types default to mixed Gregorian/Julian as defined by UDUnits.

An Example: Consider the following netCDF variable

```c
variables:
  double lon_cal(lon_cal);
  lon_cal:long_name = "lon_cal";
  lon_cal:units = "days since 1964-2-28 0:0:0";
```
lon_cal:calendar = "365_day" ;
data:
    lon_cal = 1,2,3,4,5,6,7,8,9,10;

'ncks -v lon_cal -d lon_cal,'1964-3-1 0:00:0.0','1964-3-4 00:00:0.0' results in lon_cal=1,2,3,4.

netCDF variables should always be stored with MKS (i.e., God’s) units, so that application programs may assume MKS dimensions apply to all input variables. The UDUnits feature is intended to alleviate some of the NCO user’s pain when handling MKS units. It connects users who think in human-friendly units (e.g., miles, millibars, days) to extract data which are always stored in God’s units, MKS (e.g., meters, Pascals, seconds). The feature is not intended to encourage writers to store data in esoteric units (e.g., furlongs, pounds per square inch, fortights).

3.23 Rebasing Time Coordinate

<table>
<thead>
<tr>
<th>Availability: ncra, nrcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: None</td>
</tr>
</tbody>
</table>

Time rebasing is invoked when numerous files share a common record coordinate, and the record coordinate units change among input files. The rebasing is performed automatically if and only if UDUnits is installed. Usually rebasing occurs when the record coordinate is a time-based variable, and times are recorded in units of a time-since-basetime, and the basetime changes from file to file. Since the output file can have only one unit (i.e., one basetime) for the record coordinate, NCO, in such cases, chooses the units of the first input file to be the units of the output file. It is necessary to “rebase” all the input record variables to this output time unit in order for the output file to have the correct values.

For example suppose the time coordinate is in hours and each day in January is stored in its own daily file. Each daily file records the temperature variable tpt(time) with an (unadjusted) time coordinate value between 0–23 hours, and uses the units attribute to advance the base time:

```
file01.nc time:units="hours since 1990-1-1"
file02.nc time:units="hours since 1990-1-2"
...
file31.nc time:units="hours since 1990-1-31"
// Mean noontime temperature in January
ncra -v tpt -d time,"1990-1-1 12:00:00","1990-1-31 23:59:59",24 \file??.nc noon.nc
```

```
// Concatenate day2 noon through day3 noon records
nrcat -v tpt -d time,"1990-1-2 12:00:00","1990-1-3 11:59:59" \file01.nc file02.nc file03.nc noon.nc
```

```
// Results: time is "re-based" to the time units in "file01.nc"
time=36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 
```
51, 52, 53, 54, 55, 56, 57, 58, 59 ;

// If we repeat the above command but with only two input files...
ncrcat -v tpt -d time,"1990-1-2 12:00:00","1990-1-3 11:59:59" \
   file02.nc file03 noon.nc

// ...then output time coordinate is based on time units in "file02.nc"
time = 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, \
   26, 27, 28, 29, 30, 31, 32, 33, 34, 35 ;

As of NCO version 4.2.1 (August, 2012), NCO automatically rebases not only the record coordinate (time, here) but also any bounds associated with the record coordinate (e.g., time_bnds) (see Section 3.38 [CF Conventions], page 109).

### 3.24 Multiple Record Dimensions

<table>
<thead>
<tr>
<th>Availability: ncecat, ncpdq</th>
<th>Short options: None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long options: <code>--mrd</code></td>
<td></td>
</tr>
</tbody>
</table>

The netCDF3 file format allows only one record dimension, and that dimension must be the first dimension (i.e., the least rapidly varying dimension) of any variable in which it appears. This imposes certain rules on how operators must perform operations that alter the ordering of dimensions or the number of record variables. The netCDF4 file format has no such restrictions. Files and variables may have any number of record dimensions in any order. This additional flexibility of netCDF4 can only be realized by selectively abandoning the constraints that would make operations behave completely consistently between netCDF3 and netCDF4 files.

NCO chooses, by default, to impose netCDF3-based constraints on netCDF4 files. This reduces the number of unanticipated consequences and keeps the operators functioning in a familiar way. Put another way, NCO limits production of additional record dimensions so processing netCDF4 files leads to the same results as processing netCDF4 files. Users can override this default with the `--mrd` (or `--multiple_record_dimension`) switch, which enables netCDF4 variables to accumulate additional record dimensions.

How can additional record dimensions be produced? Most commonly ncecat (in record-aggregate mode) defines a new leading record dimension. In netCDF4 files this becomes an additional record dimension unless the original record dimension is changed to a fixed dimension (as must be done in netCDF3 files). Also when ncpdq reorders dimensions it can preserve the “record” property of record variables. ncpdq tries to define as a record dimension whichever dimension ends up first in a record variable, and, in netCDF4 files, this becomes an additional record dimension unless the original record dimension is changed to a fixed dimension (as must be done in netCDF3 files). It it easier if ncpdq and ncecat do not increase the number of record dimensions in a variable so that is the default. Use `--mrd` to override this.
3.25 Missing values

The phrase missing data refers to data points that are missing, invalid, or for any reason not intended to be arithmetically processed in the same fashion as valid data. The NCO arithmetic operators attempt to handle missing data in an intelligent fashion. There are four steps in the NCO treatment of missing data:

1. Identifying variables that may contain missing data.

   NCO follows the convention that missing data should be stored with the _FillValue specified in the variable’s _FillValue attributes. The only way NCO recognizes that a variable may contain missing data is if the variable has a _FillValue attribute. In this case, any elements of the variable which are numerically equal to the _FillValue are treated as missing data.

   NCO adopted the behavior that the default attribute name, if any, assumed to specify the value of data to ignore is _FillValue with version 3.9.2 (August, 2007). Prior to that, the missing_value attribute, if any, was assumed to specify the value of data to ignore. Supporting both of these attributes simultaneously is not practical. Hence the behavior NCO once applied to missing_value it now applies to any _FillValue. NCO now treats any missing_value as normal data.

   It has been and remains most advisable to create both _FillValue and missing_value attributes with identical values in datasets. Many legacy datasets contain only missing_value attributes. NCO can help migrating datasets between these conventions. One may use ncrename (see Section 4.11 [ncrename netCDF Renamer], page 214) to rename all missing_value attributes to _FillValue:

   
   ncrename -a .missing_value,_FillValue inout.nc
  
   Alternatively, one may use ncatted (see Section 4.2 [ncatted netCDF Attribute Editor], page 168) to add a _FillValue attribute to all variables

   
   ncatted -O -a _FillValue,,o,f,1.0e36 inout.nc
   
   2. Converting the _FillValue to the type of the variable, if necessary.

   Consider a variable var of type var_type with a _FillValue attribute of type att_type containing the value _FillValue. As a guideline, the type of the _FillValue attribute should be the same as the type of the variable it is attached to. If var_type equals att_type then NCO straightforwardly compares each value of var to _FillValue to determine which elements of var are to be treated as missing data. If not, then NCO converts _FillValue from att_type to var_type by using the implicit conversion rules of C, or, if att_type is NC_CHAR, by typecasting the results of the C function strtd(_FillValue). You may use the NCO operator ncatted to change the _FillValue at-

---

18 The old functionality, i.e., where the ignored values are indicated by missing_value not _FillValue, may still be selected at NCO build time by compiling NCO with the token definition CPPFLAGS='''-UNCO_USE_FILL_VALUE'''.

19 For example, the DOE ARM program often uses att_type = NC_CHAR and _FillValue = '-99999.'
tribute and all data whose data is `_FillValue` to a new value (see Section 4.2 [netCDF Attribute Editor], page 168).

3. Identifying missing data during arithmetic operations.

When an NCO arithmetic operator processes a variable `var` with a `_FillValue` attribute, it compares each value of `var` to `_FillValue` before performing an operation. Note the `_FillValue` comparison imposes a performance penalty on the operator. Arithmetic processing of variables which contain the `_FillValue` attribute always incurs this penalty, even when none of the data are missing. Conversely, arithmetic processing of variables which do not contain the `_FillValue` attribute never incurs this penalty. In other words, do not attach a `_FillValue` attribute to a variable which does not contain missing data. This exhortation can usually be obeyed for model generated data, but it may be harder to know in advance whether all observational data will be valid or not.

4. Treatment of any data identified as missing in arithmetic operators.

NCO averagers (ncra, nces, ncwa) do not count any element with the value `_FillValue` towards the average. ncbo and ncflint define a `_FillValue` result when either of the input values is a `_FillValue`. Sometimes the `_FillValue` may change from file to file in a multi-file operator, e.g., ncra. NCO is written to account for this (it always compares a variable to the `_FillValue` assigned to that variable in the current file). Suffice it to say that, in all known cases, NCO does “the right thing”.

It is impossible to determine and store the correct result of a binary operation in a single variable. One such corner case occurs when both operands have differing `_FillValue` attributes, i.e., attributes with different numerical values. Since the output (result) of the operation can only have one `_FillValue`, some information may be lost. In this case, NCO always defines the output variable to have the same `_FillValue` as the first input variable. Prior to performing the arithmetic operation, all values of the second operand equal to the second `_FillValue` are replaced with the first `_FillValue`. Then the arithmetic operation proceeds as normal, comparing each element of each operand to a single `_FillValue`. Comparing each element to two distinct `_FillValue's` would be much slower and would be no likelier to yield a more satisfactory answer. In practice, judicious choice of `_FillValue` values prevents any important information from being lost.

3.26 Chunking

| Availability: ncap2, ncbo, nces, ncecat, ncflint, ncks, ncpdq, ncra, ncrcat, ncwa |
| Short options: none |
| Long options: `--cnk_byt cnk_sz`, `--chunk_byte cnk_sz` |
| `--cnk_dmn dmn_nm,cnk_sz`, `--chunk_dimension dmn_nm,cnk_sz` |
| `--cnk_map cnk_map`, `--chunk_map cnk_map` |
| `--cnk_min var_sz`, `--chunk_min var_sz` |
| `--cnk_plc cnk_plc`, `--chunk_policy cnk_plc` |
| `--cnk_scl cnk_sz`, `--chunk_scalar cnk_sz` |
All netCDF4-enabled NCO operators that define variables support a plethora of chunk-size options. Chunking can significantly accelerate or degrade read/write access to large datasets. Dataset chunking issues are described by THG and Unidata here, here, and here. NCO authors are working on generalized algorithms and applications of chunking strategies (stay tuned for more in 2015).

The NCO chunking implementation is designed to be flexible. Users control four aspects of the chunking implementation. These are the chunking policy, chunking map, chunksize, and minimum chunksize. The chunking policy determines which variables to chunk, and the chunking map determines how (with what exact sizes) to chunk those variables. These are high-level mechanisms that apply to an entire file and all variables and dimensions. The chunksize option allows per-dimension specification of sizes that will override the selected (or default) chunking map.

The var_siz parameter is the minimum size in bytes (not elements) of variables to chunk. This threshold is intended to restrict use of chunking to variables for which it is efficient. By default this minimum variable size for chunking is twice the system blocksize (when available) and is 8192 bytes otherwise. Users may set this to any value with the ‘--cnk_min’ switch.

The chunking implementation is similar to a hybrid of the ncpdq packing policies (see Section 4.8 [ncpdq netCDF Permute Dimensions Quickly], page 202) and hyperslab specifications (see Section 3.15 [Hyperslabs], page 55). Each aspect is intended to have a sensible default, so that many users only need to set one switch to obtain sensible chunking. Power users can tune chunking with the three switches in tandem to obtain optimal performance.

By default, NCO preserves the chunking characteristics of the input file in the output file\(^\text{20}\). In other words, preserving chunking requires no switches or user intervention.

Users specify the desired chunking policy with the ‘-P’ switch (or its long option equivalents, ‘--cnk_plc’ and ‘--chunk_policy’) and its cnk_plc argument. As of August, 2014, six chunking policies are implemented:

**Chunk All Variables**
- Definition: Chunk all variables possible. For obvious reasons, scalar variables cannot be chunked.
- Alternate invocation: nchunk
- cnk_plc key values: ‘all’, ‘cnk_all’, ‘plc_all’
- Mnemonic: All

**Chunk Variables with at least Two Dimensions [default]**
- Definition: Chunk all variables possible with at least two dimensions
- Alternate invocation: none
- cnk_plc key values: ‘g2d’, ‘cnk_g2d’, ‘plc_g2d’

---

\(^{20}\) This behavior became the default in November 2014 with NCO version 4.4.7. Prior versions would always use netCDF default chunking in the output file when no NCO chunking switches were activated, regardless of the chunking in the input file.
Mnemonic: Greater than or equal to 2 Dimensions

Chunk Variables with at least Three Dimensions
Definition: Chunk all variables possible with at least three dimensions
Alternate invocation: none
cnk_plc key values: ‘g3d’, ‘cnk_g3d’, ‘plc_g3d’
Mnemonic: Greater than or equal to 3 Dimensions

Chunk One-Dimensional Record Variables
Definition: Chunk all 1-D record variables
Alternate invocation: none
Any specified (with ‘--cnk_dmn’) record dimension chunksizes will be applied only to 1-D record variables (and to no other variables). Other dimensions may be chunked with their own ‘--cnk_dmn’ options that will apply to all variables.
cnk_plc key values: ‘rid’, ‘cnk_rid’, ‘plc_rid’
Mnemonic: Record 1-D variables

Chunk Variables Containing Explicitly Chunked Dimensions
Definition: Chunk all variables possible that contain at least one dimension whose chunksize was explicitly set with the ‘--cnk_dmn’ option. Alternate invocation: none
cnk_plc key values: ‘xpl’, ‘cnk_xpl’, ‘plc_xpl’
Mnemonic: EXPLicitly specified dimensions

Chunk Variables that are already Chunked
Definition: Chunk only variables that are already chunked in the input file. When used in conjunction with ‘cnk_map=xst’ this option preserves and copies the chunking parameters from the input to the output file. Alternate invocation: none
cnk_plc key values: ‘xst’, ‘cnk_xst’, ‘plc_xst’
Mnemonic: EXiSTing chunked variables

Chunk Variables with NCO recommendations
Definition: Chunk all variables according to NCO best practices. This is a virtual option that ensures the chunking policy is (in the subjective opinion of the authors) the best policy for typical usage. As of NCO version 4.4.8 (February, 2015), this virtual policy implements ‘map_rew’ for 3-D variables and ‘map_lfp’ for all other variables.
Alternate invocation: none
cnk_plc key values: ‘nco’, ‘cnk_nco’, ‘plc_nco’
Mnemonic: NetCDFOperator

Unchunking
Definition: Unchunk all variables possible. The HDF5 storage layer requires that record variables (i.e., variables that contain at least one record dimension)
must be chunked. Also variables that are compressed or use checksums must be chunked. Such variables cannot be unchunked.
Alternate invocation: `ncunchunk`

chunk_plc key values: ‘uck’, ‘cnk_uck’, ‘plc_uck’, ‘unchunk’
Mnemonic: $UnChunK$

Equivalent key values are fully interchangeable. Multiple equivalent options are provided to satisfy disparate needs and tastes of NCO users working with scripts and from the command line.

The chunking algorithms must know the chunksizes of each dimension of each variable to be chunked. The correspondence between the input variable shape and the chunksizes is called the chunking map. The user specifies the desired chunking map with the ‘-M’ switch (or its long option equivalents, ‘--cnk_map’ and ‘--chunk_map’) and its `cnk_map` argument. Eight chunking maps are currently implemented:

**Chunksize Equals Dimension Size** [default]
Definition: Chunksize defaults to dimension size. Explicitly specify chunksizes for particular dimensions with ‘--cnk_dmn’ option.

chunk_map key values: ‘dmn’, ‘cnk_dmn’, ‘map_dmn’
Mnemonic: $DiMeNsion$

**Chunksize Equals Dimension Size except Record Dimension**
Definition: Chunksize equals dimension size except record dimension has size one. Explicitly specify chunksizes for particular dimensions with ‘--cnk_dmn’ option.

chunk_map key values: ‘rd1’, ‘cnk_rd1’, ‘map_rd1’
Mnemonic: $RecData$ision size 1

**Chunksize Equals Scalar Size Specified**
Definition: Chunksize for all dimensions is set with the ‘--cnk_scl’ option.

chunk_map key values: ‘xpl’, ‘cnk_xpl’, ‘map_xpl’
Mnemonic: $EXPlici$ty specified dimensions

**Chunksize Product Matches Scalar Size Specified**
Definition: The product of the chunksizes for each variable matches (approximately equals) the size specified with the ‘--cnk_scl’ option. A dimension of size one is said to be degenerate. For a variable of rank $R$ (i.e., with $R$ non-degenerate dimensions), the chunksize in each non-degenerate dimension is the $R$th root of `cnk_scl`.

chunk_map key values: ‘prd’, ‘cnk_prd’, ‘map_prd’
Mnemonic: $PRoduct$
Chunksize Lefter Product Matches Scalar Size Specified

Definition: The product of the chunksizes for each variable (approximately) equals the size specified with the ‘--cnk_scl’ option. This is accomplished by using dimension sizes as chunksizes for the rightmost (most rapidly varying) dimensions, and then “flexing” the chunksize of the leftmost (least rapidly varying) dimensions such that the product of all chunksizes matches the specified size. All dimensions to the left of and including the first record dimension define the left-hand side. This map was first proposed by Chris Barker.

\texttt{cnk\_map} key values: ‘lfp’, ‘cnk\_lfp’, ‘map\_lfp’

Mnemonic: \textit{L}eft \textit{P}roduct

---

Chunksize Equals Existing Chunksize

Definition: Chunksizes are copied from the input to the output file for every variable that is chunked in the input file. Variables not chunked in the input file will be chunked with default mappings.

\texttt{cnk\_map} key values: ‘xst’, ‘cnk\_xst’, ‘map\_xst’

Mnemonic: \textit{EXiST}

---

Chunksize Balances 1D and (N-1)-D Access to N-D Variable

Definition: Chunksizes are chosen so that 1-D and ((N-1))-D hyperslabs of 3-D variables (e.g., point-timeseries or latitude/longitude surfaces of 3-D fields) both require approximately the number of chunks. Hence their access time should be balanced. Russ Rew explains the motivation and derivation for this strategy \textit{here}.

\texttt{cnk\_map} key values: ‘rew’, ‘cnk\_rew’, ‘map\_rew’

Mnemonic: Russ \textit{REW}

---

Chunksizes use netCDF4 defaults

Definition: Chunksizes are determined by the underlying netCDF library. All variables selected by the current chunking policy have their chunksizes determined by netCDF library defaults. The default algorithm netCDF uses to determine chunksizes has changed through the years, and thus depends on the netCDF library version. This map can be used to reset (portions of) previously chunked files to default chunking values.

\texttt{cnk\_map} key values: ‘nc4’, ‘cnk\_nc4’, ‘map\_nc4’

Mnemonic: \textit{NetCDF4}

---

Chunksizes use NCO recommendations

Definition: Chunksizes are determined by the currently recommended NCO map. This is a virtual option that ensures the chunking map is (in the subjective opinion of the authors) the best map for typical usage. As of NCO version 4.4.8 (February, 2015), this virtual map calls ‘map\_rew’ for 3-D variables and ‘map\_lfp’ for all others.

\texttt{cnk\_map} key values: ‘nco’, ‘cnk\_nco’, ‘map\_nco’
Mnemonic: NetCDFOperator

It is possible to combine the above chunking map algorithms with user-specified per-dimension (though not per-variable) chunksizes that override specific chunksizes determined by the maps above. The user specifies the per-dimension chunksizes with the (equivalent) long options ‘--cnk_dmn’ or ‘--chunk_dimension’). The option takes two comma-separated arguments, dmn_um,cnk_sz, which are the dimension name and its chunksize, respectively. The ‘--cnk_dmn’ option may be used as many times as necessary.

The default behavior of chunking depends on several factors. As mentioned above, when no chunking options are explicitly specified by the user, then NCO preserves the chunking characteristics of the input file in the output file. This is equivalent to specifying both cnk_plc and cnk_map as “existing”, i.e., ‘--cnk_plc=xst --cnk_map=xst’. If output netCDF4 files are chunked with the default behavior of the netCDF4 library.

When any chunking parameter except ‘cnk_plc’ or ‘cnk_map’ is specified (such as ‘cnk_dmn’ or ‘cnk_sz_scl’), then the “existing” policy and map are retained and the output chunksizes are modified where necessary in accord with the user-specified parameter. When ‘cnk_map’ is specified and ‘cnk_plc’ is not, then NCO picks (what it thinks is) the optimal chunking policy. This has always been policy ‘map_g2d’. When ‘cnk_plc’ is specified and ‘cnk_map’ is not, then NCO picks (what it thinks is) the optimal chunking map. This has always been map ‘map_rd1’.

To start afresh and return to netCDF4 chunking defaults, select ‘cnk_map=nc4’.

```
# Simple chunking and unchunking
ncks -O -4 --cnk_plc=all in.nc out.nc # Chunk in.nc
ncks -O -4 --cnk_plc=unchunk in.nc out.nc # Unchunk in.nc

# Chunk data then unchunk it, printing informative metadata
ncks -O -4 -D 4 --cnk_plc=all ~/nco/data/in.nc ~/foo.nc
ncks -O -4 -D 4 --cnk_plc=uck ~/foo.nc ~/foo.nc

# Set total chunksize to 8192 B
ncks -O -4 -D 4 --cnk_plc=all --cnk_byt=8192 ~/nco/data/in.nc ~/foo.nc

# More complex chunking procedures, with informative metadata
ncks -O -4 -D 4 --cnk_scl=8 ~/nco/data/in.nc ~/foo.nc
ncks -O -4 -D 4 --cnk_scl=8 dstmch90_clm.nc ~/foo.nc
ncks -O -4 -D 4 --cnk_dmn lat,64 --cnk_dmn lon,128 dstmch90_clm.nc
   ~/foo.nc
ncks -O -4 -D 4 --cnk_plc=uck ~/foo.nc ~/foo.nc
ncks -O -4 -D 4 --cnk_plc=g2d --cnk_map=rd1 --cnk_dmn lat,32
   --cnk_dmn lon,128 dstmch90_clm_0112.nc ~/foo.nc

# Chunking works with all operators...
ncap2 -O -4 -D 4 --cnk_scl=8 -S ~/nco/data/ncap2_tst.nco
   ~/nco/data/in.nc ~/foo.nc
```
Chunking policy ‘r1d’ changes the chunksize of 1-D record variables (and no other variables) to that specified (with ‘--cnk_dmn’) chunksize. Any specified record dimension chunk-sizes will be applied to 1-D record variables only. Other dimensions may be chunked with their own ‘--cnk_dmn’ options that will apply to all variables. For example,

```
ncks --cnk_plc=r1d --cnk_dmn=time,1000. in.nc out.nc
```

This sets time chunks to 1000 only in 1-D record variables. Without the ‘r1d’ policy, time chunks would change in all variables.

It is appropriate to conclude by informing users about an aspect of chunking that may not be expected. Three types of variables are always chunked: Record variables, Deflated (compressed) variables, and Checksummed variables. Hence all variables that contain a record dimension are also chunked (since data must be chunked in all dimensions, not just one). Unless otherwise specified by the user, the other (fixed, non-record) dimensions of record variables are assigned default chunk sizes. The HDF5 layer does all this automatically to optimize the on-disk variable/file storage geometry of record variables. Do not be surprised to learn that files created without any explicit instructions to activate chunking nevertheless contain chunked variables.

### 3.27 Compression

| Availability: ncbo, ncecat, nces, ncflint, ncks, ncpdq, ncra, ncrca, ncra, ncwa |
| Short options: None |
| Long options: ‘--ppc var1[,var2[,...]]=prc’, ‘--precision_preserving_compression var1[,var2[,...]]=prc’, ‘--quantize var1[,var2[,...]]=prc’ |

NCO implements or accesses four different compression algorithms, the standard lossless DEFLATE algorithm and three lossy compression algorithms. All four algorithms reduce the on-disk size of a dataset while sacrificing no (lossless) or a tolerable amount (lossy) of precision. First, NCO can access the lossless DEFLATE algorithm, a combination of Lempel-Ziv encoding and Huffman coding, algorithm on any netCDF4 dataset (see Section 3.28 [Deflation], page 88). Because it is lossless, this algorithm re-inflates deflated data to their full original precision. This algorithm is accessed via the HDF5 library layer (which itself calls the zlib library also used by gzip), and is unavailable with netCDF3.
The three lossy compression algorithms are Packing (see Section 3.32 [Packed data], page 92), and two precision-preserving algorithms. Packing quantizes data of a higher precision type into a lower precision type (often NC_SHORT) that thus stores a fewer (though constant) number of bytes per value. Packed data almost unpacks into a (much) smaller dynamic range than the floating-point data can represent. The type-conversion and reduced dynamic range of the data allows packing to eliminate bits typically used to store an exponent, thus improving its packing efficiency. Packed data also can also be deflated for additional space savings.

A limitation of packing is that unpacking data stored as integers into the linear range defined by scale_factor and add_offset rapidly loses precision outside of a narrow range of floating point values. Variables packed as NC_SHORT, for example, can represent only about 64000 discrete values in the range \(-32768 \times \text{scale} \text{factor} + \text{add.offset}\) to \(32767 \times \text{scale.factor} + \text{add.offset}\). The precision of packed data equals the value of scale_factor, and scale_factor is usually chosen to span the range of valid data, not to represent the intrinsic precision of the variable. In other words, the precision of packed data cannot be specified in advance because it depends on the range of values to quantize.

NCO implemented the final two lossy compression algorithms in version 4.4.8 (February, 2015). These are both Precision-Preserving Compression (PPC) algorithms and since standard terminology for precision is remarkably imprecise, so is our nomenclature. The operational definition of “significant digit” in our precision preserving algorithms is that the exact value, before rounding or quantization, is within one-half the value of the decimal place occupied by the Least Significant Digit (LSD) of the rounded value. For example, the value \(\pi = 3.14\) correctly represents the exact mathematical constant \(\pi\) to three significant digits because the LSD of the rounded value (i.e., 4) is in the one-hundredths digit place, and the difference between the exact value and the rounded value is less than one-half of one one-hundredth, i.e., \((3.14159265358979323844 - 3.14) < 0.005\).

One PPC algorithm preserves the specified total Number of Significant Digits (NSD) of the value. For example there is only one significant digit in the weight of most “eight-hundred pound gorillas” that you will encounter, i.e., so nsd = 1. This is the most straightforward measure of precision, and thus NSD is the default PPC algorithm.

The other PPC algorithm preserves the number of Decimal Significant Digits (DSD), i.e., the number of significant digits following (positive, by convention) or preceding (negative) the decimal point. For example, “0.008” and “800” have, respectively, three and negative two digits digits following the decimal point, corresponding dsd = 3 and dsd = -2.

The only justifiable NSD for a given value depends on intrinsic accuracy and error characteristics of the model or measurements, and not on the units with which the value is stored. The appropriate DSD for a given value depends on these intrinsic characteristics and, in addition, the units of storage. This is the fundamental difference between the NSD and DSD approaches. The eight-hundred pound gorilla always has nsd = 1 regardless of whether the value is store in pounds or in some other unit. DSD corresponding to this weight is dsd = -2 if the value is store in pounds, dsd = 4 if stored in megapounds.

Users may wish to express the precision to be preserved as either NSD or DSD. Invoke PPC with the long option ‘--ppc var=prc’, or give the same arguments to the synonyms ‘--precision_preserving_compression’, or to ‘--quantize’. Here var is the variable
to quantize, and prc is the precision. The default algorithm assumes prc specifies NSD precision, e.g., ‘T=2’ means nsd = 2. Prepend prc with a decimal point to specify DSD precision, e.g., ‘T=.2’ means dsd = 2. NSD precision must be specified as a positive integer. DSD precision may be a positive or negative integer; and is specified as the negative base 10 logarithm of the desired precision, in accord with common usage. For example, specifying ‘T=.3’ or ‘T=.-2’ tells the DSD algorithm to store only enough bits to preserve the value of T rounded to the nearest thousandth or hundred, respectively.

Setting var to default has the special meaning of applying the associated NSD or DSD algorithm to all floating point variables except coordinate variables. Variables not affected by default include integer and non-numeric atomic types, coordinates, and variables mentioned in the coordinates or bounds attribute of any variable. NCO applies PPC to coordinate variables only if those variables are explicitly specified (i.e., not with the ‘default=prc’ mechanism. NCO applies PPC to integer-type variables only if those variables are explicitly specified (i.e., not with the ‘default=prc’, and only if the DSD algorithm is invoked with a negative prc. To prevent PPC from applying to certain non-coordinate variables (e.g., gridcell_area or gaussian_weight), explicitly specify a precision exceeding 7 (for NC_FLOAT) or 15 (for NC_DOUBLE) for those variables. Since these are the maximum representable precisions in decimal digits, NCO turns-off PPC (i.e., does nothing) when more precision is requested.

The time penalty for compressing and uncompressing data varies according to the algorithm. The Number of Significant Digit (NSD) algorithm quantizes by bitmasking, and employs no floating point math. The Decimal Significant Digit (DSD) algorithm quantizes by rounding, which does require floating point math. Hence NSD is likely faster than DSD, though the difference has not been measured. NSD creates a bitmask to alter the significand of IEEE 754 floating point data. The bitmask is one for all bits to be retained and zero or one for all bits to be ignored. The algorithm assumes that the number of binary digits (i.e., bits) necessary to represent a single base-10 digit is \( \ln(10)/\ln(2) = 3.32 \). The exact numbers of bits Nbit retained for single and double precision values are \( \text{ceil}(3.32 \times \text{nsd}) + 1 \) and \( \text{ceil}(3.32 \times \text{nsd}) + 2 \), respectively. Once these reach 23 and 53, respectively, bitmasking is completely ineffective. This occurs at nsd = 6.3 and 15.4, respectively.

The DSD algorithm, by contrast, uses rounding to remove undesired precision. The rounding zeroes the greatest number of significand bits consistent with the desired precision.

To demonstrate the change in IEEE representation caused by PPC rounding algorithms, consider again the case of pi, represented as an NC_FLOAT. The IEEE 754 single precision representations of the exact value (3.141592...), the value with only three significant digits treated as exact (3.140000...), and the value as stored (3.140625) after PPC-rounding with either the NSD (prc = 3) or DSD (prc = 2) algorithm are, respectively,

<table>
<thead>
<tr>
<th>Exponent Fraction (Significand)</th>
<th>Decimal</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 100000001 0010010000111111101111</td>
<td># 3.14159265 Exact</td>
<td></td>
</tr>
</tbody>
</table>

\[21\] Rounding is performed by the internal math library rint() family of functions that were standardized in C99. The exact algorithm employed is \( \text{val} := \text{rint}(\text{scale} \times \text{val})/\text{scale} \) where \( \text{scale} \) is the nearest power of 2 that exceeds \( 10 \times \text{prc} \), and the inverse of \( \text{scale} \) is used when \( \text{prc} < 0 \). For \( \text{prc} = 3 \) or \( \text{prc} = -2 \), for example, we have \( \text{scale} = 1024 \) and \( \text{scale} = 1/128 \).
The string of trailing zero-bits in the rounded values facilitates byte-stream compression. Note that the NSD and DSD algorithms do not always produce results that are bit-for-bit identical, although they do in this particular case.

Reducing the preserved precision of NSD-rounding produces increasingly long strings of identical-bits amenable to compression:

<table>
<thead>
<tr>
<th>S</th>
<th>Exponent</th>
<th>Fraction (Significand)</th>
<th>Decimal</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100000001</td>
<td>00100100000000000000000000</td>
<td># 3.14062500</td>
<td>NSD = 3</td>
</tr>
<tr>
<td>0</td>
<td>100000001</td>
<td>00100100000000000000000000</td>
<td># 3.14062500</td>
<td>DSD = 2</td>
</tr>
</tbody>
</table>

The consumption of about 3 bits per digit of base-10 precision is evident, as is the coincidence of a quantized value that greatly exceeds the mandated precision for NSD = 2. Although the NSD algorithm generally masks some bits for all nsd <= 7 (for NC_FLOAT), compression algorithms like DEFLATE may need byte-size-or-greater (i.e., at least eight-bit) bit patterns before their algorithms can take advantage of of encoding such patterns for compression. Do not expect significantly enhanced compression from nsd > 5 (for NC_FLOAT) or nsd > 14 (for NC_DOUBLE). Clearly values stored as NC_DOUBLE (i.e., eight-bytes) are susceptible to much greater compression than NC_FLOAT for a given precision because their significands explicitly contain 53 bits rather than 23 bits.

Maintaining non-biased statistical properties during lossy compression requires special attention. The DSD algorithm uses rint(), which rounds toward the nearest even integer. Thus DSD has no systematic bias. However, the NSD algorithm uses a bitmask technique susceptible to statistical bias. Zeroing all non-significant bits is guaranteed to produce numbers quantized to the specified tolerance, i.e., half of the decimal value of the position occupied by the LSD. However, always zeroing the non-significant bits results in quantized numbers that never exceed the exact number. This would produce a negative bias in statistical quantities (e.g., the average) subsequently derived from the quantized numbers. To avoid this bias, NSD rounds non-significant bits down (to zero) or up (to one) in an alternating fashion when processing array data. In general, the first element is rounded down, the second up, and so on. This results in a mean bias quite close to zero. The only exception is that the floating point value of zero is never quantized upwards. For simplicity, NSD always rounds scalars downwards.

Although NSD or DSD are different algorithms under the hood, they both replace the (unwanted) least significant bits of the IEEE significand with a string of consecutive zeroes. Byte-stream compression techniques, such as the gzip DEFLATE algorithm compression available in HDF5, always compress zero-strings more efficiently than random digits. The
net result is netCDF files that utilize compression can be significantly reduced in size. This feature only works when the data are compressed, either internally (by netCDF) or externally (by another user-supplied mechanism). It is most straightforward to compress data internally using the built-in compression and decompression supported by netCDF4. For convenience, NCO automatically activates file-wide Lempel-Ziv deflation (see Section 3.28 [Deflation], page 88) level one (i.e., ‘-L 1’) when PPC is invoked on any variable in a netCDF4 output file. This makes PPC easier to use effectively, since the user need not explicitly specify deflation. Any explicitly specified deflation (including no deflation, ‘-L 0’) will override the PPC deflation default. If the output file is a netCDF3 format, NCO will emit a message suggesting internal netCDF4 or external netCDF3 compression. netCDF3 files compressed by an external utility such as gzip accrete approximately the same benefits (shrinkage) as netCDF4, although with netCDF3 the user or provider must uncompress (e.g., gunzip) the file before accessing the data. There is no benefit to rounding numbers and storing them in netCDF3 files unless such custom compression/decompression is employed. Without that, one may as well maintain the undesired precision.

The user accesses PPC through a single switch, ‘--ppc’, repeated as many times as necessary. To apply the NSD algorithm to variable $u$ use, e.g.,

\texttt{ncks -7 --ppc u=2 in.nc out.nc}

The output file will preserve only two significant digits of $u$. The options ‘-4’ or ‘-7’ ensure a netCDF4-format output (regardless of the input file format) to support internal compression. It is recommended though not required to write netCDF4 files after PPC. For clarity the ‘-4/-7’ switches are omitted in subsequent examples. NCO attaches attributes that indicate the algorithm used and degree of precision retained for each variable affected by PPC. The NSD and DSD algorithms store the attributes \texttt{number_of_significant_digits} and \texttt{least_significant_digit}\textsuperscript{22}, respectively.

It is safe to attempt PPC on input that has already been rounded. Variables can be made rounder, not sharper, i.e., variables cannot be “un-rounded”. Thus PPC attempted on an input variable with an existing PPC attribute proceeds only if the new rounding level exceeds the old, otherwise no new rounding occurs (i.e., a “no-op”), and the original PPC attribute is retained rather than replaced with the newer value of $prc$.

To request, say, five significant digits ($nsd = 5$) for all fields, except, say, wind speeds which are only known to integer values ($dsd = 0$) in the supplied units, requires ‘--ppc’ twice:

\texttt{ncks -4 --ppc default=5 --ppc u,v=.0 in.nc out.nc}

To preserve five digits in all variables except coordinate variables and $u$ and $v$, use the ‘default’ option and separately specify the exceptions:

\texttt{ncks --ppc default=5 --ppc u,v=20 in.nc out.nc}

\textsuperscript{22} The \texttt{nc3tonc4} tool by Jeff Whitaker inspired NCO to implement PPC. NCO implements a different DSD algorithm than \texttt{nc3tonc4}, and produces slightly different (not bit-for-bit) though self-consistent and equivalent results. \texttt{nc3tonc4} records the precision of its DSD algorithm in the attribute \texttt{least_significant_digit} and NCO does the same for consistency.
The `--ppc` option may be specified any number of times to support varying precision types and levels, and each option may aggregate all the variables with the same precision:

```
ncks --ppc p,w,z=5 --ppc q,RH=4 --ppc T,u,v=3 in.nc out.nc
```

Any `var` argument may be a regular expression. This simplifies generating lists of related variables:

```
ncks --ppc Q.?=5 --ppc FS.?,FL.?=4 --ppc RH.=3 in.nc out.nc
```

Although PPC-rounding instantly reduces data precision, on-disk storage reduction only occurs once the data are compressed.

How can one be sure the lossy data are sufficiently precise? PPC preserves all significant digits of every value. The DSD algorithm uses floating point math to round each value optimally so that it has the maximum number of zeroed bits that preserve the specified precision. The NSD algorithm uses a theoretical approach (3.2 bits per base-10 digit), tuned and tested to ensure the worst case quantization error is less than half the value of the minimum increment in the least significant digit.

We define several metrics to quantify the quantization error. The mean error $\bar{\epsilon}$ and mean absolute error $\bar{\epsilon}^+$ incurred in quantizing a variable from its true values $x_i$ to quantized values $q_i$ are, respectively,

$$
\bar{\epsilon} = \frac{\sum_{i=1}^{N} \mu_i m_i w_i (x_i - q_i)}{\sum_{i=1}^{N} \mu_i m_i w_i}
$$

$$
\bar{\epsilon}^+ = \frac{\sum_{i=1}^{N} \mu_i m_i w_i |x_i - q_i|}{\sum_{i=1}^{N} \mu_i m_i w_i}
$$

where $\mu_i$ is 1 unless $x_i$ equals the missing value, $m_i$ is 1 unless $x_i$ is masked, and $w_i$ is the weight. The maximum and minimum errors $\epsilon_{\text{max}}$ and $\epsilon_{\text{min}}$ are both signed

$$
\epsilon_{\text{max}} = \max(x_i - q_i)
$$

$$
\epsilon_{\text{min}} = \min(x_i - q_i)
$$

while the maximum and minimum absolute errors $\epsilon_{\text{mabs}}^+$ are positive-definite.

$$
\epsilon_{\text{mabs}}^+ = \max|x_i - q_i| = \max(|\epsilon_{\text{max}}|, |\epsilon_{\text{min}}|)
$$

$$
\epsilon_{\text{mibs}}^+ = \min|x_i - q_i| = \min(|\epsilon_{\text{max}}|, |\epsilon_{\text{min}}|)
$$

Typically $\epsilon_{\text{mibs}}^+ = 0$ for quantization, since many exact values need no quantization. Bit-shifting zeros into the least significant bits (LSBs) always underestimates true values so that $\epsilon_{\text{max}} = 0$. Conversely, bit-shifting ones into the LSBs always overestimates true values so that $\epsilon_{\text{min}} = 0$. Our NSD algorithm is balanced because it alternates bit-shifting zeroes and ones. Balanced algorithms should yield $\epsilon_{\text{max}} \approx -\epsilon_{\text{min}}$, $\epsilon_{\text{mabs}}^+ \approx \epsilon_{\text{mibs}}^+$, and $\bar{\epsilon} \approx 0$.

The three most important error metrics for quantization are $\epsilon_{\text{mabs}}^+$, $\epsilon^+$, and $\bar{\epsilon}$. The upper bound (worst case) quantization performance is $\epsilon_{\text{mabs}}^+$. $\epsilon^+$ measures the absolute mean accuracy of quantization, and does not allow positive and negative offsets to compensate eachother and conceal poor performance. The difference between $\epsilon_{\text{mabs}}^+$ and $\epsilon^+$ indicates how much of an outlier the worst case is. The mean accuracy $\bar{\epsilon}$ indicates whether statistical properties of quantized numbers will accurately reflect the true values.

All three metrics are expressed in terms of the fraction of the ten’s place occupied by the LSD. If the LSD is the hundreds digit or the thousandths digit, then the metrics are fractions of 100, or of 1/100, respectively. PPC algorithms should produce maximum absolute
errors no greater than 0.5 in these units. If the LSD is the hundreds digit, then quantized versions of true values will be within fifty of the true value. It is much easier to satisfy this tolerance for a true value of 100 (only 50% accuracy required) than for 999 (5% accuracy required). Thus the minimum accuracy guaranteed for \( nsd = 1 \) ranges from 5–50%. For this reason, the best and worst cast performance usually occurs for true values whose LSD value is close to one and nine, respectively. Of course most users prefer \( prc > 1 \) because accuracies increase exponentially with \( prc \). Continuing the previous example to \( prc = 2 \), quantized versions of true values from 1000–9999 will also be within 50 of the true value, i.e., have accuracies from 0.5–5%. In other words, only two significant digits are necessary to guarantee better than 5% accuracy in quantization. We recommend that dataset producers and users consider quantizing datasets with \( nsd = 3 \). This guarantees accuracy of 0.05–0.5% for individual values. Statistics computed from ensembles of quantized values will, assuming the mean error \( \bar{\epsilon} \) is small, have much better accuracy than 0.5%. This accuracy is the most that can be justified for many applications.

To demonstrate these principles we conduct error analyses on an artificial, reproducible dataset, and on an actual dataset of observational analysis values.\(^{23}\) The table summarizes quantization accuracy based on the three metrics.

<table>
<thead>
<tr>
<th>NSD</th>
<th>Number of Significant Digits.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emabs</td>
<td>Maximum absolute error.</td>
</tr>
<tr>
<td>Emebs</td>
<td>Mean absolute error.</td>
</tr>
<tr>
<td>Emean</td>
<td>Mean error.</td>
</tr>
</tbody>
</table>

**Artificial Data: \( N=1000000 \) values in \([1.0,2.0)\) in steps of \( 1.0e^{-6} \)**

<table>
<thead>
<tr>
<th>NSD</th>
<th>Emabs</th>
<th>Emebs</th>
<th>Emean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.31</td>
<td>0.11</td>
<td>4.1e-4</td>
</tr>
<tr>
<td>2</td>
<td>0.39</td>
<td>0.14</td>
<td>6.8e-5</td>
</tr>
<tr>
<td>3</td>
<td>0.49</td>
<td>0.17</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>4</td>
<td>0.30</td>
<td>0.11</td>
<td>3.2e-7</td>
</tr>
<tr>
<td>5</td>
<td>0.37</td>
<td>0.13</td>
<td>3.1e-7</td>
</tr>
<tr>
<td>6</td>
<td>0.36</td>
<td>0.12</td>
<td>4.4e-7</td>
</tr>
<tr>
<td>7</td>
<td>0.00</td>
<td>0.00</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Double-Precision**

<table>
<thead>
<tr>
<th>Emabs</th>
<th>Emebs</th>
<th>Emean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.31</td>
<td>0.11</td>
</tr>
<tr>
<td>2</td>
<td>0.39</td>
<td>0.14</td>
</tr>
<tr>
<td>3</td>
<td>0.49</td>
<td>0.17</td>
</tr>
<tr>
<td>4</td>
<td>0.30</td>
<td>0.11</td>
</tr>
<tr>
<td>5</td>
<td>0.37</td>
<td>0.13</td>
</tr>
<tr>
<td>6</td>
<td>0.36</td>
<td>0.13</td>
</tr>
</tbody>
</table>

**Observational Analysis: \( N=13934592 \) values MERRA Temperature 20130601**

**Double-Precision**

<table>
<thead>
<tr>
<th>NSD</th>
<th>Emabs</th>
<th>Emebs</th>
<th>Emean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.31</td>
<td>0.11</td>
<td>2.4e-3</td>
</tr>
<tr>
<td>2</td>
<td>0.39</td>
<td>0.14</td>
<td>3.8e-4</td>
</tr>
<tr>
<td>3</td>
<td>0.49</td>
<td>0.17</td>
<td>9.6e-5</td>
</tr>
<tr>
<td>4</td>
<td>0.30</td>
<td>0.11</td>
<td>2.3e-3</td>
</tr>
<tr>
<td>5</td>
<td>0.37</td>
<td>0.13</td>
<td>2.2e-3</td>
</tr>
<tr>
<td>6</td>
<td>0.36</td>
<td>0.13</td>
<td>1.7e-2</td>
</tr>
</tbody>
</table>

\(^{23}\) The artificial dataset employed is one million evenly spaced values from 1.0–2.0. The analysis data are \( N = 13934592 \) values of the temperature field from the NASA MERRA analysis of 20130601.
All results show that PPC quantization performs as expected. Absolute maximum errors $Emabs < 0.5$ for all $prc$. For $1 \leq prc \leq 6$, quantization results in comparable maximum absolute and mean absolute errors $Emabs$ and $Emebs$, respectively. Mean errors $Emean$ are orders of magnitude smaller because quantization produces over- and under-estimated values in balance. When $prc = 7$, quantization of single-precision values is ineffective, because all available bits are used to represent the maximum precision of seven digits. The maximum and mean absolute errors $Emabs$ and $Emebs$ are nearly identical across algorithms, precisions, and dataset types. This is consistent with both the artificial data and empirical data being random, and thus exercising equally strengths and weaknesses of the algorithms over the course of millions of input values. We generated artificial arrays with many different starting values and interval spacing and all gave qualitatively similar results. The results presented are the worst obtained.

The artificial data has much smaller mean error $Emean$ than the observational analysis. The reason why is unclear. It may be because the temperature field is concentrated in particular ranges of values (and associated quantization errors) prevalent on Earth, e.g., $200 < T < 320$. It is worth noting that the mean error $Emean < 0.01$ for $1 \leq prc < 6$, and that $Emean$ is typically at least two or more orders of magnitude less than $Emabs$. Thus quantized values with precisions as low as $prc = 1$ still yield highly significant statistics by contemporary scientific standards.

Testing shows that PPC quantization enhances compression of typical climate datasets. The degree of enhancement depends, of course, on the required precision. Model results are often computed as NC\_DOUBLE then archived as NC\_FLOAT to save space. This table summarizes the performance of lossless and lossy compression on two typical, or at least random, netCDF data files. The files were taken from representative model-simulated and satellite-retrieved datasets. Only floating point data were compressed. No attempt was made to compress integer-type variables as they occupy an insignificant fraction of every dataset. The columns are

| Type | File-type: N3 for netCDF CLASSIC, N4 for NETCDF4, N7 for NETCDF4\_CLASSIC (which comprises netCDF3 data types and structures with netCDF4 storage features like compression), H4 for HDF4, and H5 for HDF5. N4/7 means results apply to both N4 and N7 filetypes. | LLC | Type of lossless compression employed, if any. Bare numbers refer to the strength of the DEFLATE algorithm employed internally by netCDF4/HDF5, while numbers prefixed with B refer to the block size employed by the Burrows-Wheeler algorithm in bzip2. | PPC | Number of significant digits retained by the precision-preserving compression NSD algorithm. | Pck | Y if the default ncpdq packing algorithm (convert floating point types to NC\_SHORT) was employed. | Size | Resulting filesize in MB. | % | Compression ratio, i.e., resulting filesize relative to original size, in percent. In some cases the original files is already losslessly compressed. The compression |
ratios reported are relative to the size of the original file as distributed, not as optimally losslessly compressed.

A dash (−) indicates the associated compression feature was not employed.

<table>
<thead>
<tr>
<th>File Path</th>
<th>Type</th>
<th>LLC</th>
<th>PPC</th>
<th>Pck</th>
<th>Size</th>
<th>%</th>
<th>Flags and Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td># dstmch90_clm.nc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>34.7</td>
<td>100.0</td>
<td>Original is not compressed</td>
</tr>
<tr>
<td></td>
<td>N3</td>
<td>B1</td>
<td>-</td>
<td>-</td>
<td>28.9</td>
<td>83.2</td>
<td>bzip2 -1</td>
</tr>
<tr>
<td></td>
<td>N3</td>
<td>B9</td>
<td>-</td>
<td>-</td>
<td>29.3</td>
<td>84.4</td>
<td>bzip2 -9</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>35.0</td>
<td>101.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>28.2</td>
<td>81.3</td>
<td>-L 1</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>28.0</td>
<td>80.8</td>
<td>-L 9</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>17.6</td>
<td>50.9</td>
<td>ncpdq -L 0</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>Y</td>
<td>-</td>
<td>7.9</td>
<td>22.8</td>
<td>ncpdq -L 1</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>7</td>
<td>-</td>
<td>28.2</td>
<td>81.3</td>
<td>--ppc default=7</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>6</td>
<td>-</td>
<td>27.9</td>
<td>80.6</td>
<td>--ppc default=6</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>5</td>
<td>-</td>
<td>25.9</td>
<td>74.6</td>
<td>--ppc default=5</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>4</td>
<td>-</td>
<td>22.3</td>
<td>64.3</td>
<td>--ppc default=4</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>3</td>
<td>-</td>
<td>18.9</td>
<td>54.6</td>
<td>--ppc default=3</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>14.5</td>
<td>43.2</td>
<td>--ppc default=2</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>10.0</td>
<td>29.0</td>
<td>--ppc default=1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># b1850c5cn_doe_polar_merged_0_cesm1_2_0_HD+MAM4+tun2b.hp.e003.cam.h0.0001-01.nc</th>
<th>Type</th>
<th>LLC</th>
<th>PPC</th>
<th>Pck</th>
<th>Size</th>
<th>%</th>
<th>Flags and Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>119.8</td>
<td>100.0</td>
<td>Original is not compressed</td>
</tr>
<tr>
<td></td>
<td>N3</td>
<td>B1</td>
<td>-</td>
<td>-</td>
<td>84.2</td>
<td>70.3</td>
<td>bzip2 -1</td>
</tr>
<tr>
<td></td>
<td>N3</td>
<td>B9</td>
<td>-</td>
<td>-</td>
<td>84.8</td>
<td>70.9</td>
<td>bzip2 -9</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>120.5</td>
<td>100.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>82.6</td>
<td>69.0</td>
<td>-L 1</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>82.1</td>
<td>68.6</td>
<td>-L 9</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>60.7</td>
<td>50.7</td>
<td>ncpdq -L 0</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>Y</td>
<td>-</td>
<td>26.0</td>
<td>21.8</td>
<td>ncpdq -L 1</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>7</td>
<td>-</td>
<td>82.6</td>
<td>69.0</td>
<td>--ppc default=7</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>6</td>
<td>-</td>
<td>81.9</td>
<td>68.4</td>
<td>--ppc default=6</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>5</td>
<td>-</td>
<td>77.2</td>
<td>64.5</td>
<td>--ppc default=5</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>4</td>
<td>-</td>
<td>69.0</td>
<td>57.6</td>
<td>--ppc default=4</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>3</td>
<td>-</td>
<td>59.3</td>
<td>49.5</td>
<td>--ppc default=3</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>49.5</td>
<td>41.3</td>
<td>--ppc default=2</td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>38.2</td>
<td>31.9</td>
<td>--ppc default=1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># MERRA300.prod.assim.inst3_3d_asm_Cp.20130601.hdf</th>
<th>Type</th>
<th>LLC</th>
<th>PPC</th>
<th>Pck</th>
<th>Size</th>
<th>%</th>
<th>Flags and Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H4</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>244.3</td>
<td>100.0</td>
<td>Original is compressed</td>
</tr>
<tr>
<td></td>
<td>H4</td>
<td>B1</td>
<td>-</td>
<td>-</td>
<td>244.7</td>
<td>100.1</td>
<td>bzip2 -1</td>
</tr>
<tr>
<td></td>
<td>N4</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>214.5</td>
<td>87.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N7</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>210.6</td>
<td>86.2</td>
<td></td>
</tr>
</tbody>
</table>
A selective, per-variable approach to PPC yields the best balance of precision and compression yet requires the dataset producer to understand the intrinsic precision of each variable. Such a specification for a GCM dataset might look like this (using names for the NCAR CAM model):

```bash
# Be conservative on non-explicit quantities, so default=5
# Some quantities deserve four significant digits
# Many quantities, such as aerosol optical depths and burdens, are
# highly uncertain and only useful to three significant digits.
ncks -7 -O
--ppc default=5
--ppc AN.?,AQ.?=4
```

3.28 Deflation

All NCO operators that define variables support the netCDF4 feature of storing variables compressed with the lossless DEFLATE compression algorithm. DEFLATE combines the Lempel-Ziv encoding with Huffman coding. The specific version used by netCDF4/HDF5 is that implemented in the zlib library used by gzip. Activate deflation with the \(-L dfl\_lvl\) short option (or with the same argument to the \(--dfl\_lvl\) or \(--deflate\) long options). Specify the deflation level \(dfl\_lvl\) on a scale from no deflation \((dfl\_lvl = 0)\) to maximum deflation \((dfl\_lvl = 9)\). Under the hood, this selects the compression blocksize. Minimal deflation \((dfl\_lvl = 1)\) achieves considerable storage compression with little time penalty. Higher deflation levels require more time for compression. File sizes resulting from minimal \((dfl\_lvl = 1)\) and maximal \((dfl\_lvl = 9)\) deflation levels typically differ by less than 10\% in size.

To compress an entire file using deflation, use

\[
\text{ncks -4 -L 0 in.nc out.nc # No deflation (fast, no time penalty)} \\
\text{ncks -4 -L 1 in.nc out.nc # Minimal deflation (little time penalty)} \\
\text{ncks -4 -L 9 in.nc out.nc # Maximal deflation (much slower)}
\]

Unscientific testing shows that deflation compresses typical climate datasets by 30-60\%. Packing, a lossy compression technique available for all netCDF files (see Section 3.32 [Packed data], page 92), can easily compress files by 50\%. Packed data may be deflated to squeeze datasets by about 80\%:

\[
\text{ncks -4 -L 1 in.nc out.nc # Minimal deflation (~30-60\% compression)} \\
\text{ncks -4 -L 9 in.nc out.nc # Maximal deflation (~31-63\% compression)} \\
\text{ncpdq in.nc out.nc # Standard packing (~50\% compression)} \\
\text{ncpdq -4 -L 9 in.nc out.nc # Deflated packing (~80\% compression)}
\]

ncks prints deflation parameters, if any, to screen (see Section 4.7 [ncks netCDF Kitchen Sink], page 188).

3.29 MD5 digests

Availability: ncecat, ncks, ncrcat

Short options:

Long options: \(--md5\_dgs\), \(--md5\_digest\), \(--md5\_wrt\_att\), \(--md5\_write\_attribute\)
As of NCO version 4.1.0 (April, 2012), NCO supports data integrity verification using the MD5 digest algorithm. This support is currently implemented in `ncks` and in the multifile concatenators `ncecat` and `ncrcat`. Activate it with the `--md5_dgs` or `--md5_digest` long options. As of NCO version 4.3.3 (July, 2013), NCO will write the MD5 digest of each variable as an NC_CHAR attribute named MD5. This support is currently implemented in `ncks` and in the multifile concatenators `ncecat` and `ncrcat`. Activate it with the `--md5_wrt_att` or `--md5_write_attribute` long options.

The behavior and verbosity of the MD5 digest is operator-dependent. When activating MD5 digests with `ncks` it is assumed that the user simply wishes to see the digest of every variable and this is done when the debugging level exceeds one. This incurs only the minor overhead of performing the hash algorithm for each variable read. MD5 digests may be activated in both the one- and two-filename argument forms of `ncks`, which are used for printing and for sub-setting, respectively. The MD5 digests are shown as a 32-character hexadecimal string in which each two characters represent one byte of the 16-byte digest:

```bash
> ncks -O -D 2 -C --md5 -v md5_a,md5_abc ~/nco/data/in.nc
...ncks: INFO MD5(md5_a) = 0cc175b9c0f1b6a831c399e269772661
    md5_a = 'a'
ncks: INFO MD5(md5_abc) = 900150983cd24fb0d6963f7d28e17f72
    lev[0]=100 md5_abc[0--2] = 'abc'
> ncks -O -D 2 -C -d lev,0 --md5 -v md5_a,md5_abc ~/nco/data/in.nc
...ncks: INFO MD5(md5_a) = 0cc175b9c0f1b6a831c399e269772661
    md5_a = 'a'
ncks: INFO MD5(md5_abc) = 0cc175b9c0f1b6a831c399e269772661
    lev[0]=100 md5_abc[0--0] = 'a'
```

In fact these examples demonstrate the validity of the hash algorithm since the MD5 hashes of the strings “a” and “abc” are widely known. The second example shows that the hyperslab of variable `md5_abc` (= “abc”) consisting of only its first letter (= “a”) has the same hash as the variable `md5_a` (“a”). This illustrates that MD5 digests act only on variable data, not on metadata.

When activating MD5 digests with `ncecat` or `ncrcat` it is assumed that the user wishes to verify that every variable written to disk has the same MD5 digest as when it is subsequently read from disk. This incurs the major additional overhead of reading in each variable after it is written and performing the hash algorithm again on that to compare to the original hash. Moreover, it is assumed that such operations are generally done “production mode” where the user is not interested in actually examining the digests herself. The digests proceed silently unless the debugging level exceeds three:

```bash
> ncecat -O -D 4 --md5 -p ~/nco/data in.nc in.nc ~/foo.nc | grep MD5
...ncecat: INFO MD5(wnd_spd) = bec190dd944f2ce2794a7a4abf224b28
ncecat: INFO MD5 digests of RAM and disk contents for wnd_spd agree
```

```bash
> ncrcat -O -D 4 --md5 -p ~/nco/data in.nc in.nc ~/foo.nc | grep MD5
...
```
ncrcat: INFO MD5(wnd_spd) = 74699bb0a72b7f16456badb2c995f1a1
ncrcat: INFO MD5 digests of RAM and disk contents for wnd_spd agree

Regardless of the debugging level, an error is returned when the digests of the variable read from the source file and from the output file disagree.

These rules are evolving and as NCO pays more attention to data integrity. We welcome feedback and suggestions from users.

### 3.30 Buffer sizes

<table>
<thead>
<tr>
<th>Availability: All operators</th>
<th>Short options:</th>
<th>Long options:</th>
<th><code>--bfr_sz_hnt</code>, <code>--buffer_size_hint</code></th>
</tr>
</thead>
</table>

As of NCO version 4.2.0 (May, 2012), NCO allows the user to request specific buffer sizes to allocate for reading and writing files. This buffer size determines how many system calls the netCDF layer must invoke to read and write files. By default, netCDF uses the preferred I/O block size returned as the `st_blksize` member of the `stat` structure returned by the `stat()` system call. Otherwise, netCDF uses twice the system pagesize. Larger sizes can increase access speed by reducing the number of system calls netCDF makes to read/write data from/to disk. Because netCDF cannot guarantee the buffer size request will be met, the actual buffer size granted by the system is printed as an INFO statement.

```
# Request 2 MB file buffer instead of default 8 kB buffer
> ncks -O -D 3 --bfr_sz=2097152 ~/nco/data/in.nc ~/foo.nc
...
ncks: INFO nc__open() will request file buffer size = 2097152 bytes
ncks: INFO nc__open() opened file with buffer size = 2097152 bytes
...```

### 3.31 RAM disks

<table>
<thead>
<tr>
<th>Availability: All operators</th>
<th>Short options:</th>
<th>Long options:</th>
<th><code>--ram_all</code>, <code>--create_ram</code>, <code>--open_ram</code>, <code>--diskless_all</code></th>
</tr>
</thead>
</table>

As of NCO version 4.2.1 (August, 2012), NCO supports the use of diskless files, aka RAM disks, for file access and creation. Two independent switches, `--open_ram` and `--create_ram`, control this feature. Before describing the specifics of these switches, we describe why many NCO operations will not benefit from them. Essentially, reading/writing from/to RAM rather than disk only hastens the task when reads/writes to disk are avoided. Most NCO operations are simple enough that they require a single read-from/write-to disk. On modern Linux systems the block size defaults to 8192 B. The GLADE filesystem at NCAR has a block size of 512 kB.
for every block of input/output. Diskless access does not change this, but it does add an extra read-from/write-to RAM. However this extra RAM write/read does avoid contention for limited system resources like disk-head access. Operators which may benefit from RAM disks include ncwa, which may need to read weighting variables multiple times, the multi-file operators ncra, ncrcat, and ncecat, which may try to write output at least once per input file, and ncap2 scripts which may be arbitrarily long and convoluted.

The ‘--open_ram’ switch causes input files to copied to RAM when opened. All further metadata and data access occurs in RAM and thus avoids access time delays caused by disk-head movement. Usually input data is read at most once so it is unlikely that requesting input files be stored in RAM will save much time. The likeliest exceptions are files that are accessed numerous times, such as those analyzed extensively analyzed by ncap2.

Invoking ‘--open_ram’, ‘--ram_all’, or ‘--diskless_all’ uses much more system memory. To copy the input file to RAM increases the sustained memory use by exactly the on-disk filesize of the input file, i.e., $MS^+ = FT$. For large input files this can be a huge memory burden that starves the rest of the NCO analysis of sufficient RAM. To be safe, use ‘--open_ram’, ‘--ram_all’, or ‘--diskless_all’ only on files that are much (say at least a factor of four) smaller than your available system RAM. See Section 2.9 [Memory Requirements], page 22 for further details.

The ‘--create_ram’ switch causes output files to be created in RAM, rather than on disk. These files are copied to disk only when closed, i.e., when the operator completes. Creating files in RAM may save time, especially with ncap2 computations that are iterative, e.g., loops, and for multi-file operators that write output every record (timestep) or file. RAM files provide many of the same benefits as RAM variables in such cases (see Section 4.1.12 [RAM variables], page 134).

Two switches, ‘--ram_all’ and ‘--diskless_all’, are convenient shortcuts for specifying both ‘--create_ram’ and ‘--diskless_ram’. Thus

```bash
ncks in.nc out.nc # Default: Open in.nc on disk, write out.nc to disk
ncks --open_ram in.nc out.nc # Open in.nc in RAM, write out.nc to disk
ncks --create_ram in.nc out.nc # Create out.nc in RAM, write to disk
# Open in.nc in RAM, create out.nc in RAM, then write out.nc to disk
ncks --open_ram --create_ram in.nc out.nc
ncks --ram_all in.nc out.nc # Same as above
ncks --diskless_all in.nc out.nc # Same as above
```

It is straightforward to demonstrate the efficacy of RAM disks. For NASA we constructed a test that employs ncecat an arbitrary number (set to one hundred thousand) of files are all symbolically linked to the same file. Everything is on the local filesystem (not DAP).

```bash
# Create symbolic links for benchmark
cd ${DATA}/nco # Do all work here
for idx in {1..99999}; do
  idx_fmt='printf "%05d" ${idx}''
  /bin/ln -s ${DATA}/nco/LPRM-AMSR_E_L3_D_SOILM3_V002-20120512T111931Z_20020619.nc \
    ${DATA}/nco/${idx_fmt}.nc
done
```
# Benchmark time to ncecat one hundred thousand files

```
# Benchmark time to ncecat one hundred thousand files
time ncecat --create_ram -O -u time -v ts -d Latitude,40.0 -
-d Longitude,-105.0 -p ${DATA}/nco -n 99999,5,1 00001.nc ~/foo.nc
```

Run normally on a laptop in 201303, this completes in 21 seconds. The `--create_ram` reduces the elapsed time to 9 seconds. Some of this speed may be due to using symlinks and caching. However, the efficacy of `--create_ram` is clear. Placing the output file in RAM avoids thousands of disk writes. It is not unreasonable to for NCO to process a million files like this in a few minutes. However, there is no substitute for benchmarking with real files.

A completely independent way to reduce time spent writing files is to refrain from writing temporary output files. This is accomplished with the `--no_tmp_fl` switch (see Section 2.3 [Temporary Output Files], page 15).

## 3.32 Packed data

| Availability: | ncap2, ncbo, ncfl, ncdf, ncdf, ncflint, ncflint, ncpdq, ncpdq, ncra, ncra, ncwa |
| Short options: | None |
| Long options: | `--hdf_upk`, `--hdf_unpack` |

The phrase *packed data* refers to data which are stored in the standard netCDF3 packing format which employs a lossy algorithm. See Section 4.7 [ncks netCDF Kitchen Sink], page 188 for a description of deflation, a lossless compression technique available with netCDF4 only. Packed data may be deflated to save additional space.

### Packing Algorithm

*Packing* The standard netCDF packing algorithm (described here) produces data with the same dynamic range as the original but which requires no more than half the space to store. Like all packing algorithms, it is *lossy*. The packed variable is stored (usually) as type NC_SHORT with the two attributes required to unpack the variable, `scale_factor` and `add_offset`, stored at the original (unpacked) precision of the variable. Let \( \min \) and \( \max \) be the minimum and maximum values of \( x \).

\[
\begin{align*}
scale\_factor &= (\max - \min)/ndrv \\
add\_offset &= (\min + \max)/2 \\
pck &= (\upk - add\_offset)/scale\_factor \\
&= \frac{ndrv \times [\upk - (\min + \max)/2]}{\max - \min}
\end{align*}
\]

where \( ndrv \) is the number of discrete representable values for given type of packed variable. The theoretical maximum value for \( ndrv \) is two raised to the number of bits used to store the packed variable. Thus if the variable is packed into type NC_SHORT, a two-byte datatype, then there are at most \( 2^{16} = 65536 \) distinct values representable. In practice, the number of

\footnote{Although not a part of the standard, NCO enforces the policy that the \_FillValue attribute, if any, of a packed variable is also stored at the original precision.}
discretely representible values is taken to be two less than the theoretical maximum. This leaves space for a missing value and solves potential problems with rounding that may occur during the unpacking of the variable. Thus for NC_SHORT, \( ndrv = 65536 - 2 = 65534 \). Less often, the variable may be packed into type NC_CHAR, where \( ndrv = 2^8 - 2 = 256 - 2 = 254 \), or type NC_INT where \( ndrv = 2^{32} - 2 = 4294967295 - 2 = 4294967293 \). One useful feature of (lossy) netCDF packing algorithm is that additional, loss-less packing algorithms perform well on top of it.

**Unpacking Algorithm**

*Unpacking* The unpacking algorithm depends on the presence of two attributes, `scale_factor` and `add_offset`. If `scale_factor` is present for a variable, the data are multiplied by the value `scale_factor` after the data are read. If `add_offset` is present for a variable, then the `add_offset` value is added to the data after the data are read. If both `scale_factor` and `add_offset` attributes are present, the data are first scaled by `scale_factor` before the offset `add_offset` is added.

\[
upk = \text{scale_factor} \times \text{pck} + \text{add_offset} = \frac{\text{pck} \times (\max - \min)}{\text{ndrv}} + \frac{\min + \max}{2}
\]

When `scale_factor` and `add_offset` are used for packing, the associated variable (containing the packed data) is typically of type `byte` or `short`, whereas the unpacked values are intended to be of type `int`, `float`, or `double`. An attribute’s `scale_factor` and `add_offset` and `_FillValue`, if any, should all be of the type intended for the unpacked data, i.e., `int`, `float` or `double`.

**Default Handling of Packed Data**

Most files originally written in HDF format use the HDF packing/unpacking algorithm. This algorithm is incompatible with the netCDF packing algorithm described above. The unpacking component of the HDF algorithm (described here) is

\[
upk = \text{scale_factor} \times (\text{pck} - \text{add_offset})
\]

Confusingly, the (incompatible) netCDF and HDF algorithms both store their parameters in attributes with the same names (`scale_factor` and `add_offset`). Data packed with one algorithm should never be unpacked with the other; doing so will result in incorrect answers. Unfortunately, few users are aware that their datasets may be packed, and fewer know the details of the packing algorithm employed. This is what we in the “bizness” call an *interoperability* issue because it hampers data analysis performed on heterogeneous systems.

As described below, NCO automatically unpacks data before performing arithmetic. This automatic unpacking occurs silently since there is usually no reason to bother users with these details. There is as yet no generic way for NCO to know which packing convention was used, so NCO *assumes* the netCDF convention was used. NCO uses the same convention for unpacking unless explicitly told otherwise with the ‘-hdf_upk’ (also ‘-hdf_unpack’) switch. Until and unless a method of automatically detecting the packing method is devised, it must remain the user’s responsibility to tell NCO when to use the HDF convention instead of the netCDF convention to unpack.
If your data originally came from an HDF file (e.g., NASA EOS) then it was likely packed with the HDF convention and must be unpacked with the same convention. Our recommendation is to only request HDF unpacking when you are certain. Most packed datasets encountered by NCO will have used the netCDF convention. Those that were not will hopefully produce noticeably weird values when unpacked by the wrong algorithm. Before or after panicking, treat this as a clue to re-try your commands with the `--hdf_upk` switch. See Section 4.8 [ncpdq netCDF Permute Dimensions Quickly], page 202 for an easy technique to unpack data packed with the HDF convention, and then re-pack it with the netCDF convention.

**Default Handling of Packed Data**

All NCO arithmetic operators understand packed data. The operators automatically unpack any packed variable in the input file which will be arithmetically processed. For example, `ncra` unpacks all record variables, and `ncwa` unpacks all variable which contain a dimension to be averaged. These variables are stored unpacked in the output file.

On the other hand, arithmetic operators do not unpack non-processed variables. For example, `ncra` leaves all non-record variables packed, and `ncwa` leaves packed all variables lacking an averaged dimension. These variables (called fixed variables) are passed unaltered from the input to the output file. Hence fixed variables which are packed in input files remain packed in output files. Completely packing and unpacking files is easily accomplished with `ncpdq` (see Section 4.8 [ncpdq netCDF Permute Dimensions Quickly], page 202). Pack and unpack individual variables with `ncpdq` and the `ncap2 pack()` and `unpack()` functions (see Section 4.1.11 [Methods and functions], page 131).

### 3.33 Operation Types

| Availability: | ncap2, ncra, nces, ncwa |
| Short options: | `-y` |
| Long options: | `--operation`, `--op_typ` |

The `--y op_typ` switch allows specification of many different types of operations Set `op_typ` to the abbreviated key for the corresponding operation:

- **avg**  Mean value
- **sqravg**  Square of the mean
- **avgsqr**  Mean of sum of squares
- **max**  Maximum value
- **min**  Minimum value
- **mabs**  Maximum absolute value
- **mebs**  Mean absolute value
- **mabs**  Minimum absolute value
**rms**  
Root-mean-square (normalized by $N$)

**rmssdn**  
Root-mean square (normalized by $N-1$)

**sqrt**  
Square root of the mean

**ttl**  
Sum of values

NCO assumes coordinate variables represent grid axes, e.g., longitude. The only rank-reduction which makes sense for coordinate variables is averaging. Hence NCO implements the operation type requested with ‘-y’ on all non-coordinate variables, not on coordinate variables. When an operation requires a coordinate variable to be reduced in rank, i.e., from one dimension to a scalar or from one dimension to a degenerate (single value) array, then NCO *always averages* the coordinate variable regardless of the arithmetic operation type performed on the non-coordinate variables.

The mathematical definition of each arithmetic operation is given below. See Section 4.12 [ncwa netCDF Weighted Averager], page 219, for additional information on masks and normalization. If an operation type is not specified with ‘-y’ then the operator performs an arithmetic average by default. Averaging is described first so the terminology for the other operations is familiar.

The masked, weighted average of a variable $x$ can be generally represented as

$$
\bar{x}_j = \frac{\sum_{i=1}^{N} \mu_i m_i w_i x_i}{\sum_{i=1}^{N} \mu_i m_i w_i}
$$

where $\bar{x}_j$ is the $j$’th element of the output hyperslab, $x_i$ is the $i$’th element of the input hyperslab, $\mu_i$ is 1 unless $x_i$ equals the missing value, $m_i$ is 1 unless $x_i$ is masked, and $w_i$ is the weight. This formidable looking formula represents a simple weighted average whose bells and whistles are all explained below. It is not too early to note, however, that when $\mu_i = m_i = w_i = 1$, the generic averaging expression above reduces to a simple arithmetic average. Furthermore, $m_i = w_i = 1$ for all operators except ncwa. These variables are included in the discussion below for completeness, and for possible future use in other operators.

The size $J$ of the output hyperslab for a given variable is the product of all the dimensions of the input variable which are not averaged over. The size $N$ of the input hyperslab contributing to each $\bar{x}_j$ is simply the product of the sizes of all dimensions which are averaged over (i.e., dimensions specified with ‘-a’). Thus $N$ is the number of input elements which potentially contribute to each output element. An input element $x_i$ contributes to the output element $x_j$ except in two conditions:

1. $x_i$ equals the missing value (see Section 3.25 [Missing Values], page 71) for the variable.
2. $x_i$ is located at a point where the mask condition (see Section 4.12.1 [Mask condition], page 220) is false.

Points $x_i$ in either of these two categories do not contribute to $x_j$—they are ignored. We now define these criteria more rigorously.

Each $x_i$ has an associated Boolean weight $\mu_i$ whose value is 0 or 1 (false or true). The value of $\mu_i$ is 1 (true) unless $x_i$ equals the missing value (see Section 3.25 [Missing...
Values], page 71) for the variable. Thus, for a variable with no _FillValue attribute, \( \mu_i \) is always 1. All NCO arithmetic operators (\texttt{ncbo, ncra, nces, ncflint, ncwa}) treat missing values analogously.

Besides (weighted) averaging, \texttt{ncwa, ncra,} and \texttt{nces} also compute some common non-linear operations which may be specified with the ‘-y’ switch (see Section 3.33 [Operation Types], page 94). The other rank-reducing operations are simple variations of the generic weighted mean described above. The total value of \( x \) (\texttt{-y ttl}) is

\[
\bar{x}_j = \sum_{i=1}^{i=N} \mu_i m_i w_i x_i
\]

Note that the total is the same as the numerator of the mean of \( x \), and may also be obtained in \texttt{ncwa} by using the ‘-N’ switch (see Section 4.12 [ncwa netCDF Weighted Averager], page 219).

The minimum value of \( x \) (\texttt{-y min}) is

\[
\bar{x}_j = \min[\mu_1 m_1 w_1 x_1, \mu_2 m_2 w_2 x_2, \ldots, \mu_N m_N w_N x_N]
\]

Analogously, the maximum value of \( x \) (\texttt{-y max}) is

\[
\bar{x}_j = \max[\mu_1 m_1 w_1 x_1, \mu_2 m_2 w_2 x_2, \ldots, \mu_N m_N w_N x_N]
\]

Thus the minima and maxima are determined after any weights are applied.

The minimum absolute value of \( x \) (\texttt{-y mibs}) is

\[
\bar{x}_j = \min[\mu_1 m_1 w_1 |x_1|, \mu_2 m_2 w_2 |x_2|, \ldots, \mu_N m_N w_N |x_N|]
\]

Analogously, the maximum absolute value of \( x \) (\texttt{-y mabs}) is

\[
\bar{x}_j = \max[\mu_1 m_1 w_1 |x_1|, \mu_2 m_2 w_2 |x_2|, \ldots, \mu_N m_N w_N |x_N|]
\]

Thus the minimum and maximum absolute values are determined after any weights are applied. The mean absolute value of \( x \) (\texttt{-y mebs}) is

\[
\bar{x}_j = \left( \frac{\sum_{i=1}^{i=N} \mu_i m_i w_i |x_i|}{\sum_{i=1}^{i=N} \mu_i m_i w_i} \right)^2
\]

The square of the mean value of \( x \) (\texttt{-y sqravg}) is

\[
\bar{x}_j = \left( \frac{\sum_{i=1}^{i=N} \mu_i m_i w_i x_i}{\sum_{i=1}^{i=N} \mu_i m_i w_i} \right)^2
\]

The mean of the sum of squares of \( x \) (\texttt{-y avgsqr}) is

\[
\bar{x}_j = \frac{\sum_{i=1}^{i=N} \mu_i m_i w_i x_i^2}{\sum_{i=1}^{i=N} \mu_i m_i w_i}
\]
If \( x \) represents a deviation from the mean of another variable, \( x_i = y_i - \bar{y} \) (possibly created by \texttt{ncko} in a previous step), then applying \texttt{avgsqr} to \( x \) computes the approximate variance of \( y \). Computing the true variance of \( y \) requires subtracting 1 from the denominator, discussed below. For a large sample size however, the two results will be nearly indistinguishable.

The root mean square of \( x \) (\(-y \text{ rms}\)) is

\[
\bar{x}_j = \sqrt{\frac{\sum_{i=1}^{N} \mu_i m_i w_i x_i^2}{\sum_{i=1}^{N} \mu_i m_i w_i}}
\]

Thus \texttt{rms} simply computes the squareroot of the quantity computed by \texttt{avgsqr}.

The root mean square of \( x \) with standard-deviation-like normalization (\(-y \text{ rmssdn}\)) is implemented as follows. When weights are not specified, this function is the same as the root mean square of \( x \) except one is subtracted from the sum in the denominator

\[
\bar{x}_j = \sqrt{\frac{\sum_{i=1}^{N} \mu_i m_i x_i^2}{\sum_{i=1}^{N} \mu_i m_i - 1 + \sum_{i=1}^{N} \mu_i}}
\]

If \( x \) represents the deviation from the mean of another variable, \( x_i = y_i - \bar{y} \), then applying \texttt{rmssdn} to \( x \) computes the standard deviation of \( y \). In this case the \(-1\) in the denominator compensates for the degree of freedom already used in computing \( \bar{y} \) in the numerator. Consult a statistics book for more details.

When weights are specified it is unclear how to compensate for this extra degree of freedom. Weighting the numerator and denominator of the above by \( w_i \) and subtracting one from the denominator is only appropriate when all the weights are 1.0. When the weights are arbitrary (e.g., Gaussian weights), subtracting one from the sum in the denominator does not necessarily remove one degree of freedom. Therefore when \(-y \text{ rmssdn}\) is requested and weights are specified, \texttt{ncwa} actually implements the \texttt{rms} procedure. \texttt{nces} and \texttt{ncra}, which do not allow weights to be specified, always implement the \texttt{rmssdn} procedure when asked.

\texttt{20130827: Fedora Core 19 (FC19) broke here with "./nco.texi:6394: Missing dollarsign inserted.” Ubuntu always built nco.texi fine Adding a dollarsign character right here breaks Ubuntu builds too Hence I must carefully spell-out the word dollarsign instead 20130829: Making many smaller TeX environments does not solve problem 20130910: Using latest texinfo.tex from GNU does not solve problem 20130910: Karl Berry solved problem by fixing bug in texinfo.tex Bug was triggered in Fedora by apostrophe in ”User’s Guide” (manual title) Bug not present in texinfo.tex version 2008-04-18.10 (used by Ubuntu 13.04) Bug present in texinfo.tex version 2013-02-01.11 (used by FC19) Bug just fixed in texinfo.tex version 2013-09-11 (committed by Karl) nco/autobld/texinfo.tex now contains fixed version Breakage always occurs near here The square root of the mean of \( x \) (\(-y \text{ sqrt}\)) is

\[
\bar{x}_j = \sqrt{\frac{\sum_{i=1}^{N} \mu_i m_i w_i x_i^2}{\sum_{i=1}^{N} \mu_i m_i w_i}}
\]

The definitions of some of these operations are not universally useful. Mostly they were chosen to facilitate standard statistical computations within the NCO framework. We are
open to redefining and or adding to the above. If you are interested in having other statistical quantities defined in NCO please contact the NCO project (see Section 1.7 [Help Requests and Bug Reports], page 13).

EXAMPLES

Suppose you wish to examine the variable \texttt{prs\_sfc(time,lat,lon)} which contains a time series of the surface pressure as a function of latitude and longitude. Find the minimum value of \texttt{prs\_sfc} over all dimensions:

\begin{verbatim}
ncwa -y min -v prs_sfc in.nc foo.nc
\end{verbatim}

Find the maximum value of \texttt{prs\_sfc} at each time interval for each latitude:

\begin{verbatim}
ncwa -y max -v prs_sfc -a lon in.nc foo.nc
\end{verbatim}

Find the root-mean-square value of the time-series of \texttt{prs\_sfc} at every gridpoint:

\begin{verbatim}
ncra -y rms -v prs_sfc in.nc foo.nc
ncwa -y rms -v prs_sfc -a time in.nc foo.nc
\end{verbatim}

The previous two commands give the same answer but \texttt{ncra} is preferred because it has a smaller memory footprint. A dimension of size one is said to be degenerate. By default, \texttt{ncra} leaves the (degenerate) \texttt{time} dimension in the output file (which is usually useful) whereas \texttt{ncwa} removes the \texttt{time} dimension (unless ‘-b’ is given).

These operations work as expected in multi-file operators. Suppose that \texttt{prs\_sfc} is stored in multiple timesteps per file across multiple files, say \texttt{jan.nc}, \texttt{feb.nc}, \texttt{march.nc}. We can now find the three month maximum surface pressure at every point.

\begin{verbatim}
nces -y max -v prs_sfc jan.nc feb.nc march.nc out.nc
\end{verbatim}

It is possible to use a combination of these operations to compute the variance and standard deviation of a field stored in a single file or across multiple files. The procedure to compute the temporal standard deviation of the surface pressure at all points in a single file \texttt{in.nc} involves three steps.

\begin{verbatim}
ncwa -O -v prs_sfc -a time in.nc out.nc
ncbo -O -v prs_sfc in.nc out.nc out.nc
ncra -O -y rmssdn out.nc out.nc
\end{verbatim}

First construct the temporal mean of \texttt{prs\_sfc} in the file \texttt{out.nc}. Next overwrite \texttt{out.nc} with the anomaly (deviation from the mean). Finally overwrite \texttt{out.nc} with the root-mean-square of itself. Note the use of ‘-y rmssdn’ (rather than ‘-y rms’) in the final step. This ensures the standard deviation is correctly normalized by one fewer than the number of time samples. The procedure to compute the variance is identical except for the use of ‘-y var’ instead of ‘-y rmssdn’ in the final step.

\begin{verbatim}
ncap2 can also compute statistics like standard deviations. Brute-force implementation of formulae is one option, e.g.,
\end{verbatim}

\begin{verbatim}
ncap2 -s 'prs\_sfc\_sdn=sqrt((prs\_sfc-prs\_sfc.\_avg($time)\_^2).\_total($time)/(\$time.\_size-1))
in.nc out.nc
\end{verbatim}
The operation may, of course, be broken into multiple steps in order to archive intermediate quantities, such as the time-anomalies:

```bash
ncap2 -s 'prs_sfc_anm=prs_sfc-prs_sfc.avg($time)' -s 'prs_sfc_sdn=sqrt((prs_sfc_anm^2).total($time)/($time.size-1))' in.nc out.nc
```

`ncap2` supports intrinsic standard deviation functions (see Section 3.33 [Operation Types], page 94) which simplify the above expression to:

```bash
ncap2 -s 'prs_sfc_sdn=(prs_sfc-prs_sfc.avg($time)).rmssdn($time)' in.nc out.nc
```

These intrinsic functions compute the answer quickly and concisely.

The procedure to compute the spatial standard deviation of a field in a single file `in.nc` involves three steps:

```bash
ncwa -O -v prs_sfc,gw -a lat,lon -w gw in.nc out.nc
ncbo -O -v prs_sfc,gw in.nc out.nc out.nc
ncwa -O -y rmssdn -v prs_sfc -a lat,lon -w gw out.nc out.nc
```

First the appropriately weighted (with `-w gw`) spatial mean values are written to the output file. This example includes the use of a weighted variable specified with `-w gw`. When using weights to compute standard deviations one must remember to include the weights in the initial output files so that they may be used again in the final step. The initial output file is then overwritten with the gridpoint deviations from the spatial mean. Finally the root-mean-square of the appropriately weighted spatial deviations is taken.

The `ncap2` solution to the spatially-weighted standard deviation problem is:

```bash
ncap2 -s 'prs_sfc_sdn=(prs_sfc*gw-prs_sfc*gw.avg($lat,$lon)).rmssdn($lat,$lon)' in.nc out.nc
```

Be sure to multiply the variable by the weight prior to computing the anomalies and the standard deviation.

The procedure to compute the standard deviation of a time-series across multiple files involves one extra step since all the input must first be collected into one file:

```bash
ncrcat -O -v tpt in.nc in.nc foo1.nc
ncwa -O -a time foo1.nc foo2.nc
ncbo -O -v tpt foo1.nc foo2.nc foo3.nc
ncra -O -y rmssdn foo3.nc out.nc
```

The first step assembles all the data into a single file. Though this may consume a lot of temporary disk space, it is more or less required by the `ncbo` operation in the third step.

### 3.34 Type Conversion
Type conversion refers to the casting or coercion of one fundamental or atomic data type to another, e.g., converting NC_SHORT (two bytes) to NC_DOUBLE (eight bytes). Type conversion always promotes or demotes the range and/or precision of the values a variable can hold. Type conversion is automatic when the language carries out this promotion according to an internal set of rules without explicit user intervention. In contrast, manual type conversion refers to explicit user commands to change the type of a variable or attribute. Most type conversion happens automatically, yet there are situations in which manual type conversion is advantageous.

3.34.1 Automatic type conversion

There are at least two reasons to avoid type conversions. First, type conversions are expensive since they require creating (temporary) buffers and casting each element of a variable from its storage type to some other type and then, often, converting it back. Second, a dataset’s creator perhaps had a good reason for storing data as, say, NC_FLOAT rather than NC_DOUBLE. In a scientific framework there is no reason to store data with more precision than the observations merit. Normally this is single-precision, which guarantees 6–9 digits of precision. Reasons to engage in type conversion include avoiding rounding errors and out-of-range limitations of less-precise types. This is the case with most integers. Thus NCO defaults to automatically promote integer types to floating point when performing lengthy arithmetic, yet NCO defaults to not promoting single to double-precision floats.

Before discussing the more subtle floating point issues, we first examine integer promotion. We will show how following parsimonious conversion rules dogmatically can cause problems, and what NCO does about that. That said, there are situations in which implicit conversion of single- to double-precision is also warranted. Understanding the narrowness of these situations takes time, and we hope the reader appreciates the following detailed discussion.

Consider the average of the two NC_SHORTs 17000s and 17000s. A straightforward average without promotion results in garbage since the intermediate value which holds their sum is also of type NC_SHORT and thus overflows on (i.e., cannot represent) values greater than 32,76726. There are valid reasons for expecting this operation to succeed and the NCO philosophy is to make operators do what you want, not what is purest. Thus, unlike C and Fortran, but like many other higher level interpreted languages, NCO arithmetic operators will perform automatic type conversion on integers when all the following conditions are met27:

1. The requested operation is arithmetic. This is why type conversion is limited to the operators ncap2, ncbo, nces, ncflint, ncra, and ncwa.

26 $32767 = 2^{15} - 1$
27 Operators began performing automatic type conversions before arithmetic in NCO version 1.2, August, 2000. Previous versions never performed unnecessary type conversion for arithmetic.
2. The arithmetic operation could benefit from type conversion. Operations that could benefit include averaging, summation, or any “hard” arithmetic that could overflow or underflow. Larger representable sums help avoid overflow, and more precision helps to avoid underflow. Type conversion does not benefit searching for minima and maxima (‘-y min’, or ‘-y max’).

3. The variable on disk is of type `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, or `NC_INT`. Type `NC_DOUBLE` is not promoted because there is no type of higher precision. Conversion of type `NC_FLOAT` is discussed in detail below. When it occurs, it follows the same procedure (promotion then arithmetic then demotion) as conversion of integer types.

When these criteria are all met, the operator promotes the variable in question to type `NC_DOUBLE`, performs all the arithmetic operations, casts the `NC_DOUBLE` type back to the original type, and finally writes the result to disk. The result written to disk may not be what you expect, because of incommensurate ranges represented by different types, and because of (lack of) rounding. First, continuing the above example, the average (e.g., ‘-y avg’) of 17000s and 17000s is written to disk as 17000s. The type conversion feature of NCO makes this possible since the arithmetic and intermediate values are stored as `NC_DOUBLE`s, i.e., 34000.0d and only the final result must be represented as an `NC_SHORT`. Without the type conversion feature of NCO, the average would have been garbage (albeit predictable garbage near -15768s). Similarly, the total (e.g., ‘-y ttl’) of 17000s and 17000s written to disk is garbage (actually -31536s) since the final result (the true total) of 34000 is outside the range of type `NC_SHORT`.

After arithmetic is computed in double-precision for promoted variables, the intermediate double-precision values must be demoted to the variables’ original storage type (e.g., from `NC_DOUBLE` to `NC_SHORT`). NCO has handled this demotion in three ways in its history. Prior to October, 2011 (version 4.0.8), NCO employed the C library truncate function, `trunc()`\(^{28}\). Truncation rounds \(x\) to the nearest integer not larger in absolute value. For example, truncation rounds 1.0d, 1.5d, and 1.8d to the same value, 1s. Clearly, truncation does not round floating point numbers to the nearest integer! Yet truncation is how the C language performs implicit conversion of real numbers to integers.

NCO stopped using truncation for demotion when an alert user (Neil Davis) informed us that this caused a small bias in the packing algorithm employed by `ncpdq`. This led to NCO adopting rounding functions for demotion. Rounding functions eliminated the small bias in the packing algorithm.

From February, 2012 through March, 2013 (versions 4.0.9–4.2.6), NCO employed the C library family of rounding functions, `lround()`. These functions round \(x\) to the nearest integer, halfway cases away from zero. The problem with `lround()` is that it always rounds real values ending in .5 away from zero. This rounds, for example, 1.5d and 2.5d to 2s and 3s, respectively.

Since April, 2013 (version 4.3.0), NCO has employed the other C library family of rounding functions, `lrint()`. This algorithm rounds \(x\) to the nearest integer, using the current rounding direction. Halfway cases are rounded to the nearest even integer. This rounds,

---

\(^{28}\) The actual type conversions with truncation were handled by intrinsic type conversion, so the `trunc()` function was never explicitly called, although the results would be the same if it were.
for example, both \texttt{1.5d} and \texttt{2.5d} to the same value, \texttt{2s}, as recommended by the IEEE. This rounding is symmetric: up half the time, down half the time. This is the current and hopefully final demotion algorithm employed by NCO.

Hence because of automatic conversion, NCO will compute the average of \texttt{2s} and \texttt{3s} in double-precision arithmetic as \((2.0d + 3.0d)/2.0d) = \texttt{2.5d}\). It then demotes this intermediate result back to \texttt{NC_SHORT} and stores it on disk as \texttt{trunc(2.5d) = 2s} (versions up to 4.0.8), \texttt{lround(2.5d) = 3s} (versions 4.0.9–4.2.6), and \texttt{lrint(2.5d) = 2s} (versions 4.3.0 and later).

### 3.34.2 Promoting Single-precision to Double

Promotion of real numbers from single- to double-precision is fundamental to scientific computing. When it should occur depends on the precision of the inputs and the number of operations. Single-precision (four-byte) numbers contain about seven significant figures, while double-precision contain about sixteen. More, err, precisely, the IEEE single-precision representation gives from 6 to 9 significant decimal digits precision\(^{29}\). And the IEEE double-precision representation gives from 15 to 17 significant decimal digits precision\(^{30}\). Hence double-precision numbers represent about nine digits more precision than single-precision numbers.

Given these properties, there are at least two possible arithmetic conventions for the treatment of real numbers:

1. Conservative, aka Fortran Convention Automatic type conversion during arithmetic in the Fortran language is, by default, performed only when necessary. All operands in an operation are converted to the most precise type involved the operation before the arithmetic operation. Expressions which involve only single-precision numbers are computed entirely in single-precision. Expressions involving mixed precision types are computed in the type of higher precision. NCO by default employs the Fortran Convention for promotion.

2. Aggressive, aka C Convention The C language is by default much more aggressive (and thus wasteful) than Fortran, and will always implicitly convert single- to double-precision numbers for no good reason. All real-number standard C library functions are double-precision, and C programmers must take extra steps to only utilize single precision arithmetic. The high level interpreted data analysis languages IDL, Matlab, and NCL all adopt the C Convention.

NCO does not automatically promote \texttt{NC_FLOAT} because, in our judgement, the performance penalty of always doing so would outweigh the potential benefits. The now-classic

\(^{29}\) According to Wikipedia’s summary of IEEE standard 754, “If a decimal string with at most 6 significant digits is converted to IEEE 754 single-precision and then converted back to the same number of significant decimal, then the final string should match the original; and if an IEEE 754 single-precision is converted to a decimal string with at least 9 significant decimal and then converted back to single, then the final number must match the original”.

\(^{30}\) According to Wikipedia’s summary of IEEE standard 754, “If a decimal string with at most 15 significant digits is converted to IEEE 754 double-precision representation and then converted back to a string with the same number of significant digits, then the final string should match the original; and if an IEEE 754 double precision is converted to a decimal string with at least 17 significant digits and then converted back to double, then the final number must match the original”.
text “Numerical Recipes in C” discusses this point under the section “Implicit Conversion of Float to Double”\textsuperscript{31}. That said, such promotion is warranted in some circumstances.

For example, rounding errors can accumulate to worrisome levels during arithmetic performed on large arrays of single-precision floats. This use-case occurs often in geoscientific studies of climate where thousands-to-millions of gridpoints may contribute to a single average. If the inputs are all single-precision, then so should be the output. However the intermediate results where running sums are accumulated may suffer from too much rounding or from underflow unless computed in double-precision.

The order of operations matters to floating point math even when the analytic expressions are equal. Cautious users feel disquieted when results from equally valid analyses differ in the final bits instead of agreeing bit-for-bit. For example, averaging arrays in multiple stages produces different answers than averaging them in one step. This is easily seen in the computation of ensemble averages by two different methods. The NCO test file \texttt{in.nc} contains single- and double-precision representations of the same temperature timeseries as \texttt{tpt_flt} and \texttt{tpt_dbl}. Pretend each datapoint in this timeseries represents a monthly-mean temperature. We will mimic the derivation of a fifteen-year ensemble-mean January temperature by concatenating the input file five times, and then averaging the datapoints representing January two different ways. In Method 1 we derive the 15-year ensemble January average in two steps, as the average of three five-year averages. This method is naturally used when each input file contains multiple years and multiple input files are needed\textsuperscript{32}. In Method 2 we obtain 15-year ensemble January average in a single step, by averaging all 15 Januaries at one time:

```bash
# tpt_flt and tpt_dbl are identical except for precision
ncks --cdl -C -v tpt_flt,tpt_dbl ~/nco/data/in.nc
# Create file with five "ten-month years" (i.e., 50 timesteps) of temperature data
ncrcat -O -v tpt_flt,tpt_dbl ~/nco/data in.nc in.nc in.nc in.nc in.nc ~/foo.nc
# Average 1st five "Januaries" (elements 1, 11, 21, 31, 41)
ncra --flt -O -F -d time,1,,10 ~/foo.nc ~/foo_avg1.nc
# Average 2nd five "Januaries" (elements 2, 12, 22, 32, 42)
ncra --flt -O -F -d time,2,,10 ~/foo.nc ~/foo_avg2.nc
# Average 3rd five "Januaries" (elements 3, 13, 23, 33, 43)
ncra --flt -O -F -d time,3,,10 ~/foo.nc ~/foo_avg3.nc
# Method 1: Obtain ensemble January average by averaging the averages
ncra --flt -O ~/foo_avg1.nc ~/foo_avg2.nc ~/foo_avg3.nc ~/foo_avg_mth1.nc
# Method 2: Obtain ensemble January average by averaging the raw data
# Employ ncra's "subcycle" feature (http://nco.sf.net/nco.html#ssc)
ncra --flt -O -F -d time,1,,10,3 ~/foo.nc ~/foo_avg_mth2.nc
```

\textsuperscript{31} See page 21 in Section 1.2 of the First edition for this gem:

One does not need much experience in scientific computing to recognize that the implicit conversion rules are, in fact, sheer madness! In effect, they make it impossible to write efficient numerical programs.

\textsuperscript{32} For example, the CMIP5 archive tends to distribute monthly average timeseries in 50-year chunks.
Although the two methods are arithmetically equivalent, they produce slightly different answers due to the different order of operations. Moreover, it appears at first glance that the single-precision answers suffer from greater error than the double-precision answers. In fact both precisions suffer from non-zero rounding errors. The answers differ negligibly to machine precision, which is about seven significant figures for single precision floats (tpt_flt), and sixteen significant figures for double precision (tpt_dbl). The input precision determines the answer precision.

IEEE arithmetic guarantees that two methods will produce bit-for-bit identical answers only if they compute the same operations in the same order. Bit-for-bit identical answers may also occur by happenstance when rounding errors exactly compensate one another. This is demonstrated by repeating the example above with the '--dbl' (or '--rth_dbl' for clarity) option which forces conversion of single-precision numbers to double-precision prior to arithmetic. Now ncra will treat the first value of tpt_flt, 273.1000f, as 273.1000000000000d. Arithmetic on tpt_flt then proceeds in double-precision until the final answer, which is converted back to single-precision for final storage.

The '--dbl' switch has no effect on the results computed from double-precision inputs. But now the two methods produce bit-for-bit identical results from the single-precision inputs! This is due to the happenstance of rounding along with the effects of the '--dbl' switch. The '--flt' and '--rth_flt' switches are provided for symmetry. They enforce the traditional NCO and Fortran convention of keeping single-precision arithmetic in single-precision unless a double-precision number is explicitly involved.
We have shown that forced promotion of single- to double-precision prior to arithmetic has advantages and disadvantages. The primary disadvantages are speed and size. Double-precision arithmetic is 10–60% slower than, and requires twice the memory of single-precision arithmetic. The primary advantage is that rounding errors in double-precision are much less likely to accumulate to values near the precision of the underlying geophysical variable.

For example, if we know temperature to five significant digits, then a rounding error of 1-bit could affect the least precise digit of temperature after 1,000–10,000 consecutive one-sided rounding errors under the worst possible scenario. Many geophysical grids have tens-of-thousands to millions of points that must be summed prior to normalization to compute an average. It is possible for single-precision rounding errors to accumulate and degrade the precision in such situations. Double-precision arithmetic mitigates this problem, so ‘--dbl’ would be warranted.

This can be seen with another example, averaging a global surface temperature field with ncwa. The input contains a single-precision global temperature field (stored in TREFHT) produced by the CAM3 general circulation model (GCM) run and stored at 1.9 by 2.5 degrees resolution. This requires 94 latitudes and 144 longitudes, or 13,824 total surface gridpoints, a typical GCM resolution these days. These input characteristics are provided only to show the context to the interested reader, equivalent results would be found in statistics of any dataset of comparable size. Models often represent Earth on a spherical grid where global averages must be created by weighting each gridcell by its latitude-dependent weight (i.e., the Gaussian weight stored in \( gw \)), or by the surface area of each contributing gridpoint (stored in area).

Like many geophysical models and most GCMs, CAM3 runs completely in double-precision yet stores its archival output in single-precision to save space. In practice such models usually save multi-dimensional prognostic and diagnostic fields (like TREFHT(lat,lon) and area(lat,lon)) as single-precision, while saving all one-dimensional coordinates and weights (here lat, lon, and \( gw(\text{lon}) \)) as double-precision. To obtain pure double-precision arithmetic and storage of the global mean temperature, we first create and store double-precision versions of the single-precision fields:

\[
\text{ncap2} -O -s 'TREFHT\_dbl=double(TREFHT);area\_dbl=double(area)\}' in.nc in.nc
\]

The single- and double-precision temperatures may each be averaged globally using four permutations for the precision of the weight and of the intermediate arithmetic representation:

1. Single-precision weight (area), single-precision arithmetic
2. Double-precision weight (gw), single-precision arithmetic
3. Single-precision weight (area), double-precision arithmetic
4. Double-precision weight (gw), double-precision arithmetic

# NB: Values below are printed with C-format %5.6f using
# ncks -H -C -s '%5.6f' -v TREFHT,TREFHT\_dbl out.nc

ncwa --flt -O -a lat,lon -w area in.nc out.nc

# TREFHT = 289.246735
\texttt{\# TREFHT\textsubscript{dbl} = 289.239964}\n\texttt{\# Double-precision weight (gw), single-precision arithmetic}\nncwa --flt -0 -a lat,lon -w gw in.nc out.nc\n\texttt{\# TREFHT = 289.226135}\n\texttt{\# TREFHT\textsubscript{dbl} = 289.239964}\n\texttt{\# Single-precision weight (area), double-precision arithmetic}\nncwa --dbl -0 -a lat,lon -w area in.nc out.nc\n\texttt{\# TREFHT = 289.239960}\n\texttt{\# TREFHT\textsubscript{dbl} = 289.239964}\n\texttt{\# Double-precision weight (gw), double-precision arithmetic}\nncwa --dbl -0 -a lat,lon -w gw in.nc out.nc\n\texttt{\# TREFHT = 289.239960}\n\texttt{\# TREFHT\textsubscript{dbl} = 289.239964}\n
First note that the TREFHT\textsubscript{dbl} average never changes because TREFHT\textsubscript{dbl}(lat,lon) is double-precision in the input file. As described above, NCO automatically converts all operands involving to the highest precision involved in the operation. So specifying ‘--dbl’ is redundant for double-precision inputs.

Second, the single-precision arithmetic averages of the single-precision input TREFHT differ by 289.246735 – 289.226135 = 0.0206 from eachother, and, more importantly, by as much as 289.239964 – 289.226135 = 0.013829 from the correct (double-precision) answer. These averages differ in the fifth digit, i.e., they agree only to four significant figures! Given that climate scientists are concerned about global temperature variations of a tenth of a degree or less, this difference is large. It means that the global mean temperature changes scientists are looking for are comparable in size to the numerical artifacts produced by the averaging procedure.

Why are the single-precision numerical artifacts so large? Each global average is the result of multiplying almost 15,000 elements each by its weight, summing those, and then dividing by the summed weights. Thus about 50,000 single-precision floating point operations caused the loss of two to three significant digits of precision. The net error of a series of independent rounding errors is a random walk phenomena\textsuperscript{33}. Successive rounding errors displace the answer further from the truth. An ensemble of such averages will, on average, have no net bias. In other words, the expectation value of a series of IEEE rounding errors is zero. And the error of any given sequence of rounding errors obeys, for large series, a Gaussian distribution centered on zero.

Single-precision numbers use three of their four eight-bit bytes to represent the mantissa so the smallest representable single-precision mantissa is $\epsilon \equiv 2^{-23} = 1.19209 \times 10^{-7}$. This $\epsilon$ is the smallest $x$ such that $1.0 + x \neq 1.0$. This is the rounding error for non-exact precision-numbers. Applying random walk theory to rounding, it can be shown that the expected rounding error after $n$ inexact operations is $\sqrt{2n/\pi}$ for large $n$. The expected (i.e., mean absolute) rounding error in our example with 13,824 additions is about $\sqrt{2 \times 13824/\pi} = 91.96$. Hence, addition alone of about fifteen thousand single-precision floats is expected to consume about two significant digits of precision. This neglects the error due to the inner product (weights times values) and normalization (division by tally) aspects of a weighted

\textsuperscript{33} Thanks to Michael J. Prather for explaining this to me.
average. the ratio of two numbers each containing a numerical bias can magnify the size of the bias. In summary, a global mean number computed from about 15,000 gridpoints each with weights can be expected to lose up to three significant digits. Since single-precision starts with about seven significant digits, we should not expect to retain more than four significant digits after computing weighted averages in single-precision. The above example with TREFHT shows the expected four digits of agreement.

The NCO results have been independently validated to the extent possible in three other languages: C, Matlab, and NCL. C and NCO are the only languages that permit single-precision numbers to be treated with single precision arithmetic:

```plaintext
# Double-precision weight (gw), single-precision arithmetic (C)
ncwa_3528514.exe
# TREFHT = 289.240112
# Double-precision weight (gw), double-precision arithmetic (C)
# TREFHT = 289.239964
# Single-precision weight (area), double-precision arithmetic (Matlab)
# TREFHT = 289.239964
# Double-precision weight (gw), double-precision arithmetic (Matlab)
# TREFHT = 289.239964
# Single-precision weight (area), double-precision arithmetic (NCL)
ncl < ncwa_3528514.ncl
# TREFHT = 289.239960
# TREFHT_dbl = 289.239964
# Double-precision weight (gw), double-precision arithmetic (NCL)
# TREFHT = 289.239960
# TREFHT_dbl = 289.239964
```

All languages tested (C, Matlab, NCL, and NCO) agree to machine precision with double-precision arithmetic. Users are fortunate to have a variety of high quality software that liberates them from the drudgery of coding their own. Many packages are free (as in beer)! As shown above NCO permits one to shift to their float-promotion preferences as desired. No other language allows this with a simple switch.

To summarize, until version 4.3.6 (September, 2013), the default arithmetic convention of NCO adhered to Fortran behavior, and automatically promoted single-precision to double-precision in all mixed-precision expressions, and left-alone pure single-precision expressions. This is faster and more memory efficient than other conventions. However, pure single-precision arithmetic can lose too much precision when used to condense (e.g., average) large arrays. Statistics involving about \( n = 10,000 \) single-precision inputs will lose about 2–3 digits if not promoted to double-precision prior to arithmetic. The loss scales with the squareroot of \( n \). For larger \( n \), users should promote floats with the ‘--dbl’ option if they want to preserve more than four significant digits in their results.

The ‘--dbl’ and ‘--flt’ switches are only available with the NCO arithmetic operators that could potentially perform more than a few single-precision floating point operations per result. These are ncex, ncra, and ncwa. Each is capable of thousands to millions or more operations per result. By contrast, the arithmetic operators ncbo and ncflint perform at
most one floating point operation per result. Providing the `--dbl` option for such trivial operations makes little sense, so the option is not currently made available.

We are interested in users’ opinions on these matters. The default behavior was changed from `--flt` to `--dbl` with the release of NCO version 4.3.6 (October 2013). We will change the default back to `--flt` if users prefer. Or we could set a threshold (e.g., \( n \geq 10000 \)) after which single- to double-precision promotion is automatically invoked. Or we could make the default promotion convention settable via an environment variable (GSL does this a lot). Please let us know what you think of the selected defaults and options.

### 3.34.3 Manual type conversion

ncap2 provides intrinsic functions for performing manual type conversions. This, for example, converts variable `tpt` to external type `NC_SHORT` (a C-type `short`), and variable `prs` to external type `NC_DOUBLE` (a C-type `double`).

```bash
ncap2 -s 'tpt=short(tpt);prs=double(prs)' in.nc out.nc
```

See Section 4.1 [ncap2 netCDF Arithmetic Processor], page 116, for more details.

### 3.35 Batch Mode

<table>
<thead>
<tr>
<th>Availability: All operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: <code>--O</code>, <code>--A</code></td>
</tr>
<tr>
<td>Long options: <code>--ovr</code>, <code>--overwrite</code>, <code>--apn</code>, <code>--append</code></td>
</tr>
</tbody>
</table>

If the `output-file` specified for a command is a pre-existing file, then the operator will prompt the user whether to overwrite (erase) the existing `output-file`, attempt to append to it, or abort the operation. However, interactive questions reduce productivity when processing large amounts of data. Therefore NCO also implements two ways to override its own safety features, the `--O` and `--A` switches. Specifying `--O` tells the operator to overwrite any existing `output-file` without prompting the user interactively. Specifying `--A` tells the operator to attempt to append to any existing `output-file` without prompting the user interactively. These switches are useful in batch environments because they suppress interactive keyboard input. NB: As of 20120515, ncap2 is unable to append to files that already contain the appended dimensions.

### 3.36 History Attribute

<table>
<thead>
<tr>
<th>Availability: All operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: <code>--h</code></td>
</tr>
<tr>
<td>Long options: <code>--hist</code>, <code>--history</code></td>
</tr>
</tbody>
</table>

All operators automatically append a `history` global attribute to any file they create or modify. The `history` attribute consists of a timestamp and the full string of the invocation command to the operator, e.g., `Mon May 26 20:10:24 1997: ncks in.nc foo.nc`. The full
Chapter 3: Shared Features

Contents of an existing `history` attribute are copied from the first `input-file` to the `output-file`. The timestamps appear in reverse chronological order, with the most recent timestamp appearing first in the `history` attribute. Since NCO and many other netCDF operators adhere to the `history` convention, the entire data processing path of a given netCDF file may often be deduced from examination of its `history` attribute. As of May, 2002, NCO is case-insensitive to the spelling of the `history` attribute name. Thus attributes named `History` or `HISTORY` (which are non-standard and not recommended) will be treated as valid `history` attributes. When more than one global attribute fits the case-insensitive search for “history”, the first one found will be used. `history` attribute To avoid information overkill, all operators have an optional switch (`-h`, `--hst`, or `--history`) to override automatically appending the `history` attribute (see Section 4.2 [ncatted netCDF Attribute Editor], page 168). Note that the ‘-h’ switch also turns off writing the `nco_input_file_list` attribute for multi-file operators (see Section 3.37 [File List Attributes], page 109).

### 3.37 File List Attributes

<table>
<thead>
<tr>
<th>Availability: nces, ncecat, ncra, nrcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: ‘-H’</td>
</tr>
<tr>
<td>Long options: ‘--fl_lst_in’, ‘--file_list’</td>
</tr>
</tbody>
</table>

Many methods of specifying large numbers of input file names pass these names via pipes, encodings, or argument transfer programs (see Section 2.7 [Large Numbers of Files], page 19). When these methods are used, the input file list is not explicitly passed on the command line. This results in a loss of information since the `history` attribute no longer contains the exact command by which the file was created.

NCO solves this dilemma by archiving input file list attributes. When the input file list to a multi-file operator is specified via `stdin`, the operator, by default, attaches two global attributes to any file they create or modify. The `nco_input_file_number` global attribute contains the number of input files, and `nco_input_file_list` contains the file names, specified as standard input to the multi-file operator. This information helps to verify that all input files the user thinks were piped through `stdin` actually arrived. Without the `nco_input_file_list` attribute, the information is lost forever and the “chain of evidence” would be broken.

The ‘-H’ switch overrides (turns off) the default behavior of writing the input file list global attributes when input is from `stdin`. The ‘-h’ switch does this too, and turns off the `history` attribute as well (see Section 3.36 [History Attribute], page 108). Hence both switches allows space-conscious users to avoid storing what may amount to many thousands of filenames in a metadata attribute.

### 3.38 CF Conventions
NCO recognizes some Climate and Forecast (CF) metadata conventions, and applies special rules to such data. NCO was contemporaneous with COARDS and still contains some rules to handle older model datasets that pre-date CF, such as NCAR CCM and early CCSM datasets. Such datasets may not contain an explicit Conventions attribute (e.g., ‘CF-1.0’). Nevertheless, we refer to all such metadata collectively as CF metadata. Skip this section if you never work with CF metadata.

The latest CF netCDF conventions are described here. Most CF netCDF conventions are transparent to NCO. There are no known pitfalls associated with using any NCO operator on files adhering to these conventions. To increase user friendliness, NCO applies special rules to certain variables in CF files. Rules not required by CF are because experience shows that they simplify data analysis.

Here is a general sense of NCO’s CF-support: Understanding and implementing NUG recommendations such as the history attribute, packing conventions, and attention to units. Special handling of variables designated as coordinates, bounds, or ancillary variables, so that users subsetting a certain variable automatically obtain all related variables. Special handling and prevention of meaningless operations (e.g., the root-mean-square of latitude) so that coordinates and bounds preserve meaningful information even as normal fields are statistically transformed. Understanding units and certain calendars so that hyperslabs may be specified in physical units, and so that user needs not manually decode per-file time specifications. Understanding auxiliary coordinates so that irregular hyperslabs may be specified on complex geometric grids. Checking for CF-compliance on netCDF3 and netCDF4 and HDF files. Converting netCDF4 and HDF files to netCDF3 for strict CF-compliance. Finally, a main use of NCO is to “produce CF”, i.e., to improve CF-compliance by annotating metadata, renaming objects (attributes, variables, and dimensions), permuting and inverting dimensions, recomputing values, and data compression.

Currently, NCO determines whether a datafile is a CF output datafile simply by checking (case-insensitively) whether the value of the global attribute Conventions (if any) equals ‘CF-1.0’ or ‘NCAR-CSM’ Should Conventions equal either of these in the (first) input-file, NCO will apply special rules to certain variables because of their usual meaning in CF files. NCO will not average the following variables often found in CF files: ntrm, ntrn, ntrk, ndbase, nsbase, nbdate, nbsec, mdt, mhisf. These variables contain scalar metadata such as the resolution of the host geophysical model and it makes no sense to change their values.

Furthermore, the size and rank-preserving arithmetic operators try not to operate on certain grid properties. These operators are ncap2, ncbo, nces, ncflint, and ncpdq (when used for packing, not for permutation). These operators do not operate, by default, on (i.e., add, subtract, pack, etc.) the following variables: ORO, area, datesec, date, gw, hyai, hyam, hybi, hybm, lat_bnds, lon_bnds, msk_. These variables represent Gaussian weights, land/sea masks, time fields, hybrid pressure coefficients, and latitude/longitude boundaries. We call these fields non-coordinate grid properties. Coordinate grid properties are easy to identify because they are coordinate variables such as latitude and longitude.
Users usually want all grid properties to remain unaltered in the output file. To be treated as a grid property, the variable name must exactly match a name in the above list, or be a coordinate variable. The handling of msk_* is exceptional in that any variable name beginning with the string msk_ is considered to be a “mask” and is thus preserved (not operated on arithmetically).

You must spoof NCO if you would like any grid properties or other special CF fields processed normally. For example rename the variables first with ncrename, or alter the Conventions attribute.

As of NCO version 4.0.8 (April, 2011), NCO supports the CF bounds convention for cell boundaries described here. This convention allows coordinate variables (including multidimensional coordinates) to describe the boundaries of their cells. This is done by naming the variable which contains the bounds in the bounds attribute. Note that coordinates of rank $N$ have bounds of rank $N + 1$. NCO-generated subsets of CF-compliant files with bounds attributes will include the coordinates specified by the bounds attribute, if any. Hence the subsets will themselves be CF-compliant. Bounds are subject to the user-specified override switches (including ‘-c’ and ‘-C’) described in Section 3.12 [Subsetting Coordinate Variables], page 45.

As of NCO version 4.4.5 (July, 2014), NCO supports the CF ancillary_variables convention for described here. This convention allows ancillary variables to be associated with one or more primary variables. NCO attaches any such variables to the extraction list along with the primary variable and its usual (one-dimensional) coordinates, if any. Ancillary variables are subject to the user-specified override switches (including ‘-c’ and ‘-C’) described in Section 3.12 [Subsetting Coordinate Variables], page 45.

As of NCO version 3.9.6 (January, 2009), NCO supports the CF coordinates convention described here. This convention allows variables to specify additional coordinates (including multi-dimensional coordinates) in a space-separated string attribute named coordinates. NCO attaches any such coordinates to the extraction list along with the variable and its usual (one-dimensional) coordinates, if any. These auxiliary coordinates are subject to the user-specified override switches (including ‘-c’ and ‘-C’) described in Section 3.12 [Subsetting Coordinate Variables], page 45.

Elimination of reduced dimensions from the coordinates attribute helps ensure that rank-reduced variables become completely independent from their former dimensions. As of NCO version 4.4.9 (March, 2015), NCO may modify the coordinates attribute to assist this. In particular, nwa eliminates from the coordinates attribute any dimension that it collapses, e.g., by averaging. The former presence of this dimension will usually be indicated by the CF cell_methods convention described here. Hence the CF cell_methods and coordinates conventions can be said to work in tandem to characterize the state and history of a variable’s analysis.

As of NCO version 4.4.2 (February, 2014), NCO supports some of the CF cell_methods convention to describe the analysis procedures that have been applied to data. The convention creates (or appends to an existing) cell_methods attribute a space-separated list of couplets of the form dmn: op where dmn is a comma-separated list of dimensions previously contained in the variable that have been reduced by the arithmetic operation op. For example, the cell_methods value time: mean says that the variable in question was
averaged over the time dimension. In such cases time will either be a scalar variable or a degenerate dimension or coordinate. This simply means that it has been averaged-over. The value time, lon: mean lat: max says that the variable in question is the maximum zonal mean of the time averaged original variable. Which is to say that the variable was first averaged over time and longitude, and then the residual latitudinal array was reduced by choosing the maximum value. Since the cell methods convention may alter metadata in an undesirable (or possibly incorrect) fashion, we provide switches to ensure it is always or never used. Use long-options ‘--cell_mth’ or ‘--cell_methods’ to invoke the algorithm (true by default), and options ‘--no_cell_mth’ or ‘--no_cell_methods’ to turn it off. These options are only available in the operators ncwa and ncra.

### 3.39 ARM Conventions

<table>
<thead>
<tr>
<th>Availability: ncrcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: None</td>
</tr>
</tbody>
</table>

ncrcat has been programmed to correctly handle data files which utilize the Atmospheric Radiation Measurement (ARM) Program convention for time and time offsets. If you do not work with ARM data then you may skip this section. ARM data files store time information in two variables, a scalar, base_time, and a record variable, time_offset. Subtle but serious problems can arise when these type of files are blindly concatenated without CF or ARM support. NCO implements rebasing (see Section 3.23 [Rebasing Time Coordinate], page 69) as necessary on both CF and ARM files. Rebasing chains together consecutive input-files and produces an output-file which contains the correct time information. For ARM files this is especially complex because the time coordinates are often stored as type NC_CHAR. Currently, ncrcat determines whether a datafile is an ARM datafile simply by testing for the existence of the variables base_time, time_offset, and the dimension time. If these are found in the input-file then ncrcat will automatically perform two non-standard, but hopefully useful, procedures. First, ncrcat will ensure that values of time_offset appearing in the output-file are relative to the base_time appearing in the first input-file (and presumably, though not necessarily, also appearing in the output-file). Second, if a coordinate variable named time is not found in the input-files, then ncrcat automatically creates the time coordinate in the output-file. The values of time are defined by the ARM conventions time = base_time + time_offset. Thus, if output-file contains the time_offset variable, it will also contain the time coordinate. A short message is added to the history global attribute whenever these ARM-specific procedures are executed.

### 3.40 Operator Version

<table>
<thead>
<tr>
<th>Availability: All operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short options: ‘-r’</td>
</tr>
<tr>
<td>Long options: ‘--revision’, ‘--version’, or ‘--vrs’</td>
</tr>
</tbody>
</table>
All operators can be told to print their version information, library version, copyright notice, and compile-time configuration with the ‘-r’ switch, or its long-option equivalent ‘revision’. The ‘--version’ or ‘--vrs’ switches print the operator version information only. The internal version number varies between operators, and indicates the most recent change to a particular operator’s source code. This is useful in making sure you are working with the most recent operators. The version of NCO you are using might be, e.g., 3.9.5. Using ‘-r’ on, say, ncks, produces something like ‘NCO netCDF Operators version "3.9.5" last modified 2008/05/11 built May 12 2008 on neige by zender Copyright (C) 1995--2008 Charlie Zender ncks version 20090918’. This tells you that ncks contains all patches up to version 3.9.5, which dates from May 11, 2008.
4 Reference Manual

This chapter presents reference pages for each of the operators individually. The operators are presented in alphabetical order. All valid command line switches are included in the syntax statement. Recall that descriptions of many of these command line switches are provided only in Chapter 3 [Shared features], page 25, to avoid redundancy. Only options specific to, or most useful with, a particular operator are described in any detail in the sections below.
ncap2 understands a relatively full-featured language of operations, including loops, conditionals, arrays, and math functions. ncap2 is the most rapidly changing NCO operator and its documentation is incomplete. The distribution file data/ncap2_tst.nco contains an up-to-date overview of its syntax and capabilities. The data/*.nco distribution files (especially bin_cnt.nco, psd_wrf.nco, and rgr.nco) contain in-depth examples of ncap2 solutions to complex problems.

SYNTAX

[-s algebra] [-S fl.nco] [-t thr_nbr] [-v]
input-file [output-file]

DESCRIPTION

ncap2 arithmetically processes netCDF files. The processing instructions are contained either in the NCO script file fl.nco or in a sequence of command line arguments. The options ‘-s’ (or long options ‘--spt’ or ‘--script’) are used for in-line scripts and ‘-S’ (or long options ‘--fl_spt’ or ‘--script-file’) are used to provide the filename where (usually multiple) scripting commands are pre-stored. ncap2 was written to perform arbitrary algebraic transformations of data and archive the results as easily as possible. See Section 3.25 [Missing Values], page 71, for treatment of missing values. The results of the algebraic manipulations are called derived fields.

Unlike the other operators, ncap2 does not accept a list of variables to be operated on as an argument to ‘-v’ (see Section 3.11 [Subsetting Files], page 40). Rather, the ‘-v’ switch takes no arguments and indicates that ncap2 should output only user-defined variables. ncap2 neither accepts nor understands the -x switch. NB: As of 20120515, ncap2 is unable to append to files that already contain the appended dimensions.

Defining new variables in terms of existing variables is a powerful feature of ncap2. Derived fields inherit the metadata (i.e., attributes) of their ancestors, if any, in the script or input file. When the derived field is completely new (no identically-named ancestors exist), then it inherits the metadata (if any) of the left-most variable on the right hand side of the defining expression. This metadata inheritance is called attribute propagation. Attribute propagation is intended to facilitate well-documented data analysis, and we welcome suggestions to improve this feature.

The only exception to this rule of attribute propagation is in cases of left hand casting (see Section 4.1.4 [Left hand casting], page 122). The user must manually define the proper metadata for variables defined using left hand casting.

\[\text{nccap2 is the successor to ncap which was put into maintenance mode in November, 2006. This documentation refers to ncap2, which has a superset of the ncap functionality. Eventually ncap will be deprecated in favor ncap2. ncap2 may be renamed ncap in 2013.}\]
4.1.1 Syntax of ncap2 statements

Mastering ncap2 is relatively simple. Each valid statement consists of standard forward algebraic expression. The fl.nco, if present, is simply a list of such statements, whitespace, and comments. The syntax of statements is most like the computer language C. The following characteristics of C are preserved:

Array syntax
Arrays elements are placed within [] characters;

Array indexing
Arrays are 0-based;

Array storage
Last dimension is most rapidly varying;

Assignment statements
A semi-colon ';' indicates the end of an assignment statement.

Comments
Multi-line comments are enclosed within /* */ characters. Single line comments are preceded by // characters.

Nesting
Files may be nested in scripts using #include script. Note that the #include command is not followed by a semi-colon because it is a pre-processor directive, not an assignment statement. The filename script is interpreted relative to the run directory.

Attribute syntax
The at-sign @ is used to delineate an attribute name from a variable name.

4.1.2 Expressions

Expressions are the fundamental building block of ncap2. Expressions are composed of variables, numbers, literals, and attributes. The following C operators are “overloaded” and work with scalars and multi-dimensional arrays:

Arithmetic Operators: * / % + - ^
Binary Operators: > >= < <= == != == || && >> <<
Unary Operators: + - ++ -- !
Conditional Operator: exp1 ? exp2 : exp3
Assign Operators: = += -= /= *=

In the following section a variable also refers to a number literal which is read in as a scalar variable:

Arithmetic and Binary Operators
Consider var1 'op' var2

Precision
- When both operands are variables, the result has the precision of the higher precision operand.
- When one operand is a variable and the other an attribute, the result has the precision of the variable.
• When both operands are attributes, the result has the precision of the more precise attribute.
• The exponentiation operator “^” is an exception to the above rules. When both operands have type less than NC_FLOAT, the result is NC_FLOAT. When either type is NC_DOUBLE, the result is also NC_DOUBLE.

Rank
• The Rank of the result is generally equal to Rank of the operand that has the greatest number of dimensions.
• If the dimensions in var2 are a subset of the dimensions in var1 then it’s possible to make var2 conform to var1 through broadcasting and or dimension reordering.
• Broadcasting a variable means creating data in non-existing dimensions by copying data in existing dimensions.
• More specifically: If the numbers of dimensions in var1 is greater than or equal to the number of dimensions in var2 then an attempt is made to make var2 conform to var1, else var1 is made to conform to var2. If conformance is not possible then an error message will be emitted and script execution will cease.

Even though the logical operators return True(1) or False(0) they are treated in the same way as the arithmetic operators with regard to precision and rank.

Examples:

```
dimensions: time=10, lat=2, lon=4
Suppose we have the two variables:

double P(time,lat,lon);
float PZ0(lon,lat); // PZ0=1,2,3,4,5,6,7,8;

Consider now the expression:
PZ=P-PZ0

PZ0 is made to conform to P and the result is
PZ0 =
  1,3,5,7,2,4,6,8,
  1,3,5,7,2,4,6,8,
  1,3,5,7,2,4,6,8,
  1,3,5,7,2,4,6,8,
  1,3,5,7,2,4,6,8,
  1,3,5,7,2,4,6,8,
  1,3,5,7,2,4,6,8,
  1,3,5,7,2,4,6,8,

Once the expression is evaluated then PZ will be of type double;
```
Consider now

\[
\begin{align*}
\text{start} &= \text{four-att_var}@\text{double_att}; \quad // \text{start} = -69 \quad \text{and is of type integer;} \\
\text{four_pow} &= \text{four}^3.0f \quad // \text{four_pow}=64 \quad \text{and is of type float} \\
\text{three nw} &= \text{three_dmnn_var_sht}*1.0f; \quad // \text{type is now float} \\
\text{start@n1} &= \text{att_var}@\text{short_att}*\text{att_var}@\text{int_att}; \\
&\quad // \text{start@n1}=5329 \quad \text{and is type int}
\end{align*}
\]

**Binary Operators**

Unlike C the binary operators return an array of values. There is no such thing as short circuiting with the AND/OR operators. Missing values are carried into the result in the same way they are with the arithmetic operators. When an expression is evaluated in an if() the missing values are treated as true.

The binary operators are, in order of precedence:

\[
\begin{align*}
! & \quad \text{Logical Not} \\
<< & \quad \text{Less Than Selection} \\
>> & \quad \text{Greater Than Selection} \\
> & \quad \text{Greater than} \\
>= & \quad \text{Greater than or equal to} \\
< & \quad \text{Less than} \\
<= & \quad \text{Less than or equal to} \\
== & \quad \text{Equal to} \\
!= & \quad \text{Not equal to} \\
&& & \quad \text{Logical AND} \\
|| & \quad \text{Logical OR}
\end{align*}
\]

To see all operators: see Section 4.1.26 [Operator precedence and associativity], page 165

**Examples:**

\[
\begin{align*}
\text{tm1} &= \text{time}>2 \&\& \text{time}<7; \quad // \text{tm1}=0, 0, 1, 1, 1, 1, 0, 0, 0, 0 \quad \text{double} \\
\text{tm2} &= \text{time==3} \mid\mid \text{time}>=6; \quad // \text{tm2}=0, 0, 1, 0, 0, 1, 1, 1, 1, 1 \quad \text{double} \\
\text{tm3} &= \text{int}(!\text{tm1}); \quad // \text{tm3}=1, 1, 0, 0, 0, 1, 1, 1, 1, 1 \quad \text{int} \\
\text{tm4} &= \text{tm1} \&\& \text{tm2}; \quad // \text{tm4}=0, 0, 1, 0, 0, 1, 0, 0, 0, 0 \quad \text{double} \\
\text{tm5} &= !\text{tm4}; \quad // \text{tm5}=1, 1, 0, 1, 0, 1, 1, 1, 1, 1 \quad \text{double}
\end{align*}
\]

**Regular Assign Operator**

\[\text{var1 }=\text{’exp1’}\]

If var1 does not already exist in Output then var1 is written to Output with the values and dimensions from exp1. If var1 already exists in Output, then the only requirement on exp1 is that the number of elements must match the number already on disk. The type of exp1 is converted if necessary to the disk type.
Other Assign Operators ++,-,=,*=/=

`var1 'ass_op' exp1`

If `exp1` is a variable and it doesn’t conform to `var1` then an attempt is made to make it conform to `var1`. If `exp1` is an attribute it must have unity size or else have the same number of elements as `var1`. If `exp1` has a different type to `var1` the it is converted to the `var1` type.

```
    z1=four+=one*=10 // z1=14 four=14 one=10;
    time-=2        // time= -1,0,1,2,3,4,5,6,7,8
```

Increment/Decrement Operators

These work in a similar fashion to their regular C counterparts. If say the variable `four` is input only then the statement `++four` effectively means read `four` from input increment each element by one, then write the new values to Output;

Example:

```
n2=++four;  n2=5, four=5
n3=one--+20; n3=21 one=0;
n4=--time;  n4=time=0.,1.,2.,3.,4.,5.,6.,7.,8.,9.;
```

Conditional Operator ?:

`exp1 ? exp2 : exp3`

The conditional operator (or ternary Operator) is a succinct way of writing an if/then/else. If `exp1` evaluates to true then `exp2` is returned else `exp3` is returned.

Example:

```
weight_avg=weight.avg();
weight_avg@units= (weight_avg == 1 ? "kilo" : "kilos");
PS_nw=PS-(PS.min() > 100000 ? 100000 : 0);
```

Clipping Operators

<< Less-than Clipping

For arrays, the less-than selection operator selects all values in the left operand that are less than the corresponding value in the right operand. If the value of the left side is greater than or equal to the corresponding value of the right side, then the right side value is placed in the result

>> Greater-than Clipping

For arrays, the greater-than selection operator selects all values in the left operand that are greater than the corresponding value in the right operand. If the value of the left side is less than or equal to the corresponding value of the right side, then the right side value is placed in the result.

Example:

```
RDM2=RDM >> 100.0 // 100,100,100,100,126,126,100,100,100,100  double
RDM2=RDM << 90s  // 1, 9, 36, 84, 90, 90, 84, 36, 9, 1   int
```
4.1.3 Dimensions

Dimensions are defined in Output using the `defdim()` function.

```plaintext
defdim("cnt",10);  # Dimension size is fixed by default
defdim("cnt",10,NC_UNLIMITED);  # Dimension is unlimited (record dimension)
defdim("cnt",10,0);   # Dimension is unlimited (record dimension)
defdim("cnt",10,1);   # Dimension size is fixed
defdim("cnt",10,737); # All non-zero values indicate dimension size is fixed
```

This dimension name must then be prefixed with a dollar-sign ‘$’ when referred to in method arguments or left-hand-casting, e.g.,

```plaintext
new_var[$cnt]=time;
temperature[$time,$lat,$lon]=35.5;
temp_avg=temperature.avg($time);
```

The `size` method allows the dimension size to be used in an arithmetic expression:

```plaintext
time_avg=time.total() / $time.size;
```

Increase the size of a new variable by one and set new member to zero:

```plaintext
defdim("cnt_new",$cnt.size+1);
new_var[$cnt_new]=0.0;
new_var(0:($cnt_new.size-2))=old_var;
```

**Dimension Abbreviations**

It is possible to use dimension abbreviations as method arguments:

- $0$ is the first dimension of a variable
- $1$ is the second dimension of a variable
- $n$ is the \(n+1\) dimension of a variable

```plaintext
float four_dmn_rec_var(time,lat,lev,lon);
double three_dmn_var_dbl(time,lat,lon);

four_nw=four_dmn_rev_var.reverse($time,$lon)
four_nw=four_dmn_rec_var.reverse($0,$3);

four_avg=four_dmn_rec_var.avg($lat,$lev);
four_avg=four_dmn_rec_var.avg($1,$2);

three_mw=three_dmn_var_dbl.permute($time,$lon,$lat);
three_mw=three_dmn_var_dbl.permute($0,$2,$1);
```

**ID Quoting**

If the dimension name contains non-regular characters use ID quoting. See Section 4.1.27 [ID Quoting], page 166

```plaintext
defdim("a--list.A",10);
A1['$a--list.A']=30.0;
```
GOTCHA
It is not possible to manually define in Output any dimensions that exist in Input. When a variable from Input appears in an expression or statement its dimensions in Input are automagically copied to Output (if they are not already present)

4.1.4 Left hand casting
The following examples demonstrate the utility of the left hand casting ability of ncap2. Consider first this simple, artificial, example. If lat and lon are one dimensional coordinates of dimensions lat and lon, respectively, then addition of these two one-dimensional arrays is intrinsically ill-defined because whether lat_lon should be dimensioned lat by lon or lon by lat is ambiguous (assuming that addition is to remain a commutative procedure, i.e., one that does not depend on the order of its arguments). Differing dimensions are said to be orthogonal to one another, and sets of dimensions which are mutually exclusive are orthogonal as a set and any arithmetic operation between variables in orthogonal dimensional spaces is ambiguous without further information.

The ambiguity may be resolved by enumerating the desired dimension ordering of the output expression inside square brackets on the left hand side (LHS) of the equals sign. This is called left hand casting because the user resolves the dimensional ordering of the RHS of the expression by specifying the desired ordering on the LHS.

\[ \text{ncap2} -s \ '\text{lat_lon}[\text{lat},\text{lon}]=\text{lat}+\text{lon}' \ \text{in.nc} \ \text{out.nc} \]

\[ \text{ncap2} -s \ '\text{lon_lat}[\text{lon},\text{lat}]=\text{lat}+\text{lon}' \ \text{in.nc} \ \text{out.nc} \]

The explicit list of dimensions on the LHS, [lat,lon] resolves the otherwise ambiguous ordering of dimensions in lat_lon. In effect, the LHS casts its rank properties onto the RHS. Without LHS casting, the dimensional ordering of lat_lon would be undefined and, hopefully, ncap2 would print an error message.

Consider now a slightly more complex example. In geophysical models, a coordinate system based on a blend of terrain-following and density-following surfaces is called a hybrid coordinate system. In this coordinate system, four variables must be manipulated to obtain the pressure of the vertical coordinate: PO is the domain-mean surface pressure offset (a scalar), PS is the local (time-varying) surface pressure (usually two horizontal spatial dimensions, i.e. latitude by longitude), hyam is the weight given to surfaces of constant density (one spatial dimension, pressure, which is orthogonal to the horizontal dimensions), and hybm is the weight given to surfaces of constant elevation (also one spatial dimension). This command constructs a four-dimensional pressure prs_mdp from the four input variables of mixed rank and orthogonality:

\[ \text{ncap2} -s \ '\text{prs_mdp}[\text{time},\text{lat},\text{lon},\text{lev}]=P0*\text{hyam}+PS*\text{hybm}' \ \text{in.nc} \ \text{out.nc} \]

Manipulating the four fields which define the pressure in a hybrid coordinate system is easy with left hand casting.

4.1.5 Arrays and hyperslabs
Generating a regularly spaced one-dimensional array with ncap2 is simple with the array() function. The syntax is

\[ \text{var_out} = \text{array(val_srt, val_ncr, $dmn_nm)}; \quad // \text{One-dimensional output} \]
\texttt{var\_out=\texttt{array}(\texttt{val\_srt,\texttt{val\_ncr,\texttt{var\_tpl}}); // Multi-dimensional output}

where the arguments are the starting value \texttt{val\_srt}, incremental value \texttt{val\_ncr}, and, for one-dimensional output, the single dimension \texttt{$dmn\_nm$}, or, for multi-dimensional output, a template variable \texttt{var\_tpl}, i.e., a variable with the same shape as the desired output. The type of \texttt{var\_out} will be the same as \texttt{val\_srt}. Be sure to encode this type with the appropriate decimal point and floating point suffix when \texttt{val\_srt} is a “naked constant” rather than a variable. For example, to produce an array of shorts (signed two-byte integers), integers (signed four-byte integers), unsigned 64-bit integers, floats, or doubles use

\begin{verbatim}
\texttt{var\_out=\texttt{array}(1s,\texttt{val\_ncr,}\texttt{$dmn\_nm}); // NC\_SHORT array}
\texttt{var\_out=\texttt{array}(1,\texttt{val\_ncr,}\texttt{$dmn\_nm}); // NC\_INT array}
\texttt{var\_out=\texttt{array}(1ull,\texttt{val\_ncr,}\texttt{$dmn\_nm}); // NC\_UINT64 array}
\texttt{var\_out=\texttt{array}(1f,\texttt{val\_ncr,}\texttt{$dmn\_nm}); // NC\_FLOAT array}
\texttt{var\_out=\texttt{array}(1.,\texttt{val\_ncr,}\texttt{$dmn\_nm}); // NC\_DOUBLE array}
\end{verbatim}

Once the associated dimensions have been defined, the start and increment arguments may be supplied as values, mathematical expressions, or variables:

\begin{verbatim}
\texttt{var\_out=\texttt{array}(1,1,\texttt{$time}); // 1,2,3,4,5,6,7,8,9,10}
\texttt{var\_out=\texttt{array}(1+2-2,\texttt{one,}\texttt{$time}); // 1,2,3,4,5,6,7,8,9,10}
\texttt{var\_out=\texttt{array}(1,2,\texttt{three\_dmn\_rec\_var}); // 1,3,5,...155,157,159}
\end{verbatim}

Hyperslabs in \texttt{ncap2} are more limited than hyperslabs with the other NCO operators. \texttt{ncap2} does not understand the shell command-line syntax used to specify multi-slabs, wrapped co-ordinates, negative stride or coordinate value limits. However with a bit of syntactic magic they are all are possible. \texttt{ncap2} accepts (in fact, it requires) \textit{N}-hyperslab arguments for a variable of rank \textit{N}:

\begin{verbatim}
\texttt{var1(\texttt{arg1,\texttt{arg2 ... arg\textit{N}});}
\end{verbatim}

where each hyperslab argument is of the form

\texttt{start:end:stride}

and the arguments for different dimensions are separated by commas. If \texttt{start} is omitted, it defaults to zero. If \texttt{end} is omitted, it defaults to dimension size minus one. If \texttt{stride} is omitted, it defaults to one.

If a single value is present then it is assumed that that dimension collapses to a single value (i.e., a cross-section). The number of hyperslab arguments MUST equal the variable’s rank.

**Hyperslabs on the Right Hand Side of an assign**

A simple 1D example:

\begin{verbatim}
($\texttt{time.size=10}$)
\texttt{od[$\texttt{time}]=\{20,22,24,26,28,30,32,34,36,38\};}
\texttt{od(7); // 34}
\texttt{od(7:); // 34,36,38}
\end{verbatim}
od(:7);  // 20,22,24,26,28,30,32,34
od(:4);   // 20,28,36
od(1:6:2) // 22,26,30
od(:)     // 20,22,24,26,28,30,32,34,36,38

A more complex three dimensional example:

($lat.size=2,$lon.size=4)

\[
th[\text{\$time,\$lat,\$lon}] =
\begin{array}{cccccccc}
  1, & 2, & 3, & 4, & 5, & 6, & 7, & 8, \\
  9, & 10, & 11, & 12, & 13, & 14, & 15, & 16, \\
  17, & 18, & 19, & 20, & 21, & 22, & 23, & 24, \\
  33, & 34, & 35, & 36, & 37, & 38, & 39, & 40, \\
  41, & 42, & 43, & 44, & 45, & 46, & 47, & 48, \\
  49, & 50, & 51, & 52, & 53, & 54, & 55, & 56, \\
  -99, & 58, & 59, & 60, & 61, & 62, & 63, & 64, \\
  65, & 66, & 67, & 68, & 69, & 70, & 71, & 72, \\
  -99, & 74, & 75, & 76, & 77, & 78, & 79, & 99 \\
\end{array}
\]

\[
th(1,1,3);   // 16
\]
\[
th(2,0,:);   // 17, 18, 19, 20
\]
\[
th(:,1,3);  // 8, 16, 24, -99, 40, 48, 56, 64, 72, -99
\]
\[
th(:,5,:),0:3:2); // 1, 3, 5, 7, 41, 43, 45, 47
\]

If hyperslab arguments collapse to a single value (a cross-section has been specified), then that dimension is removed from the returned variable. If all the values collapse then a scalar variable is returned. So, for example, the following is valid:

\[
\text{th\\_nw=th(0,:,:)+th(9,:,:)};
\]

// th\\_nw has dimensions $lon,$lat
// NB: the time dimension has become degenerate

The following is invalid:

\[
\text{th\\_nw=th(0,:,:0:1)+th(9,:,:0:1)};
\]

because the $lon$ dimension now only has two elements. The above can be calculated by using a LHS cast with $lon\\_nw$ as replacement dim for $lon$:

\[
defdim("lon\\_nw",2);
\]
\[
\text{th\\_nw[$lat,$lon\\_nw]=th(0,:,:0:1) +th(9,:,:0:1)};
\]

**Hyperslabs on the Left Hand Side of an assign**

When hyperslabing on the LHS, the expression on the RHS must evaluate to a scalar or a variable/attribute with the same number of elements as the LHS hyperslab. Set all elements of the last record to zero:

\[
\text{th(9,:,:)=0.0};
\]

Set first element of each lon element to 1.0:

\[
\text{th(:,0,:0)=1.0};
\]
One may hyperslab on both sides of an assign. For example, this sets the last record to the first record:

\[ \text{th}(9,:,:)=\text{th}(0,:,:); \]

Say \( \text{th0} \) represents pressure at height=0 and \( \text{th1} \) represents pressure at height=1. Then it is possible to insert these hyperslabs into the records

\[
\text{prs}[:,0,:,:]=\text{th0};
\]
\[
\text{prs}[:,1,:,:]=\text{th1}
\]

**Reverse method**

Use the `reverse()` method to reverse a dimension’s elements in a variable with at least one dimension. This is equivalent to a negative stride, e.g.,

\[
\text{th\_rv}=\text{th}(1,:,:).reverse($\text{lon}$);
\]
\[
\text{od\_rv}=\text{od}.reverse($\text{time}$);
\]

**Permute method**

Use the `permute()` method to swap the dimensions of a variable. The number and names of dimension arguments must match the dimensions in the variable. If the first dimension in the variable is of record type then this must remain the first dimension. If you want to change the record dimension then consider using `ncpdq`.

Consider the variable:

```plaintext
float three_dmn_var(lat,lev,lon);
three_dmn_var_prm=three_dmn_var.permute($\text{lon}$,$\text{lat}$,$\text{lev}$);
// The permuted values are
three_dmn_var_prm=
0,4,8,
12,16,20,
1,5,9,
13,17,21,
2,6,10,
14,18,22,
3,7,11,
15,19,23;
```

### 4.1.6 Attributes

Attributes are referred to by `var_nm@att_nm`

All the following are valid statements:

```plaintext
global@text="Test Attributes"; /* Assign a global variable attribute */
a1[$\text{time}$]=time*20;
a1@long_name="Kelvin";
a1@min=a1.min();
a1@max=a1.max();
a1@min++;
--a1@max; q
```
\begin{verbatim}
a1(0)=a1@min;
a1($time.size-1)=a1@max;

A value list can be used on the RHS of an assign...
a1@trip1={1,2,3};
a1@triplet={a1@min,(a1@min+a1@max)/2,a1@max};
\end{verbatim}

The netCDF specification allows all attribute types to have a size greater than one. The maximum is defined by NC_MAX_ATTRS. The following is an ncdump of the metadata for variable a1

\begin{verbatim}
double a1(time);
a1:long_name = "Kelvin";
a1:max = 199.;
a1:min = 21.;
a1:trip1 = 1, 2, 3;
a1:triplet = 21., 110., 199.;
\end{verbatim}

The size() method can be used with attributes. For example, to save an attribute text string in a variable,

\begin{verbatim}
defdim("sng_len", a1@long_name.size());
sng_arr[$sng_len]=a1@long_name; // sng_arr now contains "Kelvin"
\end{verbatim}

Attributes defined in a script are stored in memory and are written to Output after script completion. To stop the attribute being written use the ram_delete() method or use a bogus variable name.

\textbf{Attribute Propagation and Inheritance}

- Attribute propagation occurs in a regular assign statement. The variable being defined on the LHS gets copies of the attributes from the leftermost variable on the RHS.
- Attribute Inheritance: The LHS variable “inherits” attributes from an Input variable with the same name
- It is possible to have a regular assign statement for which both propagation and inheritance occur.

// prs_mdp inherits attributes from P0:
prs_mdp[time,lat,lon,lev]=P0*hyam+hybm*PS;
// th_min inherits attributes from three_dmn_var_dbl:
th_min=1.0 + 2*three_dmn_var_dbl.min($time);

If the attribute name contains non-regular characters use ID quoting. See see Section 4.1.27 [ID Quoting], page 166

'\texttt{b..m1@c--lost}=23;'

\textbf{4.1.7 Number literals}

The table below lists the postfix character(s) to add to a number literal for type cohesion. To use the new netCDF4 types NCO must be compiled/linked to the netCDF4 library and the output file must be HDF5.
n1[$time]=1UL; // n1 will now be of type @code{NC_UINT}
n2[$lon]=4b; // n2 will be of type @code{NC_BYTE}
n3[$lat]=5ull; // n3 will be of type @code{NC_UINT64}
n3@a1=6.0d; // attribute will be of type @code{NC_DOUBLE}
n3@a2=-666L; // attribute will be of type @code{NC_INT}

A floating point number without a postfix will default to @code{NC_DOUBLE}. An integer without a postfix will default to type @code{NC_INT}. There is no postfix for characters, use a quoted string instead.
n4[$rlev]=0.1 // n4 will be of type @code{NC_DOUBLE}
n5[$lon_grd]=2.0 // n5 will be of type @code{NC_DOUBLE}
n6[$gds_crd]=2e3; // n6 will be of type @code{NC_DOUBLE}
n7[$gds_crd]=2e3f; // n7 will be of type @code{NC_FLOAT}
n6@a1=41; // attribute will be of type @code{NC_INT}
n6@a2=-21; // attribute will be of type @code{NC_INT}
n6@units="kelvin" // attribute will be of type @code{NC_CHAR}

netCDF3/4 Types
b|B NC_BYTE, a signed 1-byte integer
none NC_CHAR, an ISO/ASCII character
s|S NC_SHORT, a signed 2-byte integer
l|L NC_INT, a signed 4-byte integer
f|F NC_FLOAT, a single-precision (4-byte) floating point number
d|D NC_DOUBLE, a double-precision (8-byte) floating point number

netCDF4 Types
ub|UB NC_UBYTE, an unsigned 1-byte integer
us|US NC_USHORT, an unsigned 2-byte integer
u|U|ull|UL NC_UINT, an unsigned 4-byte integer
ll|LL NC_INT64, a signed 8-byte integer
ull|ULL NC_UINT64, an unsigned 8-byte integer

4.1.8 if statement
The syntax of the if statement is similar to its C counterpart. The Conditional Operator (ternary operator) has also been implemented.

```c
if(exp1)
  stmt1;
else if(exp2)
  stmt2;
else
  stmt3;
```
# Can use code blocks as well:
if(exp1){
    stmt1;
    stmt1a;
    stmt1b;
}else if(exp2)
    stmt2;
else{
    stmt3;
    stmt3a;
    stmt3b;
}

For a variable or attribute expression to be logically true all its non-missing value elements must be logically true, i.e., non-zero. The expression can be of any type. Unlike C there is no short-circuiting of an expression with the OR (||) and AND (&&) operators. The whole expression is evaluated regardless if one of the AND/OR operands are True/False.

# Simple example
if(time>0)
    print("All values of time are greater than zero\n");
else if( time<0)
    print("All values of time are less than zero\n");
else {
    time_max=time.max();
    time_min=time.min();
    print("min value of time=");print(time_min,"%f");
    print("max value of time=");print(time_max,"%f");
}

# Example from ddra.nco
if(fl_typ==fl_typ_gcm){
    var_nbr_apx=32;
    lmn_nbr=1.0*var_nbr_apx*varsz_gcm_4D; /* [nbr] Variable size */
    if(nco_op_typ==nco_op_typ_avg){
        lmn_nbr_avg=1.0*var_nbr_apx*varsz_gcm_4D; // Block size
        lmn_nbr_wgt=dmnsz_gcm_lat; /* [nbr] Weight size */
    } // !nco_op_typ_avg
}else if(fl_typ==fl_typ_stl){
    var_nbr_apx=8;
    lmn_nbr=1.0*var_nbr_apx*varsz_stl_2D; /* [nbr] Variable size */
    if(nco_op_typ==nco_op_typ_avg){
        lmn_nbr_avg=1.0*var_nbr_apx*varsz_stl_2D; // Block size
        lmn_nbr_wgt=dmnsz_stl_lat; /* [nbr] Weight size */
    } // !nco_op_typ_avg
} // !fl_typ
Conditional Operator

    // netCDF4 needed for this example
    th_nw=(three_dmn_var_sht >= 0 ? three_dmn_var_sht.uint() : \
            three_dmn_var_sht.int());

4.1.9 print statement

    print(variable_name/attribute name/string, format string);

The print function takes a variable name or attribute name or a quoted string and prints
the contents in a in a similar fashion to ncks -H. There is also an optional C-language
style format string argument. Currently the print function cannot print RAM variables or
expressions such as 'print(var_msk*3+4)'. To print an expression, first evaluate it as a
non-RAM variable (so it will be saved and can be printed), and then print the variable.

examples

    print(lon);
    lon[0]=0
    lon[1]=90
    lon[2]=180
    lon[3]=270

    print(lon_2D_rrg,"%3.2f, ");
    0.00,0.00,180.00,0.00,180.00,0.00,180.00,0.00,

    print(mss_val_fst@_FillValue);
    mss_val_fst@_FillValue, size = 1 NC_FLOAT, value = -999

    print("This function \t is monotonic\n");
    This function is    monotonic

4.1.10 Missing values ncap2

Missing values operate slightly differently in ncap2 Consider the expression where op is any
of the following operators (excluding '=')

    Arithmetic operators ( * / % + - ^ )
    Binary Operators   ( >, >= <, <= ==, !=,==,||,&&, >>,<< )
    Assign Operators   ( + =, -= /=, *= )

    var1 'op' var2

If var1 has a missing value then this is the value used in the operation, otherwise the missing
value for var2 is used. If during the element-by-element operation an element from either
operand is equal to the missing value then the missing value is carried through. In this way
missing values 'percolate' or propagate through an expression.

Missing values associated with Output variables are stored in memory and are written to
disk after the script finishes. During script execution its possible (and legal) for the missing
value of a variable to take on several different values.
Consider the variable:

```c
int rec_var_int_mss_val_int(time); = -999, 2, 3, 4, 5, 6, 7, 8, -999, -999;
rec_var_int_mss_val_int::_FillValue = -999;
```

```c
n2 = rec_var_int_mss_val_int + rec_var_int_mss_val_int.reverse($time);
```

```c
n2 = -999, -999, 11, 11, 11, 11, 11, 999, -999;
```

The following methods manipulate missing value information associated with a variable. They only work on variables in Output.

**set_miss(expr)**

The numeric argument `expr` becomes the new missing value, overwriting the old missing value, if any. The argument given is converted if necessary to the variable type. NB: This only changes the missing value attribute. Missing values in the original variable remain unchanged, and thus are no longer considered missing values. They are "orphaned". Thus `set_miss()` is normally used only when creating new variables. The intrinsic function `change_miss()` (see below) is typically used to edit values of existing variables.

**change_miss(expr)**

Sets or changes (any pre-existing) missing value attribute and missing data values to `expr`. NB: This is an expensive function since all values must be examined. Use this function when changing missing values for pre-existing variables.

**get_miss()**

Returns the missing value of a variable. If the variable exists in Input and Output then the missing value of the variable in Output is returned. If the variable has no missing value then an error is returned.

**delete_miss()**

Delete the missing value associated with a variable.

**number_miss()**

Count the number of missing values a variable contains.

```c
th = three_dmn_var_dbl;
th.change_miss(-1e10d);
/* Set values less than 0 or greater than 50 to missing value */
where(th < 0.0 || th > 50.0) th = th.get_miss();
```

```c
# Another example:
new[$time,$lat,$lon]=1.0;
new.set_miss(-997.0);
```

```c
// Extract only elements divisible by 3
where (three_dmn_var_dbl%3 == 0)
   new=three_dmn_var_dbl;
elsewhere
   new=new.get_miss();
```
// Print missing value and variable summary
mss_val_nbr=three_dmn_var_dbl.number_miss();
print(three_dmn_var_dbl@_FillValue);
print("Number of missing values in three_dmn_var_dbl: ");
print(mss_val_nbr,"%d");
print(three_dmn_var_dbl);

4.1.11 Methods and functions

The convention within this document is that methods can be used as functions. However, functions are not and cannot be used as methods. Methods can be daisy-chained and their syntax is cleaner than functions. Method names are reserved words and CANNOT be used as variable names. The command `ncap2 -f` shows the complete list of methods available on your build.

\[
\begin{align*}
    n2 &= \sin(\theta) \\
    n2 &= \theta \cdot \sin() \\
    n2 &= \sin(\theta)^2 + \cos(\theta)^2 \\
    n2 &= \theta \cdot \sin() \cdot \text{pow}(2) + \theta \cdot \cos()^2
\end{align*}
\]

This statement chains together methods to convert `three_dmn_var_sht` to type double, average it, then convert this back to type short:

\[
\text{three_avg}=\text{three_dmn_var_sht} \cdot \text{double().avg().short();}
\]

Aggregate Methods

These methods mirror the averaging types available in `ncwa`. The arguments to the methods are the dimensions to average over. Specifying no dimensions is equivalent to specifying all dimensions i.e., averaging over all dimensions. A masking variable and a weighting variable can be manually created and applied as needed.

\[
\begin{align*}
    \text{avg()} &= \text{Mean value} \\
    \text{sqravg()} &= \text{Square of the mean} \\
    \text{avgsqr()} &= \text{Mean of sum of squares} \\
    \text{max()} &= \text{Maximum value} \\
    \text{min()} &= \text{Minimum value} \\
    \text{mabs()} &= \text{Maximum absolute value} \\
    \text{mebs()} &= \text{Mean absolute value} \\
    \text{mibs()} &= \text{Minimum absolute value} \\
    \text{rms()} &= \text{Root-mean-square (normalize by } N) \\
    \text{rmssdn()} &= \text{Root-mean square (normalize by } N-1) \\
    \text{ttl() or total()} &= \text{Sum of values}
\end{align*}
\]
// Average a variable over time
four_time_avg=four_dmn_rec_var($time);

Packing Methods
For more information see Section 3.32 [Packed data], page 92 and see Section 4.8 [ncpdq
netCDF Permute Dimensions Quickly], page 202

pack() & pack_short()
   The default packing algorithm is applied and variable is packed to NC_SHORT
pack_byte()
   Variable is packed to NC_BYTE
pack_short()
   Variable is packed to NC_SHORT
pack_int()
   Variable is packed to NC_INT
unpack() The standard unpacking algorithm is applied.

Basic Methods
These methods work with variables and attributes. They have no arguments
size() Total number of elements
ndims() Number of dimensions in variable
type() Returns the netcdf type (see previous section)

Utility Methods
These functions are used to manipulate missing values and RAM variables. see
Section 4.1.10 [Missing values ncap2], page 129

set_miss(expr)
   Takes one argument the missing value. Sets or overwrites the existing missing
   value. The argument given is converted if necessary to the variable type
change_miss(expr)
   Changes the missing value elements of the variable to the new missing value
   (n.b. an expensive function).
get_miss()
   Returns the missing value of a variable in Input or Output
delete_miss()
   Deletes the missing value associated with a variable.
ram_write()
   Writes a RAM variable to disk i.e., converts it to a regular disk type variable
ram_delete()
   Deletes a RAM variable or an attribute
PDQ Methods
See see Section 4.8 [ncpdq netCDF Permute Dimensions Quickly], page 202

reverse(dim args)
Reverses the dimension ordering of elements in a variable.

permute(dim args)
Re-shapes variables by re-ordering the dimensions. All the dims of the variable must be specified in the arguments. A limitation of this permute (unlike ncpdq) is that the record dimension cannot be re-assigned.

// Swap dimensions about and reorder along lon
lat_2D_rrg_new=lat_2D_rrg.permute($lon,$lat).reverse($lon);
lat_2D_rrg_new=0,90,-30,30,-30,30,-90,0

Type Conversion Methods
These methods allow ncap2 to convert variables and attributes to the different netcdf types. For more details on automatic and manual type conversion see (see Section 3.34 [Type Conversion], page 99). You may only use the new netCDF4 types if you have compiled/links NCO with the netCDF4 library and the Output file is HDF5.

netCDF3/4 Types
byte() convert to NC_BYTE, a signed 1-byte integer
char() convert to NC_CHAR, an ISO/ASCII character
short() convert to NC_SHORT, a signed 2-byte integer
int() convert to NC_INT, a signed 4-byte integer
float() convert to NC_FLOAT, a single-precision (4-byte) floating point number
double() convert to NC_DOUBLE, a double-precision (8-byte) floating point number

netCDF4 Types
ubyte() convert to NC_UBYTE, an unsigned 1-byte integer
ushort() convert to NC_USHORT, an unsigned 2-byte integer
uint() convert to NC_UINT, an unsigned 4-byte integer
int64() convert to NC_INT64, a signed 8-byte integer
uint64() convert to NC_UINT64, an unsigned 8-byte integer

Intrinsic Mathematical Methods
The list of mathematical methods is system dependant. For the full list see Section 4.1.25 [Intrinsic mathematical methods], page 163

All the mathematical methods take a single argument except atan2() and pow() which take two. If the operand type is less than float then the result will be of type float. Arguments of type double yield results of type double. Like the other methods, you are free to use the mathematical methods as functions.
n1=pow(2,3.0f) // n1 type float
n2=atan2(2,3.0) // n2 type double
n3=1/(three_dmn_var_dbl.cos().pow(2))-tan(three_dmn_var_dbl)^2; // n3 type double

4.1.12 RAM variables
Unlike regular variables, RAM variables are never written to disk. Hence using RAM variables in place of regular variables (especially within loops) significantly increases execution speed. Variables that are frequently accessed within for or where clauses provide the greatest opportunities for optimization. To declare and define a RAM variable simply prefix the variable name with an asterisk (*) when the variable is declared/initialized. To delete a RAM variables (and recover their memory) use the `ram_delete()` method. To write a RAM variable to disk (like a regular variable) use `ram_write()`.

\[
*\text{temp}[$\text{time},\text{lat},\text{lon}]=10.0; \hspace{1em} \text{Cast}
\]

\[
*\text{temp\_avg}=\text{temp\_avg}($\text{time}); \hspace{1em} \text{Regular assign}
\]

\[
\text{temp}\_\text{ram\_delete();} \hspace{1em} \text{Delete RAM variable}
\]

\[
\text{temp\_avg}\_\text{ram\_write();} \hspace{1em} \text{Write Variable to output}
\]

// Create and increment a RAM variable from "one" in Input
*\text{one}++;
// Create RAM variables from the variables three and four in Input.
// Multiply three by 10 and add it to four.
*\text{four}+=*\text{three}*=10; // three=30, four=34

4.1.13 Where statement
A `where()` combines the definition and application of a mask all in one go and can lead to succinct code. The full syntax of a `where()` statement is as follows:

\[
\text{where(mask)} \begin{cases} 
\text{var1}=\text{expr1}; \\
\text{elsewhere} \\
\text{var1}=\text{expr2};
\end{cases}
\]

\[
\text{where(mask)}\{ 
\text{var1}=\text{expr1}; \\
\text{var2}=\text{expr2}; \\
\ldots 
\} \text{elsewhere\{ 
\text{var1}=\text{expr3} \\
\text{var2}=\text{expr4} \\
\text{var3}=\text{expr5}; \\
\ldots
\\}
\]

- The only expression allowed in the predicate of a where is assign, i.e., 'var=expr'. This assign differs from a regular `ncap2` assign. The LHS var must already exist in Input or
Output. The RHS expression must evaluate to a scalar or a variable/attribute of the same size as the LHS variable.

- Consider when both the LHS and RHS are variables: For every element where mask condition is True, the corresponding LHS variable element is re-assigned to its partner element on the RHS. In the elsewhere part the mask is logically inverted and the assign process proceeds as before.
- If the mask dimensions are a subset of the LHS variable's dimensions, then it is made to conform; if it cannot be made to conform then script execution halts.
- Missing values in the mask evaluate to False in the where code/block statement and to True in the elsewhere block/statement. LHS variable elements set to missing value are not re-assigned. For these reasons, do not explicitly reference missing values in the masking condition, e.g., \texttt{where(foo=foo.get\_missing()) foo=1;} will not work as expected.

Example: Consider the variables \texttt{float lon\_2D\_rct(lat,lon);} and \texttt{float var\_msk(lat,lon);}. Suppose we wish to multiply by two the elements for which \texttt{var\_msk} equals 1:

\begin{verbatim}
where(var\_msk==1) lon\_2D\_rct=2*lon\_2D\_rct;
\end{verbatim}

Suppose that we have the variable \texttt{int RDM(time)} and that we want to set its values less than 8 or greater than 80 to 0:

\begin{verbatim}
where(RDM < 8 \texttt{||} RDM > 80) RDM=0;
\end{verbatim}

Consider irregularly gridded data, described using rank 2 coordinates: \texttt{double lat(south\_north,east\_west), double lon(south\_north,east\_west), double temperature(south\_north,east\_west)}. This type of structure is often found in regional weather/climate model (such as WRF) output, and in satellite swath data. For this reason we call it “Swath-like Data”, or SLD. To find the average temperature in a region bounded by \texttt{[lat\_min,\texttt{lat\_max}]} and \texttt{[lon\_min,\texttt{lon\_max}]}:

\begin{verbatim}
temperature\_msk[$south\_north,$east\_west]=0.0;
where((lat >= lat\_min \texttt{&&} lat <= lat\_max) \texttt{&&} (lon >= lon\_min \texttt{&&} lon <= lon\_max))
temperature\_msk=temperature;
elsewhere
  temperature\_msk=temperature@\_FillValue;
\end{verbatim}

\begin{verbatim}
temp\_avg=temperature\_msk.avg();
temp\_max=temperature\_msk.max();
\end{verbatim}

For North American Regional Reanalysis (NARR) data (example \texttt{dataset}) the procedure looks like this

\begin{verbatim}
ncap2 -O -v -S ~/narr.nco ${DATA}/hdf/narr\_uwnd.199605.nc ~/foo.nc
\end{verbatim}

where \texttt{narr.nco} is an \texttt{ncap2} script like this:

\begin{verbatim}
/* North American Regional Reanalysis (NARR) Statistics
NARR stores grids with 2-D latitude and longitude, aka Swath-like Data (SLD)
\end{verbatim}
Here we work with three variables:
lat(y,x), lon(y,x), and uwnd(time,level,y,x);
To study sub-regions of SLD, we use masking techniques:
1. Define mask as zero times variable to be masked
   Then mask automatically inherits variable attributes
   And average below will inherit mask attributes
2. Optionally, create mask as RAM variable (as below with asterisk *)
   NCO does not write RAM variable to output
   Masks are often unwanted, and can be big, so this speeds execution
3. Example could be extended to preserve mean lat and lon of sub-region
   Follow uwnd example to do this: lat_msk=0.0*lat ... lat_avg=lat.avg($y,$x) */
   *uwnd_msk=0.0*uwnd;
   where((lat >= 35.6 && lat <= 37.0) && (lon >= -100.5 && lon <= -99.0))
   uwnd_msk=uwnd;
   elsewhere
   uwnd_msk=uwnd@_FillValue;

   // Average only over horizontal dimensions x and y (preserve level and time)
   uwnd_avg=uwnd_msk.avg($y,$x);

Stripped of comments and formatting, this example is a three-statement script executed
by a one-line command. NCO needs only this meagre input to unpack and copy the input
data and attributes, compute the statistics, and then define and write the output file. Unless the comments pointed out that wind variable (uwnd) was four-dimensional and the latitude/longitude grid variables were both two-dimensional, there would be no way to tell. This shows how NCO hides from the user the complexity of analyzing multi-dimensional SLD. We plan to extend such SLD features to more operators soon.

4.1.14 Loops
ncap2 supplies for() loops and while() loops. They are completely unoptimized so use
them only with RAM variables unless you want thrash your disk to death. To break out of
a loop use the break command. To iterate to the next cycle use the continue command.

    // Set elements in variable double temp(time,lat)
    // If element < 0 set to 0, if element > 100 set to 100
    *sz_idx=$time.size;
    *sz_jdx=$lat.size;
    for(*idx=0;idx<sz_idx;idx++)
        for(*jdx=0;jdx<sz_jdx;jdx++)
            if(temp(idx,jdx) > 100) temp(idx,jdx)=100.0;
            else if(temp(idx,jdx) < 0) temp(idx,jdx)=0.0;

    // Are values of co-ordinate variable double lat(lat) monotonic?
    *sz=$lat.size;
    for(*idx=1;idx<sz;idx++)
        if(lat(idx)-lat(idx-1) < 0.0) break;
if(idx == sz) print("lat co-ordinate is monotonic\n");
    else print("lat co-ordinate is NOT monotonic\n");

    // Sum odd elements
*idx=0;
*sz=$lat_nw.size;
*sum=0.0;

    while(idx<sz){
        if(lat(idx)%2) sum+=lat(idx);
        idx++;
    }

    ram_write(sum);
    print("Total of odd elements ");print(sum);print("\n");

4.1.15 Include files

The syntax of an include-file is:

    #include "script.nco"

The script filename is searched relative to the run directory. It is possible to nest include
files to an arbitrary depth. A handy use of include files is to store often used constants.
Use RAM variables if you do not want these constants written to output-file.

    // script.nco
    // Sample file to #include in ncap2 script
    *pi=3.1415926535; // RAM variable, not written to output
    *h=6.62607095e-34; // RAM variable, not written to output
    e=2.71828; // Regular (disk) variable, written to output

4.1.16 sort methods

In ncap2 there are multiple ways to sort data. Beginning with NCO 4.1.0 (March, 2012),
ncap2 support six sorting functions:

    var_out=sort(var_in,&srt_map); // Ascending sort
    var_out=asort(var_in,&srt_map); // Ascending sort
    var_out=dsort(var_in,&srt_map); // Desending sort
    var_out=remap(var_in,srt_map); // Apply srt_map to var_in
    var_out=unmap(var_in,srt_map); // Reverse what srt_map did to var_in
    dsr_map=invert_map(srt_map); // Produce "de-sort" map that inverts srt_map

The first two functions, sort() and asort() sort, in ascending order, all the elements of
var_in (which can be a variable or attribute) without regard to any dimensions. The third
function, dsort() does the same but sorts in descending order. Remember that ascending
and descending sorts are specified by asort() and dsort(), respectively.

These three functions are overloaded to take a second, optional argument called the sort
map srt_map, which should be supplied as a call-by-reference variable, i.e., preceded with
an ampersand. If the sort map does not yet exist, then it will be created and returned as an integer type the same shape as the input variable.

The output var_out of each sort function is a sorted version of the input, var_in. The output var_out of the two mapping functions the result of applying (with remap()) or un-applying (with unmap()) the sort map srt_map to the input var_in. To apply the sort map with remap() the size of the variable must be exactly divisible by the size of the sort map.

The final function invert_map() returns the so-called de-sorting map dsr_map which is inverse map of the input map srt_map. This gives the user access to both the forward and inverse sorting maps which can be useful in special situations.

```cpp
a1[$time]={10,2,3,4,6,5,7,3,4,1};
a1_sort=sort(a1);
print(a1_sort);
// 1, 2, 3, 3, 4, 4, 5, 6, 7, 10;

a2[$lon]={2,1,4,3};
a2_sort=sort(a2,&a2_map);
print(a2);
// 1, 2, 3, 4
print(a2_map);
// 1, 0, 3, 2;
```

If the map variable does not exist prior to the sort() call, then it will be created with the same shape as the input variable and be of type NC_INT. If the map variable already exists, then the only restriction is that it be of at least the same size as the input variable.

To apply a map use remap(var_in,srt_map).

```cpp
defdim("nlat",5);

a3[$lon]={2,5,3,7};
a4[$nlat,$lon]={
  1, 2, 3, 4,
  5, 6, 7, 8,
  9,10,11,12,
  13,14,15,16,
  17,18,19,20};

a3_sort=sort(a3,&a3_map);
print(a3_map);
// 0, 2, 1, 3;

a4_sort=remap(a4,a3_map);
print(a4_sort);
// 1, 3, 2, 4,
// 5, 7, 6, 8,
// 9,11,10,12,
// 13,15,14,16,
```
a3_map2[$nlat] = {4,3,0,2,1};

a4_sort2 = remap(a4, a3_map2);
print(a4_sort2);
// 3, 5, 4, 2, 1
// 8, 10, 9, 7, 6,
// 13, 15, 14, 12, 11,
// 18, 20, 19, 17, 16

As in the above example you may create your own sort map. To sort in descending order, apply the reverse() method after the sort().

Here is an extended example of how to use ncap2 features to hyperslab an irregular region based on the values of a variable not a coordinate. The distinction is crucial: hyperslabbing based on dimensional indices or coordinate values is straightforward. Using the values of single or multi-dimensional variable to define a hyperslab is quite different.

cat > ~/ncap2_foo.nco << 'EOF'
// Purpose: Save irregular 1-D regions based on variable values

// Included in NCO User Guide at http://nco.sf.net/nco.html#sort

/* NB: Single quotes around EOF above turn off shell parameter expansion in "here documents". This in turn prevents the need for protecting dollarsign characters in NCO scripts with backslashes when the script is cut-and-pasted (aka "moused") from an editor or e-mail into a shell console window */

/* Copy coordinates and variable(s) of interest into RAM variable(s)
Benefits:
1. ncap2 defines writes all variables on LHS of expression to disk
   Only exception is RAM variables, which are stored in RAM only
   Repeated operations on regular variables takes more time,
   because changes are written to disk copy after every change.
   RAM variables are only changed in RAM so script works faster
   RAM variables can be written to disk at end with ram_write()
2. Script permutes variables of interest during processing
   Safer to work with copies that have different names
   This discourages accidental, mistaken use of permuted versions
3. Makes this script a more generic template:
   var_in instead of specific variable names everywhere */
*var_in=one_dmn_rec_var;
*crd_in=time;
*dmn_in_sz=$time.size; // [nbr] Size of input arrays

/* Create all other "intermediate" variables as RAM variables
to prevent them from cluttering the output file.
Mask flag and sort map are same size as variable of interest */
*msk_flg=var_in;
*srt_map=var_in;

/* In this example we mask for all values evenly divisible by 3
This is the key, problem-specific portion of the template
Replace this where() condition by that for your problem
Mask variable is Boolean: 1=Meets condition, 0=Fails condition */
where(var_in % 3 == 0) msk_flg=1; elsewhere msk_flg=0;

// print("msk_flg = ");print(msk_flg); // For debugging...

/* The sort() routine is overloaded, and takes one or two arguments
The second argument (optional) is the "sort map" (srt_map below)
Pass the sort map by reference, i.e., prefix with an ampersand
If the sort map does not yet exist, then it will be created and
returned as an integer type the same shape as the input variable.
The output of sort(), on the LHS, is a sorted version of the input
msk_flg is not needed in its original order after sort()
Hence we use msk_flg as both input to and output from sort()
Doing this prevents the need to define a new, unneeded variable */
msk_flg=sort(msk_flg,&srt_map);

// Count number of valid points in mask by summing the one’s
*msk_nbr=msk_flg.total();

// Define output dimension equal in size to number of valid points
defdim("crd_out",msk_nbr);

/* Now sort the variable of interest using the sort map and remap()
The output, on the LHS, is the input re-arranged so that all points
meeting the mask condition are contiguous at the end of the array
Use same srt_map to hyperslab multiple variables of the same shape
Remember to apply srt_map to the coordinate variables */
crd_in=remap(crd_in,srt_map);
var_in=remap(var_in,srt_map);

/* Hyperslab last msk_nbr values of variable(s) of interest */
crd_out[crd_out]=crd_in((dmn_in_sz-msk_nbr):(dmn_in_sz-1));
var_out[crd_out]=var_in((dmn_in_sz-msk_nbr):(dmn_in_sz-1));

/* NB: Even though we created all variables possible as RAM variables,
the original coordinate of interest, time, is written to the output.
I’m not exactly sure why. For now, delete it from the output with:
ncks -O -x -v time ~/foo.nc ~/foo.nc */
EOF
ncap2 -O -v -S ~/ncap2_foo.nco ~/nco/data/in.nc ~/foo.nc
ncks -O -x -v time ~/foo.nc ~/foo.nc
ncks ~/foo.nc

Here is an extended example of how to use ncap2 features to sort multi-dimensional arrays based on the coordinate values along a single dimension.

cat > ~/ncap2_foo.nco << 'EOF'
/* Purpose: Sort multi-dimensional array based on coordinate values
   This example sorts the variable three_dmn_rec_var(time,lat,lon)
   based on the values of the time coordinate. */

// Included in NCO User Guide at http://nco.sf.net/nco.html#sort

// Randomize the time coordinate
time=10.0*gsl_rng_uniform(time);
//print("original randomized time =\n");print(time);

/* The sort() routine is overloaded, and takes one or two arguments
   The first argument is a one dimensional array
   The second argument (optional) is the "sort map" (srt_map below)
   Pass the sort map by reference, i.e., prefix with an ampersand
   If the sort map does not yet exist, then it will be created and
   returned as an integer type the same shape as the input variable.
   The output of sort(), on the LHS, is a sorted version of the input */

time=sort(time,&srt_map);
//print("sorted time (ascending order) and associated sort map =\n");print(time);print(srt_map);

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:

/* sort() always sorts in ascending order
   The associated sort map therefore re-arranges the original,
   randomized time array into ascending order.
   There are two methods to obtain the descending order the user wants
   1) We could solve the problem in ascending order (the default)
      and then apply the reverse() method to re-arrange the results.
   2) We could change the sort map to return things in descending
      order of time and solve the problem directly in descending order. */

// Following shows how to do method one:
7. Use reverse to obtain descending in time order
Loops could accomplish the same thing (exercise left for reader)
However, loops are slow for large datasets */

/* Following index manipulation requires understanding correspondence
between 1-d (unrolled, memory order of storage) and access into that
memory as a multidimensional (3-d, in this case) rectangular array.
Key idea to understand is how dimensionality affects offsets */

// Copy 1-d sort map into 3-d sort map
srt_map_3d=(0*int(three_dmn_rec_var))+srt_map;
// Multiply base offset by factorial of lesser dimensions
srt_map_3d*=$lat.size*$lon.size;
lon_idx=array(0,1,$lon);
lat_idx=array(0,1,$lat)*$lon.size;
lat_lon_idx[$lat,$lon]=lat_idx+lon_idx;
srt_map_3d+=lat_lon_idx;
print("sort map 3d =\n");print(srt_map_3d);

// Use remap() to re-map the data
three_dmn_rec_var=remap(three_dmn_rec_var,srt_map_3d);

// Finally, reverse data so time coordinate is descending
time=time.reverse($time);
//print("sorted time (descending order) =\n");print(time);
three_dmn_rec_var=three_dmn_rec_var.reverse($time);

// Method two: Key difference is srt_map=$time.size-srt_map-1;
EOF
ncap2 -O -v -S ~/ncap2_foo.nco ~/nco/data/in.nc ~/foo.nc

4.1.17 Irregular Grids

NCO is capable of analyzing datasets for many different underlying coordinate grid types.
netCDF was developed for and initially used with grids comprised of orthogonal dimensions
forming a rectangular coordinate system. We call such grids standard grids. It is increas-
ingly common for datasets to use metadata to describe much more complex grids. Let us
first define three important coordinate grid properties: rectangularity, regularity, and fxm.

Grids are regular if the spacing between adjacent is constant. For example, a 4-by-5
degree latitude-longitude grid is regular because the spacings between adjacent latitudes
(4 degrees) are constant as are the (5 degrees) spacings between adjacent longitudes. Spac-
ing in irregular grids depends on the location along the coordinate. Grids such as Gaussian
grids have uneven spacing in latitude (points cluster near the equator) and so are irregular.

Grids are rectangular if the number of elements in any dimension is not a function
of any other dimension. For example, a T42 Gaussian latitude-longitude grid is rectangular
because there are the same number of longitudes (128) for each of the (64) latitudes. Grids
are *non-rectangular* if the elements in any dimension depend on another dimension. Non-rectangular grids present many special challenges to analysis software like NCO.

Wrapped coordinates (see Section 3.20 [Wrapped Coordinates], page 63), such as longitude, are independent of these grid properties (regularity, rectangularity).

The preferred NCO technique to analyze data on non-standard coordinate grids is to create a region mask with ncap2, and then to use the mask within ncap2 for variable-specific processing, and/or with other operators (e.g., ncwa, ncdiff) for entire file processing.

Before describing the construction of masks, let us review how irregularly gridded geoscience data are described. Say that latitude and longitude are stored as \(R\)-dimensional arrays and the product of the dimension sizes is the total number of elements \(N\) in the other variables. Geoscience applications tend to use \(R=1, R=2, \) and \(R=3\).

If the grid is has no simple representation (e.g., discontinuous) then it makes sense to store all coordinates as 1D arrays with the same size as the number of grid points. These gridpoints can be completely independent of all the other (own weight, area, etc.).

\(R=1\): lat(number_of_gridpoints) and lon(number_of_gridpoints)

If the horizontal grid is time-invariant then \(R=2\) is common:

\(R=2\): lat(south,north,east,west) and lon(south,north,east,west)

The Weather and Research Forecast (WRF) model uses \(R=3\):

\(R=3\): lat(time,south,north,east,west), lon(time,south,north,east,west)

and so supports grids that change with time.

Grids with \(R > 1\) often use missing values to indicated empty points. For example, so-called “staggered grids” will use fewer east/west points near the poles and more near the equator. netCDF only accepts rectangular arrays so space must be allocated for the maximum number of east/west points at all latitudes. Then the application writes missing values into the unused points near the poles.

We demonstrate the ncap2 analysis technique for irregular regions by constructing a mask for an \(R=2\) grid. We wish to find, say, the mean temperature within \([\text{lat\_min}, \text{lat\_max}]\) and \([\text{lon\_min}, \text{lon\_max}]\):

\[
\text{ncap2 -s 'mask\_var= (lat >= lat\_min && lat <= lat\_max) && \\
(lon >= lon\_min && lon <= lon\_max);'} \text{ in.nc out.nc}
\]

Arbitrarily shaped regions can be defined by more complex conditional statements. Once defined, masks can be applied to specific variables, and to entire files:

\[
\text{ncap2 -s 'temperature\_avg=(temperature*mask\_var).avg()'} \text{ in.nc out.nc}
\]

\[
\text{ncwa -a lat,lon -m mask\_var -w area in.nc out.nc}
\]

Crafting such commands on the command line is possible though unwieldy. In such cases, a script is often cleaner and allows you to document the procedure:

\[
\text{cat > ncap2.in \&\& 'EOF'}
\]

\[
\text{mask\_var = (lat >= lat\_min && lat <= lat\_max) && (lon >= lon\_min && lon <= lon\_max); \text{ \&\& Check that mask contains some valid values}}
\]
temperature_avg=(temperature*mask_var).avg(); // Average temperature
temperature_max=(temperature*mask_var).max(); // Maximum temperature
}
EOF
ncap2 -S ncap2.in in.nc out.nc

Grids like those produced by the WRF model are complex because one must use global
metadata to determine the grid staggering and offsets to translate XLAT and XLONG into real
latitudes, longitudes, and missing points. The WRF grid documentation should describe
this. For WRF files creating regional masks looks, in general, like

\[
\text{mask\_var} = (\text{XLAT} \geq \text{lat\_min} \&\& \text{XLAT} \leq \text{lat\_max}) \&\& (\text{XLONG} \geq \text{lon\_min} \&\& \text{XLONG} \leq \text{lon\_max})
\]

A few notes: Irregular regions are the union of arrays lat/lon_min/max’s. The mask
procedure is identical for all R.

4.1.18 Bilinear interpolation

As of version 4.0.0 NCO has internal routines to perform bilinear interpolation on gridded
data sets. In mathematics, bilinear interpolation is an extension of linear interpolation for
interpolating functions of two variables on a regular grid. The idea is to perform linear
interpolation first in one direction, and then again in the other direction.

Suppose we have an irregular grid of data temperature[lat,lon], with co-ordinate vars
lat[lat], lon[lon]. We wish to find the temperature at an arbitrary point [X,Y] within
the grid. If we can locate lat_min,lat_max and lon_min,lon_max such that \( \text{lat\_min} \leq X \leq \text{lat\_max} \) and \( \text{lon\_min} \leq Y \leq \text{lon\_max} \) then we can interpolate in two dimensions the
temperature at [X,Y].

The general form of the ncap2 interpolation function is

\[
\text{var\_out} = \text{bilinear\_interp}(\text{grid\_in}, \text{grid\_out}, \text{grid\_out\_x}, \text{grid\_out\_y}, \text{grid\_in\_x}, \text{grid\_in\_y})
\]

where

- \text{grid\_in} Input function data. Usually a two dimensional variable. It must be of size \text{grid\_in\_x.size()}*\text{grid\_in\_y.size()}
- \text{grid\_out} This variable is the shape of \text{var\_out}. Usually a two dimensional variable. It
  must be of size \text{grid\_out\_x.size()}*\text{grid\_out\_y.size()}
- \text{grid\_out\_x} X output values
- \text{grid\_out\_y} Y output values
- \text{grid\_in\_x} X input values values. Must be monotonic (increasing or decreasing).
- \text{grid\_in\_y} Y input values values. Must be monotonic (increasing or decreasing).

Prior to calculations all arguments are converted to type \text{NC\_DOUBLE}. After calculations
\text{var\_out} is converted to the input type of \text{grid\_in}. 
Suppose the first part of an ncap2 script is

```python
defdim("X",4);
defdim("Y",5);

// Temperature
T_in[$X,$Y]=
{100, 200, 300, 400, 500,
  101, 202, 303, 404, 505,
  102, 204, 306, 408, 510,
  103, 206, 309, 412, 515.0 };

// Coordinate variables
x_in[$X]={0.0,1.0,2.0,3.01};
y_in[$Y]={1.0,2.0,3.0,4.0,5};
```

Now we interpolate with the following variables:

```python
defdim("Xn",3);
defdim("Yn",4);
T_out[$Xn,$Yn]=0.0;
x_out[$Xn]={0.0,0.02,3.01};
y_out[$Yn]={1.1,2.0,3,4};
var_out=bilinear_interp(T_in,T_out,x_out,y_out,x_in,y_in);
print(var_out);
```

It is possible to interpolate a single point:

```python
var_out=bilinear_interp(T_in,0.0,3.0,4.99,x_in,y_in);
print(var_out);
```

Wrapping and Extrapolation
The function `bilinear_interp_wrap()` takes the same arguments as `bilinear_interp()` but performs wrapping (Y) and extrapolation (X) for points off the edge of the grid. If the given range of longitude is say (25-335) and we have a point at 20 degrees, then the endpoints of the range are used for the interpolation. This is what wrapping means. For wrapping to occur Y must be longitude and must be in the range (0,360) or (-180,180). There are no restrictions on the longitude (X) values, though typically these are in the range (-90,90). This ncap2 script illustrates both wrapping and extrapolation of end points:

```python
defdim("lat_in",6);
defdim("lon_in",5);

// Coordinate input vars
lat_in[$lat_in]={-80,-40,0,30,60.0,85.0};
```
lon_in[$lon_in$] = {30, 110, 190, 270, 350.0};

T_in[$lat_in,$lon_in$] =
{10, 40, 50, 30, 15,
12, 43, 52, 31, 16,
14, 46, 54, 32, 17,
16, 49, 56, 33, 18,
18, 52, 58, 34, 19,
20, 55, 60, 35, 20.0};

defdim("lat_out", 4);
defdim("lon_out", 3);

// Coordinate variables
lat_out[$lat_out$] = {-90, 0, 70, 88.0};
lon_out[$lon_out$] = {0, 190, 355.0};

T_out[$lat_out,$lon_out$] = 0.0;

T_out = bilinear_interp_wrap(T_in, T_out, lat_out, lon_out, lat_in, lon_in);
print(T_out);

// 13.4375, 49.5, 14.09375,
// 16.25, 54, 16.625,
// 19.25, 58.8, 19.325,
// 20.15, 60.24, 20.135;

4.1.19 GSL special functions

As of version 3.9.6 (released January, 2009), NCO can link to the GNU Scientific Library (GSL). ncap2 can access most GSL special functions including Airy, Bessel, error, gamma, beta, hypergeometric, and Legendre functions and elliptical integrals. GSL must be version 1.4 or later. To list the GSL functions available with your NCO build, use ncap2 -f | grep ^gsl.

The function names used by ncap2 mirror their GSL names. The NCO wrappers for GSL functions automatically call the error-handling version of the GSL function when available\(^2\). This allows NCO to return a missing value when the GSL library encounters a domain error or a floating point exception. The slow-down due to calling the error-handling version of the GSL numerical functions was found to be negligible (please let us know if you find otherwise).

Consider the gamma function.
The GSL function prototype is
int gsl_sf_gamma_e(const double x, gsl_sf_result * result) The ncap2 script would be:
lon_in[lon] = {-1, 0.1, 0, 2, 0.3};

\(^2\) These are the GSL standard function names postfixed with _e. NCO calls these functions automatically, without the NCO command having to specifically indicate the _e function suffix.
lon_out=gsl_sf_gamma(lon_in);
lon_out= _, 9.5135, 4.5908, 2.9915

The first value is set to _FillValue since the gamma function is undefined for negative integers. If the input variable has a missing value then this value is used. Otherwise, the default double fill value is used (defined in the netCDF header netcdf.h as NC_FILL_DOUBLE = 9.969e+36).

Consider a call to a Bessel function with GSL prototype

```c
int gsl_sf_bessel_Jn_e(int n, double x, gsl_sf_result * result)
```

An `ncap2` script would be

```c
lon_out=gsl_sf_bessel_Jn(2,lon_in);
lon_out=0.11490, 0.0012, 0.00498, 0.011165
```

This computes the Bessel function of order $n=2$ for every value in `lon_in`. The Bessel order argument, an integer, can also be a non-scalar variable, i.e., an array.

```c
n_in[lon]={0,1,2,3};
lon_out=gsl_sf_bessel_Jn(n_in,0.5);
lon_out= 0.93846, 0.24226, 0.03060, 0.00256
```

Arguments to GSL wrapper functions in `ncap2` must conform to one another, i.e., they must share the same sub-set of dimensions. For example: `three_out=gsl_sf_bessel_Jn(n_in,three_dmn_var_dbl)` is valid because the variable `three_dmn_var_dbl` has a `lon` dimension, so `n_in` in can be broadcast to conform to `three_dmn_var_dbl`. However `time_out=gsl_sf_bessel_Jn(n_in,time)` is invalid.

Consider the elliptical integral with prototype

```c
int gsl_sf_ellint_RD_e(double x, double y, double z, gsl_mode_t mode, gsl_sf_result * result)
```

```c
three_out=gsl_sf_ellint_RD(0.5,time,three_dmn_var_dbl);
```

The three arguments are all conformable so the above `ncap2` call is valid. The mode argument in the function prototype controls the convergence of the algorithm. It also appears in the Airy Function prototypes. It can be set by defining the environment variable `GSL_PREC_MODE`. If unset it defaults to the value `GSL_PREC_DOUBLE`. See the GSL manual for more details.

```c
export GSL_PREC_MODE=0 // GSL_PREC_DOUBLE
export GSL_PREC_MODE=1 // GSL_PREC_SINGLE
export GSL_PREC_MODE=2 // GSL_PREC_APPROX
```

The `ncap2` wrappers to the array functions are slightly different. Consider the following GSL prototype

```c
int gsl_sf_bessel_Jn_array(int nmin, int nmax, double x, double *result_array)
```

```c
b1=lon.double();
x=0.5;
status=gsl_sf_bessel_Jn_array(1,4,x,&b1);
print(status);
b1=0.24226,0.0306,0.00256,0.00016;
```
This calculates the Bessel function of $x=0.5$ for $n=1$ to 4. The first three arguments are scalar values. If a non-scalar variable is supplied as an argument then only the first value is used. The final argument is the variable where the results are stored (NB: the & indicates this is a call by reference). This final argument must be of type double and must be of least size $n_{\text{max}}-n_{\text{min}}+1$. If either of these conditions is not met then the function returns an error message. The functionwrapper returns a status flag. Zero indicates success.

Consider another array function

```c
int gsl_sf_legendre_Pl_array(int lmax, double x, double *result_array);
```

```c
a1=time.double();
    x=0.3;
    status=gsl_sf_legendre_Pl_array(a1.size()-1, x,&a1);
    print(status);
```

This call calculates $P_l(0.3)$ for $l=0..9$. Note that $|x|<1$, otherwise there will be a domain error. See the GSL documentation for more details.

The GSL functions implemented in NCO are listed in the table below. This table is correct for GSL version 1.10. To see what functions are available on your build run the command `ncap2 -f |grep ^gsl`. To see this table along with the GSL C-function prototypes look at the spreadsheet `doc/nco.gsl.ods`.

<table>
<thead>
<tr>
<th>GSL NAME</th>
<th>I</th>
<th>NCAP FUNCTION CALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>gsl_sf_airy_Ai</td>
<td>Y</td>
<td>gsl_sf_airy_Ai(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_airy_Bi</td>
<td>Y</td>
<td>gsl_sf_airy_Bi(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_airy_Ai_scaled</td>
<td>Y</td>
<td>gsl_sf_airy_Ai_scaled(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_airy_Bi_scaled</td>
<td>Y</td>
<td>gsl_sf_airy_Bi_scaled(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_airy_Ai_deriv</td>
<td>Y</td>
<td>gsl_sf_airy_Ai_deriv(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_airy_Bi_deriv</td>
<td>Y</td>
<td>gsl_sf_airy_Bi_deriv(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_airy_Ai_deriv_scaled</td>
<td>Y</td>
<td>gsl_sf_airy_Ai_deriv_scaled(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_airy_Bi_deriv_scaled</td>
<td>Y</td>
<td>gsl_sf_airy_Bi_deriv_scaled(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_airy_zero_Ai</td>
<td>Y</td>
<td>gsl_sf_airy_zero_Ai(uint_expr)</td>
</tr>
<tr>
<td>gsl_sf_airy_zero_Bi</td>
<td>Y</td>
<td>gsl_sf_airy_zero_Bi(uint_expr)</td>
</tr>
<tr>
<td>gsl_sf_airy_zero_Ai_deriv</td>
<td>Y</td>
<td>gsl_sf_airy_zero_Ai_deriv(uint_expr)</td>
</tr>
<tr>
<td>gsl_sf_airy_zero_Bi_deriv</td>
<td>Y</td>
<td>gsl_sf_airy_zero_Bi_deriv(uint_expr)</td>
</tr>
<tr>
<td>gsl_sf_bessel_J0</td>
<td>Y</td>
<td>gsl_sf_bessel_J0(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_bessel_J1</td>
<td>Y</td>
<td>gsl_sf_bessel_J1(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_bessel_Jn</td>
<td>Y</td>
<td>gsl_sf_bessel_Jn(int_expr,dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_bessel_Jn_array</td>
<td>Y</td>
<td>status=gsl_sf_bessel_Jn_array(int,int,double,&amp;var_out)</td>
</tr>
<tr>
<td>gsl_sf_bessel_Y0</td>
<td>Y</td>
<td>gsl_sf_bessel_Y0(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_bessel_Y1</td>
<td>Y</td>
<td>gsl_sf_bessel_Y1(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_bessel_Yn</td>
<td>Y</td>
<td>gsl_sf_bessel_Yn(int_expr,dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_bessel_Yn_array</td>
<td>Y</td>
<td>gsl_sf_bessel_Yn_array</td>
</tr>
<tr>
<td>gsl_sf_bessel_I0</td>
<td>Y</td>
<td>gsl_sf_bessel_I0(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_bessel_I1</td>
<td>Y</td>
<td>gsl_sf_bessel_I1(dbl_expr)</td>
</tr>
<tr>
<td>gsl_sf_bessel_In</td>
<td>Y</td>
<td>gsl_sf_bessel_In(int_expr,dbl_expr)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_In_array</td>
<td>Y status = gsl_sf_bessel_In_array(int, int, double, &amp;var_out)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_I0_scaled_e</td>
<td>Y gsl_sf_bessel_I0_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_I1_scaled_e</td>
<td>Y gsl_sf_bessel_I1_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_In_scaled_e</td>
<td>Y gsl_sf_bessel_In_scaled(int_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_In_scaled_array</td>
<td>Y status = gsl_sf_bessel_In_scaled_array(int, int, double, &amp;var_out)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_K0_e</td>
<td>Y gsl_sf_bessel_K0(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_K1_e</td>
<td>Y gsl_sf_bessel_K1(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_Kn_e</td>
<td>Y status = gsl_sf_bessel_Kn_array(int, int, double, &amp;var_out)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_K0_scaled_e</td>
<td>Y gsl_sf_bessel_K0_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_K1_scaled_e</td>
<td>Y gsl_sf_bessel_K1_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_Kn_scaled_e</td>
<td>Y gsl_sf_bessel_Kn_scaled(int_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_Kn_scaled_array</td>
<td>Y status = gsl_sf_bessel_Kn_scaled_array(int, int, double, &amp;var_out)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_K0_scaled</td>
<td>Y gsl_sf_bessel_K0_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_K1_scaled</td>
<td>Y gsl_sf_bessel_K1_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_Kn_scaled</td>
<td>Y gsl_sf_bessel_Kn_scaled(int_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_Kn_scaled_array</td>
<td>Y status = gsl_sf_bessel_Kn_scaled_array(int, int, double, &amp;var_out)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_j0_e</td>
<td>Y gsl_sf_bessel_j0(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_j1_e</td>
<td>Y gsl_sf_bessel_j1(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_j2_e</td>
<td>Y gsl_sf_bessel_j2(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_jl_e</td>
<td>Y gsl_sf_bessel_jl(int_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_jl_array</td>
<td>Y status = gsl_sf_bessel_jl_array(int, double, &amp;var_out)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_jl_steed_array</td>
<td>Y gsl_sf_bessel_jl_steed_array</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_y0_e</td>
<td>Y gsl_sf_bessel_y0(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_y1_e</td>
<td>Y gsl_sf_bessel_y1(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_y2_e</td>
<td>Y gsl_sf_bessel_y2(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_yl_e</td>
<td>Y gsl_sf_bessel_yl(int_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_yl_array</td>
<td>Y status = gsl_sf_bessel_yl_array(int, double, &amp;var_out)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_i0_scaled_e</td>
<td>Y gsl_sf_bessel_i0_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_i1_scaled_e</td>
<td>Y gsl_sf_bessel_i1_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_i2_scaled_e</td>
<td>Y gsl_sf_bessel_i2_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_il_scaled_e</td>
<td>Y gsl_sf_bessel_il_scaled(int_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_il_scaled_array</td>
<td>Y status = gsl_sf_bessel_il_scaled_array(int, double, &amp;var_out)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_k0_scaled_e</td>
<td>Y gsl_sf_bessel_k0_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_k1_scaled_e</td>
<td>Y gsl_sf_bessel_k1_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_k2_scaled_e</td>
<td>Y gsl_sf_bessel_k2_scaled(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_kl_scaled_e</td>
<td>Y gsl_sf_bessel_kl_scaled(int_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_kl_scaled_array</td>
<td>Y status = gsl_sf_bessel_kl_scaled_array(int, double, &amp;var_out)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_Jnu_e</td>
<td>Y gsl_sf_bessel_Jnu(dbl_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_Ynu_e</td>
<td>Y gsl_sf_bessel_Ynu(dbl_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_sequence_Jnu_e</td>
<td>N gsl_sf_bessel_sequence_Jnu</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_Inu_e</td>
<td>Y gsl_sf_bessel_Inu_scaled(dbl_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_Inu_e</td>
<td>Y gsl_sf_bessel_Inu(dbl_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_Knu_scaled_e</td>
<td>Y gsl_sf_bessel_Knu_scaled(dbl_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_Knu_e</td>
<td>Y gsl_sf_bessel_Knu(dbl_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_InKnu_e</td>
<td>Y gsl_sf_bessel_InKnu(dbl_expr, dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_zero_J0_e</td>
<td>Y gsl_sf_bessel_zero_J0(uint_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_zero_J1_e</td>
<td>Y gsl_sf_bessel_zero_J1(uint_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_bessel_zero_Jnu_e</td>
<td>N gsl_sf_bessel_zero_Jnu</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_clausen_e</td>
<td>Y gsl_sf_clausen(dbl_expr)</td>
<td></td>
</tr>
<tr>
<td>gsl_sf_hydrogenicR_1_e</td>
<td>N gsl_sf_hydrogenicR_1</td>
<td></td>
</tr>
</tbody>
</table>
gsl sf hydrogenicR_e  N    gsl sf hydrogenicR
[gsl sf coulomb wave FG_e  N    gsl sf coulomb wave FG
[gsl sf coulomb wave F array  N    gsl sf coulomb wave F array
[gsl sf coulomb wave FG array  N    gsl sf coulomb wave FG array
[gsl sf coulomb wave FGp array  N    gsl sf coulomb wave FGp array
[gsl sf coulomb CL_e  N    gsl sf coulomb CL
[gsl sf coulomb CL array  N    gsl sf coulomb CL_array
[gsl sf coupling 3j e  N    gsl sf coupling 3j
[gsl sf coupling 6j e  N    gsl sf coupling 6j
[gsl sf coupling RacahW_e  N    gsl sf coupling RacahW
[gsl sf coupling 9j e  N    gsl sf coupling 9j
[gsl sf coupling 6j INCORRECT e N    gsl sf coupling 6j INCORRECT
[gsl sf dawson e  Y    gsl sf dawson dbl expr
[gsl sf debye 1 e  Y    gsl sf debye 1 dbl expr
[gsl sf debye 2 e  Y    gsl sf debye 2 dbl expr
[gsl sf debye 3 e  Y    gsl sf debye 3 dbl expr
[gsl sf debye 4 e  Y    gsl sf debye 4 dbl expr
[gsl sf debye 5 e  Y    gsl sf debye 5 dbl expr
[gsl sf debye 6 e  Y    gsl sf debye 6 dbl expr
[gsl sf dilog e  N    gsl sf dilog
[gsl sf complex dilog xy e  N    gsl sf complex dilog xy e
[gsl sf complex dilog e  N    gsl sf complex dilog
[gsl sf complex spence xy e  N    gsl sf complex spence xy e
[gsl sf multiply e  N    gsl sf multiply
[gsl sf multiply err e  N    gsl sf multiply err
[gsl sf ellint K comp e  Y    gsl sf ellint K comp dbl expr
[gsl sf ellint E comp e  Y    gsl sf ellint E comp dbl expr
[gsl sf ellint p comp e  Y    gsl sf ellint P comp dbl expr, dbl expres
[gsl sf ellint D comp e  Y    gsl sf ellint D comp dbl expr
[gsl sf ellint F e  Y    gsl sf ellint F dbl expr, dbl expres
[gsl sf ellint E e  Y    gsl sf ellint E dbl expres, dbl expres
[gsl sf ellint P e  Y    gsl sf ellint P dbl expres, dbl expres, dbl expres
[gsl sf ellint D e  Y    gsl sf ellint D dbl expres, dbl expres, dbl expres
[gsl sf ellint RC e  Y    gsl sf ellint RC dbl expres, dbl expres
[gsl sf ellint RD e  Y    gsl sf ellint RD dbl expres, dbl expres, dbl expres
[gsl sf ellint RF e  Y    gsl sf ellint RF dbl expres, dbl expres, dbl expres
[gsl sf ellint RJ e  Y    gsl sf ellint RJ dbl expres, dbl expres, dbl expres, dbl expres
[gsl sf elljac e  N    gsl sf elljac
[gsl sf erfc e  Y    gsl sf erfc dbl expres
[gsl sf log erfc e  Y    gsl sf log erfc dbl expres
[gsl sf erf e  Y    gsl sf erf dbl expres
[gsl sf erf Z e  Y    gsl sf erf Z dbl expres
[gsl sf erf Q e  Y    gsl sf erf Q dbl expres
[gsl sf hazard e  Y    gsl sf hazard dbl expres
[gsl sf exp e  Y    gsl sf exp dbl expres
[gsl sf exp e10 e  N    gsl sf exp e10
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gsl_sf_exp_mult_e</code></td>
<td>Y <code>gsl_sf_exp_mult(dbl_expr,dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_exp_mult_e10_e</code></td>
<td>N <code>gsl_sf_exp_mult_e10</code></td>
</tr>
<tr>
<td><code>gsl_sf_expm1_e</code></td>
<td>Y <code>gsl_sf_expm1(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_exprel_e</code></td>
<td>Y <code>gsl_sf_exprel(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_exprel_2_e</code></td>
<td>Y <code>gsl_sf_exprel_2(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_exprel_n_e</code></td>
<td>Y <code>gsl_sf_exprel_n(int_expr,dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_exp_err_e</code></td>
<td>Y <code>gsl_sf_exp_err(dbl_expr,dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_exp_err_e10_e</code></td>
<td>N <code>gsl_sf_exp_err_e10</code></td>
</tr>
<tr>
<td><code>gsl_sf_exp_mult_err_e</code></td>
<td>N <code>gsl_sf_exp_mult_err</code></td>
</tr>
<tr>
<td><code>gsl_sf_exp_mult_err_e10_e</code></td>
<td>N <code>gsl_sf_exp_mult_err_e10</code></td>
</tr>
<tr>
<td><code>gsl_sf_expint_E1_e</code></td>
<td>Y <code>gsl_sf_expint_E1(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_expint_E2_e</code></td>
<td>Y <code>gsl_sf_expint_E2(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_expint_En_e</code></td>
<td>Y <code>gsl_sf_expint_En(int_expr,dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_expint_E1_scaled_e</code></td>
<td>Y <code>gsl_sf_expint_E1_scaled(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_expint_E2_scaled_e</code></td>
<td>Y <code>gsl_sf_expint_E2_scaled(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_expint_En_scaled_e</code></td>
<td>Y <code>gsl_sf_expint_En_scaled(int_expr,dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_expint_Ei_e</code></td>
<td>Y <code>gsl_sf_expint_Ei(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_expint_Ei_scaled_e</code></td>
<td>Y <code>gsl_sf_expint_Ei_scaled(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_shi_e</code></td>
<td>Y <code>gsl_sf_shi(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_chi_e</code></td>
<td>Y <code>gsl_sf_chi(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_exprel_3_e</code></td>
<td>Y <code>gsl_sf_exprel_3(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_sil_e</code></td>
<td>Y <code>gsl_sf_sil(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_cil_e</code></td>
<td>Y <code>gsl_sf_cil(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_3half_e</code></td>
<td>Y <code>gsl_sf_3half(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_3half_e</code></td>
<td>Y <code>gsl_sf_3half(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_3half_e</code></td>
<td>Y <code>gsl_sf_3half(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_lgamma_e</code></td>
<td>Y <code>gsl_sf_lgamma(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_lgamma_sgn_e</code></td>
<td>N <code>gsl_sf_lgamma_sgn</code></td>
</tr>
<tr>
<td><code>gsl_sf_fact_e</code></td>
<td>Y <code>gsl_sf_fact(uint_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_doublefact_e</code></td>
<td>Y <code>gsl_sf_doublefact(uint_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_lgamma_e</code></td>
<td>Y <code>gsl_sf_lgamma(dbl_expr)</code></td>
</tr>
<tr>
<td><code>gsl_sf_lncnchoose_e</code></td>
<td>N <code>gsl_sf_lncnchoose</code></td>
</tr>
<tr>
<td><code>gsl_sf_cncnchoose_e</code></td>
<td>N <code>gsl_sf_cncnchoose</code></td>
</tr>
<tr>
<td><code>gsl_sf_lnpoch_e</code></td>
<td>Y <code>gsl_sf_lnpoch(dbl_expr,dbl_expr)</code></td>
</tr>
</tbody>
</table>
4.1.20 GSL interpolation

As of version 3.9.9 (released July, 2009), NCO has wrappers to the GSL interpolation functions.

Given a set of data points \((x_1,y_1),...,(x_n,y_n)\) the GSL functions computes a continuous interpolating function \(Y(x)\) such that \(Y(x_i) = y_i\). The interpolation is piecewise smooth, and its behavior at the end-points is determined by the type of interpolation used. For more information consult the GSL manual.

Interpolation with \texttt{ncap2} is a two stage process. In the first stage, a RAM variable is created from the chosen interpolating function and the data set. This RAM variable holds in memory a GSL interpolation object. In the second stage, points along the interpolating function are calculated. If you have a very large data set or are interpolating many sets then consider deleting the RAM variable when it is redundant. Use the command \texttt{ram_delete(var nm)}.

A simple example

\begin{verbatim}
x_in[$lon] = {1.0, 2.0, 3.0, 4.0};
y_in[$lon] = {1.1, 1.2, 1.5, 1.8};

// Ram variable is declared and defined here
gsl_interp_cspline(&ram_sp, x_in, y_in);

x_out[$lon_grd] = {1.1, 2.0, 3.0, 3.1, 3.99};
\end{verbatim}
y_out=gsl_spline_eval(ram_sp,x_out);
y2=gsl_spline_eval(ram_sp,1.3);
y3=gsl_spline_eval(ram_sp,0.0);
ram_delete(ram_sp);

print(y_out); // 1.10472, 1.2, 1.4, 1.42658, 1.69680002
print(y2);    // 1.12454
print(y3);    // '_'

Note in the above example y3 is set to 'missing value' because 0.0 isn’t within the input X range.

**GSL Interpolation Types**
All the interpolation functions have been implemented. These are:
gsl_interp_linear()
gsl_interp_polynomial()
gsl_interp_cspline()
gsl_interp_cspline_periodic()
gsl_interp_akima()
gsl_interp_akima_periodic()

**Evaluation of Interpolating Types**

**Implemented**
gsl_spline_eval()

**Unimplemented**
gsl_spline_deriv()
gsl_spline_deriv2()
gsl_spline_integ()

### 4.1.21 GSL least-squares fitting
Least Squares fitting is a method of calculating a straight line through a set of experimental data points in the XY plane. The data maybe weighted or unweighted. For more information please refer to the GSL manual.

These GSL functions fall into three categories:

**A)** Fitting data to \( Y = c_0 + c_1 X \)

**B)** Fitting data (through the origin) \( Y = c_1 X \)

**C)** Multi-parameter fitting (not yet implemented)

**Section A**

status=gsl_fit_linear
(data_x,stride_x,data_y,stride_y,n,&co,&c1,&cov00,&cov01,&cov11,&sumsq)
**Input variables:** data\_x, stride\_x, data\_y, stride\_y, n
From the above variables an X and Y vector both of length 'n' are derived. If data\_x or data\_y is less than type double then it is converted to type double. It is up to you to do bounds checking on the input data. For example if stride\_x=3 and n=8 then the size of data\_x must be at least 24

**Output variables:** c0, c1, cov00, cov01, cov11,sumsq
The '&' prefix indicates that these are call-by-reference variables. If any of the output variables don’t exist prior to the call then they are created on the fly as scalar variables of type double. If they already exist then their existing value is overwritten. If the function call is successful then status=0.

```c
status= gsl_fit_wlinear(data\_x,stride\_x,data\_w,stride\_w,data\_y,stride\_y,n,&co,&c1,&cov00,&cov01,&cov11,&chisq)
```

Similar to the above call except it creates an additional weighting vector from the variables data\_w, stride\_w, n

```c
data\_y\_out=gsl_fit_linear\_est(data\_x,c0,c1,cov00,cov01,cov11)
```

This function calculates y values along the line Y=c0+c1\*X

---

**Section B**

```c
status=gsl_fit_mul(data\_x,stride\_x,data\_y,stride\_y,n,&c1,&cov11,&sumsq)
```

**Input variables:** data\_x, stride\_x, data\_y, stride\_y, n
From the above variables an X and Y vector both of length 'n' are derived. If data\_x or data\_y is less than type double then it is converted to type double.

**Output variables:** c1,cov11,sumsq

```c
status= gsl_fit_wmul(data\_x,stride\_x,data\_w,stride\_w,data\_y,stride\_y,n,&c1,&cov11,&sumsq)
```

Similar to the above call except it creates an additional weighting vector from the variables data\_w, stride\_w, n

```c
data\_y\_out=gsl_fit_mul\_est(data\_x,c0,c1,cov11)
```

This function calculates y values along the line Y=c1\*X

The below example shows **gsl_fit_linear()** in action

```c
defdim("d1",10);
xin[d1]={1,2,3,4,5,6,7,8,9,10.0};
yin[d1]={3.1,6.2,9.1,12.2,15.1,18.2,21.3,24.0,27.0,30.0};
gsl_fit_linear(xin,1,yin,1,$d1.size,&c0,&c1,&cov00,&cov01,&cov11,&sumsq);
```
print(c0); // 0.2
print(c1); // 2.98545454545

defdim("e1",4);
xout[e1]={1.0,3.0,4.0,11};
yout[e1]=0.0;

yout=gsl_fit_linear_est(xout, c0,c1, cov00,cov01, cov11, sumsq);

print(yout); // 3.18545454545 ,9.15636363636, ,12.1418181818 ,33.04

4.1.22 GSL statistics

Wrappers for most of the GSL Statistical functions have been implemented. The GSL function names include a type specifier (except for type double functions). To obtain the equivalent NCO name simply remove the type specifier; then depending on the data type the appropriate GSL function is called. The weighed statistical functions e.g., gsl_stats_wvariance() are only defined in GSL for floating point types; so your data must of type float or double otherwise ncap2 will emit an error message. To view the implemented functions use the shell command ncap2 -f|grep_stats

GSL Functions

short gsl_stats_max (short data[], size_t stride, size_t n);
double gsl_stats_int_mean (int data[], size_t stride, size_t n);
double gsl_stats_short_sd_with_fixed_mean (short data[], size_t stride, size_t n, double mean);
double gsl_stats_wmean (double w[], size_t wstride, double data[], size_t stride, size_t n);
double gsl_stats_quantile_from_sorted_data (double sorted_data[], size_t stride, size_t n);

Equivalent ncap2 wrapper functions

short gsl_stats_max (var_data, data_stride, n);
double gsl_stats_mean (var_data, data_stride, n);
double gsl_stats_sd_with_fixed_mean (var_data, data_stride, n, var_mean);
double gsl_stats_wmean (var_weight, weight_stride, var_data, data_stride, n, var_mean);
double gsl_stats_quantile_from_sorted_data (var_sorted_data, data_stride, n, var_f);

GSL has no notion of missing values or dimensionality beyond one. If your data has missing values which you want ignored in the calculations then use the ncap2 built in aggregate functions( Section 4.1.11 [Methods and functions], page 131 ). The GSL functions operate on a vector of values created from the var_data/stride/n arguments. The ncap wrappers check that there is no bounding error with regard to the size of the data and the final value in the vector.

Some examples

a1[time]={1,2,3,4,5,6,7,8,9,10};
a1_avg=gsl_stats_mean(a1,1,10);
print(a1_avg); // 5.5

a1_var=gsl_stats_variance(a1,4,3);
print(a1_var); // 16.0

// bounding error, vector attempts to access element a1(10)
a1_sd=gsl_stats_sd(a1,5,3);

For functions with the signature func (var data, data stride, n), one may omit the second or third arguments. The default value for stride is 1. The default value for n is 1+(data.size()-1)/stride.

// Following statements are equivalent
n2=gsl_stats_max(a1,1,10)
n2=gsl_stats_max(a1,1);
n2=gsl_stats_max(a1);

// Following statements are equivalent
n3=gsl_stats_median_from_sorted_data(a1,2,5);
n3=gsl_stats_median_from_sorted_data(a1,2);

// Following statements are NOT equivalent
n4=gsl_stats_kurtosis(a1,3,2);
n4=gsl_stats_kurtosis(a1,3); //default n=4

The following example illustrates some of the weighted functions. The data are randomly generated. In this case the value of the weight for each datum is either 0.0 or 1.0

defdim("r1",2000);
data[r1]=1.0;

// Fill with random numbers [0.0,10.0)
data=10.0*gsl_rng_uniform(data);

// Create a weighting variable
weight=(data>4.0);

wmean=gsl_stats_wmean(weight,1,data,1,$r1.size);
print(wmean);

wsd=gsl_stats_wsd(weight,1,data,1,$r1.size);
print(wsd);

// number of values in data that are greater than 4
weight_size=weight.total();
print(weight_size);
// print min/max of data

dmin=data.gsl_stats_min();
dmax=data.gsl_stats_max();
print(dmin);print(dmax);

4.1.23 GSL random number generation

The GSL library has a large number of random number generators. In addition there are a large set of functions for turning uniform random numbers into discrete or continuous probability distributions. The random number generator algorithms vary in terms of quality numbers output, speed of execution and maximum number output. For more information see the GSL documentation. The algorithm and seed are set via environment variables, these are picked up by the ncap2 code.

Setup

The number algorithm is set by the environment variable GSL_RNG_TYPE. If this variable isn’t set then the default rng algorithm is gsl_rng_19937. The seed is set with the environment variable GSL_RNG_SEED. The following wrapper functions in ncap2 provide information about the chosen algorithm.

gsl_rng_min()
the minimum value returned by the rng algorithm.

gsl_rng_max()
the maximum value returned by the rng algorithm.

Uniformly Distributed Random Numbers

gsl_rng_get(var_in)
This function returns var_in with integers from the chosen rng algorithm. The min and max values depend upon the chosen rng algorithm.

gsl_rng_uniform_int(var_in)
This function returns var_in with random integers from 0 to n-1. The value n must be less than or equal to the maximum value of the chosen rng algorithm.

gsl_rng_uniform(var_in)
This function returns var_in with double-precision numbers in the range [0.0,1). The range includes 0.0 and excludes 1.0.

gsl_rng_uniform_pos(var_in)
This function returns var_in with double-precision numbers in the range (0.0,1), excluding both 0.0 and 1.0.

Below are examples of gsl_rng_get() and gsl_rng_uniform_int() in action.

    export GSL_RNG_TYPE=ranlux
    export GSL_RNG_SEED=10
    ncap2 -v -O -s 'a1(time)=0;a2=gsl_rng_get(a1);' in.nc foo.nc
    // 10 random numbers from the range 0 - 16777215
    // a2=9056646, 12776696, 1011656, 13354708, 5139066, 1388751, 11163902, 7730127, 15531355, 10387694 ;
ncap2 -v -O -s 'a1\[time\]=21;a2=gsl_rng_uniform_int(a1).sort();' in.nc foo.nc
  // 10 random numbers from the range 0 - 20
  a2 = 1, 1, 6, 9, 11, 13, 13, 15, 16, 19;

The following example produces an ncap2 runtime error. This is because the chose rng algorithm has a maximum value greater than NC_MAX_INT=2147483647; the wrapper functions to gsl_rng_get() and gsl_rng_uniform_int() return variable of type NC_INT. Please be aware of this when using random number distribution functions from the GSL library which return unsigned int. Examples of these are gsl_ran_geometric() and gsl_ran_pascal().

    export GSL_RNG_TYPE=mt19937
    ncap2 -v -O -s 'a1\[time\]=0;a2=gsl_rng_get(a1);' in.nc foo.nc

To find the maximum value of the chosen rng algorithm use the following code snippet.

    ncap2 -v -O -s 'rng_max=gsl_rng_max();print(rng_max)' in.nc foo.nc

Random Number Distributions
The GSL library has a rich set of random number distribution functions. The library also provides cumulative distribution functions and inverse cumulative distribution functions sometimes referred to a quantile functions. To see what's available on your build use the shell command ncap2 -f|grep -e _ran -e _cdf.

The following examples all return variables of type NC_INT

    defdim("out",15);
    a1\[$out\]=0.5;
    a2=gsl_ran_binomial(a1,30).sort();
    //a2 = 10, 11, 12, 13, 14, 14, 15, 15, 16, 16, 16, 17, 22;
    a3=gsl_ran_geometric(a2).sort();
    //a2 = 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 4, 5;
    a4=gsl_ran_pascal(a2,50);
    //a5 = 37, 40, 40, 42, 43, 45, 46, 49, 52, 58, 60, 62, 62, 65, 67;

The following all return variables of type NC_DOUBLE

    defdim("b1",1000);
    b1\[$b1\]=0.8;
    b2=gsl_ran_exponential(b1);
    b2_avg=b2.avg();
    print(b2_avg);
    // b2_avg = 0.756047976787

    b3=gsl_ran_gaussian(b1);
    b3_avg=b3.avg();
    b3_rms=b3.rms();
    print(b3_avg);
4.1.24 Examples ncap2

See the ncap.in and ncap2.in scripts released with NCO for more complete demonstrations of ncap2 functionality (script available on-line at http://nco.sf.net/ncap2.in).

Define new attribute new for existing variable one as twice the existing attribute double_att of variable att_var:

```bash
cap2 -s 'one@new=2*att_var@double_att' in.nc out.nc
```

Average variables of mixed types (result is of type double):

```bash
cap2 -s 'average=(var_float+var_double+var_int)/3' in.nc out.nc
```

Multiple commands may be given to ncap2 in three ways. First, the commands may be placed in a script which is executed, e.g., tst.nco. Second, the commands may be individually specified with multiple `-s` arguments to the same ncap2 invocation. Third, the commands may be chained into a single `-s` argument to ncap2. Assuming the file tst.nco contains the commands `a=3; b=4; c=sqrt(a^2+b^2)`; then the following ncap2 invocations produce identical results:

```bash
ncap2 -v -S tst.nco in.nc out.nc
ncap2 -v -s 'a=3' -s 'b=4' -s 'c=sqrt(a^2+b^2)' in.nc out.nc
ncap2 -v -s 'a=3;b=4;c=sqrt(a^2+b^2)' in.nc out.nc
```

The second and third examples show that ncap2 does not require that a trailing semi-colon `;` be placed at the end of a `-s` argument, although a trailing semi-colon `;` is always allowed. However, semi-colons are required to separate individual assignment statements chained together as a single `-s` argument.

ncap2 may be used to “grow” dimensions, i.e., to increase dimension sizes without altering existing data. Say in.nc has ORO(lat,lon) and the user wishes a new file with new_ORO(new_lat,new_lon) that contains zeros in the undefined portions of the new grid.

```bash
defdim("new_lat",$lat.size+1); // Define new dimension sizes
defdim("new_lon",$lon.size+1);
new_ORO[$new_lat,$new_lon]=0.0f; // Initialize to zero
new_ORO(0:$lat.size-1,0:$lon.size-1)=ORO; // Fill valid data
```

The commands to define new coordinate variables new_lat and new_lon in the output file follow a similar pattern. One would might store these commands in a script grow.nco and then execute the script with
ncap2 -v -S grow.nco in.nc out.nc

Imagine you wish to create a binary flag based on the value of an array. The flag should have value 1.0 where the array exceeds 1.0, and value 0.0 elsewhere. This example creates the binary flag ORO_flg in out.nc from the continuous array named ORO in in.nc.

ncap2 -s 'ORO_flg=(ORO > 1.0)' in.nc out.nc

Suppose your task is to change all values of ORO which equal 2.0 to the new value 3.0:

ncap2 -s 'ORO_msk=(ORO==2.0);ORO=ORO_msk*3.0+!ORO_msk*ORO' in.nc out.nc

This creates and uses ORO_msk to mask the subsequent arithmetic operation. Values of ORO are only changed where ORO_msk is true, i.e., where ORO equals 2.0 Using the where statement the above code simplifies to:

ncap2 -s 'where(ORO==2.0) ORO=3.0;' in.nc foo.nc

This example uses ncap2 to compute the covariance of two variables. Let the variables $u$ and $v$ be the horizontal wind components. The covariance of $u$ and $v$ is defined as the time mean product of the deviations of $u$ and $v$ from their respective time means. Symbolically, the covariance $[u'v'] = [uv] - [u][v]$ where $[x]$ denotes the time-average of $x$, $[x] = \frac{1}{T} \int_{t=0}^{T} x(t) \, dt$ and $x'$ denotes the deviation from the time-mean. The covariance tells us how much of the correlation of two signals arises from the signal fluctuations versus the mean signals. Sometimes this is called the eddy covariance. We will store the covariance in the variable uprmvprm.

ncwa -O -a time -v u,v in.nc foo.nc # Compute time mean of u,v
ncrename -O -v u,uavg -v v,vavg foo.nc # Rename to avoid conflict
ncks -A -v uavg,vavg foo.nc in.nc # Place time means with originals
ncap2 -O -s 'uprmvprm=u*v-uavg*vavg' in.nc in.nc # Covariance
ncra -O -v uprmvprm in.nc foo.nc # Time-mean covariance

The mathematically inclined will note that the same covariance would be obtained by replacing the step involving ncap2 with

ncap2 -O -s 'uprmvprm=(u-uavg)*(v-vavg)' foo.nc foo.nc # Covariance

As of NCO version 3.1.8 (December, 2006), ncap2 can compute averages, and thus covariances, by itself:

ncap2 -s 'uavg=u.avg($time);vavg=v.avg($time);uprmvprm=u*v-uavg*vavg' \ 
   -s 'uprmvprmvavg=uprmvprm.avg($time)' in.nc foo.nc

We have not seen a simpler method to script and execute powerful arithmetic than ncap2.

ncap2 utilizes many meta-characters (e.g., ‘$’, ‘?’ , ‘;’ , ‘( )’, ‘[ ]’) that can confuse the command-line shell if not quoted properly. The issues are the same as those which arise in utilizing extended regular expressions to subset variables (see Section 3.11 [Subsetting Files], page 40). The example above will fail with no quotes and with double quotes. This is because shell globbing tries to interpolate the value of $time from the shell environment unless it is quoted:

ncap2 -s 'uavg=u.avg($time)' in.nc foo.nc # Correct (recommended)
ncap2 -s uavg=u.avg('$time') in.nc foo.nc # Correct (and dangerous)
ncap2 -s uavg=u.avg($time) in.nc foo.nc # Fails ($time = '')
ncap2 -s "uavg=u.avg($time)" in.nc foo.nc # Fails ($time = '')

Without the single quotes, the shell replaces $time with an empty string. The command ncap2 receives from the shell is uavg=u.avg(). This causes ncap2 to average over all dimensions rather than just the time dimension, and unintended consequence.

We recommend using single quotes to protect ncap2 command-line scripts from the shell, even when such protection is not strictly necessary. Expert users may violate this rule to exploit the ability to use shell variables in ncap2 command-line scripts (see Chapter 9 [CCSM Example], page 259). In such cases it may be necessary to use the shell backslash character ‘\’ to protect the ncap2 meta-character.

A dimension of size one is said to be degenerate. Whether a degenerate record dimension is desirable or not depends on the application. Often a degenerate time dimension is useful, e.g., for concatenating, but it may cause problems with arithmetic. Such is the case in the above example, where the first step employs ncwa rather than ncra for the time-averaging. Of course the numerical results are the same with both operators. The difference is that, unless ‘-b’ is specified, ncwa writes no time dimension to the output file, while ncra defaults to keeping time as a degenerate (size 1) dimension.Appending u and v to the output file would cause ncks to try to expand the degenerate time axis of uavg and vavg to the size of the non-degenerate time dimension in the input file. Thus the append (ncks -A) command would be undefined (and should fail) in this case. Equally important is the ‘-C’ argument (see Section 3.12 [Subsetting Coordinate Variables], page 45) to ncwa to prevent any scalar time variable from being written to the output file. Knowing when to use ncwa -a time rather than the default ncra for time-averaging takes, well, time.

4.1.25 Intrinsic mathematical methods

ncap2 supports the standard mathematical functions supplied with most operating systems. Standard calculator notation is used for addition +, subtraction -, multiplication *, division /, exponentiation ^, and modulus %. The available elementary mathematical functions are:

abs(x) Absolute value Absolute value of x, |x|. Example: abs(−1) = 1
acos(x) Arc-cosine Arc-cosine of x where x is specified in radians. Example: acos(1.0) = 0.0
acosh(x) Hyperbolic arc-cosine Hyperbolic arc-cosine of x where x is specified in radians. Example: acosh(1.0) = 0.0
asin(x) Arc-sine Arc-sine of x where x is specified in radians. Example: asin(1.0) = 1.57079632679489661922
asinh(x) Hyperbolic arc-sine Hyperbolic arc-sine of x where x is specified in radians. Example: asinh(1.0) = 0.88137358702
atan(x) Arc-tangent Arc-tangent of x where x is specified in radians between −π/2 and π/2. Example: atan(1.0) = 0.78539816339744830961
atan2(y,x) Arc-tangent2 Arc-tangent of y/x
atanh(x)  *Hyperbolic arc-tangent* Hyperbolic arc-tangent of \( x \) where \( x \) is specified in radians between \(-\pi/2\) and \(\pi/2\). Example: \( \text{atanh}(3.14159265358979323844) = 1.0 \)

ceil(x)  *Ceil* Ceiling of \( x \). Smallest integral value not less than argument. Example: \( \text{ceil}(0.1) = 1.0 \)

cos(x)  *Cosine* Cosine of \( x \) where \( x \) is specified in radians. Example: \( \cos(0.0) = 1.0 \)

cosh(x)  *Hyperbolic cosine* Hyperbolic cosine of \( x \) where \( x \) is specified in radians. Example: \( \cosh(0.0) = 1.0 \)

erf(x)  *Error function* Error function of \( x \) where \( x \) is specified between \(-1\) and \(1\). Example: \( \text{erf}(1.0) = 0.842701 \)

erfc(x)  *Complementary error function* Complementary error function of \( x \) where \( x \) is specified between \(-1\) and \(1\). Example: \( \text{erfc}(1.0) = 0.1579920705 \)

exp(x)  *Exponential* Exponential of \( x \), \( e^x \). Example: \( \exp(1.0) = 2.71828182845904523536 \)

cos(x)  *Floor* Floor of \( x \). Largest integral value not greater than argument. Example: \( \text{floor}(1.9) = 1 \)

gamma(x)  *Gamma function* Gamma function of \( x \), \( \Gamma(x) \). The well-known and loved continuous factorial function. Example: \( \gamma(0.5) = \sqrt{\pi} \)

gamma_inc_P(x)  *Incomplete Gamma function* Incomplete Gamma function of parameter \( a \) and variable \( x \), \( P(a,x) \). One of the four incomplete gamma functions. Example: \( \gamma_{inc,P}(1,1) = 1 - e^{-1} \)

ln(x)  *Natural Logarithm* Natural logarithm of \( x \), \( \ln(x) \). Example: \( \ln(2.71828182845904523536) = 1.0 \)

log(x)  *Natural Logarithm* Exact synonym for \( \ln(x) \).

log10(x)  *Base 10 Logarithm* Base 10 logarithm of \( x \), \( \log_{10}(x) \). Example: \( \log(10.0) = 1.0 \)

nearbyint(x)  *Round inexactly* Nearest integer to \( x \) is returned in floating point format. No exceptions are raised for *inexact conversions*. Example: \( \text{nearbyint}(0.1) = 0.0 \)

pow(x,y)  *Power* Value of \( x \) is raised to the power of \( y \). Exceptions are raised for *domain errors*. Due to type-limitations in the C language \texttt{pow} function, integer arguments are promoted (see Section 3.34 [Type Conversion], page 99) to type \texttt{NC_FL0AT} before evaluation. Example: \( \text{pow}(2,3) = 8 \)

rint(x)  *Round exactly* Nearest integer to \( x \) is returned in floating point format. Exceptions are raised for *inexact conversions*. Example: \( rint(0.1) = 0 \)

round(x)  *Round* Nearest integer to \( x \) is returned in floating point format. Round halfway cases away from zero, regardless of current IEEE rounding direction. Example: \( \text{round}(0.5) = 1.0 \)

sin(x)  *Sine* Sine of \( x \) where \( x \) is specified in radians. Example: \( \sin(1.57079632679489661922) = 1.0 \)
\( \sinh(x) \)  \hspace{1em} \textit{Hyperbolic sine} \hspace{1em} \text{Hyperbolic sine of } x \text{ where } x \text{ is specified in radians. Example: } \sinh(1.0) = 1.1752

\( \sqrt{x} \)  \hspace{1em} \textit{Square Root} \hspace{1em} \text{Square Root of } x, \sqrt{x}. \text{ Example: } \sqrt{4.0} = 2.0

\( \tan(x) \)  \hspace{1em} \textit{Tangent} \hspace{1em} \text{Tangent of } x \text{ where } x \text{ is specified in radians. Example: } \tan(0.78539816339744830961) = 1.0

\( \tanh(x) \)  \hspace{1em} \textit{Hyperbolic tangent} \hspace{1em} \text{Hyperbolic tangent of } x \text{ where } x \text{ is specified in radians. Example: } \tanh(1.0) = 0.761594155956

\( \text{trunc}(x) \)  \hspace{1em} \textit{Truncate} \hspace{1em} \text{Nearest integer to } x \text{ is returned in floating point format. Round half-way cases toward zero, regardless of current IEEE rounding direction. Example: } \text{trunc}(0.5) = 0.0

The complete list of mathematical functions supported is platform-specific. Functions mandated by ANSI C are \textit{guaranteed} to be present and are indicated with an asterisk \(^3\) and are indicated with an asterisk. Use the ‘-f’ (or ‘fnc_tbl’ or ‘prn_fnc_tbl’) switch to print a complete list of functions supported on your platform. \(^4\)

4.1.26 Operator precedence and associativity

This page lists the \texttt{ncap2} operators in order of precedence (highest to lowest). Their associativity indicates in what order operators of equal precedence in an expression are applied.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ --</td>
<td>Postfix Increment/Decrement</td>
<td>Right to Left</td>
</tr>
<tr>
<td>()</td>
<td>Parentheses (function call)</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>Method call</td>
<td></td>
</tr>
<tr>
<td>++ --</td>
<td>Prefix Increment/Decrement</td>
<td>Right to Left</td>
</tr>
<tr>
<td>+ -</td>
<td>Unary Plus/Minus</td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Logical Not</td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>Power of Operator</td>
<td>Right to Left</td>
</tr>
<tr>
<td>* / %</td>
<td>Multiply/Divide/Modulus</td>
<td>Left To Right</td>
</tr>
<tr>
<td>+ -</td>
<td>Addition/Subtraction</td>
<td>Left To Right</td>
</tr>
<tr>
<td>&gt;&gt; &lt;&lt;</td>
<td>Fortran style array clipping</td>
<td>Left to Right</td>
</tr>
<tr>
<td>&lt; &lt;=</td>
<td>Less than/Less than or equal to</td>
<td>Left to Right</td>
</tr>
<tr>
<td>&gt; &gt;=</td>
<td>Greater than/Greater than or equal to</td>
<td>Left to Right</td>
</tr>
<tr>
<td>== !=</td>
<td>Equal to/Not equal to</td>
<td>Left to Right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
<td>Left to Right</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?:</td>
<td>Ternary Operator</td>
<td>Right to Left</td>
</tr>
</tbody>
</table>

\(^3\) ANSI C compilers are guaranteed to support double-precision versions of these functions. These functions normally operate on netCDF variables of type \texttt{NC\_DOUBLE} without having to perform intrinsic conversions. For example, ANSI compilers provide \texttt{sin} for the sine of C-type \texttt{double} variables. The ANSI standard does not require, but many compilers provide, an extended set of mathematical functions that apply to single (\texttt{float}) and quadruple (\texttt{long double}) precision variables. Using these functions (e.g., \texttt{sinf} for \texttt{float}, \texttt{sinl} for \texttt{long double}), when available, is (presumably) more efficient than casting variables to type \texttt{double}, performing the operation, and then re-casting. NCO uses the faster intrinsic functions when they are available, and uses the casting method when they are not.

\(^4\) Linux supports more of these intrinsic functions than other OSs.
4.1.27 ID Quoting

In this section when I refer to a name I mean a variable name, attribute name or a dimension name. The allowed characters in a valid netCDF name vary from release to release. (See end section). If you want to use metacharacters in a name or use a method name as a variable name then the name has to be quoted wherever it occurs.

The default NCO name is specified by the regular expressions:

\[
\begin{align*}
DGT: & \ ('0'..'9') ; \\
LPH: & \ ('a'..'z' | 'A'..'Z' | '_' ) ; \\
name: & \ (LPH)(LPH|DGT)+
\end{align*}
\]

The first character of a valid name must be alphabetic or the underscore. Any subsequent characters must be alphanumeric or underscore. (e.g., a1_23, hell_is_666)

The valid characters in a quoted name are specified by the regular expressions:

\[
\begin{align*}
LPHDGT: & \ ('a'..'z' | 'A'..'Z' | '_' | '0'..'9') ; \\
name: & \ (LPHDGT|'-'|'+'|'.|'|(|')|'|')|'+');'
\end{align*}
\]

Quote a variable:
'avg', '10+10', 'set_miss' '+-90field', '-test'=10.0d

Quote a attribute:
'three@10', 'set_mss@+10', '666@hell', 't1@+units'='kelvin'

Quote a dimension:
'$10', '$t1-', '$-odd', c1['$10','$t1-']=23.0d

The following comments are from the netCDF library definitions and detail the naming conventions for each release. netcdf-3.5.1
netcdf-3.6.0-p1
netcdf-3.6.1
netcdf-3.6.2

/*
* ( [a-zA-Z][0-9]|'_' |'-' |'+' |'.|'|':|'|@' |'(' |')' ) +
* Verify that name string is valid CDL syntax, i.e., all characters are
* alphanumeric, '-', '_', '+', or '.).
* Also permit ':', '@', '(', or ')' in names for chemists currently making
* use of these characters, but don’t document until ncgen and ncdump can
* also handle these characters in names.
*/
/*
 * Verify that a name string is valid syntax. The allowed name
 * syntax (in RE form) is:
 * *
 * ([a-zA-Z_]\{UTF8\})([\x00-\x1F\x7F/]\{UTF8\})*
 * *
 * where UTF8 represents a multibyte UTF-8 encoding. Also, no
 * trailing spaces are permitted in names. This definition
 * must be consistent with the one in ncgen.l. We do not allow '/'
 * because HDF5 does not permit slashes in names as slash is used as a
 * group separator. If UTF-8 is supported, then a multi-byte UTF-8
 * character can occur anywhere within an identifier. We later
 * normalize UTF-8 strings to NFC to facilitate matching and queries.
 */
4.2 ncatted netCDF Attribute Editor

SYNTAX

```
ncatted [-a att_dsc] [-a ...] [-D dbg] [-h] [--hdr_pad nbr]
infile [[out_file]]
```

DESCRIPTION

ncatted edits attributes in a netCDF file. If you are editing attributes then you are spending too much time in the world of metadata, and ncatted was written to get you back out as quickly and painlessly as possible. ncatted can append, create, delete, modify, and overwrite attributes (all explained below). ncatted allows each editing operation to be applied to every variable in a file. This saves time when changing attribute conventions throughout a file. ncatted is for writing attributes. To read attribute values in plain text, use ncks -m -M, or define something like ncattget as a shell command (see Section 4.7.2 [Filters for ncks], page 196).

Because repeated use of ncatted can considerably increase the size of the history global attribute (see Section 3.36 [History Attribute], page 108), the ‘-h’ switch is provided to override automatically appending the command to the history global attribute in the output-file.

According to the netCDF User Guide, altering metadata in netCDF files does not incur the penalty of recopying the entire file when the new metadata occupies less space than the old metadata. Thus ncatted may run much faster (at least on netCDF3 files) if judicious use of header padding (see Section 3.2 [Metadata Optimization], page 25) was made when producing the input-file. Similarly, using the ‘--hdr_pad’ option with ncatted helps ensure that future metadata changes to output-file occur as swiftly as possible.

When ncatted is used to change the _FillValue attribute, it changes the associated missing data self-consistently. If the internal floating point representation of a missing value, e.g., 1.0e36, differs between two machines then netCDF files produced on those machines will have incompatible missing values. This allows ncatted to change the missing values in files from different machines to a single value so that the files may then be concatenated, e.g., by ncrccat, without losing information. See Section 3.25 [Missing Values], page 71, for more information.

To master ncatted one must understand the meaning of the structure that describes the attribute modification, att_dsc specified by the required option ‘-a’ or ‘--attribute’. This option is repeatable and may be used multiple time in a single ncatted invocation to increase the efficiency of altering multiple attributes. Each att_dsc contains five elements. This makes using ncatted somewhat complicated, though powerful. The att_dsc fields are in the following order:

```
att_dsc = att_nm, var_nm, mode, att_type, att_val
```

att_nm Attribute name. Example: units
var

Variable name. Example: pressure, ^H2O$. Regular expressions (see Section 3.11 [Subsetting Files], page 40) are accepted and will select any matching variable (not attribute) names. The names global and group have special meaning.

mode

Edit mode abbreviation. Example: a. See below for complete listing of valid values of mode.

att_type

Attribute type abbreviation. Example: c. See below for complete listing of valid values of att_type.

att_val

Attribute value. Example: pascal.

There should be no empty space between these five consecutive arguments. The description of these arguments follows in their order of appearance.

The value of att_num is the name of the attribute to edit. This meaning of this should be clear to all ncatted users. Recall, as mentioned above, that var_num (but not att_num) may be specified as a regular expression. If att_num is omitted (i.e., left blank) and Delete mode is selected, then all attributes associated with the specified variable will be deleted.

The value of var_num is the name of the variable containing the attribute (named att_num) that you want to edit. There are three very important and useful exceptions to this rule. The value of var_num can also be used to direct ncatted to edit global attributes, or to repeat the editing operation for every group or variable in a file. A value of var_num of global indicates that att_num refers to a global (i.e., root-level) attribute, rather than to a particular variable’s attribute. This is the method ncatted supports for editing global attributes. A value of var_num of group indicates that att_num refers to all groups, rather than to a particular variable’s or group’s attribute. The operation will proceed to edit group metadata for every group. Finally, if var_num is left blank, then ncatted attempts to perform the editing operation on every variable in the file. This option may be convenient to use if you decide to change the conventions you use for describing the data.

The value of mode is a single character abbreviation (a, c, d, m, or o) standing for one of five editing modes:

a  Append. Append value att_val to current var_num attribute att_num value att_val, if any. If var_num does not have an attribute att_num, there is no effect.

c  Create. Create variable var_num attribute att_num with att_val if att_num does not yet exist. If var_num already has an attribute att_num, there is no effect, so the existing attribute is preserved without change.

d  Delete. Delete current var_num attribute att_num. If var_num does not have an attribute att_num, there is no effect. If att_num is omitted (left blank), then all attributes associated with the specified variable are automatically deleted. When Delete mode is selected, the att_type and att_val arguments are superfluous and may be left blank.

m  Modify. Change value of current var_num attribute att_num to value att_val. If var_num does not have an attribute att_num, there is no effect.
Overwrite. Write attribute \texttt{att nm} with value \texttt{att val} to variable \texttt{var nm}, overwriting existing attribute \texttt{att nm}, if any. This is the default mode.

The value of \texttt{att type} is a single character abbreviation (f, d, l, i, s, c, b, u) or a short string standing for one of the twelve primitive netCDF data types:

- \texttt{f} \textit{Float}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_FLOAT.
- \texttt{d} \textit{Double}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_DOUBLE.
- \texttt{i, l} \textit{Integer} or \textit{Long}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_INT.
- \texttt{s} \textit{Short}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_SHORT.
- \texttt{c} \textit{Char}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_CHAR.
- \texttt{b} \textit{Byte}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_BYTE.
- \texttt{ub} \textit{Unsigned Byte}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_UBYTE.
- \texttt{us} \textit{Unsigned Short}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_USHORT.
- \texttt{u, ui, ul} \textit{Unsigned Int}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_UINT.
- \texttt{ll, int64} \textit{Int64}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_INT64.
- \texttt{ull, uint64} \textit{Uint64}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_UINT64.
- \texttt{sng, string} \textit{String}. Value(s) specified in \texttt{att val} will be stored as netCDF intrinsic type NC\_STRING. Note that ncatted handles type NC\_STRING attributes correctly beginning with version 4.3.3 released in July, 2013. Earlier versions fail when asked to handle NC\_STRING attributes.

In \textit{Delete} mode the specification of \texttt{att type} is optional (and is ignored if supplied).

The value of \texttt{att val} is what you want to change attribute \texttt{att nm} to contain. The specification of \texttt{att val} is optional in \textit{Delete} (and is ignored) mode. Attribute values for all types besides NC\_CHAR must have an attribute length of at least one. Thus \texttt{att val} may be a single value or one-dimensional array of elements of type \texttt{att type}. If the \texttt{att val} is not set or is set to empty space, and the \texttt{att type} is NC\_CHAR, e.g., -a units,T,o,c,"
or -a units,T,o,c,, then the corresponding attribute is set to have zero length. When
specifying an array of values, it is safest to enclose att_val in single or double quotes,
e.g., -a levels,T,o,s,"1,2,3,4" or -a levels,T,o,s, ’1,2,3,4’ . The quotes are strictly
unnecessary around att_val except when att_val contains characters which would confuse
the calling shell, such as spaces, commas, and wildcard characters.

NCO processing of NC_CHAR attributes is a bit like Perl in that it attempts to do what
you want by default (but this sometimes causes unexpected results if you want unusual
data storage). If the att_type is NC_CHAR then the argument is interpreted as a string and it
may contain C-language escape sequences, e.g., \n, which NCO will interpret before writing
anything to disk. NCO translates valid escape sequences and stores the appropriate ASCII
code instead. Since two byte escape sequences, e.g., \n, represent one-byte ASCII codes,
e.g., ASCII 10 (decimal), the stored string attribute is one byte shorter than the input string
length for each embedded escape sequence. The most frequently used C-language escape
sequences are \n (for linefeed) and \t (for horizontal tab). These sequences in particular
allow convenient editing of formatted text attributes. The other valid ASCII codes are
\a, \b, \f, \r, \v, and \\. See Section 4.7 [ncks netCDF Kitchen Sink], page 188, for more
examples of string formatting (with the ncks ‘-s’ option) with special characters.

Analogous to printf, other special characters are also allowed by ncatted if they are
“protected” by a backslash. The characters ", ’, ?, and \ may be input to the shell as \", \', \?, and \\. NCO simply strips away the leading backslash from these characters before
editing the attribute. No other characters require protection by a backslash. Backslashes
which precede any other character (e.g., 3, m, $, 1, & , @, %, {, and }) will not be filtered and
will be included in the attribute.

Note that the NUL character \0 which terminates C language strings is assumed and
need not be explicitly specified. If \0 is input, it is translated to the NUL character.
However, this will make the subsequent portion of the string, if any, invisible to C standard
library string functions. And that may cause unintended consequences. Because of these
context-sensitive rules, one must use ncatted with care in order to store data, rather than
text strings, in an attribute of type NC_CHAR.

Note that ncatted interprets character attributes (i.e., attributes of type NC_CHAR) as
strings. EXAMPLES

Append the string Data version 2.0.\n to the global attribute history:

    ncatted -a history,global,a,c,'Data version 2.0\n' in.nc

Note the use of embedded C language printf()-style escape sequences.

Change the value of the long_name attribute for variable T from whatever it currently
is to “temperature”:

    ncatted -a long_name,T,o,c,temperature in.nc

NCO arithmetic operators may not work as expected on IEEE NaN (short for Not-a-
Number) and NaN-like numbers such as positive infinity and negative infinity\footnote{Arithmetic
comparisons to NaN and NaN-like numbers always return False, contrary to the behavior of
all other numbers. This is difficult to inuit, yet is the behavior defined for NaN by IEEE 754. Using NaN
for the missing value in datasets is legal. We strongly discourage it.}. One way
to work-around this problem is to change IEEE NaNs to normal missing values. As of NCO 4.1.0 (March, 2012), ncatted works with NaNs. First set the missing value (i.e., the value of the _FillValue attribute) for the variable(s) in question to the IEEE NaN value.

```
ncatted -a _FillValue,,o,f,NaN in.nc
```

Then change the missing value from the IEEE NaN value to a normal IEEE number, like 1.0e36 (or to whatever the original missing value was).

```
ncatted -a _FillValue,,m,f,1.0e36 in.nc
```

Delete all existing units attributes:

```
ncatted -a units,,d,, in.nc
```

The value of var_num was left blank in order to select all variables in the file. The values of att_type and att_val were left blank because they are superfluous in Delete mode.

Delete all attributes associated with the tpt variable, and delete all global attributes

```
ncatted -a ,tpt,d,, -a ,global,d,, in.nc
```

The value of att_num was left blank in order to select all attributes associated with the variable. To delete all global attributes, simply replace tpt with global in the above.

Modify all existing units attributes to meter second-1:

```
ncatted -a units,,m,c,'meter second-1' in.nc
```

Add a units attribute of kilogram kilogram-1 to all variables whose first three characters are ‘H2O’:

```
ncatted -a units,‘^H2O’*,c,c,'kilogram kilogram-1' in.nc
```

Overwrite the quanta attribute of variable energy to an array of four integers.

```
ncatted -a quanta,energy,o,s,'010,101,111,121' in.nc
```

As of NCO 3.9.6 (January, 2009), ncatted accepts extended regular expressions as arguments for variable names, though not for attribute names. Create isotope attributes for all variables containing ‘H2O’ in their names.

```
ncatted -a isotope,'^H2O*$',c,s,'18' in.nc
```

See Section 3.11 [Subsetting Files], page 40 for more details.

As of NCO 4.3.8 (November, 2013), ncatted accepts full and partial group paths in names of attributes, variables, dimensions, and groups.

```
# Overwrite units attribute of specific 'lon' variable
ncatted -O -a units,/g1/lon,o,c,'degrees_west' in_grp.nc
# Overwrite units attribute of all 'lon' variables
ncatted -O -a units,lon,o,c,'degrees_west' in_grp.nc
# Delete units attribute of all 'lon' variables
ncatted -O -a units,lon,d,, in_grp.nc
# Overwrite units attribute with new type for specific 'lon' variable
ncatted -O -a units,/g1/lon,o,sng,'degrees_west' in_grp.nc
```
# Add new_att attribute to all variables
ncatted -O -a new_att,,c,sng,'new variable attribute’ in_grp.nc
# Add new_grp_att group attribute to all groups
ncatted -O -a new_grp_att,group,c,sng,’new group attribute’ in_grp.nc
# Add new_grp_att group attribute to single group
ncatted -O -a g1_grp_att,g1,c,sng,’new group attribute’ in_grp.nc
# Add new_glb_att global attribute to root group
ncatted -O -a new_glb_att,global,c,sng,’new global attribute’ in_grp.nc

Demonstrate input of C-language escape sequences (e.g., \n) and other special characters (e.g., \"")

ncatted -h -a special,global,o,c,
  ’\nDouble quote: "\nTwo consecutive double quotes: ""
Single quote: Beyond my shell abilities!\nBackslash: \\
Two consecutive backslashes: \\
Question mark: ?\n’ in.nc

Note that the entire attribute is protected from the shell by single quotes. These outer single quotes are necessary for interactive use, but may be omitted in batch scripts.
4.3 ncbo netCDF Binary Operator

SYNTAX

```
    [-D dbg] [-d dim,[min][,[max][,[stride]]] [-F]
    [-G gpe_dsc] [-g grp[,...]] [-h] [--hdr_pad nbr]
    [-L dfl_lvl] [-l path] [--no_tmp_fl]
    [-t thr_nbr] [--unn] [-v var[,...]] [-X ...] [-x] [-y op_typ]
file_1 file_2 [file_3]
```

DESCRIPTION

ncbo performs binary operations on variables in file_1 and the corresponding variables (those with the same name) in file_2 and stores the results in file_3. The binary operation operates on the entire files (modulo any excluded variables). See Section 3.25 [Missing Values], page 71, for treatment of missing values. One of the four standard arithmetic binary operations currently supported must be selected with the ‘-y op_typ’ switch (or long options ‘--op_typ’ or ‘--operation’). The valid binary operations for ncbo, their definitions, corresponding values of the op_typ key, and alternate invocations are:

**Addition**

Definition: `file_3 = file_1 + file_2`
Alternate invocation: `ncadd`
op_typ key values: ‘add’, ‘+’, ‘addition’
Examples: `ncbo --op_typ=add 1.nc 2.nc 3.nc`, `ncadd 1.nc 2.nc 3.nc`

**Subtraction**

Definition: `file_3 = file_1 - file_2`
Alternate invocations: `ncdiff`, `ncsub`, `ncsubtract`
Examples: `ncbo --op_typ=- 1.nc 2.nc 3.nc`, `ncdiff 1.nc 2.nc 3.nc`

**Multiplication**

Definition: `file_3 = file_1 * file_2`
Alternate invocations: `ncmult`, `ncmultiply`
op_typ key values: ‘mlt’, ‘*’, ‘mult’, ‘multiply’, ‘multiplication’
Examples: `ncbo --op_typ=mlt 1.nc 2.nc 3.nc`, `ncmult 1.nc 2.nc 3.nc`

**Division**

Definition: `file_3 = file_1 / file_2`
Alternate invocation: `ncdivide`
op_typ key values: ‘dvd’, ‘/’, ‘divide’, ‘division’
Examples: `ncbo --op_typ=/ 1.nc 2.nc 3.nc`, `ncdivide 1.nc 2.nc 3.nc`
Care should be taken when using the shortest form of key values, i.e., ‘+’, ‘-’, ‘*’, and ‘/’. Some of these single characters may have special meanings to the shell. Place these characters inside quotes to keep them from being interpreted (globbed) by the shell. For example, the following commands are equivalent:

```
ncbo --op_typ=* 1.nc 2.nc 3.nc # Dangerous (shell may try to glob)
ncbo --op_typ="*" 1.nc 2.nc 3.nc # Safe ('*' protected from shell)
ncbo --op_typ=mlt 1.nc 2.nc 3.nc
ncbo --op_typ=mult 1.nc 2.nc 3.nc
ncbo --op_typ=multiply 1.nc 2.nc 3.nc
ncbo --op_typ=multiplication 1.nc 2.nc 3.n
cnmult 1.nc 2.nc 3.nc # First do 'ln -s ncbo ncmult'
ncmultiply 1.nc 2.nc 3.nc # First do 'ln -s ncbo ncmultiply'
```

No particular argument or invocation form is preferred. Users are encouraged to use the forms which are most intuitive to them.

Normally, `ncbo` will fail unless an operation type is specified with ‘-y’ (equivalent to ‘--op_typ’). You may create exceptions to this rule to suit your particular tastes, in conformance with your site’s policy on symbolic links to executables (files of a different name point to the actual executable). For many years, `ncdiff` was the main binary file operator. As a result, many users prefer to continue invoking `ncdiff` rather than memorizing a new command (`ncbo -y sbt`) which behaves identically to the original `ncdiff` command. However, from a software maintenance standpoint, maintaining a distinct executable for each binary operation (e.g., `ncadd`) is untenable, and a single executable, `ncbo`, is desirable. To maintain backward compatibility, therefore, NCO automatically creates a symbolic link from `ncbo` to `ncdiff`. Thus `ncdiff` is called an alternate invocation of `ncbo`. `ncbo` supports many additional alternate invocations which must be manually activated. Should users or system administrators decide to activate them, the procedure is simple. For example, to use `ncadd` instead of `ncbo --op_typ=add`, simply create a symbolic link from `ncbo` to `ncadd`. The alternate invocations supported for each operation type are listed above. Alternatively, users may always define `ncadd` as an alias to `ncbo --op_typ=add`.

It is important to maintain portability in NCO scripts. Therefore we recommend that site-specific invocations (e.g., `ncadd`) be used only in interactive sessions from the command-line. For scripts, we recommend using the full invocation (e.g., `ncbo --op_typ=add`). This ensures portability of scripts between users and sites.

`ncbo` operates (e.g., adds) variables in `file_2` with the corresponding variables (those with the same name) in `file_1` and stores the results in `file_3`. Variables in `file_1` or `file_2` are:

---

1. A naked (i.e., unprotected or unquoted) ‘*’ is a wildcard character. A naked ‘-’ may confuse the command line parser. A naked ‘+’ and ‘/’ are relatively harmless.
2. The widely used shell Bash correctly interprets all these special characters even when they are not quoted. That is, Bash does not prevent NCO from correctly interpreting the intended arithmetic operation when the following arguments are given (without quotes) to `ncbo`: ‘--op_typ=+’, ‘--op_typ=-’, ‘--op_typ=’, and ‘--op_typ=/’
3. The command to do this is ‘ln -s -f ncbo ncadd’
4. The command to do this is ‘alias ncadd="ncbo --op_typ=add"’
are broadcast to conform to the corresponding variable in the other input file if necessary. Now ncbo is completely symmetric with respect to file_1 and file_2, i.e., file_1 − file_2 = −(file_2 − file_1).

Broadcasting a variable means creating data in non-existing dimensions by copying data in existing dimensions. For example, a two dimensional variable in file_2 can be subtracted from a four, three, or two (not one or zero) dimensional variable (of the same name) in file_1. This functionality allows the user to compute anomalies from the mean. In the future, we will broadcast variables in file_1, if necessary to conform to their counterparts in file_2. Thus, presently, the number of dimensions, or rank, of any processed variable in file_1 must be greater than or equal to the rank of the same variable in file_2. Of course, the size of all dimensions common to both file_1 and file_2 must be equal.

When computing anomalies from the mean it is often the case that file_2 was created by applying an averaging operator to a file with initially the same dimensions as file_1 (often file_1 itself). In these cases, creating file_2 with ncra rather than ncwa will cause the ncbo operation to fail. For concreteness say the record dimension in file_1 is time. If file_2 was created by averaging file_1 over the time dimension with the ncra operator (rather than with the ncwa operator), then file_2 will have a time dimension of size 1 rather than having no time dimension at all. In this case the input files to ncbo, file_1 and file_2, will have unequally sized time dimensions which causes ncbo to fail. To prevent this from occuring, use ncwa to remove the time dimension from file_2. See the example below.

ncbo never operates on coordinate variables or variables of type NC_CHAR or NC_STRING. This ensures that coordinates like (e.g., latitude and longitude) are physically meaningful in the output file, file_3. This behavior is hardcoded. ncbo applies special rules to some CF-defined (and/or NCAR CCSM or NCAR CCM fields) such as ORO. See Section 3.38 [CF Conventions], page 109 for a complete description. Finally, we note that ncflint (see Section 4.6 [ncflint netCDF File Interpolator], page 185) is designed for file interpolation. As such, it also performs file subtraction, addition, multiplication, albeit in a more convoluted way than ncbo.

Beginning with NCO version 4.3.1 (May, 2013), ncbo supports group broadcasting. Group broadcasting means processing data based on group patterns in the input file(s) and automatically transferring or transforming groups to the output file. Consider the case where file_1 contains multiple groups each with the variable v1, while file_2 contains v1 only in its top-level (i.e., root) group. Then ncbo will replicate the group structure of file_1 in the output file, file_3. Each group in file_3 contains the output of the corresponding group in file_1 operating on the data in the single group in file_2. An example is provided below.

EXAMPLES

Say files 85_0112.nc and 86_0112.nc each contain 12 months of data. Compute the change in the monthly averages from 1985 to 1986:

---

5 Prior to NCO version 4.3.1 (May, 2013), ncbo would only broadcast variables in file_2 to conform to file_1. Variables in file_1 were never broadcast to conform to the dimensions in file_2.

6 This is because ncra collapses the record dimension to a size of 1 (making it a degenerate dimension), but does not remove it, while, unless ‘-b’ is given, ncwa removes all averaged dimensions. In other words, by default ncra changes variable size though not rank, while, ncwa changes both variable size and rank.
These commands are all different ways of expressing the same thing.

The following examples demonstrate the broadcasting feature of ncbo. Say we wish to compute the monthly anomalies of T from the yearly average of T for the year 1985. First we create the 1985 average from the monthly data, which is stored with the record dimension time.

```bash
ncra 85_0112.nc 85.nc
ncwa -O -a time 85.nc 85.nc
```

The second command, ncwa, gets rid of the time dimension of size 1 that ncra left in 85.nc. Now none of the variables in 85.nc has a time dimension. A quicker way to accomplish this is to use ncwa from the beginning:

```bash
ncwa -a time 85_0112.nc 85.nc
```

We are now ready to use ncbo to compute the anomalies for 1985:

```bash
ncdiff -v T 85_0112.nc 85.nc t_anm_85_0112.nc
```

Each of the 12 records in t_anm_85_0112.nc now contains the monthly deviation of T from the annual mean of T for each gridpoint.

Say we wish to compute the monthly gridpoint anomalies from the zonal annual mean. A zonal mean is a quantity that has been averaged over the longitudinal (or x) direction. First we use ncwa to average over longitudinal direction lon, creating 85_x.nc, the zonal mean of 85.nc. Then we use ncbo to subtract the zonal annual means from the monthly gridpoint data:

```bash
ncwa -a lon 85.nc 85_x.nc
ncdiff 85_0112.nc 85_x.nc tx_anm_85_0112.nc
```

This examples works assuming 85_0112.nc has dimensions time and lon, and that 85_x.nc has no time or lon dimension.

Group broadcasting simplifies evaluation of multiple models against observations. Consider the input file cmip5.nc which contains multiple top-level groups cesm, ecmwf, and giss, each of which contains the surface air temperature field tas. We wish to compare these models to observations stored in obs.nc which contains tas only in its top-level (i.e., root) group. It is often the case that many models and/or model simulations exist, whereas only one observational dataset does. We evaluate the models and obtain the bias (difference) between models and observations by subtracting obs.nc from cmip5.nc. Then ncbo “broadcasts” (i.e., replicates) the observational data to match the group structure of cmip5.nc, subtracts, and then stores the results in the output file, bias.nc which has the same group structure as cmip5.nc.

```bash
% ncbo -O cmip5.nc obs.nc bias.nc
% ncks -H -v tas -d time,3 bias.nc
/cesm/tas
```
As a final example, say we have five years of monthly data (i.e., 60 months) stored in 8501_8912.nc and we wish to create a file which contains the twelve month seasonal cycle of the average monthly anomaly from the five-year mean of this data. The following method is just one permutation of many which will accomplish the same result. First use `ncwa` to create the five-year mean:

```bash
ncwa -a time 8501_8912.nc 8589.nc
```

Next use `ncbo` to create a file containing the difference of each month’s data from the five-year mean:

```bash
ncbo 8501_8912.nc 8589.nc t_anm_8501_8912.nc
```

Now use `ncks` to group together the five January anomalies in one file, and use `ncra` to create the average anomaly for all five Januarys. These commands are embedded in a shell loop so they are repeated for all twelve months:

```bash
for idx in {1..12}; do # Bash Shell (version 3.0+)
  idx='printf "%02d" ${idx}'' # Zero-pad to preserve order
  ncks -F -d time,${idx},,12 t_anm_8501_8912.nc foo.$(idx)
  ncra foo.$(idx) t_anm_8589_${idx}.nc
done
```

for idx in 01 02 03 04 05 06 07 08 09 10 11 12; do # Bourne Shell
  ncks -F -d time,${idx},,12 t_anm_8501_8912.nc foo.$(idx)
  ncra foo.$(idx) t_anm_8589_${idx}.nc
done

for each idx (01 02 03 04 05 06 07 08 09 10 11 12) # C Shell
  ncks -F -d time,${idx},,12 t_anm_8501_8912.nc foo.$(idx)
  ncra foo.$(idx) t_anm_8589_${idx}.nc
end

Note that `ncra` understands the `stride` argument so the two commands inside the loop may be combined into the single command:

```
ncra -F -d time,${idx},,12 t_anm_8501_8912.nc foo.$(idx)
```

Finally, use `ncrcat` to concatenate the 12 average monthly anomaly files into one twelve-record file which contains the entire seasonal cycle of the monthly anomalies:

```
ncrcat t_anm_8589_??.nc t_anm_8589_0112.nc
```
4.4 nces netCDF Ensemble Statistics

SYNTAX


[-D dbg] [-d dim,[min],[max],[stride]] [-F]

[-G gpe_dsc] [-g grp[,...]] [-h] [--hdf] [--hdr_pad nbr]


[[-t thr_nbr] [--unn] [-v var[,...]] [-X ...] [-x] [-y op_typ]}

[input-files] [output-file]

DESCRIPTION

nces performs gridpoint statistics on variables across an arbitrary number (an ensemble) of input-files and/or of input groups within each file. Each file (or group) receives an equal weight. nces was formerly (until NCO version 4.3.9, released December, 2013) known as ncea (netCDF Ensemble Averager). For example, nces will average a set of files or groups, weighting each file or group evenly. This is distinct from ncra, which performs statistics only over the record dimension(s) (e.g., time), and weights each record in each record dimension evenly.

The file or group is the logical unit of organization for the results of many scientific studies. Often one wishes to generate a file or group which is the statistical product (e.g., average) of many separate files or groups. This may be to reduce statistical noise by combining the results of a large number of experiments, or it may simply be a step in a procedure whose goal is to compute anomalies from a mean state. In any case, when one desires to generate a file whose statistical properties are equally influenced by all the inputs, then nces is the operator to use.

Variables in the output-file are the same size as the variable hyperslab in each input file or group, and each input file or group must be the same size after hyperslabbing. nces does allow files to differ in the record dimension size if the requested record hyperslab (see Section 3.15 [Hyperslabs], page 55) resolves to the same size for all files. nces recomputes the record dimension hyperslab limits for each input file so that coordinate limits may be used to select equal length timeseries from unequal length files. This simplifies analysis of unequal length timeseries from simulation ensembles (e.g., the CMIP3 IPCC AR4 archive).

nces works in one of two modes, file ensembles or group ensembles. File ensembles are the default (equivalent to the old ncea) and may also be explicitly specified by the ‘--nsm_fl’ or ‘--ensemble_file’ switches. To perform statistics on ensembles of groups, a

---

1 The old ncea command was deprecated in NCO version 4.3.9, released December, 2013. NCO will attempt to maintain back-compatibility and work as expected with invocations of ncea for as long as possible. Please replace ncea by nces in all future work.

2 As of NCO version 4.4.2 (released February, 2014) nces allows hyperslabs in all dimensions so long as the hyperslabs resolve to the same size. The fixed (i.e., non-record) dimensions should be the same size in all ensemble members both before and after hyperslabbing, although the hyperslabs may (and usually do) change the size of the dimensions from the input to the output files. Prior to this, nces was only guaranteed to work on hyperslabs in the record dimension that resolved to the same size.
newer feature, use ‘--nsm_grp’ or ‘--ensemble_group’. Members of a group ensemble are
groups that share the same structure, parent group, and nesting level. Members must be
leaf groups, i.e., not contain any sub-groups. Their contents usually have different values
because they are realizations of replicated experiments. In group ensemble mode nces
computes the statistics across the ensemble, which may span multiple input files. Files may
contain members of multiple, distinct ensembles. However, all ensembles must have at least
one member in the first input file. Group ensembles behave as an unlimited dimension of
datasets: they may contain an arbitrary and extensible number of realizations in each file,
and may be composed from multiple files.

Output statistics in group ensemble mode are stored in the parent group by default. If the
ensemble members are /cesm/csm_01 and /cesm/csm_02, then the computed statistic will
be in /cesm in the output file. The ‘--nsm_sfx’ option instructs nces to instead store output
in a new child group of the parent created by attaching the suffix to the parent group’s name,
e.g., ‘--nsm_sfx=_avg’ would store results in the output group /cesm/csm_avg:

    nces --nsm_grp mdl1.nc mdl2.nc mdl3.nc out.nc
    nces --nsm_grp --nsm_sfx=_avg mdl1.nc mdl2.nc mdl3.nc out.nc

See Section 2.6 [Statistics vs. Concatenation], page 18, for a description of the distinc-
tions between the statistics tools and concatenators. As a multi-file operator, nces will
read the list of input-files from stdin if they are not specified as positional arguments on
the command line (see Section 2.7 [Large Numbers of Files], page 19).

Like ncra and ncwa, nces treats coordinate variables as a special case. Coordinate
variables are assumed to be the same in all ensemble members, so nces simply copies the
coordinate variables that appear in ensemble members directly to the output file. This
has the same effect as averaging the coordinate variable across the ensemble, yet does not
incur the time- or precision- penalties of actually averaging them. ncra and ncwa allow
coordinate variables to be processed only by the linear average operation, regardless of
the arithmetic operation type performed on the non-coordinate variables (see Section 3.33
[Operation Types], page 94). Thus it can be said that the three operators (ncra, ncwa, and
nces) all average coordinate variables (even though nces simply copies them). All other
requested arithmetic operations (e.g., maximization, square-root, RMS) are applied only to
non-coordinate variables. In these cases the linear average of the coordinate variable will
be returned.

EXAMPLES

Consider a model experiment which generated five realizations of one year of data, say
1985. Imagine that the experimenter slightly perturbs the initial conditions of the problem
before generating each new solution. Assume each file contains all twelve months (a seasonal
cycle) of data and we want to produce a single file containing the ensemble average (mean)
seasonal cycle. Here the numeric filename suffix denotes the realization number (/not the
month):

    nces 85_01.nc 85_02.nc 85_03.nc 85_04.nc 85_05.nc 85.nc
    nces 85_0[1-5].nc 85.nc
    nces -n 5,2,1 85_01.nc 85.nc
These three commands produce identical answers. See Section 3.5 [Specifying Input Files], page 28, for an explanation of the distinctions between these methods. The output file, 85.nc, is the same size as the inputs files. It contains 12 months of data (which might or might not be stored in the record dimension, depending on the input files), but each value in the output file is the average of the five values in the input files.

In the previous example, the user could have obtained the ensemble average values in a particular spatio-temporal region by adding a hyperslab argument to the command, e.g.,

```bash
nces -d time,0,2 -d lat,-23.5,23.5 85_??\.nc 85.nc
```

In this case the output file would contain only three slices of data in the time dimension. These three slices are the average of the first three slices from the input files. Additionally, only data inside the tropics is included.

As of NCO version 4.3.9 (released December, 2013) nces also works with groups (rather than files) as the fundamental unit of the ensemble. Consider two ensembles, /ecmwf and /cesm stored across three input files mdl1.nc, mdl2.nc, and mdl3.nc. Ensemble members would be leaf groups with names like /ecmwf/01, /ecmwf/02 etc. and /cesm/01, /cesm/02, etc. These commands average both ensembles:

```bash
nces --nsm_grp mdl1.nc mdl2.nc mdl3.nc out.nc
nces --nsm_grp --nsm_sfx='\_min' --op_typ=min -n 3,1,1 mdl1.nc out.nc
nces --nsm_grp -g cesm -v tas -d time,0,3 -n 3,1,1 mdl1.nc out.nc
```

The first command stores averages in the output groups /cesm and /ecmwf, while the second stores minima in the output groups /cesm/cesm_min and /ecmwf/ecmwf_min: The third command demonstrates that sub-setting and hyperslabbing work as expected. Note that each input file may contain different numbers of members of each ensemble, as long as all distinct ensembles contain at least one member in the first file.
4.5 ncecat netCDF Ensemble Concatenator

SYNTAX

```
  [-D dbg] [-d dim,[min],[max],[stride]] [-F]
  [-G gpe_dsc] [-g grp[,...]] [--gag] [-h] [-hdf] [-hdr_pad nbr]
  [-t thr_nbr] [-u ulm_nm] [-unn] [-v var[,...]] [-X ...] [-x]
  [input-files] [output-file]
```

DESCRIPTION

ncecat aggregates an arbitrary number of input files into a single output file using one of two methods. **Record AGgregation** (RAG), the traditional method employed on (flat) netCDF3 files and still the default method, stores input-files as consecutive records in the **output-file**. **Group AGgregation** (GAG) stores input-files as top-level groups in the netCDF4 **output-file**. Record Aggregation (RAG) makes numerous assumptions about the structure of input files whereas Group Aggregation (GAG) makes none. Both methods are described in detail below. Since ncecat aggregates all the contents of the input files, it can easily produce large output files so it is often helpful to invoke subsetting simultaneously (see Section 3.11 [Subsetting Files], page 40).

RAG makes each variable (except coordinate variables) in each input file into a single record of the same variable in the output file. Coordinate variables are not concatenated, they are instead simply copied from the first input file to the output-file. All input-files must contain all extracted variables (or else there would be “gaps” in the output file).

A new record dimension is the glue which binds together the input file data. The new record dimension is defined in the root group of the output file so it is visible to all subgroups. Its name is, by default, “record”. This default name can be overridden with the `-u ulm_nm` short option (or the `--ulm_nm` or `rcd_nm` long options).

Each extracted variable must be constant in size and rank across all input-files. The only exception is that ncecat allows files to differ in the record dimension size if the requested record hyperslab (see Section 3.15 [Hyperslabs], page 55) resolves to the same size for all files. This allows easier gluing/averaging of unequal length timeseries from simulation ensembles (e.g., the CMIP rchive).

Classic (i.e., all netCDF3 and `NETCDF4_CLASSIC`) output files can contain only one record dimension. ncecat makes room for the new glue record dimension by changing the pre-existing record dimension, if any, in the input files into a fixed dimension in the output file. netCDF4 output files may contain any number of record dimensions, so ncecat need not and does not alter the record dimensions, if any, of the input files as it copies them to the output file.

**Group AGgregation** (GAG) stores input-files as top-level groups in the **output-file**. No assumption is made about the size or shape or type of a given object (variable or dimension or group) in the input file. The entire contents of the extracted portion of each input
file is placed in its own top-level group in output-file, which is automatically made as a netCDF4-format file.

GAG has two methods to specify group names for the output-file. The ‘-G’ option, or its long-option equivalent ‘--gpe’, takes as argument a group path editing description gpe_dsc of where to place the results. Each input file needs a distinct output group name to avoid namespace conflicts in the output-file. Hence ncecat automatically creates unique output group names based on either the input filenames or the gpe_dsc arguments. When the user provides gpe_dsc (i.e., with ‘-G’), then the output groups are formed by enumerating sequential two-digit numeric suffixes starting with zero, and appending them to the specified group path (see Section 3.13 [Group Path Editing], page 46). When gpe_dsc is not provided (i.e., user requests GAG with ‘--gag’ instead of ‘-G’), then ncecat forms the output groups by stripping the input file name of any type-suffix (e.g., .nc), and all but the final component of the full filename.

\[
\text{ncecat} \quad --gag \quad 85.\text{nc} \quad 86.\text{nc} \quad 87.\text{nc} \quad 8587.\text{nc} \quad \text{# Output groups } 85, 86, 87
\]
\[
\text{ncecat} \quad -G \quad 85_\quad \text{a.\text{nc}} \quad \text{b.\text{nc}} \quad \text{c.\text{nc}} \quad 8589.\text{nc} \quad \text{# Output groups } 85_00, 85_01, 85_02
\]
\[
\text{ncecat} \quad -G \quad 85/ \quad \text{a.\text{nc}} \quad \text{b.\text{nc}} \quad \text{c.\text{nc}} \quad 8589.\text{nc} \quad \text{# Output groups } 85/00, 85/01, 85/02
\]

With both RAG and GAG the output-file size is the sum of the sizes of the extracted variables in the input files. See Section 2.6 [Statistics vs. Concatenation], page 18, for a description of the distinctions between the various statistics tools and concatenators. As a multi-file operator, ncecat will read the list of input-files from stdin if they are not specified as positional arguments on the command line (see Section 2.7 [Large Numbers of Files], page 19).

Suppress global metadata copying. By default NCO’s multi-file operators copy the global metadata from the first input file into output-file. This helps to preserve the provenance of the output data. However, the use of metadata is burgeoning and is not uncommon to encounter files with excessive amounts of extraneous metadata. Extracting small bits of data from such files leads to output files which are much larger than necessary due to the automatically copied metadata. ncecat supports turning off the default copying of global metadata via the ‘-M’ switch (or its long option equivalents, ‘--global_metadata_suppress’).

Consider five realizations, 85a.nc, 85b.nc, ... 85e.nc of 1985 predictions from the same climate model. Then ncecat 85?.nc 85_ens.nc glues together the individual realizations into the single file, 85_ens.nc. If an input variable was dimensioned [lat,lon], it will by default have dimensions [record,lat,lon] in the output file. A restriction of ncecat is that the hyperslabs of the processed variables must be the same from file to file. Normally this means all the input files are the same size, and contain data on different realizations of the same variables.

Concatenating a variable packed with different scales across multiple datasets is beyond the capabilities of ncecat (and ncrat, the other concatenator (Section 2.6.1 [Concatenation], page 18). ncecat does not unpack data, it simply copies the data from the input-files, and the metadata from the first input-file, to the output-file. This means that data compressed with a packing convention must use the identical packing parameters (e.g., scale_factor and add_offset) for a given variable across all input files. Otherwise the concatenated dataset will not unpack correctly. The workaround for cases where the pack-
ing parameters differ across input-files requires three steps: First, unpack the data using ncpdq. Second, concatenate the unpacked data using ncecat. Third, re-pack the result with ncpdq.

EXAMPLES

Consider a model experiment which generated five realizations of one year of data, say 1985. You can imagine that the experimenter slightly perturbs the initial conditions of the problem before generating each new solution. Assume each file contains all twelve months (a seasonal cycle) of data and we want to produce a single file containing all the seasonal cycles. Here the numeric filename suffix denotes the experiment number (not the month):

```
ncecat 85_01.nc 85_02.nc 85_03.nc 85_04.nc 85_05.nc 85.nc
ncecat 85_0[1-5].nc 85.nc
ncecat -n 5,2,1 85_01.nc 85.nc
```

These three commands produce identical answers. See Section 3.5 [Specifying Input Files], page 28, for an explanation of the distinctions between these methods. The output file, 85.nc, is five times the size as a single input-file. It contains 60 months of data.

One often prefers that the (new) record dimension have a more descriptive, context-based name than simply “record”. This is easily accomplished with the ‘-u ulm_nm’ switch:

```
ncecat -u realization 85_0[1-5].nc 85.nc
```

Users are more likely to understand the data processing history when such descriptive coordinates are used.

Consider a file with an existing record dimension named time. and suppose the user wishes to convert time from a record dimension to a non-record dimension. This may be useful, for example, when the user has another use for the record variable. The simplest method is to use ‘ncks --fix_rec_dmn’ but another possibility is to use ncecat followed by ncwa:

```
ncecat in.nc out.nc # Convert time to non-record dimension
ncwa -a record in.nc out.nc # Remove new degenerate record dimension
```

The second step removes the degenerate record dimension. See Section 4.8 [ncpdq netCDF Permute Dimensions Quickly], page 202 and Section 4.7 [ncks netCDF Kitchen Sink], page 188 for other methods of of changing variable dimensionality, including the record dimension.
4.6 ncflint netCDF File Interpolator

SYNTAX

       [-D dbg] [-d dim,[min],[max],[strid]]
       [-F] [-fix_rec_crd] [-G gpe_dsc] [-g grp[,....]] [-h] [-hdr_pad nbr]
       [-i var,val3] [-L dl1_lvl] [-l path] [-no_tmp_fl]
       [-t thr_nbr] [-unn] [-v var[,....]] [-w wgt1[,wgt2]] [-X ...] [-x]
       file_1 file_2 [file_3]

DESCRIPTION

ncflint creates an output file that is a linear combination of the input files. This linear combination is a weighted average, a normalized weighted average, or an interpolation of the input files. Coordinate variables are not acted upon in any case, they are simply copied from file_1.

There are two conceptually distinct methods of using ncflint. The first method is to specify the weight each input file contributes to the output file. In this method, the value val3 of a variable in the output file file_3 is determined from its values val1 and val2 in the two input files according to \( \text{val3} = wgt1 \times \text{val1} + wgt2 \times \text{val2} \). Here at least wgt1, and, optionally, wgt2, are specified on the command line with the ‘-w’ (or ‘--weight’ or ‘--wgt_var’) switch. If only wgt1 is specified then wgt2 is automatically computed as \( wgt2 = 1 - wgt1 \). Note that weights larger than 1 are allowed. Thus it is possible to specify \( wgt1 = 2 \) and \( wgt2 = -3 \). One can use this functionality to multiply all the values in a given file by a constant.

The second method of using ncflint is to specify the interpolation option with ‘-i’ (or with the ‘--ntp’ or ‘--interpolate’ long options). This is the inverse of the first method in the following sense: When the user specifies the weights directly, ncflint has no work to do besides multiplying the input values by their respective weights and adding together the results to produce the output values. It makes sense to use this when the weights are known \textit{a priori}.

Another class of problems has the arrival value (i.e., val3) of a particular variable var known \textit{a priori}. In this case, the implied weights can always be inferred by examining the values of var in the input files. This results in one equation in two unknowns, wgt1 and wgt2: \( \text{val3} = wgt1 \times \text{val1} + wgt2 \times \text{val2} \). Unique determination of the weights requires imposing the additional constraint of normalization on the weights: \( wgt1 + wgt2 = 1 \). Thus, to use the interpolation option, the user specifies var and val3 with the ‘-i’ option. ncflint then computes wgt1 and wgt2, and uses these weights on all variables to generate the output file. Although var may have any number of dimensions in the input files, it must represent a single, scalar value. Thus any dimensions associated with var must be degenerate, i.e., of size one.

If neither ‘-i’ nor ‘-w’ is specified on the command line, ncflint defaults to weighting each input file equally in the output file. This is equivalent to specifying ‘-w 0.5’ or ‘-w
0.5,0.5'. Attempting to specify both `-i` and `-w` methods in the same command is an error.

`ncflint` does not interpolate variables of type `NC_CHAR` and `NC_STRING`. This behavior is hardcoded.

By default `ncflint` interpolates or multiplies record coordinate variables (e.g., time is often stored as a record coordinate) not other coordinate variables (e.g., latitude and longitude). This is because `ncflint` is often used to time-interpolate between existing files, but is rarely used to spatially interpolate. Sometimes however, users wish to multiply entire files by a constant that does not multiply any coordinate variables. The `--fix_rec_crd` switch was implemented for this purpose in NCO version 4.2.6 (March, 2013). It prevents `ncflint` from multiplying or interpolating any coordinate variables, including record coordinate variables.

Depending on your intuition, `ncflint` may treat missing values unexpectedly. Consider a point where the value in one input file, say `val1`, equals the missing value `mss_val1` and, at the same point, the corresponding value in the other input file `val2` is not missing (i.e., does not equal `mss_val2`). There are three plausible answers, and this creates ambiguity.

Option one is to set `val3 = mss_val1`. The rationale is that `ncflint` is, at heart, an interpolator and interpolation involving a missing value is intrinsically undefined. `ncflint` currently implements this behavior since it is the most conservative and least likely to lead to misinterpretation.

Option two is to output the weighted valid data point, i.e., `val3 = wgt2 × val2`. The rationale for this behavior is that interpolation is really a weighted average of known points, so `ncflint` should weight the valid point.

Option three is to return the unweighted valid point, i.e., `val3 = val2`. This behavior would appeal to those who use `ncflint` to estimate data using the closest available data. When a point is not bracketed by valid data on both sides, it is better to return the known datum than no datum at all.

The current implementation uses the first approach, Option one. If you have strong opinions on this matter, let us know, since we are willing to implement the other approaches as options if there is enough interest.

**EXAMPLES**

Although it has other uses, the interpolation feature was designed to interpolate `file.3` to a time between existing files. Consider input files `85.nc` and `87.nc` containing variables describing the state of a physical system at times `time = 85` and `time = 87`. Assume each file contains its timestamp in the scalar variable `time`. Then, to linearly interpolate to a file `86.nc` which describes the state of the system at time at `time = 86`, we would use

```
ncflint -i time,86 85.nc 87.nc 86.nc
```

Say you have observational data covering January and April 1985 in two files named `85_01.nc` and `85_04.nc`, respectively. Then you can estimate the values for February and March by interpolating the existing data as follows. Combine `85_01.nc` and `85_04.nc` in a 2:1 ratio to make `85_02.nc`:
Multiply 85.nc by 3 and by −2 and add them together to make tst.nc:

```
nclint -w 3,-2 85.nc 85.nc tst.nc
```

This is an example of a null operation, so tst.nc should be identical (within machine precision) to 85.nc.

Multiply all the variables except the coordinate variables in the file emissions.nc by 0.8:

```
nclint --fix_rec_crd -w 0.8,0.0 emissions.nc emissions.nc scaled_emissions.nc
```

The use of `--fix_rec_crd` ensures, e.g., that the time coordinate, if any, is not scaled (i.e., multiplied).

Add 85.nc to 86.nc to obtain 85p86.nc, then subtract 86.nc from 85.nc to obtain 85m86.nc

```
nclint -w 1,1 85.nc 86.nc 85p86.nc
nclint -w 1,-1 85.nc 86.nc 85m86.nc
ncdiff 85.nc 86.nc 85m86.nc
```

Thus nclint can be used to mimic some ncbo operations. However this is not a good idea in practice because nclint does not broadcast (see Section 4.3 [ncbo netCDF Binary Operator], page 174) conforming variables during arithmetic. Thus the final two commands would produce identical results except that nclint would fail if any variables needed to be broadcast.

Rescale the dimensional units of the surface pressure prs_sfc from Pascals to hectopascals (millibars)

```
nclint -C -v prs_sfc -w 0.01,0.0 in.nc in.nc out.nc
ncatted -a units,prs_sfc,o,c,millibar out.nc
```
4.7 ncks netCDF Kitchen Sink

SYNTAX

```bash
    [-D dbg] [-d dim[,min][,[max]][,[stride]]] [-F] [-fix_rec_dmn dim]
    [-G gpe_dsc] [-g grp[,...]] [-grp_xtr_var_xcl] [-H] [-h] [-hdn] [-hdr_pad nbr]
    [-Q] [-q] [-R] [-rad] [-ram_all] [-s format]
    [-u] [-unn] [-V] [-v var[,...]] [-X ...] [-x] [-xml]
    input-file [output-file]]
```

DESCRIPTION

The nickname “kitchen sink” is a catch-all because ncks combines most features of ncdump and nccopy with extra features to extract, hyperslab, multi-slab, sub-set, and translate into one versatile utility. ncks extracts (a subset of the) data from input-file and optionally writes it in netCDF format to output-file, and optionally prints it to screen.

ncks prints netCDF input data in ASCII, CDL, or NcML text formats stdout, like (an extended version of) ncdump. By default ncks prints data in a tabular format intended to be easy to search for the data you want, one datum per screen line, with all dimension subscripts and coordinate values (if any) preceding the datum. Option ‘-s’ (or long options ‘--sng_fmt’ and ‘--string’) permits the user to format data using C-style format strings, while option ‘--cdl’ outputs CDL and option ‘--xml’ outputs NcML. ncks exposes many flexible controls over printed output, including CDL and NcML.


ncks extracts (and optionally creates a new netCDF file comprised of) only selected variables from the input file (similar to the old nceextr specification). Only variables and coordinates may be specifically included or excluded—all global attributes and any attribute associated with an extracted variable are copied to the screen and/or output netCDF file. Options ‘-c’, ‘-C’, ‘-v’, and ‘-x’ (and their long option synonyms) control which variables are extracted.

ncks extracts hyperslabs from the specified variables (ncks implements the original nccut specification). Option ‘-d’ controls the hyperslab specification. Input dimensions that are not associated with any output variable do not appear in the output netCDF. This feature removes superfluous dimensions from netCDF files.

ncks will append variables and attributes from the input-file to output-file if output-file is a pre-existing netCDF file whose relevant dimensions conform to dimension sizes of input-file. The append features of ncks are intended to provide a rudimentary means of...
adding data from one netCDF file to another, conforming, netCDF file. If naming conflicts exist between the two files, data in output-file is usually overwritten by the corresponding data from input-file. Thus, when appending, the user should backup output-file in case valuable data are inadvertently overwritten.

If output-file exists, the user will be queried whether to overwrite, append, or exit the ncks call completely. Choosing overwrite destroys the existing output-file and create an entirely new one from the output of the ncks call. Append has differing effects depending on the uniqueness of the variables and attributes output by ncks: If a variable or attribute extracted from input-file does not have a name conflict with the members of output-file then it will be added to output-file without overwriting any of the existing contents of output-file. In this case the relevant dimensions must agree (conform) between the two files; new dimensions are created in output-file as required. When a name conflict occurs, a global attribute from input-file will overwrite the corresponding global attribute from output-file. If the name conflict occurs for a non-record variable, then the dimensions and type of the variable (and of its coordinate dimensions, if any) must agree (conform) in both files. Then the variable values (and any coordinate dimension values) from input-file will overwrite the corresponding variable values (and coordinate dimension values, if any) in output-file\(^1\).

Since there can only be one record dimension in a file, the record dimension must have the same name (though not necessarily the same size) in both files if a record dimension variable is to be appended. If the record dimensions are of differing sizes, the record dimension of output-file will become the greater of the two record dimension sizes, the record variable from input-file will overwrite any counterpart in output-file and fill values will be written to any gaps left in the rest of the record variables (I think). In all cases variable attributes in output-file are superseded by attributes of the same name from input-file, and left alone if there is no name conflict.

Some users may wish to avoid interactive ncks queries about whether to overwrite existing data. For example, batch scripts will fail if ncks does not receive responses to its queries. Options ‘-O’ and ‘-A’ are available to force overwriting existing files and variables, respectively.

**Options specific to ncks**

The following summarizes features unique to ncks. Features common to many operators are described in Chapter 3 [Shared features], page 25.

\[\text{\texttt{\textasciitilde 5}}\] Print data to screen alphabetically by group, and alphabetically by variable within each group. This ordering here is used by default in CDL-mode printing, and may be selected for traditional mode printing with ‘\texttt{-5}’ (The switch for invocation may change to something more descriptive in the future).

\[\text{\texttt{-a}}\] Do not alphabetize extracted fields. By default, the specified output variables are extracted, printed, and written to disk in alphabetical order. This tends to make long output lists easier to search for particular variables. Specifying \texttt{-a} results in the variables being extracted, printed, and written to disk in the

\[\text{\texttt{\textasciitilde 5}}\] Those familiar with netCDF mechanics might wish to know what is happening here: ncks does not attempt to redefine the variable in output-file to match its definition in input-file. ncks merely copies the values of the variable and its coordinate dimensions, if any, from input-file to output-file.
order in which they were saved in the input file. Thus \texttt{-a} retains the original ordering of the variables. Also \texttt{--abc} and \texttt{--alphabetize}.

\texttt{--fix_rec_dmn}\footnote{As of NCO version 4.4.0 (January, 2014), the argument \texttt{all} may be given to \texttt{--fix_rec_dmn} to convert all record dimensions to fixed dimensions in the output file. Previously, \texttt{--fix_rec_dmn} only allowed one option, the name of a single record dimension to be fixed. Now it is simple to simultaneously fix all record dimensions. This is useful (and nearly mandatory) when flattening netCDF4 files that have multiple record dimensions per group into netCDF3 files (which are limited to at most one record dimension) (see Section 3.13 [Group Path Editing], page 46).}

\texttt{--fix_rec_dmn} Change record dimension \texttt{dim} in the input file into a fixed dimension in the output file. Also \texttt{--no_rec_dmn}. Before NCO version 4.2.5 (January, 2013), the syntax for \texttt{--fix_rec_dmn} did not permit or require the specification of the dimension name \texttt{dim}. This is because the feature only worked on netCDF3 files, which support only one record dimension, so specifying its name was not necessary. netCDF4 files allow an arbitrary number of record dimensions, so the user must specify which record dimension to fix. The decision was made that starting with NCO version 4.2.5 (January, 2013), it is always required to specify the dimension name to fix regardless of the netCDF file type. This keeps the code simple, and is symmetric with the syntax for \texttt{--mk_rec_dmn}, described next.

As of NCO version 4.4.0 (January, 2014), the \texttt{--hdn} or \texttt{--hidden} options print hidden (aka special) attributes. This is equivalent to \texttt{ncdump -s}. Hidden attributes include: \_Format, \_DeflateLevel, \_Shuffle, \_Storage, \_ChunkSizes, \_Endianness, \_Fletcher32, and \_NOFILL. Previously ncks ignored all these attributes in CDL/XML modes. Now it prints these attributes as appropriate. As of NCO version 4.4.6 (September, 2014), \texttt{--hdn} also prints the extended file format (i.e., the format of the file or server supplying the data) as \_SOURCE_FORMAT. Users are referred to the Unidata netCDF Documentation, or the man pages for \texttt{ncgen} or \texttt{ncdump}, for detailed descriptions of the meanings of these attributes.
As of NCO version 4.3.3 (July, 2013), \texttt{ncks} can print extracted data and metadata to screen (i.e., \texttt{stdout}) as valid CDL (network Common data form Description Language). CDL is the human-readable “lingua franca” of netCDF ingested by \texttt{ncgen} and excreted by \texttt{ncdump}. Compare \texttt{ncks} “traditional” with CDL printing:

```
zender@roulee:~$ ncks -v one ~/nco/data/in.nc
one: type NC_FLOAT, 0 dimensions, 1 attribute, chunked? no, compressed? no, packed? no,
one size (RAM) = 1*sizeof(NC_FLOAT) = 1*4 = 4 bytes
one attribute 0: long_name, size = 3 NC_CHAR, value = one

one = 1
```

```
zender@roulee:~$ ncks --cdl -v one ~/nco/data/in.nc
netcdf in {
  variables:
  float one ;
  one:long_name = "one" ;

  data:
    one = 1 ;
}
```

\texttt{ncgen} converts CDL-mode output into a netCDF file:

```
ncks --cdl -v one ~/nco/data/in.nc > ~/in.cdl
ncgen -k netCDF-4 -b -o ~/in.nc ~/in.cdl
ncks --cdl -v one ~/nco/data/in.nc
```

The HDF version of \texttt{ncgen}, often named \texttt{hncgen} or \texttt{ncgen-hdf}, converts netCDF3 CDL into an HDF file:

```
/usr/hdf4/bin/ncgen -b -o ~/in.hdf ~/in.cdl # HDF nngen (local builds)
/usr/bin/hncgen -b -o ~/in.hdf ~/in.cdl # Same as HDF nngen (RPM package)
/usr/bin/ncgen-hdf -b -o ~/in.hdf ~/in.cdl # Same as HDF nngen (Debian package)
hdp dumpsds ~/in.hdf # ncdump/h5dump-equivalent for HDF
```

Note that HDF4 does not support netCDF-style groups, so the above commands fail when the input file contains groups. Only netCDF4 and HDF5 support groups. In our experience the HDF \texttt{ncgen} command, by whatever name installed, is not robust and can fail on valid netCDF3 CDL.

‘--mk_rec_dmn dim’

Change existing dimension \textit{dim} to a record dimension in the output file. This is the most straightforward way of changing a dimension to a/the record dimension, and works fine in most cases. See Section 4.5 [ncecat netCDF Ensemble Concatenator], page 182 and Section 4.8 [ncpdq netCDF Permute Dimensions Quickly], page 202 for other methods of changing variable dimensionality, including the record dimension.
'\texttt{-H}' Turn-on printing to screen or turn-off copying data (not metadata). Also activated using '--print' or '--prn'. By default \texttt{ncks} prints all metadata and data to screen if no netCDF output file is specified. Use '\texttt{-H}' to print data to screen if a netCDF output is specified, or to restrict printing to data (no metadata) when no netCDF output is specified. Also use '\texttt{-H}' to turn-off copying data (not metadata) to an output file. Unless otherwise specified (with \texttt{-s}), each element of the data hyperslab prints on a separate line containing the names, indices, and, values, if any, of all of the variables dimensions. The dimension and variable indices refer to the location of the corresponding data element with respect to the variable as stored on disk (i.e., not the hyperslab).

\begin{verbatim}
\% ncks -C -v three_dmn_var in.nc
lat[0]=-90  lev[0]=100  lon[0]=0  three_dmn_var[0]=0
lat[0]=-90  lev[0]=100  lon[1]=90  three_dmn_var[1]=1
...
\end{verbatim}

Printing the same variable with the '\texttt{-F}' option shows the same variable indexed with Fortran conventions

\begin{verbatim}
\% ncks -F -C -v three_dmn_var in.nc
lon(1)=0  lev(1)=100  lat(1)=90 three_dmn_var(1)=0
lon(2)=90  lev(1)=100  lat(1)=90 three_dmn_var(2)=1
lon(3)=180 lev(1)=100  lat(1)=90 three_dmn_var(3)=2
...
\end{verbatim}

Printing a hyperslab does not affect the variable or dimension indices since these indices are relative to the full variable (as stored in the input file), and the input file has not changed. However, if the hyperslab is saved to an output file and those values are printed, the indices will change:

\begin{verbatim}
\% ncks -H -d lat,90.0 -d lev,1000.0 -v three_dmn_var in.nc out.nc
...
\% ncks -C -v three_dmn_var out.nc
lat[0]=90  lev[0]=1000 lon[0]=0 three_dmn_var[0]=20
lat[0]=90  lev[0]=1000 lon[1]=90 three_dmn_var[1]=21
\end{verbatim}

'\texttt{-M}' Turn-on printing to screen or turn-off copying global and group metadata. This includes file summary information and global and group attributes. Also '--Mtd' and '--Metadata'. By default \texttt{ncks} prints global metadata to screen if no netCDF output file and no variable extraction list is specified (with '\texttt{-v}'). Use '\texttt{-M}' to print global metadata to screen if a netCDF output is specified, or if
a variable extraction list is specified (with ‘-v’). Use ‘-M’ to turn-off copying
of global and group metadata when copying, subsetting, or appending to an
output file.

The various combinations of printing switches can be confusing. In an attempt
to anticipate what most users want to do, ncks uses context-sensitive defaults
for printing. Our goal is to minimize the use of switches required to accomplish
the common operations. We assume that users creating a new file or overwriting
(e.g., with ‘-0’) an existing file usually wish to copy all global and variable-
specific attributes to the new file. In contrast, we assume that users appending
(e.g., with ‘-A’ an explicit variable list from one file to another usually wish to
copy only the variable-specific attributes to the output file. The switches ‘-H’,
‘-M’, and ‘-m’ switches are implemented as toggles which reverse the default
behavior. The most confusing aspect of this is that ‘-M’ inhibits copying global
metadata in overwrite mode and causes copying of global metadata in append
mode.

ncks in.nc # Print VAs and GAs
ncks -v one in.nc # Print VAs not GAs
ncks -M -v one in.nc # Print GAs only
ncks -m -v one in.nc # Print VAs only
ncks -M -m -v one in.nc # Print VAs and GAs
ncks -O in.nc out.nc # Copy VAs and GAs
ncks -O -v one in.nc out.nc # Copy VAs and GAs
ncks -O -M -v one in.nc out.nc # Copy VAs not GAs
ncks -O -m -v one in.nc out.nc # Copy GAs not VAs
ncks -O -M -m -v one in.nc out.nc # Copy only data (no atts)
ncks -A in.nc out.nc # Append VAs and GAs
ncks -A -v one in.nc out.nc # Append VAs not GAs
ncks -A -M -v one in.nc out.nc # Append VAs and GAs
ncks -A -m -v one in.nc out.nc # Append only data (no atts)
ncks -A -M -m -v one in.nc out.nc # Append GAs not VAs

where VAs and GAs denote variable and group/global attributes, respectively.

‘-m’

Turn-on printing to screen or turn-off copying variable metadata. Using ‘-m’
will print variable metadata to screen (similar to ncdump -h). This displays
all metadata pertaining to each variable, one variable at a time. This includes
information on the storage properties of the variable, such as whether it em-
ploys chunking, compression, or packing. Also activated using ‘-mtd’ and
‘--metadata’. The ncks default behavior is to print variable metadata to screen
if no netCDF output file is specified. Use ‘-m’ to print variable metadata to
screen if a netCDF output is specified. Also use ‘-m’ to turn-off copying of
variable metadata to an output file.

‘--no_blank’

Print numeric representation of missing values. As of NCO As of NCO version
4.2.2 (October, 2012), NCO prints missing values as blanks (i.e., the underscore
character ‘_’) by default. To enable the old behavior of printing the numeric
representation of missing values (e.g., 1.0e36), use the `--no_blank` switch. Also activated using `--noblank` or `--no-blank`.

`-P` Print data, metadata, and units to screen. The `-P` switch is a convenience abbreviation for `-C -H -M -m -u`. Also activated using `--print` or `--prn`. This set of switches is useful for exploring file contents.

`-Q` Print quietly, meaning omit dimension names, indices, and coordinate values when printing arrays. Variable (not dimension) indices are printed. Variable names appear flush left in the output:

```
zender@roulee:~$ ncks -Q -v three_dmn_rec_var -C -H ~/nco/data/in.nc
three_dmn_rec_var[0]=1
...
```

This helps locate specific variables in lists with many variables and different dimensions. See also the `-V` option, which omits all names and indices and prints only variable values.

`-q` Quench (turn-off) all printing to screen. This overrides the setting of all print-related switches, equivalent to `-H -M -m` when in single-file printing mode. When invoked with `-R` (see Section 3.8 [Retaining Retrieved Files], page 35), ncks automatically sets `-q`. This allows ncks to retrieve remote files without automatically trying to print them. Also `--quench`.

`--rad` Retain all dimensions. When invoked with `--rad` (Retain All Dimensions), ncks copies each dimension in the input file to the output file, regardless of whether the dimension is utilized by any variables. Normally ncks discards “orphan dimensions”, i.e., dimensions not referenced by any variables. This switch allows users to keep non-referenced dimensions in the workflow. When invoked in printing mode, causes orphaned dimensions to be printed (they are not printed by default). Also `--retain_all_dimensions`, `--orphan_dimensions`, and `--rph_dmn`.

`-s format` String format for text output. Accepts C language escape sequences and `printf()` formats. Also `--string` and `--sng_fmt`.

`--secret` Print summary of ncks hidden features. These hidden or secret features are used mainly by developers. They are not supported for general use and may change at any time. This demonstrates conclusively that I cannot keep a secret. Also `--ssh` and `--scr`.

`-u` Toggle the printing of a variable’s `units` attribute, if any, with its values. Also `--units`.

`-V` Print variable values only. Do not print variable and dimension names, indices, and coordinate values when printing arrays.

```
zender@roulee:~$ ncks -V -v three_dmn_rec_var -C -H ~/nco/data/in.nc
1
...
```
See also the ‘-Q’ option, which prints variable names and indices, but not dimension names, indices, or coordinate values when printing arrays. Using ‘-V’ is the same as specifying ‘-Q --no_nm_prn’.

‘--xml, --ncml’

As of NCO version 4.3.3 (July, 2013), ncks can print extracted metadata to screen (i.e., stdout) as XML in NcML, the netCDF Markup Language. ncks supports for XML more completely than of ‘ncdump -x’. With ncks one can translate entire netCDF3 and netCDF4 files into NcML, including metadata and data, using all NCO’s subsetting and hyperslabbing capabilities. Compare ncks “traditional” with XML printing:

```
zender@roulee:~$ ncks -v one ~/nco/data/in.nc
one: type NC_FLOAT, 0 dimensions, 1 attribute, chunked? no, compressed? no, packed? no
one size (RAM) = 1*sizeof(NC_FLOAT) = 1*4 = 4 bytes
one attribute 0: long_name, size = 3 NC_CHAR, value = one
one = 1
```

```
zender@roulee:~$ ncks --xml -v one ~/nco/data/in.nc
<?xml version="1.0" encoding="UTF-8"?>
<netcdf xmlns="http://www.unidata.ucar.edu/namespaces/netcdf/ncml-2.2" location="/home/zender/in.nc">
<variable name="one" type="float" shape="">
<attribute name="long_name" separator="*" value="one" />
<values separator="*">1.</values>
</variable>
</netcdf>
```

XML-mode prints variable metadata and, as of NCO version 4.3.7 (October, 2013), variable data and, as of NCO version 4.4.0 (January, 2014), hidden attributes. That ncks produces correct NcML translations of CDM files for all supported datatypes is verified by comparison to output from Unidata’s toolsUI Java program. Please let us know how to improve XML/NcML features.

ncks provides additional options to configure NcML output: ‘--xml_no_location’, ‘--xml_spr_chr’, and ‘--xml_spr_nmr’. Every NcML configuration option automatically triggers NcML printing, so that specifying ‘--xml’ in addition to a configuration option is redundant and unnecessary. The ‘--xml_no_location’ switch prevents output of the NcML location element. By default the location element is printed with a value equal to the location of the input dataset, e.g., location="/home/zender/in.nc". The ‘--xml_spr_chr’ and ‘--xml_spr_nmr’ options customize the strings used as NcML separators for attributes and variables of character-type and numeric-type, respectively. Their default separators are * and “ ” (a space):

```
zender@roulee:~$ ncks --xml -d time,0,3 -v two_dmn_rec_var_sng in.nc
...<values separator="*">abc*bcd*cde*def</values>
...
zender@roulee:~$ ncks --xml_spr_chr=’, ’ -v two_dmn_rec_var_sng in.nc
...
<values separator="",">abc, bcd, cde, def, efg, ghi, hij, jkl, klm</values>

zender@roulee:~$ ncks --xml -v one_dmn_rec_var in.nc

<values>1 2 3 4 5 6 7 8 9 10</values>

zender@roulee:~$ ncks --xml_spr_nmr=' ', -v one_dmn_rec_var in.nc

<values separator="">1, 2, 3, 4, 5, 6, 7, 8, 9, 10</values>

Separator elements for strings are a thorny issue. One must be sure that the separator element is not mistaken as a portion of the string. NCO attempts to produce valid NcML and supplies the ‘--xml_spr_chr’ option to work around any difficulties. NCO performs precautionary checks with strstr(val,spr) to identify presence of the separator string (spr) in data (val) and, when it detects a match, automatically switches to a backup separator string (*|*). However limitations of strstr() may lead to false negatives when the separator string occurs in data beyond the first string in multi-dimensional NC_CHAR arrays. Hence, results may be ambiguous to NcML parsers. If problems arise, use ‘--xml_spr_chr’ to specify a multi-character separator that does not appear in the string array and that does not include an NcML formatting characters (e.g., commas, angles, quotes).

### 4.7.2 Filters for ncks

We encourage the use of standard UNIX pipes and filters to narrow the verbose output of ncks into more precise targets. For example, to obtain an uncluttered listing of the variables in a file try

```
ncks -m in.nc | grep -E ': type' | cut -f 1 -d ' ' | sed 's/://' | sort
```

A Bash user could alias the previous filter to the shell command nclist as shown below. More complex examples could involve command line arguments. For example, a user may frequently be interested in obtaining the value of an attribute, e.g., for textual file examination or for passing to another shell command. Say the attribute is purpose, the variable is z, and the file is in.nc. In this example, ncks -m -v z is too verbose so a robust grep and cut filter is desirable, such as

```
ncks -M -m in.nc | grep -E -i '^z attribute [0-9]+: purpose' | cut -f 11- -d ' ' | sort
```

The filters are clearly too complex to remember on-the-fly so the entire procedure could be implemented as a shell command or function called, say, ncattget

```
function ncattget { ncks -M -m ${3} | grep -E -i "^${2} attribute [0-9]+: ${1}" | cut -f 11- -d ' ' | sort ; }
```

The shell ncattget is invoked with three arguments that are, in order, the names of the attribute, variable, and file to examine. Global attributes are indicated by using a variable name of global. This definition yields the following results

```
% ncattget purpose z in.nc
Height stored with a monotonically increasing coordinate
```
Chapter 4: Reference Manual

197

% ncattget Purpose Z in.nc
Height stored with a monotonically increasing coordinate
% ncattget history z in.nc
% ncattget history global in.nc
History global attribute.
Note that case sensitivity has been turned off for the variable and attribute names (and
could be turned on by removing the ‘-i’ switch to grep). Furthermore, extended regular
expressions may be used for both the variable and attribute names. The next two commands
illustrate this by searching for the values of attribute purpose in all variables, and then for
all attributes of the variable z:
% ncattget purpose .+ in.nc
1-D latitude coordinate referred to by geodesic grid variables
1-D longitude coordinate referred to by geodesic grid variables
...
% ncattget .+ Z in.nc
Height
Height stored with a monotonically increasing coordinate
meter
Extended filters are best stored as shell commands if they are used frequently. Shell
commands may be re-used when they are defined in shell configuration files. These files are
usually named .bashrc, .cshrc, and .profile for the Bash, Csh, and Sh shells, respectively.

# NB: Untested on Csh, Ksh, Sh, Zsh! Send us feedback!
# Bash shell (/bin/bash) users place these in .bashrc
# ncattget $att_nm $var_nm $fl_nm : What attributes does variable have?
function ncattget { ncks -M -m ${3} | grep -E -i "^${2} attribute [0-9]+: ${1}" | cut # ncunits $att_val $fl_nm : Which variables have given units?
function ncunits { ncks -M -m ${2} | grep -E -i " attribute [0-9]+: units.+ ${1}" | cut
# ncavg $var_nm $fl_nm : What is mean of variable?
function ncavg { ncwa -y avg -O -C -v ${1} ${2} ~/foo.nc ; ncks -H -C -v ${1} ~/foo.nc
# ncavg $var_nm $fl_nm : What is mean of variable?
function ncavg { ncap2 -O -C -v -s "foo=${1}.avg();print(foo)" ${2} ~/foo.nc | cut -f 3
# ncdmnsz $dmn_nm $fl_nm : What is dimension size?
function ncdmnsz { ncks -m -M ${2} | grep -E -i ": ${1}, size =" | cut -f 7 -d ’ ’ | un
# nclist $fl_nm : What variables are in file?
function nclist { ncks -m ${1} | grep -E ’: type’ | cut -f 1 -d ’ ’ | sed ’s/://’ | sor
# ncmax $var_nm $fl_nm : What is maximum of variable?
function ncmax { ncwa -y max -O -C -v ${1} ${2} ~/foo.nc ; ncks -H -C -v ${1} ~/foo.nc
# ncmax $var_nm $fl_nm : What is maximum of variable?
function ncmax { ncap2 -O -C -v -s "foo=${1}.max();print(foo)" ${2} ~/foo.nc | cut -f 3
# ncmdn $var_nm $fl_nm : What is median of variable?
function ncmdn { ncap2 -O -C -v -s "foo=gsl_stats_median_from_sorted_data(${1}.sort());
# ncrng $var_nm $fl_nm : What is range of variable?
function ncrng { ncap2 -O -C -v -s "foo_min=${1}.min();foo_max=${1}.max();print(foo_min
# ncmode $var_nm $fl_nm : What is mode of variable?


function ncmode { ncap2 -O -C -v -s "foo=gsl_stats_median_from_sorted_data(${1}.sort());print(foo)" ${2} ~/foo.nc | cut -f 3- -d ' ' ; }
# ncrecsz $fl_nm : What is record dimension size?
function ncrecsz { ncks -M ${1} | grep -E -i "^Record dimension:" | cut -f 8- -d ' ' ; }
# Csh shell (/bin/csh) users place these in .cshrc
ncattget() { ncks -M -m ${3} | grep -E -i "^${2} attribute \[[0-9]+\]: ${1}" | cut -f 11- -d ' ' | sort ; }
ncdmnsz() { ncks -m -M ${2} | grep -E -i "^: \${1}, size =" | cut -f 7 -d ' ' | uniq ; }
nclist() { ncks -m ${1} | grep -E -i "^: type' | cut -f 1 -d ' ' | sed 's/://' | sort ; }
ncrecsz() { ncks -M ${1} | grep -E -i "^Record dimension:" | cut -f 8- -d ' ' ; }
# Sh shell (/bin/sh) users place these in .profile
ncattget() { ncks -M -m ${3} | grep -E -i "^${2} attribute \[[0-9]+\]: ${1}" | cut -f 11- -d ' ' | sort ; }
ncdmnsz() { ncks -m -M ${2} | grep -E -i "^: \${1}, size =" | cut -f 7 -d ' ' | uniq ; }
nclist() { ncks -m ${1} | grep -E -i "^: type' | cut -f 1 -d ' ' | sed 's/://' | sort ; }
ncrecsz() { ncks -M ${1} | grep -E -i "^Record dimension:" | cut -f 8- -d ' ' ; }

EXAMPLES

View all data in netCDF in.nc, printed with Fortran indexing conventions:

ncks -F in.nc

Copy the netCDF file in.nc to file out.nc.

ncks in.nc out.nc

Now the file out.nc contains all the data from in.nc. There are, however, two differences between in.nc and out.nc. First, the history global attribute (see Section 3.36 [History Attribute], page 108) will contain the command used to create out.nc. Second, the variables in out.nc will be defined in alphabetical order. Of course the internal storage of variable in a netCDF file should be transparent to the user, but there are cases when alphabetizing a file is useful (see description of -a switch).

Copy all global attributes (and no variables) from in.nc to out.nc:

ncks -A -x ~/nco/data/in.nc ~/out.nc

The `-x` switch tells NCO to use the complement of the extraction list (see Section 3.11 [Subsetting Files], page 40). Since no extraction list is explicitly specified (with `-v`), the default is to extract all variables. The complement of all variables is no variables. Without any variables to extract, the append (`-A`) command (see Section 2.4 [Appending Variables], page 17) has only to extract and copy (i.e., append) global attributes to the output file.

Copy/append metadata (not data) from variables in one file to variables in a second file. When copying/subsetting/appending files (as opposed to printing them), the copying of data, variable metadata, and global/group metadata are now turned OFF by `-H`, `-m`, and `-M`, respectively. This is the opposite sense in which these switches work when printing a file. One can use these switches to easily replace data or metadata in one file with data or metadata from another:

# Extract naked (data-only) copies of two variables
ncks -h -M -m -O -C -v one,three_dmn_rec_var ~/nco/data/in.nc ~/out.nc
# Change values to be sure original values are not copied in following step
ncap2 -O -v -s 'one*=2;three_dmn_rec_var*=0' ~/nco/data/in.nc ~/in2.nc
# Append in2.nc metadata (not data!) to out.nc
ncks -A -C -H -v one,three_dmn_rec_var ~/in2.nc ~/out.nc

Variables in out.nc now contain data (not metadata) from in.nc and metadata (not data) from in2.nc.

Print variable three_dmn_var from file in.nc with default notations. Next print three_dmn_var as an un-annotated text column. Then print three_dmn_var signed with very high precision. Finally, print three_dmn_var as a comma-separated list.

\% ncks -C -v three_dmn_var in.nc
lat[0]=-90 lev[0]=100 lon[0]=0 three_dmn_var[0]=0
lat[0]=-90 lev[0]=100 lon[1]=90 three_dmn_var[1]=1
...
\% ncks -s '%f\n' -C -v three_dmn_var in.nc
0.000000
1.000000
...
23.000000
\% ncks -s '%+16.10f\n' -C -v three_dmn_var in.nc
+0.0000000000
+1.0000000000
...
+23.0000000000
\% ncks -s '%f, ' -C -v three_dmn_var in.nc
0.000000, 1.000000, ..., 23.000000,

Programmers will recognize these as the venerable C language printf() formatting strings. The second and third options are useful when pasting data into text files like reports or papers. See Section 4.2 [ncatted netCDF Attribute Editor], page 168, for more details on string formatting and special characters.

As of NCO version 4.2.2 (October, 2012), NCO prints missing values as blanks (i.e., the underscore character '_') by default:

\% ncks -C -H -v mss_val in.nc
lon[0]=0 mss_val[0]=73
\% ncks -s '%+5.1f, ' -H -v mss_val in.nc
+73.0, _, +73.0, _,

To print the numeric value of the missing value instead of a blank, use the '--no_blank' option.

ncks prints in a verbose fashion by default and supplies a number of switches to pare-down (or even spruce-up) the output. The interplay of the '--Q', '--V', and (otherwise undocumented) '--no_nm_prn' switches yields most desired verbosities:

\% ncks -v three_dmn_rec_var -C -H ~/nco/data/in.nc
time[0]=1 lat[0]=-90 lon[0]=0 three_dmn_rec_var[0]=1
% ncks -Q -v three_dmn_rec_var -C -H ~/nco/data/in.nc
three_dmn_rec_var[0]=1
% ncks -V -v three_dmn_rec_var -C -H ~/nco/data/in.nc
1
% ncks --no_nm_prn -v three_dmn_rec_var -C -H ~/nco/data/in.nc
1
% ncks --no_nm_prn -v three_dmn_rec_var -C -H ~/nco/data/in.nc
1 -90 0 1

One dimensional arrays of characters stored as netCDF variables are automatically printed as strings, whether or not they are NUL-terminated, e.g.,

ncks -v fl_nm in.nc

The %%c formatting code is useful for printing multidimensional arrays of characters representing fixed length strings

ncks -s '%%c' -v fl_nm_arr in.nc

Using the %%s format code on strings which are not NUL-terminated (and thus not technically strings) is likely to result in a core dump.

Create netCDF out.nc containing all variables, and any associated coordinates, except variable time, from netCDF in.nc:

ncks -x -v time in.nc out.nc

As a special case of this, consider how to remove a variable such as time_bounds that is identified in a CF Convention (see Section 3.38 [CF Conventions], page 109) compliant ancillary_variables, bounds, or coordinates attribute. NCO subsetting assumes the user wants all ancillary variables, axes, bounds and coordinates associated with all extracted variables (see Section 3.12 [Subsetting Coordinate Variables], page 45). Hence to exclude a ancillary_variables, bounds, or coordinates variable while retaining the “parent” variable (here time), one must use the ‘-C’ switch:

ncks -C -x -v time_bounds in.nc out.nc

The ‘-C’ switch tells the operator NOT to necessarily include all the CF ancillary variables, axes, bounds, and coordinates. Hence the output file will contain time and not time_bounds.

Extract variables time and pressure from netCDF in.nc. If out.nc does not exist it will be created. Otherwise the you will be prompted whether to append to or to overwrite out.nc:

ncks -v time,pressure in.nc out.nc
ncks -C -v time,pressure in.nc out.nc

The first version of the command creates an out.nc which contains time, pressure, and any coordinate variables associated with pressure. The out.nc from the second version is guaranteed to contain only two variables time and pressure.
Create netCDF out.nc containing all variables from file in.nc. Restrict the dimensions of these variables to a hyperslab. Print (with -H) the hyperslabs to the screen for good measure. The specified hyperslab is: the fifth value in dimension time; the half-open range lat > 0. in coordinate lat; the half-open range lon < 330. in coordinate lon; the closed interval 0.3 < band < 0.5 in coordinate band; and cross-section closest to 1000. in coordinate lev. Note that limits applied to coordinate values are specified with a decimal point, and limits applied to dimension indices do not have a decimal point. See Section 3.15 [Hyperslabs], page 55.

ncks -H -d time,5 -d lat,,0.0 -d lon,330.0, -d band,0.3,0.5
-d lev,1000.0 in.nc out.nc

Assume the domain of the monotonically increasing longitude coordinate lon is 0 < lon < 360. Here, lon is an example of a wrapped coordinate. ncks will extract a hyperslab which crosses the Greenwich meridian simply by specifying the westernmost longitude as min and the easternmost longitude as max, as follows:

ncks -d lon,260.0,45.0 in.nc out.nc

For more details See Section 3.20 [Wrapped Coordinates], page 63.
4.8 ncpdq netCDF Permute Dimensions Quickly

SYNTAX

```
[-D dbg] [-d dim,[min],[max]],[,stridel]
[-F] [-G gpe_dsc] [-g grp[,...]] [-h] [--hdf] [--hdr_pad nbr]
[-R] [-r] [-z ram_all] [-t thr_nbr] [-U] [--unn] [-v var[,...]] [-X ...] [-x]
input-file [output-file]
```

DESCRIPTION

ncpdq performs one (not both) of two distinct functions: packing or dimension permutation. ncpdq is optimized to perform these actions in a parallel fashion with a minimum of time and memory. The *pdq* may stand for “Permute Dimensions Quickly”, “Pack Data Quietly”, “Pillory Dan Quayle”, or other silly uses.

Packing and Unpacking Functions

The ncpdq packing (and unpacking) algorithms are described in Section 4.1.11 [Methods and functions], page 131, and are also implemented in *ncap2*. ncpdq extends the functionality of these algorithms by providing high level control of the packing policy so that users can consistently pack (and unpack) entire files with one command. The user specifies the desired packing policy with the ‘-P’ switch (or its long option equivalents, ‘--pack_policy’ and ‘--pack_policy’) and its pck_plc argument. Four packing policies are currently implemented:

**Packing (and Re-Packing) Variables [default]**
  - Definition: Pack unpacked variables, re-pack packed variables
  - Alternate invocation: ncpack
  - pck_plc key values: ‘all_new’, ‘pck_all_new_att’

**Packing (and not Re-Packing) Variables**
  - Definition: Pack unpacked variables, copy packed variables
  - Alternate invocation: none
  - pck_plc key values: ‘all_xst’, ‘pck_all_xst_att’

**Re-Packing Variables**
  - Definition: Re-pack packed variables, copy unpacked variables
  - Alternate invocation: none
  - pck_plc key values: ‘xst_new’, ‘pck_xst_new_att’

**Unpacking**
  - Definition: Unpack packed variables, copy unpacked variables
  - Alternate invocation: ncunpack
Equivalent key values are fully interchangeable. Multiple equivalent options are provided to satisfy disparate needs and tastes of NCO users working with scripts and from the command line.

Regardless of the packing policy selected, `ncpdq` no longer (as of NCO version 4.0.4 in October, 2010) packs coordinate variables, or the special variables, weights, and other grid properties described in Section 3.38 [CF Conventions], page 109. Prior `ncpdq` versions treated coordinate variables and grid properties no differently from other variables. However, coordinate variables are one-dimensional, so packing saves little space on large files, and the resulting files are difficult for humans to read. `ncpdq` will, of course, *unpack* coordinate variables and weights, for example, in case some other, non-NCO software packed them in the first place.

Concurrently, Gaussian and area weights and other grid properties are often used to derive fields in re-inflated (unpacked) files, so packing such grid properties causes a considerable loss of precision in downstream data processing. If users express strong wishes to pack grid properties, we will implement new packing policies. An immediate workaround for those needing to pack grid properties now, is to use the `ncap2` packing functions or to rename the grid properties prior to calling `ncpdq`. We welcome your feedback.

To reduce required memorization of these complex policy switches, `ncpdq` may also be invoked via a synonym or with switches that imply a particular policy. `ncpack` is a synonym for `ncpdq` and behaves the same in all respects. Both `ncpdq` and `ncpack` assume a default packing policy request of `all_new`. Hence `ncpack` may be invoked without any `-P` switch, unlike `ncpdq`. Similarly, `ncunpack` is a synonym for `ncpdq` except that `ncpack` implicitly assumes a request to unpack, i.e., `-P pck_upk`. Finally, the `ncpdq` `-U` switch (or its long option equivalents, `--upk` and `--unpack`) requires no argument. It simply requests unpacking.

Given the menagerie of synonyms, equivalent options, and implied options, a short list of some equivalent commands is appropriate. The following commands are equivalent for packing: `ncpdq -P all_new`, `ncpdq --pck_plc=all_new`, and `ncpack`. The following commands are equivalent for unpacking: `ncpdq -P upk`, `ncpdq -U`, `ncpdq --pck_plc=unpack`, and `ncunpack`. Equivalent commands for other packing policies, e.g., `all_xst`, follow by analogy. Note that `ncpdq` synonyms are subject to the same constraints and recommendations discussed in the section on `ncbo` synonyms (see Section 4.3 [ncbo netCDF Binary Operator], page 174). That is, symbolic links must exist from the synonym to `ncpdq`, or else the user must define an alias.

The `ncpdq` packing algorithms must know to which type particular types of input variables are to be packed. The correspondence between the input variable type and the output, packed type, is called the packing map. The user specifies the desired packing map with the `-M` switch (or its long option equivalents, `--pck_map` and `--map`) and its `pck_map` argument. Five packing maps are currently implemented:
Pack Floating Precisions to NC_SHORT [default]
Definition: Pack floating precision types to NC_SHORT
Map: Pack [NC_DOUBLE,NC_FLOAT] to NC_SHORT
Types copied instead of packed: [NC_INT64,NC_UINT64,NC_INT,NC_UINT,NC_SHORT,NC_USHORT,NC_CHAR,NC_BYTE,NC_UBYTE]
pck_map key values: ‘flt_sht’, ‘pck_map_flt_sht’

Pack Floating Precisions to NC_BYTE
Definition: Pack floating precision types to NC_BYTE
Map: Pack [NC_DOUBLE,NC_FLOAT] to NC_BYTE
Types copied instead of packed: [NC_INT64,NC_UINT64,NC_INT,NC_UINT,NC_SHORT,NC_USHORT,NC_CHAR,NC_BYTE,NC_UBYTE]
pck_map key values: ‘flt_byt’, ‘pck_map_flt_byt’

Pack Higher Precisions to NC_SHORT
Definition: Pack higher precision types to NC_SHORT
Map: Pack [NC_DOUBLE,NC_FLOAT,NC_INT64,NC_UINT64,NC_INT,NC_UINT] to NC_SHORT
Types copied instead of packed: [NC_SHORT,NC_USHORT,NC_CHAR,NC_BYTE,NC_UBYTE]
pck_map key values: ‘hgh_sht’, ‘pck_map_hgh_sht’

Pack Higher Precisions to NC_BYTE
Definition: Pack higher precision types to NC_BYTE
Map: Pack [NC_DOUBLE,NC_FLOAT,NC_INT64,NC_UINT64,NC_INT,NC_UINT,NC_SHORT,NC_USHORT] to NC_BYTE
Types copied instead of packed: [NC_CHAR,NC_BYTE,NC_UBYTE]
pck_map key values: ‘hgh_byt’, ‘pck_map_hgh_byt’

Pack to Next Lesser Precision
Definition: Pack each type to type of next lesser size
Map: Pack [NC_DOUBLE,NC_INT64,NC_UINT64], to NC_INT. Pack [NC_FLOAT,NC_INT,NC_UINT] to NC_SHORT. Pack [NC_SHORT,NC_USHORT] to NC_BYTE.
Types copied instead of packed: [NC_CHAR,NC_BYTE,NC_UBYTE]
pck_map key values: ‘nxt_lsr’, ‘pck_map_nxt_lsr’

The default ‘all_new’ packing policy with the default ‘flt_sht’ packing map reduces the typical NC_FLOAT-dominated file size by about 50%. ‘flt_byt’ packing reduces an NC_DOUBLE-dominated file by about 87%.

The netCDF packing algorithm (see Section 4.1.11 [Methods and functions], page 131) is lossy—once packed, the exact original data cannot be recovered without a full backup. Hence users should be aware of some packing caveats: First, the interaction of packing and data equal to the _FillValue is complex. Test the _FillValue behavior by performing a pack/unpack cycle to ensure data that are missing stay missing and data that are not
missing do not join the Air National Guard and go missing. This may lead you to elect a new _FillValue. Second, `ncpdq` actually allows packing into `NC_CHAR` (with, e.g., ‘flt_chr’). However, the intrinsic conversion of `signed char` to higher precision types is tricky for values equal to zero, i.e., for `NUL`. Hence packing to `NC_CHAR` is not documented or advertised. Pack into `NC_BYTE` (with, e.g., ‘flt_byt’) instead.

**Dimension Permutation**

`ncpdq` re-shapes variables in `input-file` by re-ordering and/or reversing dimensions specified in the dimension list. The dimension list is a whitespace-free, comma separated list of dimension names, optionally prefixed by negative signs, that follows the `'-a'` (or long options `--arrange`, `--permute`, `--re-order`, or `--rdr`) switch. To re-order variables by a subset of their dimensions, specify these dimensions in a comma-separated list following `-a`, e.g., `-a lon,lat'. To reverse a dimension, prefix its name with a negative sign in the dimension list, e.g., `-a -lat'. Re-ordering and reversal may be performed simultaneously, e.g., `-a lon,-lat,time,-lev'.

Users may specify any permutation of dimensions, including permutations which change the record dimension identity. The record dimension is re-ordered like any other dimension. This unique `ncpdq` capability makes it possible to concatenate files along any dimension. See Section 2.6.1 [Concatenation], page 18 for a detailed example. The record dimension is always the most slowly varying dimension in a record variable (see Section 3.14 [C and Fortran Index Conventions], page 54). The specified re-ordering fails if it requires creating more than one record dimension amongst all the output variables.

Two special cases of dimension re-ordering and reversal deserve special mention. First, it may be desirable to completely reverse the storage order of a variable. To do this, include all the variable’s dimensions in the dimension re-order list in their original order, and prefix each dimension name with the negative sign. Second, it may useful to transpose a variable’s storage order, e.g., from C to Fortran data storage order (see Section 3.14 [C and Fortran Index Conventions], page 54). To do this, include all the variable’s dimensions in the dimension re-order list in reversed order. Explicit examples of these two techniques appear below.

NB: fmx ncpdq documentation will evolve through Fall 2004. I will upload updates to documentation linked to by the NCO homepage. ncpdq is a powerful operator, and I am unfamiliar with the terminology needed to describe what ncpdq does. Sequences, sets, sheesh! I just know that it does “The right thing” according to my gut feelings. Now do you feel more comfortable using it?

Let \(D(x)\) represent the dimensionality of the variable \(x\). Dimensionality describes the order and sizes of dimensions. If \(x\) has rank \(N\), then we may write \(D(x)\) as the \(N\)-element vector

\[
D(x) = [D_1, D_2, D_3, \ldots, D_{n-1}, D_n, D_{n+1}, \ldots, D_{N-2}, D_{N-1}, D_N]
\]

where \(D_n\) is the size of the \(n\)'th dimension.

The dimension re-order list specified with `'-a'` is the \(R\)-element vector

\[
R = [R_1, R_2, R_3, \ldots, R_{r-1}, R_r, R_{r+1}, \ldots, R_{R-2}, R_{R-1}, R_R]
\]

\footnote{This limitation, imposed by the netCDF storage layer, may be relaxed in the future with netCDF4.}
There need be no relation between \( N \) and \( R \). Let the \( S \)-element vector \( S \) be the intersection (i.e., the ordered set of unique shared dimensions) of \( D \) and \( R \). Then

\[
S = R \cap D \\
= [S_1, S_2, S_3, \ldots, S_{s-1}, S_s, S_{s+1}, \ldots, S_{S-2}, S_{S-1}, S_S]
\]

\( S \) is empty if \( R \not\in D \).

Re-ordering (or re-shaping) a variable means mapping the input state with dimensionality \( D(x) \) to the output state with dimensionality \( D'(x') \). In practice, mapping occurs in three logically distinct steps. First, we translate the user input to a one-to-one mapping \( M \) between input and output dimensions, \( D \mapsto D' \). This tentative map is final unless external constraints (typically netCDF restrictions) impose themselves. Second, we check and, if necessary, refine the tentative mapping so that the re-shaped variables will co-exist in the same file without violating netCDF-imposed storage restrictions. This refined map specifies the final (output) dimensionality. Third, we translate the output dimensionality into one-dimensional memory offsets for each datum according to the C language convention for multi-dimensional array storage. Dimension reversal changes the ordering of data, though not the rank or dimensionality, and so is part of the third step.

Dimensions \( R \) disjoint from \( D \) play no role in re-ordering. The first step taken to re-order a variable is to determine \( S \). \( R \) is constant for all variables, whereas \( D \), and hence \( S \), is variable-specific. \( S \) is empty if \( R \not\in D \). This may be the case for some extracted variables. The user may explicitly specify the one-to-one mapping of input to output dimension order by supplying (with ‘-a’) a re-order list \( R \) such that \( S = N \). In this case \( D'_n = S_n \). The degenerate case occurs when \( D = S \). This produces the identity mapping \( D'_n = D_n \).

The mapping of input to output dimension order is more complex when \( S \neq N \). In this case \( D'_n = D_n \) for the \( N - S \) dimensions \( D'_n \not\in S \). For the \( S \) dimensions \( D'_n \in S \), \( D'_n = S_s \).

**EXAMPLES**

Pack and unpack all variables in file in.nc and store the results in out.nc:

```
ncpdq in.nc out.nc # Same as ncpack in.nc out.nc
ncpdq -P all_new -M flt_sht in.nc out.nc # Defaults
ncpdq -P all_xst in.nc out.nc
ncpdq -P upk in.nc out.nc # Same as ncunpack in.nc out.nc
ncpdq -U in.nc out.nc # Same as ncunpack in.nc out.nc
```

The first two commands pack any unpacked variable in the input file. They also unpack and then re-pack every packed variable. The third command only packs unpacked variables in the input file. If a variable is already packed, the third command copies it unchanged to the output file. The fourth and fifth commands unpack any packed variables. If a variable is not packed, the third command copies it unchanged.

The previous examples all utilized the default packing map. Suppose you wish to archive all data that are currently unpacked into a form which only preserves 256 distinct values. Then you could specify the packing map \( pck\_map \) as ‘hgh\_byt’ and the packing policy \( pck\_plc \) as ‘all\_xst’:

```
ncpdq -P all_xst -M hgh_byt in.nc out.nc
```
Many different packing maps may be used to construct a given file by performing the packing on subsets of variables (e.g., with ‘-v’) and using the append feature with ‘-A’ (see Section 2.4 [Appending Variables], page 17).

Users may wish to unpack data packed with the HDF convention, and then re-pack it with the netCDF convention so that all their datasets use the same packing convention prior to intercomparison.

# One-step procedure: For NCO 4.4.0+, netCDF 4.3.1+
# 1. Convert, unpack, and repack HDF file into netCDF file
ncpdq --hdf_upk -P xst_new modis.hdf modis.nc # HDF4 files
ncpdq --hdf_upk -P xst_new modis.h5 modis.nc # HDF5 files

# One-step procedure: For NCO 4.3.7--4.3.9
# 1. Convert, unpack, and repack HDF file into netCDF file
ncpdq --hdf4 --hdf_upk -P xst_new modis.hdf modis.nc # HDF4
ncpdq --hdf_upk -P xst_new modis.h5 modis.nc # HDF5

# Two-step procedure: For NCO 4.3.6 and earlier
# 1. Convert HDF file to netCDF file
ncl_convert2nc modis.hdf
# 2. Unpack using HDF convention and repack using netCDF convention
ncpdq --hdf_upk -P xst_new modis.nc modis.nc

NCO now\textsuperscript{2} automatically detects HDF4 files. In this case it produces an output file modis.nc which preserves the HDF packing used in the input file. The ncpdq command first unpacks all packed variables using the HDF unpacking algorithm (as specified by ‘--hdf_upk’), and then repacks those same variables using the netCDF algorithm (because that is the only algorithm NCO packs with). As described above the ‘--P xst_new’ packing policy only repacks variables that are already packed. Not-packed variables are copied directly without loss of precision\textsuperscript{3}.

Re-order file in.nc so that the dimension lon always precedes the dimension lat and store the results in out.nc:

```
ncpdq -a lon,lat in.nc out.nc
ncpdq -v three_dmn_var -a lon,lat in.nc out.nc
```

The first command re-orders every variable in the input file. The second command extracts and re-orders only the variable three_dmn_var.

\textsuperscript{2} Prior to NCO 4.4.0 and netCDF 4.3.1 (January, 2014), NCO requires the ‘--hdf4’ switch to correctly read HDF4 input files. For example, ‘ncpdq --hdf4 --hdf_upk -P xst_new modis.hdf modis.nc’. That switch is now obsolete, though harmless for backwards compatibility. Prior to version 4.3.7 (October, 2013), NCO lacked the software necessary to workaround netCDF library flaws handling HDF4 files, and thus NCO failed to convert HDF4 files to netCDF files. In those cases, use the ncl_convert2nc command distributed with NCL to convert HDF4 files to netCDF.

\textsuperscript{3} ncpdq does not support packing data using the HDF convention. Although it is now straightforward to support this, we think it might sow more confusion than it reaps. Let us know if you disagree and would like NCO to support packing data with HDF algorithm.
Suppose the dimension $\text{lat}$ represents latitude and monotonically increases from south to north. Reversing the $\text{lat}$ dimension means re-ordering the data so that latitude values decrease monotonically from north to south. Accomplish this with

```bash
% ncpdq -a -lat in.nc out.nc
% ncks -C -v lat in.nc
lat[0]=-90
lat[1]=90
% ncks -C -v lat out.nc
lat[0]=90
lat[1]=-90
```

This operation reversed the latitude dimension of all variables. Whitespace immediately preceding the negative sign that specifies dimension reversal may be dangerous. Quotes and long options can help protect negative signs that should indicate dimension reversal from being interpreted by the shell as dashes that indicate new command line switches.

```bash
ncpdq -a -lat in.nc out.nc # Dangerous? Whitespace before "-lat"
ncpdq -a '-lat' in.nc out.nc # OK. Quotes protect "-" in "-lat"
ncpdq -a lon,-lat in.nc out.nc # OK. No whitespace before "-"
ncpdq --rdr=-lat in.nc out.nc # Preferred. Uses "=" not whitespace
```

To create the mathematical transpose of a variable, place all its dimensions in the dimension re-order list in reversed order. This example creates the transpose of $\text{three_dmn_var}$:

```bash
% ncpdq -a lon,lev,lat -v three_dmn_var in.nc out.nc
% ncks -C -v three_dmn_var in.nc
lat[0]=-90 lev[0]=100 lon[0]=0 three_dmn_var[0]=0
lat[0]=-90 lev[0]=100 lon[1]=90 three_dmn_var[1]=1
% ncks -C -v three_dmn_var out.nc
lon[0]=0 lev[0]=100 lat[0]=-90 three_dmn_var[0]=0
lon[0]=0 lev[0]=100 lat[1]=90 three_dmn_var[1]=12
lon[0]=0 lev[1]=500 lat[0]=-90 three_dmn_var[2]=4
```

To completely reverse the storage order of a variable, include all its dimensions in the re-order list, each prefixed by a negative sign. This example reverses the storage order of $\text{three_dmn_var}$:

```bash
% ncpdq -a -lat,-lev,-lon -v three_dmn_var in.nc out.nc
% ncks -C -v three_dmn_var in.nc
lat[0]=-90 lev[0]=100 lon[0]=0 three_dmn_var[0]=0
```
Chapter 4: Reference Manual

Creating a record dimension named, e.g., time, in a file which has no existing record dimension is simple with ncecat:

\[
\text{ncecat -O -u time in.nc out.nc} # Create degenerate record dimension named "time"
\]

Now consider a file with all dimensions, including time, fixed (non-record). Suppose the user wishes to convert time from a fixed dimension to a record dimension. This may be useful, for example, when the user wishes to append additional time slices to the data. As of NCO version 4.0.1 (April, 2010) the preferred method for doing this is with ncks:

\[
\text{ncks -O --mk_rec_dmn time in.nc out.nc} # Change "time" to record dimension
\]

Prior to 4.0.1, the procedure to change an existing fixed dimension into a record dimension required three separate commands, ncecat followed by ncpdq, and then ncwa. The recommended method is now to use \text{'ncks --fix_rec_dmn'}, yet it is still instructive to present the original procedure, as it shows how multiple operators can achieve the same ends by different means:

\[
\text{ncecat -O in.nc out.nc} # Add degenerate record dimension named "record"
\text{ncpdq -O -a time,record out.nc out.nc} # Switch "record" and "time"
\text{ncwa -O -a record out.nc out.nc} # Remove (degenerate) "record"
\]

The first step creates a degenerate (size equals one) record dimension named (by default) record. The second step swaps the ordering of the dimensions named time and record. Since time now occupies the position of the first (least rapidly varying) dimension, it becomes the record dimension. The dimension named record is no longer a record dimension. The third step averages over this degenerate record dimension. Averaging over a degenerate dimension does not alter the data. The ordering of other dimensions in the file (lat, lon, etc.) is immaterial to this procedure. See Section 4.5 [ncecat netCDF Ensemble Concatenator], page 182 and Section 4.7 [ncks netCDF Kitchen Sink], page 188 for other methods of changing variable dimensionality, including the record dimension.
4.9 ncra netCDF Record Averager

SYNTAX

```
    [-cnck_dmn nm,sz] [-cnck_map map] [-cnck_min sz] [-cnck_plc plc] [-cnck_scl sz]
    [-D dbg] [-d dim,[min],[max],[stride],[subcycle]]) [-F]
    [-G gpe_dsc] [-g grp[,...]] [-h] [-hdf] [-hds_pad nhr]
    [-o] [-o output-
    [-t thr_nbr] [-unn] [-v var[,...]] [-X ...] [-x] [-y op_typ]
    [input-files] [output-file]
```

DESCRIPTION

ncra averages record variables across an arbitrary number of input-files. The record
dimension is, by default, retained as a degenerate (size 1) dimension in the output variables.
See Section 2.6 [Statistics vs. Concatenation], page 18, for a description of the distinctions
between the various statistics tools and concatenators. As a multi-file operator, ncra will
read the list of input-files from stdin if they are not specified as positional arguments on
the command line (see Section 2.7 [Large Numbers of Files], page 19).

Input files may vary in size, but each must have a record dimension. The record coordinate,
if any, should be monotonic (or else non-fatal warnings may be generated). Hyperslabs
of the record dimension which include more than one file work correctly. ncra supports the
stride argument to the ‘-d’ hyperslab option (see Section 3.15 [Hyperslabs], page 55) for
the record dimension only, stride is not supported for non-record dimensions.

ncra weights each record (e.g., time slice) in the input-files equally. ncra does not attempt
to see if, say, the time coordinate is irregularly spaced and thus would require a
weighted average in order to be a true time average. ncra always averages coordinate variables regardless of the arithmetic operation type performed on the non-coordinate variables.
(see Section 3.33 [Operation Types], page 94).

EXAMPLES

Average files 85.nc, 86.nc, ... 89.nc along the record dimension, and store the results
in 8589.nc:

```
ncra 85.nc 86.nc 87.nc 88.nc 89.nc 8589.nc
ncra 8[56789].nc 8589.nc
ncra -n 5,2,1 85.nc 8589.nc
```

These three methods produce identical answers. See Section 3.5 [Specifying Input Files],
page 28, for an explanation of the distinctions between these methods.

Assume the files 85.nc, 86.nc, ... 89.nc each contain a record coordinate time of
length 12 defined such that the third record in 86.nc contains data from March 1986, etc.
NCO knows how to hyperslab the record dimension across files. Thus, to average data from
December, 1985 through February, 1986:

```
ncra -d time,11,13 85.nc 86.nc 87.nc 8512_8602.nc
```
ncra -F -d time,12,14 85.nc 86.nc 87.nc 8512_8602.nc

The file 87.nc is superfluous, but does not cause an error. The ‘-F’ turns on the Fortran (1-based) indexing convention. The following uses the stride option to average all the March temperature data from multiple input files into a single output file:

ncra -F -d time,3,,12 -v temperature 85.nc 86.nc 87.nc 858687_03.nc

See Section 3.16 [Stride], page 57, for a description of the stride argument.

Assume the time coordinate is incrementally numbered such that January, 1985 = 1 and December, 1989 = 60. Assuming ‘??’ only expands to the five desired files, the following averages June, 1985–June, 1989:

ncra -d time,6.,54. ?? nc 8506_8906.nc
4.10 ncrcat netCDF Record Concatenator

SYNTAX

```bash
[--ckn_dmn nm,sz] [--ckn_map map] [--ckn_min sz] [--ckn_plc plc] [--ckn_scl sz]
[-D dbg] [-d dim,[min],[max],[[stride],[[subcycle]]]] [-F]
[-G gpe_dsc] [-g grp[,...]] [-h] [-hdr_pad nbr]
[-L df1_lvl] [-l path] [-md5_digest] [-n loop] [-no_tmp_fl]
[-t thr_nbr] [-unn] [-v var[,...]] [-X ...] [-x]
[input-files] [output-file]
```

DESCRIPTION

`ncrcat` concatenates record variables across an arbitrary number of `input-files`. The final record dimension is by default the sum of the lengths of the record dimensions in the input files. See Section 2.6 [Statistics vs. Concatenation], page 18, for a description of the distinctions between the various statistics tools and concatenators. As a multi-file operator, `ncrcat` will read the list of `input-files` from `stdin` if they are not specified as positional arguments on the command line (see Section 2.7 [Large Numbers of Files], page 19).

Input files may vary in size, but each must have a record dimension. The record coordinate, if any, should be monotonic (or else non-fatal warnings may be generated). Hyperslabs along the record dimension that span more than one file are handled correctly. `ncra` supports the `stride` argument to the `-d` hyperslab option for the record dimension only, `stride` is not supported for non-record dimensions.

Concatenating a variable packed with different scales multiple datasets is beyond the capabilities of `ncrcat` (and `ncecat`, the other concatenator (Section 2.6.1 [Concatenation], page 18). `ncrcat` does not unpack data, it simply copies the data from the `input-files`, and the metadata from the first `input-file`, to the `output-file`. This means that data compressed with a packing convention must use the identical packing parameters (e.g., `scale_factor` and `add_offset`) for a given variable across all input files. Otherwise the concatenated dataset will not unpack correctly. The workaround for cases where the packing parameters differ across `input-files` requires three steps: First, unpack the data using `ncpdq`. Second, concatenate the unpacked data using `ncrcat`. Third, re-pack the result with `ncpdq`.

`ncrcat` applies special rules to ARM convention time fields (e.g., `time_offset`). See Section 3.39 [ARM Conventions], page 112 for a complete description.

EXAMPLES

Concatenate files `85.nc`, `86.nc`, ... `89.nc` along the record dimension, and store the results in `8589.nc`:

```bash
ncrcat 85.nc 86.nc 87.nc 88.nc 89.nc 8589.nc
ncrcat [56789].nc 8589.nc
ncrcat -n 5,2,1 85.nc 8589.nc
```

These three methods produce identical answers. See Section 3.5 [Specifying Input Files], page 28, for an explanation of the distinctions between these methods.
Assume the files 85.nc, 86.nc, ... 89.nc each contain a record coordinate time of length 12 defined such that the third record in 86.nc contains data from March 1986, etc. NCO knows how to hyperslab the record dimension across files. Thus, to concatenate data from December, 1985–February, 1986:

\[ \text{ncrcat} -d \text{time},11,13 85.nc 86.nc 87.nc 8512_8602.nc \]
\[ \text{ncrcat} -F -d \text{time},12,14 85.nc 86.nc 87.nc 8512_8602.nc \]

The file 87.nc is superfluous, but does not cause an error. When ncra and ncrcat encounter a file which does contain any records that meet the specified hyperslab criteria, they disregard the file and proceed to the next file without failing. The ‘-F’ turns on the Fortran (1-based) indexing convention.

The following uses the stride option to concatenate all the March temperature data from multiple input files into a single output file

\[ \text{ncrcat} -F -d \text{time},3,,12 -v \text{temperature} 85.nc 86.nc 87.nc 858687_03.nc \]

See Section 3.16 [Stride], page 57, for a description of the stride argument.

Assume the time coordinate is incrementally numbered such that January, 1985 = 1 and December, 1989 = 60. Assuming ?? only expands to the five desired files, the following concatenates June, 1985–June, 1989:

\[ \text{ncrcat} -d \text{time},6.,54. ??\text{.nc} 8506_8906.nc \]
4.11 ncrename netCDF Renamer

SYNTAX

ncrename [-a old_name,new_name] [-a ...] [-D dbg]
[-d old_name,new_name] [-d ...] [-g old_name,new_name] [-g ...]
[-v old_name,new_name] [-v ...]
input-file [[output-file]]

DESCRIPTION

ncrename renames netCDF dimensions, variables, attributes, and groups. Each object
that has a name in the list of old names is renamed using the corresponding name in the list
of new names. All the new names must be unique. Every old name must exist in the input
file, unless the old name is preceded by the period (or “dot”) character ‘.’. The validity of
old_name is not checked prior to the renaming. Thus, if old_name is specified without the
‘.’ prefix that indicates the presence of old_name is optional, and old_name is not present
in input-file, then ncrename will abort. The new_name should never be prefixed by a ‘.’
(or else the period will be included as part of the new name). As of NCO version 4.4.6
(released October, 2014), the old_name and new_name arguments may include (or be, for
groups) partial or full group paths. The OPTIONS and EXAMPLES show how to select
specific variables whose attributes are to be renamed.

Caveat lector: Unfortunately from 2007–present (October, 2014) the netCDF library
(versions 4.0.0–4.3.2) contained bugs or limitations that sometimes prevent NCO from cor-
rectly renaming coordinate variables, dimensions, and groups in netCDF4 files. (To our
knowledge the netCDF library calls for renaming always work well on netCDF3 files so one
workaround to some netCDF4 issues is convert to netCDF3, rename, then convert back).
To understand the renaming limitations associated with particular netCDF versions, read
the ncrename documentation below in its entirety.

Although ncrename supports full pathnames for both old_name and new_name, this is
really “window dressing”. The full-path to new_name must be identical to the full-path to
old_name in all classes of objects (attributes, variables, dimensions, or groups). In other
words, ncrename can change only the local names of objects, it cannot change the location
of the object in the group hierarchy within the file. Hence using a full-path in new_name is
redundant. The object name is the terminal path component of new_name and this object
must already exist in the group specified by the old_name path.

ncrename is an exception to the normal NCO rule that the user will be interactively
prompted before an existing file is changed, and that a temporary copy of an output file
is constructed during the operation. If only input-file is specified, then ncrename changes
object names in the input-file in place without prompting and without creating a temporary
copy of input-file. This is because the renaming operation is considered reversible if the
user makes a mistake. The new_name can easily be changed back to old_name by using
ncrename one more time.
Note that renaming a dimension to the name of a dependent variable can be used to invert
the relationship between an independent coordinate variable and a dependent variable.
In this case, the named dependent variable must be one-dimensional and should have no
missing values. Such a variable will become a coordinate variable.

According to the netCDF User Guide, renaming objects in netCDF files does not incur
the penalty of recopying the entire file when the new_name is shorter than the old_name.
Thus ncrename may run much faster (at least on netCDF3 files) if judicious use of header
padding (see Section 3.2 [Metadata Optimization], page 25) was made when producing the
input-file. Similarly, using the ‘--hdr_pad’ option with ncrename helps ensure that future
metadata changes to output-file occur as swifly as possible.

OPTIONS

‘-a old_name,new_name’
Attribute renaming. The old and new names of the attribute are specified
with ‘-a’ (or ‘--attribute’) by the associated old_name and new_name values.
Global attributes are treated no differently than variable attributes. This option
may be specified more than once. As mentioned above, all occurrences of the
attribute of a given name will be renamed unless the ‘.’ form is used, with one
exception. To change the attribute name for a particular variable, specify the
old_name in the format old_var_name@old_att_name. The ‘@’ symbol delimits
the variable from the attribute name. If the attribute is uniquely named (no
other variables contain the attribute) then the old_var_name@old_att_name
syntax is redundant. The old_var_name variable names global and group have
special significance. They indicate that old_att_num should only be renamed
where it occurs as a global (i.e., root group) metadata attribute (for global),
or (for group) as any group attribute, and not where it occurs as a variable
attribute. The var_name@att_name syntax is accepted, though not required,
for the new_name.

‘-d old_name,new_name’
Dimension renaming. The old and new names of the dimension are specified
with ‘-d’ (or ‘--dmn’, ‘--dimension’) by the associated old_name and new_name values. This option
may be specified more than once.

‘-g old_name,new_name’
Group renaming. The old and new names of the group are specified with ‘-g’
(or ‘-grp’, ‘--group’) by the associated old_name and new_name values. This option
may be specified more than once. This functionality is only available
in NCO version 4.3.7 (October, 2013) or later, and only when built on netCDF
library version 4.3.1-rc1 (August, 2013) or later.

‘-v old_name,new_name’
Variable renaming. The old and new names of the variable are specified with
‘-v’ (or ‘--variable’) by the associated old_name and new_name values. This option
may be specified more than once.

EXAMPLES
Rename the variable \( p \) to \texttt{pressure} and \( t \) to \texttt{temperature} in netCDF \texttt{in.nc}. In this case \( p \) must exist in the input file (or \texttt{ncrename} will abort), but the presence of \( t \) is optional:

\begin{verbatim}
ncrename -v p,pressure -v .t,temperature in.nc
\end{verbatim}

Rename the attribute \texttt{long_name} to \texttt{largo_nombre} in the variable \texttt{u}, and no other variables in netCDF \texttt{in.nc}.

\begin{verbatim}
ncrename -a u@long_name,largo_nombre in.nc
\end{verbatim}

Rename the group \texttt{g8} to \texttt{g20} in netCDF4 file \texttt{in_grp.nc}:

\begin{verbatim}
ncrename -g g8,g20 in_grp.nc
\end{verbatim}

Rename the variable \texttt{/g1/lon} to \texttt{longitude} in netCDF4 \texttt{in_grp.nc}:

\begin{verbatim}
ncrename -v /g1/lon,longitude in_grp.nc
ncrename -v /g1/lon,/g1/longitude in_grp.nc # Alternate
\end{verbatim}

\texttt{ncrename} does not automatically attach dimensions to variables of the same name. This is done to make renaming an easy way to change whether a variable is a coordinate. If you want to rename a coordinate variable so that it remains a coordinate variable, you must separately rename both the dimension and the variable:

\begin{verbatim}
ncrename -d lon,longitude -v lon,longitude in.nc
\end{verbatim}

Unfortunately, the netCDF4 library had a longstanding bug (all versions until 4.3.1-rc5 released in December, 2013) that crashed NCO when performing this operation. Simultaneously renaming variables and dimensions in netCDF4 files with earlier versions of netCDF is impossible; it must instead be done in two separate \texttt{ncrename} invocations (e.g., first rename the variable, then rename the dimension) to avoid triggering the library bug.

A related bug causes unintended side-effects with \texttt{ncrename} also built with all versions of the netCDF4 library until 4.3.1-rc5 released in December, 2013): This bug caused renaming \textit{either} a dimension \textit{or} its associated coordinate variable (not both, which would fail as above) in a netCDF4 file to inadvertently rename both:

\begin{verbatim}
# Demonstate bug in netCDF4/HDF5 library prior to netCDF-4.3.1-rc5
ncks -O -h -m -M -4 -v lat_T42 ~/nco/data/in.nc ~/foo.nc
ncrename -O -v lat_T42,lat ~/foo.nc ~/foo2.nc # Also renames dimension
ncrename -O -d lat_T42,lat ~/foo.nc ~/foo2.nc # Also renames variable
\end{verbatim}

To avoid this faulty behavior, either build NCO with netCDF version 4.3.1-rc5 or later, or convert the file to netCDF3 first, then rename as intended, then convert back. Unfortunately while this bug and the related coordinate renaming bug were fixed in 4.3.1-rc5 (released in December, 2013), a new and related bug was discovered in October 2014.

Another netCDF bug that causes unintended side-effects with \texttt{ncrename} affects (at least) versions 4.3.1–4.3.2 and all snapshots of the netCDF4 library until January, 2015. This bug (fixed in 4.3.3 in February, 2015) corrupts values or renamed netCDF4 coordinate variables (i.e., variables with underlying dimensions of the same name) and other (non-coordinate) variables that include an underlying dimension that was renamed. In other words, \textit{re}naming coordinate variables and dimensions succeeds yet it corrupts the values contained by the
affected array variables. This bug corrupts affected variables by replacing their values with the default \_FillValue for that variable’s type:

```bash
# Demonstate bug in netCDF4/HDF5 library prior to netCDF-4.3.3
ncks -O -4 -C -M -v lat ~/nco/data/in.nc ~/bug.nc
ncrename -O -v lat,tal ~/bug.nc ~/foo.nc # Broken until netCDF-4.3.3
ncrename -O -d lat,tal ~/bug.nc ~/foo.nc # Broken until netCDF-4.3.3
ncrename -O -d lat,tal -v lat,tal ~/bug.nc ~/foo.nc # Broken too
ncks --cdl ~/foo.nc
```

To avoid this faulty behavior, either build NCO with netCDF version 4.3.3 or later, or convert the file to netCDF3 first, then rename as intended, then convert back. This bug does not affect renaming of groups or of attributes.

The final netCDF bug that causes unintended side-effects with ncrename affects only snapshots from January–February, 2015, and released version 4.3.3 (February, 2015). It was fixed in netCDF version 4.3.3.1 (March, 2015). This bug causes renamed attributes of coordinate variables in netCDF4 to files to disappear:

```bash
# Demonstate bug in netCDF4/HDF5 library netCDF-4.3.3
ncrename -O -h -a /g1/lon@units,new_units ~/nco/data/in_grp.nc ~/foo.nc
ncks -v /g1/lon ~/foo.nc # Shows units and new_units are both gone
```

As of netCDF version 4.3.3.1 (March, 2015) there are no known renaming bugs.

Create netCDF out.nc identical to in.nc except the attribute \_FillValue is changed to missing_value, the attribute units is changed to CGS\_units (but only in those variables which possess it), the attribute hieght is changed to height in the variable tpt, and in the variable prs\_sfc, if it exists.

```bash
ncrename -a _FillValue,missing_value -a .units,CGS_units \ 
-a tpt@hieght,height -a prs_sfc@.hieght,height in.nc out.nc
```

The presence and absence of the ‘.’ and ‘\@’ features cause this command to execute successfully only if a number of conditions are met. All variables must have a \_FillValue attribute and \_FillValue must also be a global attribute. The units attribute, on the other hand, will be renamed to CGS\_units wherever it is found but need not be present in the file at all (either as a global or a variable attribute). The variable tpt must contain the hieght attribute. The variable prs\_sfc need not exist, and need not contain the hieght attribute.

Rename the global or group attribute Convention to Conventions

```bash
ncrename -a Convention,Conventions in.nc # Variable and group atts.
ncrename -a .Convention,Conventions in.nc # Variable and group atts.
ncrename -a @Convention,Conventions in.nc # Group atts. only
ncrename -a @.Convention,Conventions in.nc # Group atts. only
ncrename -a global@Convention,Conventions in.nc # Group atts. only
ncrename -a .global@.Convention,Conventions in.nc # Group atts. only
ncrename -a .global@.Convention,Conventions in.nc # Global atts. only
```
The examples without the @ character attempt to change the attribute name in both Global or Group and variable attributes. The examples with the @ character attempt to change only global and group Convention attributes, and leave unchanged any Convention attributes attached directly to variables. Attributes prefixed with a period (.Convention) need not be present. Attributes not prefixed with a period (Convention) must be present. Variables prefixed with a period (or .global) need not be present. Variables not prefixed with a period (global) must be present.
4.12 ncwa netCDF Weighted Averager

SYNTAX

[-D dbg] [-d dim,[min][,[max][,[stride]]]] [-F]
[-G gpe_dsc] [-g grp[,...]] [-h] [--hdr_pad nbr] [-I]
[-T mask_comp] [-t thr_nbr] [--unn] [-v var[,...]] [-w weight]
[-X ...] [-x] [-y op_typ]
input-file [output-file]

DESCRIPTION

ncwa averages variables in a single file over arbitrary dimensions, with options to specify weights, masks, and normalization. See Section 2.6 [Statistics vs. Concatenation], page 18, for a description of the distinctions between the various statistics tools and concatenators. The default behavior of ncwa is to arithmetically average every numerical variable over all dimensions and to produce a scalar result for each.

Averaged dimensions are, by default, eliminated as dimensions. Their corresponding coordinates, if any, are output as scalar variables. The ‘-b’ switch (and its long option equivalents ‘--rdd’ and ‘--retain-degenerate-dimensions’) causes ncwa to retain averaged dimensions as degenerate (size 1) dimensions. This maintains the association between a dimension (or coordinate) and variables after averaging and simplifies, for instance, later concatenation along the degenerate dimension.

To average variables over only a subset of their dimensions, specify these dimensions in a comma-separated list following ‘-a’, e.g., ‘-a time,lat,lon’. As with all arithmetic operators, the operation may be restricted to an arbitrary hypserslab by employing the ‘-d’ option (see Section 3.15 [Hyperslabs], page 55). ncwa also handles values matching the variable’s _FillValue attribute correctly. Moreover, ncwa understands how to manipulate user-specified weights, masks, and normalization options. With these options, ncwa can compute sophisticated averages (and integrals) from the command line.

mask_var and weight, if specified, are broadcast to conform to the variables being averaged. The rank of variables is reduced by the number of dimensions which they are averaged over. Thus arrays which are one dimensional in the input-file and are averaged by ncwa appear in the output-file as scalars. This allows the user to infer which dimensions may have been averaged. Note that that it is impossible for ncwa to make make a weight or mask_var of rank W conform to a var of rank V if W > V. This situation often arises when coordinate variables (which, by definition, are one dimensional) are weighted and averaged. ncwa assumes you know this is impossible and so ncwa does not attempt to broadcast weight or mask_var to conform to var in this case, nor does ncwa print a warning message telling you this, because it is so common. Specifying dbg > 2 does cause ncwa to emit warnings in these situations, however.
Non-coordinate variables are always masked and weighted if specified. Coordinate variables, however, may be treated specially. By default, an averaged coordinate variable, e.g., \textit{latitude}, appears in \textit{output-file} averaged the same way as any other variable containing an averaged dimension. In other words, by default \texttt{ncwa} weights and masks coordinate variables like all other variables. This design decision was intended to be helpful but for some applications it may be preferable not to weight or mask coordinate variables just like all other variables. Consider the following arguments to \texttt{ncwa}: \texttt{-a latitude -w lat_wgt -d latitude,0.,90.}, where \texttt{lat_wgt} is a weight in the \textit{latitude} dimension. Since, by default \texttt{ncwa} weights coordinate variables, the value of \textit{latitude} in the \textit{output-file} depends on the weights in \texttt{lat_wgt} and is not likely to be 45.0, the midpoint latitude of the hyperslab. Option \texttt{`-I'} overrides this default behavior and causes \texttt{ncwa} not to weight or mask coordinate variables\textsuperscript{1}. In the above case, this causes the value of \textit{latitude} in the \textit{output-file} to be 45.0, an appealing result. Thus, \texttt{`-I'} specifies simple arithmetic averages for the coordinate variables. In the case of \textit{latitude}, \texttt{`-I'} specifies that you prefer to archive the arithmetic mean latitude of the averaged hyperslabs rather than the area-weighted mean latitude.\textsuperscript{2}

As explained in See Section 3.33 [Operation Types], page 94, \texttt{ncwa} always averages coordinate variables regardless of the arithmetic operation type performed on the non-coordinate variables. This is independent of the setting of the \texttt{`-I'} option. The mathematical definition of operations involving rank reduction is given above (see Section 3.33 [Operation Types], page 94).

4.12.1 Mask condition

Each \(x_i\) also has an associated masking weight \(m_i\) whose value is 0 or 1 (false or true). The value of \(m_i\) is always 1 unless a \texttt{mask_var} is specified (with \texttt{`-m'}). As noted above, \texttt{mask_var} is broadcast, if possible, to conform to the variable being averaged. In this case, the value of \(m_i\) depends on the \texttt{mask condition} also known as the \texttt{truth condition}. As expected, \(m_i = 1\) when the mask condition is \texttt{true} and \(m_i = 0\) otherwise.

The mask condition has the syntax \texttt{mask_var mask_comp mask_val}. The preferred method to specify the mask condition is in one string with the \texttt{`-B'} or \texttt{`--mask_condition'} switches. The older method is to use the three switches \texttt{`-m'}, \texttt{`-T'}, and \texttt{`-M'} to specify the \texttt{mask_var}, \texttt{mask_comp}, and \texttt{mask_val}, respectively.\textsuperscript{3} The \texttt{mask_condition} string is automatically parsed into its three constituents \texttt{mask_var}, \texttt{mask_comp}, and \texttt{mask_val}.

Here \texttt{mask_var} is the name of the masking variable (specified with \texttt{`-m'}, \texttt{`--mask-variable'}, \texttt{`--mask_variable'}, \texttt{`--msk_nm'}, or \texttt{`--msk_var'}). The truth \texttt{mask_comp} argument (specified with \texttt{`-T'}, \texttt{`--mask_comparator'}, \texttt{`--msk_cmp_typ'}, or \texttt{`--op_rlt'}) may be any one of the six arithmetic comparators: \texttt{eq}, \texttt{ne}, \texttt{gt}, \texttt{lt}, \texttt{ge}, \texttt{le}. These are the Fortran-style character abbreviations for the logical comparisons =, \(\neq\), >, <, \(\geq\), \(\leq\). The mask comparator defaults to \texttt{eq} (equality). The \texttt{mask_val} argument to \texttt{`-M'} (or \texttt{`--mask-value'}, or \texttt{`--msk_val'}) is the right hand side of the \texttt{mask condition}. Thus for

\textsuperscript{1} The default behavior of (\texttt{`-I'}) changed on 19981201—before this date the default was not to weight or mask coordinate variables.

\textsuperscript{2} If \texttt{lat_wgt} contains Gaussian weights then the value of \textit{latitude} in the \textit{output-file} will be the area-weighted centroid of the hyperslab. For the example given, this is about 30 degrees.

\textsuperscript{3} The three switches \texttt{`-m'}, \texttt{`-T'}, and \texttt{`-M'} are maintained for backward compatibility and may be deprecated in the future. It is safest to write scripts using \texttt{`--mask_condition'}. 
the \(i\)’th element of the hyperslab to be averaged, the mask condition is \(\text{mask}, \text{mask}_\text{comp}\) \(\text{mask}_\text{val}\).

Each \(x_i\) is also associated with an additional weight \(w_i\) whose value may be user-specified. The value of \(w_i\) is identically 1 unless the user specifies a weighting variable \(\text{weight}\) (with \(-w, \text{--wgt}\) or \(-\text{wgt}_\text{var}\)). In this case, the value of \(w_i\) is determined by the \(\text{weight}\) variable in the \text{input-file}. As noted above, \(\text{weight}\) is broadcast, if possible, to conform to the variable being averaged.

\(M\) is the number of input elements \(x_i\) which actually contribute to output element \(x_j\). 
\(M\) is also known as the \(\text{tally}\) and is defined as

\[
M = \sum_{i=1}^{i=N} \mu_i m_i
\]

\(M\) is identical to the denominator of the generic averaging expression except for the omission of the weight \(w_i\). Thus \(M = N\) whenever no input points are missing values or are masked. Whether an element contributes to the output, and thus increments \(M\) by one, has more to do with the above two criteria (missing value and masking) than with the numeric value of the element per se. For example, \(x_i = 0.0\) does contribute to \(x_j\) (assuming the \_FillValue\ attribute is not 0.0 and location \(i\) is not masked). The value \(x_j = 0.0\) will not change the numerator of the generic averaging expression, but it will change the denominator (unless its weight \(w_i = 0.0\) as well).

### 4.12.2 Normalization and Integration

\texttt{ncwa} has one switch which controls the normalization of the averages appearing in the \text{output-file}. Short option \texttt{-N} (or long options \texttt{--nmr} or \texttt{--numerator}) prevents \texttt{ncwa} from dividing the weighted sum of the variable (the numerator in the averaging expression) by the weighted sum of the weights (the denominator in the averaging expression). Thus \texttt{-N} tells \texttt{ncwa} to return just the numerator of the arithmetic expression defining the operation (see Section 3.33 [Operation Types], page 94).

With this normalization option, \texttt{ncwa} can integrate variables. Averages are first computed as sums, and then normalized to obtain the average. The original sum (i.e., the numerator of the expression in Section 3.33 [Operation Types], page 94) is output if default normalization is turned off (with \texttt{-N}). This sum is the integral (not the average) over the specified (with \texttt{-a}, or all, if none are specified) dimensions. The weighting variable, if specified (with \texttt{-w}), plays the role of the differential increment and thus permits more sophisticated integrals (i.e., weighted sums) to be output. For example, consider the variable \texttt{lev} where \(\text{lev} = [100, 500, 1000]\) weighted by the weight \texttt{lev}_\texttt{wgt} where \(\text{lev}_\text{wgt} = [10, 2, 1]\).

The vertical integral of \texttt{lev}, weighted by \texttt{lev}_\texttt{wgt}, is the dot product of \texttt{lev} and \texttt{lev}_\texttt{wgt}.

That this is \(3000.0\) can be seen by inspection and verified with the integration command

\[
\texttt{ncwa -N -a lev -v lev -w lev_wgt in.nc foo.nc;ncks foo.nc}
\]

#### EXAMPLES

Given file \texttt{85\_0112.nc}:

\[
\texttt{netcdf 85\_0112 { dimensions:}
\]

lat = 64 ;
lev = 18 ;
lon = 128 ;
time = UNLIMITED ; // (12 currently)

variables:
float lat(lat) ;
float lev(lev) ;
float lon(lon) ;
float time(time) ;
float scalar_var ;
float three_dmn_var(lat, lev, lon) ;
float two_dmn_var(lat, lev) ;
float mask(lat, lon) ;
float gw(lat) ;

}  

Average all variables in in.nc over all dimensions and store results in out.nc:

ncwa in.nc out.nc

All variables in in.nc are reduced to scalars in out.nc since ncwa averages over all dimensions unless otherwise specified (with ‘-a’).

Store the zonal (longitudinal) mean of in.nc in out.nc:

ncwa -a lon in.nc out1.nc
ncwa -a lon -b in.nc out2.nc

The first command turns lon into a scalar and the second retains lon as a degenerate dimension in all variables.

% ncks -C -H -v lon out1.nc
lon = 135
% ncks -C -H -v lon out2.nc
lon[0] = 135

In either case the tally is simply the size of lon, i.e., 180 for the 85_0112.nc file described by the sample header above.

Compute the meridional (latitudinal) mean, with values weighted by the corresponding element of gw:

ncwa -w gw -a lat in.nc out.nc

Here the tally is simply the size of lat, or 64. The sum of the Gaussian weights is 2.0.

Compute the area mean over the tropical Pacific:

ncwa -w gw -a lat,lon -d lat,-20.,20. -d lon,120.,270. in.nc out.nc

Here the tally is $64 \times 128 = 8192$.

\footnote{gw stands for Gaussian weight in many climate models.}
Compute the area-mean over the globe using only points for which $ORO < 0.5^5$:

```bash
ncwa -B 'ORO < 0.5' -w gw -a lat,lon in.nc out.nc
ncwa -m ORO -M 0.5 -T lt -w gw -a lat,lon in.nc out.nc
```

It is considerably simpler to specify the complete `mask_cond` with the single string argument to `'-B'` than with the three separate switches `'-m'`, `'-T'`, and `'-M'`\(^6\). If in doubt, enclose the `mask_cond` within quotes since some of the comparators have special meanings to the shell.

Assuming 70% of the gridpoints are maritime, then here the tally is $0.70 \times 8192 \approx 5734$.

Compute the global annual mean over the maritime tropical Pacific:

```bash
ncwa -B 'ORO < 0.5' -w gw -a lat,lon,time \
    -d lat,-20.0,20.0 -d lon,120.0,270.0 in.nc out.nc
ncwa -m ORO -M 0.5 -T lt -w gw -a lat,lon,time \
    -d lat,-20.0,20.0 -d lon,120.0,270.0 in.nc out.nc
```

Further examples will use the one-switch specification of `mask_cond`.

Determine the total area of the maritime tropical Pacific, assuming the variable `area` contains the area of each gridcell

```bash
ncwa -N -v area -B 'ORO < 0.5' -a lat,lon \
    -d lat,-20.0,20.0 -d lon,120.0,270.0 in.nc out.nc
```

Weighting `area` (e.g., by `gw`) is not appropriate because `area` is already area-weighted by definition. Thus the `'-N'` switch, or, equivalently, the `'-y ttl'` switch, correctly integrate the cell areas into a total regional area.

Mask a file to contain `_FillValue` everywhere except where $thr_{\text{min}} <= msk_{\text{var}} <= thr_{\text{max}}$:

```bash
# Set masking variable and its scalar thresholds
export msk_var='three_dmn_var_dbl' # Masking variable
export thr_max='20' # Maximum allowed value
export thr_min='10' # Minimum allowed value
ncecat -O in.nc out.nc # Wrap out.nc in degenerate "record" dimension
ncwa -O -a record -B "${msk_var} <= ${thr_max}" out.nc out.nc
ncecat -O out.nc out.nc # Wrap out.nc in degenerate "record" dimension
ncwa -O -a record -B "${msk_var} >= ${thr_min}" out.nc out.nc
```

After the first use of `ncwa`, `out.nc` contains `_FillValue` where $\{msk_{\text{var}}\} >= \{thr_{\text{max}}\}$. The process is then repeated on the remaining data to filter out points where $\{msk_{\text{var}}\} <= \{thr_{\text{min}}\}$. The resulting `out.nc` contains valid data only where $thr_{\text{min}} <= msk_{\text{var}} <= thr_{\text{max}}$.

---

\(^5\) **ORO** stands for *Orography* in some climate models and in those models $ORO < 0.5$ selects ocean gridpoints.

\(^6\) Unfortunately the `'-B'` and `'---mask_condition'` options are unsupported on Windows (with the MVS compiler), which lacks a free, standard parser and lexer.
5 Contributing

We welcome contributions from anyone. The project homepage at https://sf.net/projects/nco contains more information on how to contribute.

Financial contributions to NCO development may be made through PayPal. NCO has been shared for over 10 years yet only two users have contributed any money to the developers\(^1\). So you could be the third!

5.1 Contributors

NCO would not exist without the dedicated efforts of the remarkable software engineers who conceive, develop, and maintain netCDF, UDUnits, and OPeNDAP. Since 1995 NCO has received support from, I believe, the entire staff of all these projects, including Russ Rew, John Caron, Glenn Davis, Steve Emmerson, James Gallagher, Ed Hartnett, and Dennis Heimbigner. In addition to their roles in maintaining the software stack on which NCO perches, Vertl-like, some of these gentlemen have advised or contributed to NCO specifically. That support is acknowledged separately below.

The primary contributors to NCO development have been:

Charlie Zender
All concept, design and implementation from 1995–2000. Since then autotools, bug-squashing, CDL, chunking, documentation, anchoring, recursion, GPE, packing, NCO library redesign, ncap2 features, ncb0, ncpdq, SMP threading and MPI parallelization, netCDF4 integration, external funding, project management, science research, releases.

Henry Butowsky
Non-linear operations and \texttt{min()}, \texttt{max()}, \texttt{total()} support in ncra and ncwa. Type conversion for arithmetic. Migration to netCDF3 API. ncap2 parser, lexer, GSL-support, and I/O. Multislabbing algorithm. Variable wildcarding. Numerous hacks. ncap2 language.

Rorik Peterson

Joe Hamman
Python bindings (NCOpy).

Daniel Wang
Script Workflow Analysis for MultiProcessing (SWAMP). RPM support.

Harry Mangalam
Benchmarking. OPeNDAP configuration.

---

\(^1\) Happy users have sent me a few gifts, though. This includes a box of imported chocolate. Mmm. Appreciation and gifts are definitely better than money. Naturally, I’m too lazy to split and send gifts to the other developers. However, unlike some NCO developers, I have a steady "real job". My intent is to split monetary donations among the active developers and to send them their shares via PayPal.
Pedro Vicente
  Windows Visual Studio support. netCDF4 groups.

Russ Rew  Advice on NCO structural algorithms

Brian Mays
  Original packaging for Debian GNU/Linux, nroff man pages.

George Shapovalov
  Packaging for Gentoo GNU/Linux.

Bill Kocik  Memory management.

Len Makin
  NEC SX architecture support.

Jim Edwards
  AIX architecture support.

Juliana Rew
  Compatibility with large PIDs.

Karen Schuchardt
  Auxiliary coordinate support.

Gayathri Venkitachalam
  MPI implementation.

Scott Capps
  Large work-load testing

Peter Campbell, Martin Dix, Mark Flanner, Markus Liebig, Keith Lindsay, Mike Page, Martin Schmidt, Michael Schulz, Lori Sentman, Rich Signell, Gary Strand, George White Andrew Wittenberg, Remik Ziemlinski
  Excellent bug reports and feature requests.

Daniel Baumann, Nick Bower, Luk Claes, Bas Couwenberg, Barry deFreese, Aleksandar Jelenak, Francesco Lovergine, Matej Vela
  Debian packaging

Patrice Dumas, Ed Hill, Orion Poplawski
  RedHat packaging

George Shapavalov, Patrick Kursawe
  Gentoo packaging

Filipe Fernandes
  OpenSuse packaging

Takeshi Enomoto, Alexander Hansen
  Mac OS packaging

Eric Blake  Autoconf/M4 help

Gavin Burris, Kyle Wilcox
  RHEL and CentOS build scripts and bug reports.
5.2 Citation

The recommended citations for NCO software are


Use the former when referring to overall design, purpose, and optimization of NCO, and use the latter when referring to specific features and/or the User Guide itself.

Additional information on citing NCO is in the User Guide at http://nco.sf.net#ctt. A complete list of NCO publications and presentations is at http://nco.sf.net#pub. This list links to the full papers and seminars themselves.

5.3 Proposals for Institutional Funding

NSF has funded a project to improve Distributed Data Reduction & Analysis (DDRA) by evolving NCO into a suite of Scientific Data Operators called SDO. The two main components of this project are NCO parallelism (OpenMP, MPI) and Server-Side DDRA (SSDDRA) implemented through extensions to OPeNDAP and netCDF4. This project will dramatically reduce bandwidth usage for NCO DDRA.

With this first NCO proposal funded, the content of the next NCO proposal is clear. We are interested in obtaining NASA support for HDF-specific enhancements to NCO. We plan to submit a proposal to the next suitable NASA NRA or NSF opportunity.

We are considering other interesting ideas for still more proposals. Please contact us if you wish to be involved with any future NCO-related proposals. Comments on the proposals and letters of support are also very welcome.
Chapter 6: Quick Start

Simple examples in Bash shell scripts showing how to average data with different file structures. Here we include monthly, seasonal and annual average with daily or monthly data in either one file or multiple files.

6.1 Daily data in one file

Suppose we have daily data from Jan 1st, 1990 to Dec. 31, 2005 in the file of in.nc with the record dimension as time.

Monthly average:

```bash
for yyyy in {1990..2005}; do  # Loop over years
    for moy in {1..12}; do    # Loop over months
        mm=$( printf "%02d" ${moy} ) # Change to 2-digit format
        # Average specific month yyyy-mm
        ncra -O -d time,"${yyyy}-${mm}-01","${yyyy}-${mm}-31" \
        in.nc in_${yyyy}${mm}.nc
    done
done

# Concatenate monthly files together
ncrcat -O in_?????.nc out.nc
```

Annual average:

```bash
for yyyy in {1990..2005}; do  # Loop over years
    ncra -O -d time,"${yyyy}-01-01","${yyyy}-12-31" in.nc in_${yyyy}.nc
done

# Concatenate annual files together
ncrcat -O in_????.nc out.nc
```

The -O switch means to overwrite the pre-existing files (see Section 3.35 [Batch Mode], page 108). The -d option is to specify the range of hyperslabs (see Section 3.15 [Hyperslabs], page 55). There are detailed instructions on ncra (see Section 4.9 [ncra netCDF Record Averager], page 210 and ncrcat (see Section 4.10 [ncrcat netCDF Record Concatenator], page 212). NCO supports UDUnits so that we can use readable dates as time dimension (see Section 3.22 [UDUnits Support], page 66).

6.2 Monthly data in one file

Inside the input file in.nc, the record dimension time is from Jan 1990 to Dec 2005.

Seasonal average (e.g., DJF):

```bash
ncra -O --mro -d time,"1990-12-01",,12,3 in.nc out.nc
```

Annual average:
Here we use the subcycle feature (i.e., the number after the fourth comma: ‘3’ in the seasonal example and the second ‘12’ in the annual example) to retrieve groups of records separated by regular intervals (see Section 3.18 [Subcycle], page 59). The option --mro switches ncra to produce a Multi-Record Output instead of a single-record output. For example, assume snd is a 3D array with dimensions time * latitude * longitude and time includes every month from Jan. 1990 to Dec. 2005, 192 months as total, which are 16 years. Let’s look at the following two command lines.

```
ncra --mro -v snd -d time,"1990-12-01",,12,3 in.nc out_mro.nc
ncra -v snd -d time,"1990-12-01",,12,3 in.nc out_sro.nc
```

In the first output file, out_mro.nc, snd is still a 3D array with dimensions time * latitude * longitude, but the length of time now is 16, meaning 16 winters. In the second output file, out_sro.nc, the length of time is only 1. It is now the average of all the 16 winters.

when using ‘-d dim,min[,max]’ to specify the hyperslabs, you can leave it blank if you want to include the minimum or the maximum of the data, like we did above.

### 6.3 One time point one file

This means if you have daily data of 30 days, there will be 30 data files. Or if you have monthly data of 12 months, there will be 12 data files. Dealing with this kind of files, you need to specify the file names in shell scripts and pass them to NCO operators. For example, your daily data files may look like snd_19900101.nc, snd_19900102.nc, snd_19900103.nc ...

If you want to know the monthly average of Jan 1990, you can write like,

```
ncra -O snd_199001??.nc out.nc
```

You might want to use loop if you need the average of each month.

```
for moy in {1..12}; do # Loop over months
    mm=$( printf "%02d" ${moy} ) # Change to 2-digit format
    ncra -O snd_????${mm}??.nc out_${mm}.nc
done
```

### 6.4 Multiple files with multiple time points

Similar as the last one, it’s more about shell scripts. Suppose you have daily data with one month of them in one data file. The monthly average is simply to apply ncra on the specific data file. And for seasonal averages, you can specify the three months by shell scripts.
Chapter 7: CMIP5 Example

7 CMIP5 Example

The fifth phase of the Coupled Model Intercomparison Project (CMIP5) provides a multi-model framework for comparing the mechanisms and responses of climate models from around the world. However, it is a tremendous workload to retrieve a single climate statistic from all these models, each of which includes several ensemble members. Not only that, it is too often a tedious process that impedes new research and hypothesis testing. Our NASA ACCESS 2011 project simplified and accelerated this process.

Traditional geoscience data analysis requires users to work with numerous flat (data in one level or namespace) files. In that paradigm instruments or models produce, and then repositories archive and distribute, and then researchers request and analyze, collections of flat files. NCO works well with that paradigm, yet it also embodies the necessary algorithms to transition geoscience data analysis from relying solely on traditional (or “flat”) datasets to allowing newer hierarchical (or “nested”) datasets.

Hierarchical datasets support and enable combining all datastreams that meet user-specified criteria into a single or small number of files that hold all the science-relevant data. NCO (and no other software to our knowledge) exploits this capability now. Data and metadata may be aggregated into and analyzed in hierarchical structures. We call the resulting data storage, distribution, and analysis paradigm Group-Oriented Data Analysis and Distribution (GODAD). GODAD lets the scientific question organize the data, not the ad hoc granularity of all relevant datasets. This chapter illustrates GODAD techniques applied to analysis of the CMIP5 dataset.

To begin, we document below a prototypical example of CMIP5 analysis and evaluation using traditional NCO commands on netCDF3-format model and HDF-EOS format observational (NASA MODIS satellite instrument) datasets. These examples complement the NCO User Guide by detailing in-depth data analysis in a frequently encountered “real world” context. Graphical representations of the results (NCL scripts available upon request) are provided to illustrate physical meaning of the analysis. Since NCO can process hierarchical datasets, i.e., datasets stored with netCDF4 groups, we present sample scripts illustrating group-based processing as well.

7.1 Combine Files

Sometimes, the data of one ensemble member will be stored in several files to reduce single file size. It is more convenient to concatenate these files into a single timeseries, and the following script illustrates how. Key steps include:

1. Obtain number and names (or partial names) of files in a directory
2. Concatenate files along the record dimension (usually time) using `ncrcat` (see Section 4.10 [ncrcat netCDF Record Concatenator], page 212).

   ```bash
   #!/bin/bash  # shell type
   shopt -s extglob # enable extended globbing

   #===========================================================================
   # Some of the models cut one ensemble member into several files,
   ```
which include data of different time periods.
We’d better concatenate them into one at the beginning so that we won’t have to think about which files we need if we want to retrieve a specific time period later.

Method:
- Make sure 'time' is the record dimension (i.e., left-most)
- ncrcat

Input files like:
- /data/cmip5/snc_LImon_bcc-csm1-1_historical_r1i1p1_185001-190012.nc
- /data/cmip5/snc_LImon_bcc-csm1-1_historical_r1i1p1_190101-200512.nc

Output files like:
- /data/cmip5/snc_LImon_bcc-csm1-1_historical_r1i1p1_185001-200512.nc

Online: http://nco.sourceforge.net/nco.html#Combine-Files

Execute this script: bash cmb_fl.sh

drc_in='/home/wenshanw/data/cmip5/' # Directory of input files
var=('snc' 'snd') # Variables
rlm='LImon' # Realm
xpt=('historical') # Experiment (could be more)

for var_id in {0..1}; do
  # Names of all the models (ls [get file names];
  # cut [get model names];
  # sort; uniq [remove duplicates]; awk [print])
  mdl_set=$(ls $drc_in$var[var_id]_${rlm}_*.nc |
            cut -d '_' -f 3 | sort | uniq -c | awk '{print $2}' )
  # Number of models (echo [print contents]; wc [count])
  mdl_nbr=$(echo $mdl_set | wc -w)
  echo "=============================="
  echo "There are $mdl_nbr models for $var[var_id]."

  for mdl in $mdl_set; do
    # Names of all the ensemble members
    nsm_set=$(ls $drc_in$var[var_id]_${rlm}_${mdl}_${xpt[0]}_*.nc |
               cut -d '_' -f 5 | sort | uniq -c | awk '{print $2}' )
    # Number of ensemble members in each model
    nsm_nbr=$(echo $nsm_set | wc -w)
    echo "------------------------------"
    echo "Model $mdl "includes" $nsm_nbr "ensemble member(s):"
    echo $nsm_set"."
for nsm in ${nsm_set}; do  
  # Loop over ensemble members
  
  # Number of files in this ensemble member
  fl_nbr=$( ls ${drc_in}.${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_${nsm}_*.nc  
    | wc -w )

  # If there is only 1 file, continue to next loop
  if [ ${fl_nbr} -le 1 ]
  then
    echo "There is only 1 file in" ${nsm}.
    continue
  fi

  echo "There are" ${fl_nbr} "files in" ${nsm}.

  # Starting date of data
  yyyymm_str=$( ls ${drc_in}.${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_${nsm}_*.nc  
    | sed -n '1p' | cut -d '_' -f 6 | cut -d '-' -f 1 )

  # Ending date of data
  yyyymm_end=$( ls ${drc_in}.${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_${nsm}_*.nc  
    | sed -n "${fl_nbr}p" | cut -d '_' -f 6 | cut -d '-' -f 2 )

  # Concatenate one ensemble member files
  ncrecat -O ${drc_in}.${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_${nsm}_*.nc  
    ${drc_in}.${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_${nsm}_${yyyymm_str}-${yyyymm_end}

  # Remove useless files
  rm ${drc_in}.${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_${nsm}_*.nc  
    !(${yyyymm_str}-${yyyymm_end})
done
done
done

CMIP5 model data downloaded from the Earth System Grid Federation (ESGF) does not contain group features yet. Therefore users must aggregate flat files into hierarchical ones themselves. The following script shows how. Each dataset becomes a group in the output file. There can be several levels of groups. In this example, we employ two experiments ("scenarios") as the top-level. The second-level comprises different models (e.g., CCSM4, CESM1-BGC). Many models are run multiple times with slight perturbed initial conditions to produce an ensemble of realizations. These ensemble members comprise the third level of the hierarchy. The script selects two variables, snc and snd (snow cover and snow depth).

#!/bin/bash
#
Aggregate models to one group file

Method:
- Create files with groups by ncecat --gag
- Append groups level by level using ncks

Input files like:
- snc_Limon_CCSM4_historical_r1i1p1_199001-200512.nc
- snd_Limon_CESM1-BGC_esmHistorical_r1i1p1_199001-200512.nc

Output files like:
- sn_Limon_199001-200512.nc

Online: http://nco.sourceforge.net/nco.html#Combine-Files

Execute this script: bash cmb_fl_grp.sh

Directories

drc_in='..data/
drc_out='..data/grp/

Constants
rlm='Limon' # Realm: LandIce; Time frequency: monthly
tms='200001-200512' # Timeseries
flt='nc' # File Type

Geographical weights
Can be skipped when ncap2 works on group data
Loop over all snc files
for fn in $( ls $(drc_in)snc_${rlm}*_${tms}.${flt} ); do
cap2 -O -s
  'gw = float(cos(lat*3.1416/180.)); gw@long_name="geographical weight";'
  \${fn} $fn
done

var=( 'snc' 'snd' )
xpt=( 'esmHistorical' 'historical' )
mdl=( 'CCSM4' 'CESM1-BGC' 'CESM1-CAM5' )

for i in {0..1}; do # Loop over variables
  for j in {0..1}; do # Loop over experiments
    for k in {0..2}; do # Loop over models
      ncecat -O --glb_mtd_spp -G ${xpt[j]}_${mdl[k]}_${mdl[k]}_ 
       ${drc_in}snc_${var[i]}_${rlm}_${mdl[k]}_${xpt[j]}_*_${tms}.${flt} 
       ${drc_out}snc_${var[i]}_${rlm}_${mdl[k]}_${xpt[j]}_all-nsm_${tms}.${flt} 
    done
  done
done
ncks -A
${drc_out}${var}[i]_${rlm}_${mdl}[k]_${xpt}[j]_all-nsm_${tms}.${flt} \
${drc_out}${var}[i]_${rlm}_${mdl}[0]_${xpt}[j]_all-nsm_${tms}.${flt}
done  # Loop done: models

ncks -A
${drc_out}${var}[i]_${rlm}_${mdl}[0]_${xpt}[j]_all-nsm_${tms}.${flt} \
${drc_out}${var}[i]_${rlm}_${mdl}[0]_${xpt}[0]_all-nsm_${tms}.${flt}
done  # Loop done: experiments

ncks -A
${drc_out}${var}[i]_${rlm}_${mdl}[0]_${xpt}[0]_all-nsm_${tms}.${flt} \
${drc_out}${var}[0]_${rlm}_${mdl}[0]_${xpt}[0]_all-nsm_${tms}.${flt}
done  # Loop done: variables

# Rename output file
mv ${drc_out}${var}[0]_${rlm}_${mdl}[0]_${xpt}[0]_all-nsm_${tms}.${flt} \
${drc_out}sn_${rlm}_all-mdl_all-xpt_all-nsm_${tms}.${flt}

# Remove temporary files
rm ${drc_out}sn?[_]*.nc

#- Rename Group:
# E.g., file snc_LImon_CESM1-CAM5_historical_r1i1p1_199001-200512.nc
# is now group /historical/CESM1-CAM5/CESM1-CAM5_00.
# You can rename it to /historical/CESM1-CAM5/r1i1p1 to make more sense.
# Note: You don't need to write the full path of the new name.
ncrename -g ${xpt}/${mdl}/${mdl}_00,r1i1p1 \
${drc_out}${var}_${rlm}_${mdl}_all-nsm_${tms}.${flt}

#------------------------------------------------------------
# Output file structure
#------------------------------------------------------------
# esmHistorical
# {
#   CESM1-BGC
#   {
#     CESM1-BGC_00
#     {
#       snc(time, lat, lon)
#       snd(time, lat, lon)
#     }
#   }
# }
# historical
# {
#   CCSM4
#   {
#     CCSM4_00
#   }
#   }
# snc(time, lat, lon)
# snd(time, lat, lon)
#
# CCSM4_01
# {
# snc(time, lat, lon)
# snd(time, lat, lon)
# }
# CCSM4_02 { ... }
# CCSM4_03 { ... }
# CCSM4_04 { ... }
#
# CESM1-BGC
# {
# CESM1-BGC_00 { ... }
# }
# CESM1-CAM5
# {
# r1i1p1 { ... }
# CESM1-CAM5_01 { ... }
# CESM1-CAM5_02 { ... }
# }
# 
7.2 Global Distribution of Long-term Average

Figure 7.1: Global Distribution of Long-term Average.

This section illustrates how to calculate the global distribution of long-term average (see Figure 7.1) with either flat files or group file. Key steps include:

1. Average ensemble members of each model using nces (see Section 4.4 [nces netCDF Ensemble Statistics], page 179)
2. Average the record dimension using ncra (see Section 4.9 [ncra netCDF Record Averager], page 210)
3. Store results of each model as a distinct group in a single output file using ncecat (see Section 4.10 [ncecat netCDF Record Concatenator], page 212) with the --gag option

The first example shows how to process flat files.

#!/bin/bash
    
# After cmb_fl.sh
Example: Long-term average of each model globally

Input files like:
/data/cmip5/snc_LImon_bcc-csm1-1_historical_r1i1p1_185001-200512.nc

Output files like:
/data/cmip5/output/snc/snc_LImon_all-mdl_historical_all-nsm_clm.nc

Online:
http://nco.sourceforge.net/nco.html#Global-Distribution-of-Long_002dterm-Average

Execute this script: bash glb_avg.sh

Parameters

drc_in=/home/wenshanw/data/cmip5/ # Directory of input files
drc_out=/home/wenshanw/data/cmip5/output/ # Directory of output files

var=( 'snc' 'snd' ) # Variables
rlm='LImon' # Realm
xpt=( 'historical' ) # Experiment ( could be more )

fld_out=( 'snc/' 'snd/' ) # Folders of output files

for var_id in {0..1}; do # Loop over two variables
  mdl_set=$( ls $(drc_in)$var[var_id]_${rlm}_*_${xpt[0]}_*.nc | cut -d '_' -f 3 | sort | uniq -c | awk '{print $2}' )
  mdl_num=$( echo $mdl_set | wc -w )
  for mdl in $mdl_set; do # Loop over models
    nces --nsm_fl -O -4 -d time,"1956-01-01 00:00:0.0","2005-12-31 23:59:9.9" \
    $(drc_in)$var[var_id]_${rlm}_${mdl}_${xpt[0]}_*.nc \n    $(drc_out)$fld_out[var_id]${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_195601-200512.nc
    ncrs -O $(drc_out)$fld_out[var_id]${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_195601-200512.nc
  done
done

Average all the ensemble members of each model

Use nces file ensemble mode: --nsm_fl
nces --nsm_fl -O -4 -d time,"1956-01-01 00:00:0.0","2005-12-31 23:59:9.9" \\
$(drc_in)$var[var_id]_${rlm}_${mdl}_${xpt[0]}_*.nc \n$(drc_out)$fld_out[var_id]${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_195601-200512.nc

Average along time
ncra -O $(drc_out)$fld_out[var_id]${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_195601-200512.nc
$(drc_out)$fld_out[var_id]${var[var_id]}_${mdl}.nc
echo Model ${mdl} done!
done

# Remove temporary files
rm ${drc_out}${fld_out}[var_id]${var[var_id]}historical*.nc

# Store models as groups in the output file
ncecat -O --gag ${drc_out}${fld_out}[var_id]${var[var_id]}*.nc \ ${drc_out}${fld_out}[var_id]${var[var_id]}_${rlm}_\ all-mdl_${xpt[0]}_all-nsm_clm.nc

echo Var ${var[var_id]} done!
done

With the use of group, the above script will be shortened to ONE LINE.

# Data from cmb_fl_grp.sh
# ensemble averaging
nces -O --nsm_grp --nsm_sfx=’_avg’ \ sn_Limon_all-mdl_all-xpt_all-nsm_200001-200512.nc \ sn_Limon_all-mdl_all-xpt_nsm-avg.nc

The input file, sn_Limon_all-mdl_all-xpt_all-nsm_200001-200512.nc, produced by cmb_fl_grp.sh, includes all the ensemble members as groups. The option ‘--nsm_grp’ denotes that we are using group ensembles mode of nces, instead of file ensembles mode, ‘--nsm_fl’. The option ‘--nsm_sfx=’_avg’’ instructs nces to store the output as a new child group /[model]/[model name]_avg/var; otherwise, the output will be stored directly in the parent group /[model]/var. In the final output file, sn_Limon_all-mdl_all-xpt_nsm-avg_tm-avg.nc, sub-groups with a suffix of ‘avg’ are the long-term averages of each model. One thing to notice is that for now, ensembles with only one ensemble member will be left untouched.
7.3 Annual Average over Regions

This section illustrates how to calculate the annual average over specific regions (see Figure 7.2). Key steps include:

1. Spatial average using `ncap2` (see Section 4.1 [ncap2 netCDF Arithmetic Processor], page 116) and `ncwa` (see Section 4.12 [ncwa netCDF Weighted Averager], page 219);
2. Change dimension order using `ncpdq` (see Section 4.8 [ncpdq netCDF Permute Dimensions Quickly], page 202);
3. Annual average using `ncra` (see Section 4.9 [ncra netCDF Record Averager], page 210);
4. Anomaly from long-term average using `ncbo` (see Section 4.3 [ncbo netCDF Binary Operator], page 174);
5. Standard deviation using `ncbo` (see Section 4.3 [ncbo netCDF Binary Operator], page 174) and `nces` (see Section 4.4 [nces netCDF Ensemble Statistics], page 179);
6. Rename variables using `ncrename` (see Section 4.11 [ncrename netCDF Renamer], page 214);
7. Edit attributions using `ncatted` (see Section 4.2 [ncatted netCDF Attribute Editor], page 168);
8. Linear regression using `ncap2` (see Section 4.1 [ncap2 netCDF Arithmetic Processor], page 116);

Figure 7.2: Annual Average over Regions.
9. Use `ncap2` (see Section 4.1 [ncap2 netCDF Arithmetic Processor], page 116) with nco script file (i.e., .nco file);

10. Move variables around using `ncks` (see Section 4.7 [ncks netCDF Kitchen Sink], page 188).

**Flat files example**

```bash
#!/bin/bash
# Includes gsl_rgr.nco

#===========================================================================
# After cmb_f1.sh
# Example: Annual trend of each model over Greenland and Tibet
# ( time- and spatial-average, standard deviation,
# anomaly and linear regression)
#
# Input files:
# /data/cmip5/snc_LImon_bcc-csm1-1_historical_r1i1p1_185001-200512.nc
#
# Output files:
# /data/cmip5/outout/snc/snc_LImon_all-mdl_historical_all-nsm_annual.nc
#
# Online: http://nco.sourceforge.net/nco.html#Annual-Average-over-Regions
#
# Execute this script: bash ann_avg.sh
#===========================================================================

# Parameters
drc_in='/home/wenshanw/data/cmip5/' # Directory of input files
drc_out='/home/wenshanw/data/cmip5/output/' # Directory of output files

var=( 'snc' 'snd' ) # Variables
rlm='LImon' # Realm
xpt=( 'historical' ) # Experiment ( could be more )

fld_out=( 'snc/' 'snd/' ) # Folders of output files

for var_id in {0..1}; do # Loop over two variables
    mdl_set=$( ls $drc_in${var[var_id]}_${rlm}_*_${xpt[0]}_*.nc | 
              cut -d '_' -f 3 | sort | uniq -c | awk '{print $2}' )
    for mdl in ${mdl_set}; do # Loop over models
        ...
    done

for mdl in ${mdl_set}; do # Loop over models
    ...
```

```bash
```
# Loop over ensemble members
for fn in $( ls ${drc_in}${var}[var_id]_${rlm}_${mdl}_${xpt}[0]_*.nc ); do
  pfx=$( echo $fn | cut -d'/' -f6 | cut -d'_' -f1-5 )
  # Two regions
  # Geographical weight
  ncap2 -O -s 'gw = cos(lat*3.1415926/180.); gw@long_name="geographical weight"; gw@units="ratio"' ${fn} ${drc_out}${fld_out}[var_id]${pfx}_gw.nc
  # Greenland
  ncwa -O -w gw -d lat,60.0,75.0 -d lon,300.0,340.0 -a lat,lon ${drc_out}${fld_out}[var_id]${pfx}_gw_1.nc
  # Tibet
  ncwa -O -w gw -d lat,30.0,40.0 -d lon,80.0,100.0 -a lat,lon ${drc_out}${fld_out}[var_id]${pfx}_gw_2.nc
  # Aggregate 2 regions together
  ncecat -O -u rgn ${drc_out}${fld_out}[var_id]${pfx}_gw_?.nc ${drc_out}${fld_out}[var_id]${pfx}_gw_rgn4.nc
  # Change dimensions order
  ncpdq -O -a time,rgn ${drc_out}${fld_out}[var_id]${pfx}_gw_rgn4.nc
  # Remove temporary files (optional)
  rm ${drc_out}${fld_out}[var_id]${pfx}_gw_.nc
  rm ${drc_out}${fld_out}[var_id]${pfx}_gw.nc
  # Annual average (use the feature of 'Duration')
  ncra -O --mro -d time,"1956-01-01 00:00:0.0","2005-12-31 23:59:9.9",12,12 ${drc_out}${fld_out}[var_id]${pfx}_gw_rgn4.nc
  ${drc_out}${fld_out}[var_id]${pfx}_yrly.nc
  # Anomaly
  # Long-term average
  ncwa -O -a time ${drc_out}${fld_out}[var_id]${pfx}_yrly.nc
  ${drc_out}${fld_out}[var_id]${pfx}_clm.nc
  # Subtract long-term average
  ncbo -O --op_typ=- ${drc_out}${fld_out}[var_id]${pfx}_yrly.nc
  ${drc_out}${fld_out}[var_id]${pfx}_anm.nc
  done
rm ${drc_out}${fld_out}[var_id]${var}[var_id]_${rlm}_${mdl}_${xpt}[0]_*.yrly.nc
# Average over all the ensemble members
ncea -O -4 ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_*_anm.nc ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_anm.nc

# Standard deviation ------------------------------
for fn in $( ls ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_*_anm.nc ); do
  pfx=$( echo ${fn} | cut -d'/' -f8 | cut -d'_' -f1-5 )
  # Difference between each ensemble member and the average of all members
  ncbo -0 --op_typ=- ${fn} \
    ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_anm.nc \
    ${drc_out}${fld_out[var_id]}${pfx}_dlt.nc
done

# RMS
ncea -O -y rmssdn ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_*_dlt.nc \
    ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_sdv.nc

# Rename variables
ncrename -v ${var[var_id]},sdv \n    ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_sdv.nc

# Edit attributions
ncatted -a standard_name,sdv,a,c,"_standard_deviation_over_ensemble" \n    -a long_name,sdv,a,c," Standard Deviation over Ensemble" \n    -a original_name,sdv,a,c," sdv" \n    ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_sdv.nc

#------------------------------------------------------------
# Linear regression -----------------------------------------
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Have to change the name of variable in the commands file
# of gsl_rgr.nco manually (gsl_rgr.nco is listed below)
ncap2 -O -S gsl_rgr.nco \n    ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_anm_rgr.nc \n    ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_anm_rgr.nc

#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Get rid of temporary variables
ncks -O -v c0,c1,pval,${var[var_id]},gw \n    ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_anm_rgr.nc \n
# Move the variable 'sdv' into the anomaly files (i.e., *anm.nc files)
nc -A -v sdv
${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}\$
${mdl}\${xpt[0]}_all-nsm_sdv.nc
rm ${drc_out}${fld_out[var_id]}${var[var_id]}_${mdl}.nc

echo Model ${mdl} done!
done

# Store models as groups in the output file
ncecat -O --gag ${drc_out}${fld_out[var_id]}${var[var_id]}_*.nc
${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_all-mdl_${xpt[0]}_all-nsm_annual.nc

echo Var ${var[var_id]} done!
done

gsl_rgr.nco

// Linear Regression
// Called by ann_avg.sh
// Caution: make sure the variable name is
// in agreement with the main script (now is 'snd')
// Online: http://nco.sourceforge.net/nco.html#Annual-Average-over-Regions

// Declare variables
*c0[$rgn]=0.; // Intercept
*c1[$rgn]=0.; // Slope
*sdv[$rgn]=0.; // Standard deviation
*covxy[$rgn]=0.; // Covariance
*x = double(time);

for (*rgn_id=0;rgn_id<$rgn.size;rgn_id++) // Loop over regions
{
    gsl_fit_linear(time,1,snd(:,rgn_id),1,$time.size, 
&tc0, &tc1, &cov00, &cov01,&cov11,&sumsq); // Linear regression function
    c0(rgn_id) = tc0; // Output results
    c1(rgn_id) = tc1;
    covxy(rgn_id) = gsl_stats_covariance(time,1, 
$time.size,double(snd(:,rgn_id)),1,$time.size); // Covariance function
    sdv(rgn_id) = gsl_stats_sd(snd(:,rgn_id), \n1, $time.size); // Standard deviation function
}
P value------------------------------------------------------------
*time_sdv = gsl_stats_sd(time, 1, $time.size);
*r_value = covxy/(time_sdv*sdv);
*t_value = r_value/sqrt((1-r_value^2)/($time.size-2));
pval = abs(gsl_cdf_tdist_P(t_value, $time.size-2) - \  
gsl_cdf_tdist_P(-t_value, $time.size-2));
----------------------------------------------------------------

Write RAM variables to disk
// Usually NCO writes the outputs directly to disk
// Using RAM variables, declared by *, will shorten running time
// Output the final results using ram_write()
-------------------------------
ram_write(c0);
ram_write(c1);

With the group feature, all the loops over experiments, models and ensemble members can be omitted. As we are working on implementing group feature in all NCO operators, some functions (e.g., regression and standard deviation over ensemble members) may have to wait until the new versions.

#!/bin/bash
#============================================================
# Group data output by cmb_fl_grp.sh
# Annual trend of each model over Greenland and Tibet
# Time- and spatial-average, standard deviation and anomaly
# No regression yet (needs ncap2)
#
# Input files:
# sn_LImon_all-mdl_all-xpt_all-nsm_200001-200512.nc
#
# Online: http://nco.sourceforge.net/nco.html#Annual-Average-over-Regions
#
# Execute this script: bash ann_avg_grp.sh
#============================================================

drc='../data/grp/'
pfx='sn_LImon_all-mdl_all-xpt_all-nsm'
tms='200001-200512' # Time series

# Greenland
ncwa -O -w gw -d lat,60.0,75.0 -d lon,300.0,340.0 -a lat,lon 
${drc}${pfx}_${tms}.nc 
${drc}${pfx}_${tms}_grl.nc
# Tibet
ncwa -0 -w gw -d lat,30.0,40.0 -d lon,80.0,100.0 -a lat,lon \ 
${drc}${pfx}_${tms}.nc \ 
${drc}${pfx}_${tms}_tbt.nc

# Aggregate 2 regions together
ncscat -0 -u rgn ${drc}${pfx}_${tms}_???.nc \ 
${drc}${pfx}_${tms}_rgn2.nc

# Change dimensions order
ncpdq -0 -a time,rgn ${drc}${pfx}_${tms}_rgn2.nc \ 
${drc}${pfx}_${tms}_rgn2.nc

# Remove temporary files (optional)
rm ${drc}${pfx}_${tms}_???.nc

# Annual average
ncra -0 --mro -d time,,,12,12 ${drc}${pfx}_${tms}_rgn2.nc \ 
${drc}${pfx}_${tms}_rgn2_ann.nc

# Anomaly
#-----------------------------------------
# Long-term average
ncwa -0 -a time ${drc}${pfx}_${tms}_rgn2_ann.nc \ 
${drc}${pfx}_${tms}_rgn2_clm.nc
# Subtract
ncbo -0 --op_typ=- ${drc}${pfx}_${tms}_rgn2_ann.nc \ 
${drc}${pfx}_${tms}_rgn2_clm.nc \ 
${drc}${pfx}_${tms}_rgn2_anm.nc
#-----------------------------------------

# Standard Deviation: inter-annual variability
# RMS of the above anomaly
ncra -0 -y rmssdn ${drc}${pfx}_${tms}_rgn2_anm.nc \ 
${drc}${pfx}_${tms}_rgn2_stddev.nc
7.4 Monthly Cycle

This script illustrates how to calculate the monthly anomaly from the annual average (see Figure 7.3). In order to keep only the monthly cycle, we will subtract the annual average of each year from the monthly data, instead of subtracting the long-term average. This is a little more complicated in coding since we need to loop over years.

**Flat files example**

```bash
#!/bin/bash

#============================================================
# After cmb_fl.sh
# Example: Monthly cycle of each model in Greenland
#
# Input files:
# /data/cmip5/snc_LImon_bcc-csm1-1_historical_r1i1p1_185001-200512.nc
#
# Output files:
# /data/cmip5/snc/snc_LImon__all-mdl_historical_all-nsm_GN_mthly-anm.nc
#
# Online: http://nco.sourceforge.net/nco.html#Monthly-Cycle
#
# Execute this script: bash mcc.sh
#============================================================

#------------------------------------------------------------
# Parameters
# drc_in='/home/wenshanw/data/cmip5/' # Directory of input files
drc_out='/home/wenshanw/data/cmip5/output/' # Directory of output files
var=('snc' 'snd') # Variables
```
rlm='LImon'  # Realm
xpt=('historical')  # Experiment (could be more)

fld_out=('snc/' 'snd/')  # Folders of output files

for var_id in {0..1}; do  # Loop over two variables
    # names of all models
    # (ls [get file names]; cut [get the part for model names];
    # sort; uniq [remove duplicates]; awk [print])
    mdl_set=$( ls ${drc_in}${var[var_id]}_${rlm}_*_${xpt[0]}_*.nc | 
               cut -d '_' -f 3 | sort | uniq -c | awk '{print $2}' )

    for mdl in ${mdl_set}; do  # Loop over models
        # Average all the ensemble members of each model
        ncea -O -4 -d time,"1956-01-01 00:00:0.0","2005-12-31 23:59:9.9" \
           ${drc_in}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_*.nc \
           ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm.nc

        # Greenland
        # Geographical weight
        ncap2 -O -s \n          'gw = cos(lat*3.1415926/180.); gw@long_name="geographical weight"; gw@units="ratio";' \n          ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm.nc \
          ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm.nc

        ncwa -O -w gw -d lat,60.0,75.0 -d lon,300.0,340.0 -a lat,lon \
           ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm.nc \
           ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_GN.nc

    # Anomaly----------------------------------------
    for moy in {1..12}; do  # Loop over months
        mm=$( printf "%02d" ${moy} )  # Change to 2-digit format

        for yr in {1956..2005}; do  # Loop over years
            # If January, calculate the annual average
            if [ ${moy} -eq 1 ]; then
                ncka -O -d time,"${yr}-01-01 00:00:0.0","${yr}-12-31 23:59:9.9" \
                ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm.nc \
                ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm.nc

            fi

            # The specific month
            ncks -O -d time,"${yr}-${mm}-01 00:00:0.0","${yr}-${mm}-31 23:59:9.9" \
            ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm.nc \
            ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_GN.nc

        done
    done
done
Chapter 7: CMIP5 Example

# Subtract the annual average from the monthly data
ncbo -O --op_typ=- ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_GN_${yr}${mm}.nc $\ldots$

done

# Average over years
ncra -O ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_GN_????${mm}_anm.nc $\ldots$

done

# Concatenate months together
ncrcat -O ${drc_out}${fld_out[var_id]}${var[var_id]}_${rlm}_${mdl}_${xpt[0]}_all-nsm_GN_??_anm.nc $\ldots$

echo Model ${mdl} done!

done

rm -f ${drc_out}${fld_out[var_id]}${var[var_id]}*historical*

# Store models as groups in the output file
ncecat -O --gag -v ${var[var_id]} $\ldots$

echo Var ${var[var_id]} done!

done

Using group feature and hyperslabs of ncbo, the script will be shortened.

#!/bin/bash

#============================================================
# Monthly cycle of each ensemble member in Greenland
# # Input file from cmb_fl_grpsh
# sn_LImon_all-mdl_all-xpt_all-nsm_199001-200512.nc
# Online: http://nco.sourceforge.net/nco.html#Monthly-Cycle
#
7.5 Regrid MODIS Data

In order to compare the results between MODIS and CMIP5 models, one usually regrids one or both datasets so that the spatial resolutions match. Here, the script illustrates how to regrid MODIS data. Key steps include:

1. Regrid using bilinear interpolation (see Section 4.1.18 [Bilinear interpolation], page 144)
2. Rename variables, dimensions and attributions using ncrename (see Section 4.11 [ncrename netCDF Renamer], page 214).

Main Script

```
#!/bin/bash
# include bi_interp.nco
```
# Example for
# - regrid (using bi_interp.nco): the spatial resolution of MODIS data
# is much finer than those of CMIP5 models. In order to compare
# the two, we can regrid MODIS data to conform to CMIP5.

# Input files (Note: the .hdf files downloaded have to be converted to .nc at
# the present):
# /modis/mcd43c3/MCD43C3.A2000049.005.2006271205532.nc

# Output files:
# /modis/mcd43c3/cesm-grid/MCD43C3.2000049.regrid.nc

# Online: http://nco.sourceforge.net/nco.html#Regrid-MODIS-Data

# Execute this script: bash rgr.sh

for fn in $( ls ${drc_in}${fld_in}${var}.*.nc ); do # Loop over files
  sfx=$( echo $fn | cut -d '/' -f 8 | cut -d '.' -f 2 ) # Part of file names

  ncap2 -O -S bi_interp.nco ${fn} ${drc_in}${fld_out}${var}.${sfx}.regrid.nc # Regrid
  ncks -O -v wsa_sw_less,bsa_sw_less ${drc_in}${fld_out}${var}.${sfx}.regrid.nc
  ncrename -O -d latn,lat -d lonn,lon -v wsa_sw_less,wsa_sw -v bsa_sw_less,bsa_sw -a missing_value,_FillValue
  ${drc_in}${fld_out}${var}.${sfx}.regrid.nc

  echo $sfx done.
done

bi_interp.nco

// Bilinear interpolation
// Included by rgr.sh
// Online: http://nco.sourceforge.net/nco.html#Regrid-MODIS-Data

defdim("latn",192); // Define new dimension: latitude
defdim("lonn",288); // Define new dimension: longitude
Chapter 7: CMIP5 Example

73.75, 75, 76.25, 77.5, 78.75, 80, 81.25, 82.5, 83.75, 85, 86.25, 87.5, 88.75, \n90, 91.25, 92.5, 93.75, 95, 96.25, 97.5, 98.75, 100, 101.25, 102.5, 103.75, \n105, 106.25, 107.5, 108.75, 110, 111.25, 112.5, 113.75, 115, 116.25, 117.5, \n118.75, 120, 121.25, 122.5, 123.75, 125, 126.25, 127.5, 128.75, 130, 131.25, \n132.5, 133.75, 135, 136.25, 137.5, 138.75, 140, 141.25, 142.5, 143.75, 145, \n146.25, 147.5, 148.75, 150, 151.25, 152.5, 153.75, 155, 156.25, 157.5, \n158.75, 160, 161.25, 162.5, 163.75, 165, 166.25, 167.5, 168.75, 170, 171.25, \n172.5, 173.75, 175, 176.25, 177.5, 178.75, 180}; // Copy of CCSM4 longitude

*out[time, latn, lonn] = 0.0; // Output structure

// Bi-linear interpolation
bsa_sw_less = bilinear_interp_wrap(bsa_sw, out, latn, lonn, lat, lon);
wsa_sw_less = bilinear_interp_wrap(wsa_sw, out, latn, lonn, lat, lon);

// Add attributions
latn@units = "degree_north";
lonn@units = "degree_east";
latn@long_name = "latitude";
lonn@long_name = "longitude";
bsa_sw_less@hdf_name = "Albedo_BSA_shortwave";
bsa_sw_less@calibrated_nt = 5;
bsa_sw_less@missing_value = 32767.0;
bsa_sw_less@units = "albedo, no units";
bsa_sw_less@long_name = "Global_Albedo_BSA_shortwave";
wsa_sw_less@hdf_name = "Albedo_WSA_shortwave";
wsa_sw_less@calibrated_nt = 5;
wsa_sw_less@missing_value = 32767.0;
wsa_sw_less@units = "albedo, no units";
wsa_sw_less@long_name = "Global_Albedo_WSA_shortwave";

7.6 Add Coordinates to MODIS Data

Main Script

#!/bin/bash

# Example for
# - regrid (using bi_interp.nco): the spatial resolution of MODIS data is much finer than those of CMIP5 models. In order to compare the two, we can regrid MODIS data to conform to CMIP5.
# - add coordinates (using coor.nco): there is no coordinate information in MODIS data. We have to add it manually now.

# Input files:
# /modis/mcd43c3/cesm-grid/MCD43C3.2000049.regrid.nc

Raw text
# Output files:
# /modis/mcd43c3/cesm-grid/MCD43C3.2000049.regrid.nc
#
# Online: http://nco.sourceforge.net/nco.html#Add-Coordinates-to-MODIS-Data
#
# Execute this script: bash add_crd.sh
#=================================================================================

var=( 'MOD10CM' ) # Variable
fld_in=( 'snc/nc/' ) # Folder of input files
drc_in='/media/grele_data/wenshan/modis/' # directory of input files
for fn in $( ls ${drc_in}${fld_in}${var}*.nc ); do # Loop over files
    sfx=$( echo ${fn} | cut -d '/' -f 8 | cut -d '.' -f 2-4 ) # Part of file names
    echo ${sfx}

    # Rename dimension names
    ncrename -d YDim_MOD_CMG_Snow_5km,lat -d XDim_MOD_CMG_Snow_5km,lon -O
    ${drc_in}${fld_in}${var}.${sfx}.nc

    # Add coordinates
    ncap2 -O -S crd.nco ${drc_in}${fld_in}${var}.${sfx}.nc

done

crd.nco

// Add coordinates to MODIS HDF data
// Included by add_crd.sh
// Online: http://nco.sourceforge.net/nco.html#Add-Coordinates-to-MODIS-Data

lon = array(0.f, 0.05, $lon) - 180;
lat = 90.f- array(0.f, 0.05, $lat);

7.7 Permute MODIS Coordinates

MODIS orders latitude data from 90°N to -90°N, and longitude from -180°E to 180°E.
However, CMIP5 orders latitude from -90°N to 90°N, and longitude from 0°E to 360°E.
This script changes the MODIS coordinates to follow the CMIP5 convention.

#!/bin/bash

#===========================================================================
## Example for
## - permute coordinates: the grid of MODIS is
## from (-180 degE, 90 degN), the left-up corner, to
## (180 degE, -90 degN), the right-low corner. However, CMIP5 is
## from (0 degE, -90 degN) to (360 degE, 90 degN). The script
## here changes the MODIS grid to CMIP5 grid.
##==========================================================================
## Input files:
## /modis/mcd43c3/cesm-grid/MCD43C3.2000049.regrid.nc
##
## Output files:
## /modis/mcd43c3/cesm-grid/MCD43C3.2000049.regrid.nc
##
## Online: http://nco.sourceforge.net/nco.html#Permute-MODIS-Coordinates
##
## Execute this script: bash pmt_crd.sh

```
## Permute coordinates
##  - Inverse lat from (90,-90) to (-90,90)
##  - Permute lon from (-180,180) to (0,360)
for fn in $( ls MCD43C3.*.nc ); do  
  sfx=$( echo ${fn} | cut -d '.' -f 1-3 )  
  echo ${sfx}

  ## Lat
  ncpdq -O -a -lat ${fn} ${fn}  

  ## Lon
  ncks -O --msa -d lon,0.0,180.0 -d lon,-180.0,-1.25 ${fn} ${fn}

  ## Add new longitude coordinates
  ncap2 -O -s 'lon=array(0.0,1.25,$lon)' ${fn} ${fn}
done
```
8 Parallel

This section will describe scripting strategies, including the use of GNU Parallel, to NCO.

```
ls *historical*.nc | parallel ncks -O -d time,"1950-01-01","2000-01-01" {} 50y/{}
```
9 CCSM Example

This chapter illustrates how to use NCO to process and analyze the results of a CCSM climate simulation.

************************************************************************
Task 0: Finding input files
************************************************************************
The CCSM model outputs files to a local directory like:

/ptmp/zender/archive/T42x1_40

Each component model has its own subdirectory, e.g.,

/ptmp/zender/archive/T42x1_40/atm
/ptmp/zender/archive/T42x1_40/cpl
/ptmp/zender/archive/T42x1_40/ice
/ptmp/zender/archive/T42x1_40/lnd
/ptmp/zender/archive/T42x1_40/ocn

within which model output is tagged with the particular model name

/ptmp/zender/archive/T42x1_40/atm/T42x1_40.cam2.h0.0001-01.nc
/ptmp/zender/archive/T42x1_40/atm/T42x1_40.cam2.h0.0001-02.nc
/ptmp/zender/archive/T42x1_40/atm/T42x1_40.cam2.h0.0001-03.nc
...
/ptmp/zender/archive/T42x1_40/atm/T42x1_40.cam2.h0.0001-12.nc
/ptmp/zender/archive/T42x1_40/atm/T42x1_40.cam2.h0.0002-01.nc
/ptmp/zender/archive/T42x1_40/atm/T42x1_40.cam2.h0.0002-02.nc
...

or

/ptmp/zender/archive/T42x1_40/lnd/T42x1_40.clm2.h0.0001-01.nc
/ptmp/zender/archive/T42x1_40/lnd/T42x1_40.clm2.h0.0001-02.nc
/ptmp/zender/archive/T42x1_40/lnd/T42x1_40.clm2.h0.0001-03.nc
...

************************************************************************
Task 1: Regional processing
************************************************************************
The first task in data processing is often creating seasonal cycles. Imagine a 100-year simulation with its 1200 monthly mean files. Our goal is to create a single file containing 12 months of data. Each month in the output file is the mean of 100 input files.

Normally, we store the "reduced" data in a smaller, local directory.
caseid='T42x1_40'
#drc_in="${DATA}/archive/${caseid}/atm"
drc_in="${DATA}/${caseid}"
drc_out="${DATA}/${caseid}"
imkdir -p ${drc_out}
cd ${drc_out}

Method 1: Assume all data in directory applies
for mth in {1..12}; do
    mm='printf "%02d" $mth'
    ncra -O -D 1 -o ${drc_out}/${caseid}_clm${mm}.nc 
        ${drc_in}/${caseid}.cam2.h0.*-${mm}.nc
done # end loop over mth

Method 2: Use shell 'globbing' to construct input filenames
for mth in {1..12}; do
    mm='printf "%02d" $mth'
    ncra -O -D 1 -o ${drc_out}/${caseid}_clm${mm}.nc 
        ${drc_in}/${caseid}.cam2.h0.00??-${mm}.nc 
        ${drc_in}/${caseid}.cam2.h0.01??-${mm}.nc
done # end loop over mth

Method 3: Construct input filename list explicitly
for mth in {1..12}; do
    mm='printf "%02d" $mth'
    fl_lst_in=''
    for yr in {1..100}; do
        yyyy='printf "%04d" $yr'
        fl_in=${caseid}.cam2.h0.$(yyyy)-$(mm).nc
        fl_lst_in=${fl_lst_in} ${caseid}.cam2.h0.$(yyyy)-$(mm).nc"
    done # end loop over yr
    ncra -O -D 1 -o ${drc_out}/${caseid}_clm${mm}.nc -p ${drc_in} 
        ${fl_lst_in}
    done # end loop over mth

Make sure the output file averages correct input files!
ncks -M prints global metadata:

    ncks -M ${drc_out}/${caseid}_clm01.nc

The input files ncra used to create the climatological monthly mean
will appear in the global attribute named 'history'.

Use ncrcat to aggregate the climatological monthly means

    ncrcat -O -D 1 \
Finally, create climatological means for reference.
The climatological time-mean:

\[
\text{ncra -O -D 1 } \text{ }
(drc\text{-out})/\text{(caseid)}\_clm\text{??.nc } (drc\text{-out})/\text{(caseid)}\_clm\text{.nc}
\]

The climatological zonal-mean:

\[
\text{ncwa -O -D 1 -a lon } \text{ }
(drc\text{-out})/\text{(caseid)}\_clm\text{.nc } (drc\text{-out})/\text{(caseid)}\_clm\_x\text{.nc}
\]

The climatological time- and spatial-mean:

\[
\text{ncwa -O -D 1 -a lon, lat, time -w gw } \text{ }
(drc\text{-out})/\text{(caseid)}\_clm\text{.nc } (drc\text{-out})/\text{(caseid)}\_clm\_xyt\text{.nc}
\]

This file contains only scalars, e.g., "global mean temperature", used for summarizing global results of a climate experiment.

Climatological monthly anomalies = Annual Cycle:
Subtract climatological mean from climatological monthly means.
Result is annual cycle, i.e., climate-mean has been removed.

\[
\text{ncbo -O -D 1 -o (drc\text{-out})/(caseid)\_clm\text{0112\_anm.nc} } \text{ }
(drc\text{-out})/\text{(caseid)}\_clm\text{0112\_anm.nc } (drc\text{-out})/\text{(caseid)}\_clm\text{xyt.nc}
\]

************************************************************************
Task 2: Correcting monthly averages
************************************************************************

The previous step approximates all months as being equal, so, e.g., February weighs slightly too much in the climatological mean.
This approximation can be removed by weighting months appropriately.
We must add the number of days per month to the monthly mean files.
First, create a shell variable \text{dpm}:

unset \text{dpm} # Days per month
declare \text{-a dpm}
dpm=(0 31 28.25 31 30 31 30 31 31 30 31 30 31) # Allows 1-based indexing

Method 1: Create \text{dpm} directly in climatological monthly means
for \text{mth} in {1..12}; do
\text{ mm='printf "$\%02d$ (mth)'}
\text{ncap2 -O -s "dpm=0.0*date+$\{dpm[$(mth)]\}" } \text{ }
(drc\text{-out})/\text{(caseid)}\_clm\$\{\text{mm}\}.nc (drc\text{-out})/\text{(caseid)}\_clm\$\{\text{mm}\}.nc
done # end loop over mth
Method 2: Create dpm by aggregating small files
for mth in {1..12}; do
    mm='printf "%02d" ${mth}';
    ncap2 -O -v -s "dpm=${dpm[${mth}]}" ~/nco/data/in.nc \
        ${drc_out}/foo_${mm}.nc
done # end loop over mth
ncat -O -D 1 -p ${drc_out} -n 12,2,2 foo_${mm}.nc foo.nc
ncrename -O -D 1 -d record,time ${drc_out}/foo.nc
ncatted -O -h \
    -a long_name,dpm,o,c,"Days per month" \ 
    -a units,dpm,o,c,"days" \ 
    ${drc_out}/${caseid}_clm_0112.nc
ncks -A -v dpm ${drc_out}/foo.nc ${drc_out}/${caseid}_clm_0112.nc

Method 3: Create small netCDF file using ncgen

cat > foo.cdl << 'EOF'
netcdf foo {
  dimensions:
    time=unlimited;

  variables:
    float dpm(time);
    dpm:long_name="Days per month";
    dpm:units="days";

  data:
    dpm=31,28.25,31,30,31,30,31,31,30,31,30,31;
}
EOF
ncgen -b -o foo.nc foo.cdl
ncks -A -v dpm ${drc_out}/foo.nc ${drc_out}/${caseid}_clm_0112.nc

Another way to get correct monthly weighting is to average daily output files, if available.

************************************************************
Task 3: Regional processing
************************************************************
Let’s say you are interested in examining the California region.
Hyperslab your dataset to isolate the appropriate latitude/longitudes.

ncks -O -D 1 -d lat,30.0,37.0 -d lon,240.0,270.0 \ 
    ${drc_out}/${caseid}_clm_0112.nc \ 
    ${drc_out}/${caseid}_clm_0112_Cal.nc

The dataset is now much smaller!
To examine particular metrics.
Task 4: Accessing data stored remotely

**OPeNDAP server examples:**

**UCI DAP servers:**

```bash
ncks -M -p http://dust.ess.uci.edu/cgi-bin/dods/nph-dods/dodsdata in.nc
ncrcat -O -C -D 3 -p http://dust.ess.uci.edu/cgi-bin/dods/nph-dods/dodsdata -l /tmp in.nc in.nc ~/foo.nc
```

**Unidata DAP servers:**

```bash
crfs -M -p http://thredds-test.ucar.edu/thredds/dodsC/testdods in.nc
ncrcat -O -C -D 3 -p http://thredds-test.ucar.edu/thredds/dodsC/testdods -l /tmp in.nc in.nc ~/foo.nc
```

**NOAA DAP servers:**

```bash
```

**LLNL PCMDI IPCC OPeNDAP Data Portal:**

```bash
ncks -M -p http://username:password@esgcet.llnl.gov/cgi-bin/dap-cgi.py/ipcc4/sresa1b/ncar_ccsm3_0 pcmdi.ipcc4.ncar_ccsm3_0.sresa1b.run1.atm.mo.xml
```

**Earth System Grid (ESG):** [http://www.earthsystemgrid.org](http://www.earthsystemgrid.org)

```bash
caseid='b30.025.ES01'
CCSM3.0 1% increasing CO2 run, T42_gx1v3, 200 years starting in year 400
Atmospheric post-processed data, monthly averages, e.g.,
/data/zender/tmp/b30.025.ES01.cam2.h0.TREFHT.0400-01_cat_0449-12.nc
/data/zender/tmp/b30.025.ES01.cam2.h0.TREFHT.0400-01_cat_0599-12.nc
```

ESG supports password-protected FTP access by registered users
NCO uses the `.netrc` file, if present, for password-protected FTP access
Syntax for accessing single file is, e.g.,
```bash
```

# Average surface air temperature `tas` for SRESA1B scenario
# This loop is illustrative and will not work until NCO correctly
# translates `*` to FTP `mget` all remote files
for var in 'tas'; do
  for scn in 'sresa1b'; do
    for mdl in 'cccma_cgcm3_1 cccma_cgcm3_1_t63 cnrm_cm3 csiro_mk3_0 gfdl_cm2_0 gfdl_cm2_1 giss_aom giss_model_e_h giss_model_e_r'
    ```
iap_fgoals1_0_g inmcm3_0 ips1_cm4 miroc3_2_hires miroc3_2_medres \ miub_echo_g mpi_echam5 mri_cgm2_3_2a ncar_ccsm3_0 ncar_pcm1 \ ukmo_hadcm3 ukmo_hadgem1'; do
for run in 1; do
    ncks -R -O -D 3 -p ftp://climate.llnl.gov/${scn}/atm/mo/${var}/${mdl}/run${run} -l ${DATA}/${scn}/atm/mo/${var}/${mdl}/run${run} '*' ${scn}_${mdl}_${run}_${var}_${yyyymm}_${yyyymm}.nc
done # end loop over run
done # end loop over mdl
done # end loop over scn
done # end loop over var

cd sresa1b/atm/mo/tas/ukmo_hadcm3/run1/
ncks -H -m -v lat,lon,lat_bnds,lon_bnds -M tas_A1.nc | m bds -x 096 -y 073 -m 33 -o ${DATA}/data/dst_3.75x2.5.nc # ukmo_hadcm3
ncview ${DATA}/data/dst_3.75x2.5.nc

# msk_rgn is California mask on ukmo_hadcm3 grid
# area is correct area weight on ukmo_hadcm3 grid
ncks -A -v area,msk_rgn ${DATA}/data/dst_3.75x2.5.nc \ ${DATA}/sresa1b/atm/mo/tas/ukmo_hadcm3/run1/area_msk_ukmo_hadcm3.nc

Template for standardized data:
${scn}_${mdl}_${run}_${var}_${yyyymm}_${yyyymm}.nc
e.g., raw data
${DATA}/sresa1b/atm/mo/tas/ukmo_hadcm3/run1/tas_A1.nc
becomes standardized data
Level 0: raw from IPCC site--no changes except for name
Make symbolic link name match raw data
Template: ${scn}_${mdl}_${run}_${var}_${yyyymm}_${yyyymm}.nc

ln -s -f tas_A1.nc sresa1b_ukmo_hadcm3_run1_tas_200101_209911.nc
area_msk_ukmo_hadcm3.nc

/ncview sresa1b_ukmo_hadcm3_run1_200101_209911.nc

Level I: Add all variables (not standardized in time)
to file containing msk_rgn and area
Template: ${scn}_${mdl}_${run}_${yyyymm}_${yyyymm}.nc

/bin/cp area_msk_ukmo_hadcm3.nc sresa1b_ukmo_hadcm3_run1_200101_209911.nc
ncks -A -v area,msk_rgn sresa1b_ukmo_hadcm3_run1_200101_209911.nc \ sresa1b_ukmo_hadcm3_run1_200101_209911.nc

If already have file then:
mv sresa1b_ukmo_hadcm3_run1_200101_209911.nc foo.nc
/ncview sresa1b_ukmo_hadcm3_run1_200101_209911.nc
ncks -A -v tas,pr foo.nc sresa1b_ukmo_hadcm3_run1_200101_209911.nc

Level II: Correct # years, months
Template: ${scn}_${mdl}_${run}_${var}_${yyyymm}_${yyyymm}.nc

ncks -d time,....... file1.nc file2.nc
ncrcat file2.nc file3.nc sresa1b_ukmo_hadcm3_run1_200001_209912.nc

Level III: Many derived products from level II, e.g.,

A. Global mean timeseries
ncwa -w area -a lat,lon \
sresa1b_ukmo_hadcm3_run1_200001_209912.nc \
sresa1b_ukmo_hadcm3_run1_200001_209912_xy.nc

B. California average timeseries
ncwa -m msk_rgn -w area -a lat,lon \
sresa1b_ukmo_hadcm3_run1_200001_209912.nc \
sresa1b_ukmo_hadcm3_run1_200001_209912_xy_Cal.nc
10 References

General Index

""
" (double quote) ................................ 171

#
#include ........................................... 117

$
$ (wildcard character) ......................... 44

%
% (modulo) ....................................... 163

,:
, (end quote) .................................... 171

*
* (filename expansion) ....................... 174
* (multiplication) .............................. 44
* (wildcard character) ...................... 44

+
+ (addition) ....................................... 174
+ (addition) ..................................... 163
+ (addition) ...................................... 44

-
- (subtraction) .................................. 174
--3 ............................................. 163
--4 ............................................. 35
--5 ............................................. 35
--6 ............................................. 35
--64bit ......................................... 35
--7 ............................................. 35
--abc ........................................... 189
--alphabetize .................................. 189
--apn ............................................. 16, 108, 198
--append ....................................... 16, 108, 198
--auxiliary .................................... 64
--auxiliary lon_min, lon_max, lat_min, lat_max
--auxiliary ...................................... 64
--bfr_sz_hnt .................................. 90
--binary ......................................... 190
--bnr ............................................ 190
--cdl ............................................. 190
--cell_methods .................................. 111
--chunk_byte .................................... 72
--chunk_dimension ................................ 72
--chunk_map ..................................... 72
--chunk_min ...................................... 72
--chunk_policy ................................... 72
--chunk_scalar ................................... 72
--clm_mth ....................................... 111
--cnk_byt ....................................... 72
--cnk_dmn ....................................... 72
--cnk_map ....................................... 72
--cnk_mth ....................................... 75
--cnk_min ....................................... 72
--cnk_plc ....................................... 72
--cnk_scl ....................................... 72
--coords ......................................... 45, 111
--crd ............................................. 45, 111
--create_ram .................................... 16, 90
--data ............................................ 191
--dbg_lvl debug-level ........................... 191
--dbl ............................................. 102
--debug-level debug-level ..................... 13, 22, 27
--deflate ........................................ 88
--df1_lvl ........................................ 88
--dimension .................................... 55, 60, 63, 66
--dimension dim, [min], [max], [stride], [subcycle] 59
--dimension dim, [min], [max], [stride] ........ 57
--dimension dim, [min], [max], [stride] ... 55, 60, 63, 66
--diskless_all .................................. 55, 60, 63, 66
--dmn_dim, [min], [max], [stride], subcycle 59
--dmn_dim, [min], [max], subcycle 59
--dmn_dim, [min], [max], [stride] ........ 57
--dmn_dim, [min], [max], [stride] ... 55, 60, 63, 66
--ensemble_file ................................ 179
--ensemble_group ............................... 179
--ensemble_suffix .............................. 179
--exclude ....................................... 40, 198
--file_format ................................... 35
--file_list ..................................... 109
--fix_rec_dmn all ................................ 39
--fix_rec_dmn ................................... 190
--fl_bnr ......................................... 190
--fl_fmt ......................................... 35
--fl_lst ......................................... 109
--fortran ....................................... 54
--gag ............................................. 183, 233
--glob_std_spp .................................. 183
--gpl gpe_desc .................................. 46
--group grp ..................................... 40
--grp grp ........................................ 40
--grp_str_var_xcl .............................. 40
--gxx .......................................... 40
--hdf_unpack .................................. 92
--hdf_upk .................................... 92
'--hdf4' ...................................... 11
--hdn .......................................... 190
--hdr_pad hdr_pad .................................. 25
--header_pad hdr_pad .................................. 25
--hidden ........................................ 190
--hieronymus ................................... 191
--history ........................................ 108
--hst ........................................... 108
--intersection .................................. 42
--lcl_output-path .............................. 30
--local_output-path ............................ 30
--map cnk_map .................................. 75
--map pck_map .................................. 203
--mask-value mask_val ......................... 220
--mask-variable mask_var ....................... 219
--mask_comparator mask_comp ................... 220
--mask_condition mask_cond ..................... 219, 220
--mask_value mask_val ........................... 220
--mask_variable mask_var ....................... 219
--mds_dgs .................................... 88
--mds_digeat ................................... 88
--mds_write_attribute .......................... 88
--mds_wrt_att .................................. 88
--metadata ...................................... 193
--Metadata ..................................... 192
--mk_rec_dmn dim ................................ 184, 191
--mrd .......................................... 70
--mro .......................................... 59
--msa .......................................... 60
--msa_user_order ................................ 60
--msa_user_rdr .................................. 60
--msk_cmp_typ mask_comp ....................... 220
--msk_cnd mask_cond ............................ 219
--msk_cnd_sng mask_cond ....................... 220
--msk_mm mask_var ................................ 219
--mask_val mask_val ............................ 220
--mask_var mask_var ............................ 219
--mtd .......................................... 193
--Mtd .......................................... 192
--multiple_record_dimensions ................. 70
--ncml ........................................ 195
--netcdf4 ...................................... 35
--nintap loop ................................... 28
--no-blank ...................................... 193
--no-coords .................................... 45, 111
--no-crd ....................................... 45, 111
--no-blank ...................................... 193, 199
--no_cell_methods .............................. 111
--no_cll_mth ................................... 111
--no_dan_var_nm ................................ 194, 199
--no_nm_prn .................................... 194, 199
--no_rec_dmn dim ................................ 190
--no_tmp_fl ..................................... 16, 92
--noblank ...................................... 193
--nsm_fi ........................................ 179
--nsm_gr ........................................ 179
--nsm_sfx ....................................... 179
--nax ........................................... 42
'--comp_num_threads thr_nbr' .................... 25
--op_rlt mask_comp .............................. 220
--op_typ op_typ .................................. 94, 174
--open_ram ...................................... 16, 22, 90
--operation op_typ .............................. 94, 174
--orphan_dimensions ........................... 194
--output fl_out .................................. 30
--overwrite ..................................... 16, 108
--ovr ........................................... 16, 108
--pack_policy pck_plc .......................... 202
--path input-path ................................ 28, 30
--pck_map pck_map ................................ 203
--pck_plc pck_plc ................................ 202
--precision-preserving_compression ............ 78
--print ......................................... 194
--prn ............................................ 194
--pth input-path ................................ 28, 30
--prn_fnc_tbl ................................... 165
--pseudonym .................................... 8
--quiet ......................................... 194, 199
--rad ............................................ 194
--ram_all ........................................ 22, 90
--rcd_all ........................................ 182
--rec_apn ....................................... 58
--record_append .................................. 58
--retain ......................................... 35
--retain_all_dimensions ........................ 194
--revision ...................................... 13, 112
--rph_dmn ...................................... 194
--rph_dbl ...................................... 102
--rph_flt ...................................... 102
--rtn ........................................... 35
--scr ............................................ 194
--script ......................................... 116
--script-file .................................... 116
--secret ......................................... 194
--sgn_fmat ...................................... 194, 199
--spt ........................................... 116
--string ......................................... 194, 199
--thr_nbr thr_nbr ............................... 25
--threads thr_nbr ................................ 25
'--ulm nm ulm nm' ................................ 182
--un . ........................................... 40, 42
--units .......................................... 194
--unn ........................................... 40, 42
--unpack ........................................ 203
--upk ........................................... 203
--var_val ....................................... 194, 199
--variable var .................................. 40, 198
abs
absolute value
acos
acosh
add
add_offset
add_offset
adding data
addition
aggregation
Aleksandar Jelenak
Alexander Hansen
alias
alias
'al'
alphabetization
alphabetize output
alternate invocations
anchor
anchoring
ancillary variables convention
ancillary_variables attribute
Andrea Cimatoribus
Andrew Wittenberg
annual average
annual average from daily data
annual average from monthly data
anomalies
ANSI
ANSI C
appending data
appending to files
appending variables
AR4
arccosine function
arcsine function
arctangent function
area
area-averaging
arithmetic operators
arithmetic processor
ARM conventions
ARM conventions
array
array function
array indexing
array storage
array syntax
arrays
arrival value
ASCII
asin
asinh
asort
assignment statement
asynchronous file access
### General Index

- **atan** ........................................... 163
- **atanh** ......................................... 163
- attributes ....................................... 126
- attribute names ................................ 168, 214
- attribute propagation ......................... 126
- attribute syntax ................................. 117
- attribute, units ................................. 66
- attributes ....................................... 168
- attributes, appending ........................... 169
- attributes, creating ............................ 169
- attributes, deleting ............................. 169
- attributes, editing ............................. 169, 240, 250
- attributes, global .............................. 169, 240, 250
- attributes, modifying ......................... 169, 240, 250
- attributes, overwriting ....................... 169, 240, 250
- attributes capitalize .......................... 125
- `autoconf` ......................................... 14
- `autoconversion` ................................ 39
- `automagic` ....................................... 7, 19
- `automatic type conversion` ................... 100, 164
- auxiliary coordinates ......................... 111
- average ......................................... 94, 220, 229, 230, 237, 240, 247
- averaging data .................................. 71, 179, 210, 219
- `avg()` ............................................ 94
- `avgsqr` .......................................... 131
- `avgsqr()` ....................................... 94

### B

- Barry deFreese .................................... 226
- Bas Conwenberg .................................. 226
- `base_time` ....................................... 112
- `bash` ............................................. 45, 196
- Bash shell ........................................ 175, 196
- Bash Shell ........................................ 178
- batch mode ...................................... 108
- `beer` ............................................. 107
- benchmarks ...................................... 27
- Bessel function .................................. 147
- bilinear interpolation ........................... 250
- Bill Kockik ....................................... 226
- binary format .................................... 190
- binary operations ................................ 23, 174
- binary Operators ................................ 119
- bitmask .......................................... 80
- blank ............................................ 193
- `blocksize` ...................................... 73
- bounds attribute ................................ 80, 111, 200
- bounds convention ............................... 111
- Bourne Shell ..................................... 57, 178
- Brian Mays ....................................... 226
- broadcasting ..................................... 177
- broadcasting groups ............................. 10, 176, 177
- broadcasting variables ......................... 118, 175, 187, 219
- BSD ............................................... 27
- `Buffer sizes` .................................... 90
- buffering ........................................ 24
- bugs, reporting .................................. 13
- Burrows-Wheeler algorithm ...................... 85
- `byte()` ......................................... 133
- `bzip2` .......................................... 85

### C

- **C** index convention .......................... 54
- C language ........................................ 7, 71, 101, 102, 117, 194
- C Shell ............................................ 57, 178
- c++ .............................................. 6
- C++ ............................................... 6
- `C_format` ....................................... 24
- `C99` ............................................. 6, 7
- `C99` ............................................. 7, 8, 80
- `CAM` ............................................ 87
- `CAM3` .......................................... 105
- `cc` ................................................ 6
- `CC` .............................................. 6
- CCM Processor ................................... 28, 210, 212
- CCM .............................................. 227
- CCSM ............................................. 259
- CCSM conventions ................................ 109
- `CDL` ............................................. 190
- `cell` ............................................. 163
- `ceiling function` ............................... 163
- cell methods convention ....................... 111
- cell-based grids .................................. 64
- `cell_methods` ................................... 111
- CF compliance checker ........................... 51
- CF conventions ................................. 45, 64, 68, 80, 109, 176, 200
- `cfchecker` ...................................... 51
- change_miss() ................................... 130
- `char()` ......................................... 133
- characters, special ............................. 171
- Charlie Zender .................................. 1, 225
- chocolate ........................................ 225
- Chris Barker ..................................... 75
- chunking ......................................... 10, 72, 193
- chunking map .................................... 73, 75
- chunking policy .................................. 73
- chunksize ........................................ 73
- citation .......................................... 227
- `clang` .......................................... 6
- CLASSIC files ................................... 35
- client-server ..................................... 32
- Climate and Forecast Metadata Convention ... 68
- climate model ..................................... 15, 18, 29, 183, 222
- clipping operators .............................. 120
- CMIP ............................................. 179, 182
- CMIP5 ............................................. 231
- `cnk_all` ........................................ 73
- `cnk_dam` ........................................ 75
- `cnk_g2d` ......................................... 73
- `cnk_g3d` ......................................... 73
- `cnk_lfp` ......................................... 75
- `cnk_map` ........................................ 75
- `cnk_nc4` ........................................ 75
D

daily data ......................................................... 229, 230
daniel baumann ................................................. 226
daniel wang ...................................................... 225
DAP ................................................................. 32

data access protocol ........................................... 32

data safety ...................................................... 15, 214
data, missing ................................................... 71, 168
data .......................... 109
datesec .............................................................. 109
dbg_lvl .............................................................. 13, 22, 26
DDRA ............................................................... 227
Debian ............................................................. 12
depend ................................................................ 13, 22
debbuging ......................................................... 13, 22, 26
DEC ................................................................. 6
Decimal Significant Digits ..................................... 79
defdim() .............................................................. 121
defining dimensions in ncap2 .............................. 121
DEFFLATE ............................................................ 81
deflation ............................................................ 10, 88, 193
degenerate dimension ... 75, 98, 163, 176, 184, 185,
209, 210, 219, 222
delete (groups) .................................................... 46
deflete_miss() ..................................................... 130
demotion ............................................................ 100
 Dennis Heimbigner ............................................. 225
derived fields ...................................................... 116
digest ................................................................. 88
digital ................................................................. 6
dimension limits .................................................. 55
dimension names .................................................. 214
dimension order ................................................... 240
dimensions, growing ........................................... 161
disaggregate ...................................................... 49
disjoint files ....................................................... 17
diskless files ....................................................... 90
dismember ........................................................... 49
Distributed Data Reduction & Analysis .............. 227
Distributed Oceanographic Data System ........... 32
divide ................................................................. 174
dividing data ...................................................... 174
division .............................................................. 163
dimension names .................................................. 214
dimension order ................................................... 240
dimensions, growing ........................................... 161
disaggregate ...................................................... 49
disjoint files ....................................................... 17
diskless files ....................................................... 90
dismember ........................................................... 49
DODS ............................................................... 32, 35
DODS_ROOT .......................................................... 32
dot product ....................................................... 221
double() .............................................................. 133
double-precision ................................................ 165
DSB ................................................................. 79
dsort ................................................................. 137
duration ............................................................ 59
dynamic linking .................................................. 9
E

Ed Hartnett ....................................................... 225
Ed Hill .............................................................. 226
eddy covariance ................................................ 162
editing attributes .............................................. 168
egrep ............................................................... 43
Elliptic integrals ............................................... 147
ensemble .......................................................... 18, 179
ensemble average ............................................. 179
ensemble concatenation ................................... 182
crf ................................................................. 163
erfc ............................................. 163
eric Blake ..................................... 226
error function .................................. 163
error tolerance ................................ 15
Etienne Tourigny .............................. 227
exclusion ...................................... 40, 198, 200
execution time ................................. 9, 16, 24, 235, 72, 168, 215
exp ............................................ 163
exponent ...................................... 106
exponentiation ................................ 163
exponentiation function ...................... 163
expressions ................................... 117
extended file format ........................... 37
extended regular expressions ............... 20, 43, 162, 169, 172
extraction .................................... 40, 198, 200

F
f90 .............................................. 8
features, requesting .......................... 13
File buffers .................................. 90
file combination ............................... 231
file deletion .................................. 35
file multiplication ........................... 187
file removal ................................... 35
file retention .................................. 35
files, multiple ................................ 29
files, numerous input .......................... 19
Filipe Fernandes ............................... 226
filters .......................................... 196
findgen-equivalent ............................ 122
fix record dimension .......................... 182, 184, 191
fixed dimension ............................... 182, 184, 190, 191
flags ........................................... 162
flatten (groups) ............................... 46
flattening .................................... 39
float .......................................... 165
float() ......................................... 133
floor ........................................... 163
floor function .................................. 163
‘flt_byt’ ....................................... 203
‘flt_sht’ ....................................... 203
for() ........................................... 136
force append .................................. 108
force overwrite ............................... 108
foreword ...................................... 1
Fortran ......................................... 102, 210, 212
Fortran index convention ..................... 54
FORTRAN_format ............................... 24
Francesco Lovergine .......................... 226
ftp ............................................. 8, 30
FTP ............................................ 35
funding ......................................... 227

G
g++ .............................................. 6
g++ .............................................. 8
‘g2d’ ........................................... 73
‘g3d’ ........................................... 73
GAG ............................................... 182
gamma .......................................... 7, 163
gamma function ................................. 146, 163
Gary Strand .................................... 226
Gaussian weights ............................... 222
Gavin Burris ................................... 226
Gayathri Venkitachalam ....................... 226
gcc ............................................... 6
gcc .............................................. 8
GCM ............................................. 15, 87, 105
geographical weight ............................ 247
George Shapavalov ............................ 226
George Shapovalov ............................. 226
George White .................................. 226
get_miss() ...................................... 130
gethostname .................................... 8
getopt ......................................... 27
getopt.h ........................................ 27
getopt_long .................................... 27
getuid .......................................... 8
Glenn Davis .................................... 225
global attribute ............................... 172, 215
globbing ....................................... 20, 28, 45, 162, 175, 210, 212
GNU ............................................. 27, 43
gnu-win32 ..................................... 8
GNU/Linux ..................................... 21
GNUmakefile ................................... 8
God ............................................. 69
GODAD .......................................... 231
grep -E ......................................... 43
group aggregation ............................. 182, 233
group attributes ............................... 217
group names ................................... 214
group path ..................................... 46
group, aggregation ............................ 245
group, anomaly ................................ 245
group, dimension permutation ............... 245
group, spatial averaging ...................... 245
group, standard deviation .................... 245
group, temporal averaging .................... 245
groups .......................................... 10, 172
groups, averaging ............................. 239
groups, creating ................................ 233
groups, moving ................................ 48
groups, renaming .............................. 48
growing dimensions ......................... 161
GSL ............................................... 7, 146, 154
gsl_sf_bessel_In ................................ 147
gsl_sf_gamma .................................. 146
gsl_sf_legendre_P1 ............................ 148
gunzip ......................................... 81
gw .............................................. 109, 222
gzip ............................................ 78, 81
<table>
<thead>
<tr>
<th>H</th>
<th>117</th>
</tr>
</thead>
<tbody>
<tr>
<td>H14CF</td>
<td>12</td>
</tr>
<tr>
<td>h4tonccf</td>
<td>12</td>
</tr>
<tr>
<td>Harry Mangalam</td>
<td>225</td>
</tr>
<tr>
<td>hash</td>
<td>88</td>
</tr>
<tr>
<td>HDF</td>
<td>9, 35, 190, 227</td>
</tr>
<tr>
<td>HDF unpacking</td>
<td>93</td>
</tr>
<tr>
<td>hdf_name</td>
<td>12</td>
</tr>
<tr>
<td>HDF4</td>
<td>11, 190</td>
</tr>
<tr>
<td>HDF4_UNKNOWN</td>
<td>11</td>
</tr>
<tr>
<td>HDF5</td>
<td>10, 81</td>
</tr>
<tr>
<td>hdp</td>
<td>190</td>
</tr>
<tr>
<td>help</td>
<td>13</td>
</tr>
<tr>
<td>Henry Butowsky</td>
<td>225</td>
</tr>
<tr>
<td>‘hgh_byt’</td>
<td>203</td>
</tr>
<tr>
<td>‘hgh_sht’</td>
<td>203</td>
</tr>
<tr>
<td>hidden attributes</td>
<td>190</td>
</tr>
<tr>
<td>hidden features</td>
<td>194</td>
</tr>
<tr>
<td>Hierarchical Data Format</td>
<td>9</td>
</tr>
<tr>
<td>history</td>
<td>21, 30, 108, 112, 168, 198</td>
</tr>
<tr>
<td>hncgen</td>
<td>190</td>
</tr>
<tr>
<td>HP</td>
<td>6</td>
</tr>
<tr>
<td>HPSS</td>
<td>31</td>
</tr>
<tr>
<td>hsi</td>
<td>31</td>
</tr>
<tr>
<td>HTML</td>
<td>5</td>
</tr>
<tr>
<td>HTTP protocol</td>
<td>32</td>
</tr>
<tr>
<td>Huffman coding</td>
<td>78</td>
</tr>
<tr>
<td>hyai</td>
<td>109</td>
</tr>
<tr>
<td>hyam</td>
<td>109</td>
</tr>
<tr>
<td>hybi</td>
<td>109</td>
</tr>
<tr>
<td>hybi</td>
<td>109</td>
</tr>
<tr>
<td>hybrid coordinate system</td>
<td>122</td>
</tr>
<tr>
<td>hyperbolic arccosine function</td>
<td>163</td>
</tr>
<tr>
<td>hyperbolic arcsine function</td>
<td>163</td>
</tr>
<tr>
<td>hyperbolic arctangent function</td>
<td>163</td>
</tr>
<tr>
<td>hyperbolic cosine function</td>
<td>163</td>
</tr>
<tr>
<td>hyperbolic sine function</td>
<td>163</td>
</tr>
<tr>
<td>hyperbolic tangent</td>
<td>163</td>
</tr>
<tr>
<td>hyperslab</td>
<td>55, 73, 179, 182, 210, 212, 219</td>
</tr>
<tr>
<td>hyperslabs</td>
<td>123</td>
</tr>
<tr>
<td>I</td>
<td></td>
</tr>
<tr>
<td>I/O</td>
<td>34, 54, 61</td>
</tr>
<tr>
<td>I/O block size</td>
<td>90</td>
</tr>
<tr>
<td>I1SN</td>
<td>25</td>
</tr>
<tr>
<td>IBM</td>
<td>6</td>
</tr>
<tr>
<td>icc</td>
<td>6</td>
</tr>
<tr>
<td>ID Quoting</td>
<td>166</td>
</tr>
<tr>
<td>IDL</td>
<td>15</td>
</tr>
<tr>
<td>IEEE</td>
<td>80, 101</td>
</tr>
<tr>
<td>IEEE 754</td>
<td>80</td>
</tr>
<tr>
<td>IEEE NaN</td>
<td>171</td>
</tr>
<tr>
<td>if()</td>
<td>127</td>
</tr>
<tr>
<td>ilimit</td>
<td>22</td>
</tr>
<tr>
<td>illegal names</td>
<td>12</td>
</tr>
<tr>
<td>implicit conversion</td>
<td>102</td>
</tr>
<tr>
<td>include</td>
<td>137</td>
</tr>
<tr>
<td>including files</td>
<td></td>
</tr>
<tr>
<td>index conversion</td>
<td>54</td>
</tr>
<tr>
<td>indgen-equivalent</td>
<td>122</td>
</tr>
<tr>
<td>inexact conversion</td>
<td>164</td>
</tr>
<tr>
<td>Info</td>
<td>5</td>
</tr>
<tr>
<td>input files</td>
<td>20, 28, 30</td>
</tr>
<tr>
<td>input-path</td>
<td>28, 32</td>
</tr>
<tr>
<td>installation</td>
<td>6, 14</td>
</tr>
<tr>
<td>int()</td>
<td>133</td>
</tr>
<tr>
<td>int64()</td>
<td>133</td>
</tr>
<tr>
<td>integration</td>
<td>221</td>
</tr>
<tr>
<td>integrity</td>
<td>88</td>
</tr>
<tr>
<td>Intel</td>
<td>6</td>
</tr>
<tr>
<td>Internationalization</td>
<td>25</td>
</tr>
<tr>
<td>interoperability</td>
<td>93</td>
</tr>
<tr>
<td>interpolation</td>
<td>185, 250</td>
</tr>
<tr>
<td>intersection</td>
<td>40, 42</td>
</tr>
<tr>
<td>introduction</td>
<td>5</td>
</tr>
<tr>
<td>invert_map</td>
<td>137</td>
</tr>
<tr>
<td>IPCC</td>
<td>179, 227</td>
</tr>
<tr>
<td>irregular grids</td>
<td>142</td>
</tr>
<tr>
<td>ISO</td>
<td>6</td>
</tr>
<tr>
<td>J</td>
<td></td>
</tr>
<tr>
<td>James Gallagher</td>
<td>225</td>
</tr>
<tr>
<td>Jim Edwards</td>
<td>226</td>
</tr>
<tr>
<td>Joe Hamman</td>
<td>225</td>
</tr>
<tr>
<td>John Caron</td>
<td>225</td>
</tr>
<tr>
<td>Juliana Rew</td>
<td>226</td>
</tr>
<tr>
<td>K</td>
<td></td>
</tr>
<tr>
<td>Karen Schuchardt</td>
<td>226</td>
</tr>
<tr>
<td>Keith Lindsay</td>
<td>226</td>
</tr>
<tr>
<td>kitchen sink</td>
<td>188</td>
</tr>
<tr>
<td>Kyle Wilcox</td>
<td>226</td>
</tr>
<tr>
<td>L</td>
<td></td>
</tr>
<tr>
<td>L1ON</td>
<td>25</td>
</tr>
<tr>
<td>large datasets</td>
<td>21, 26</td>
</tr>
<tr>
<td>Large File Support</td>
<td>21, 40</td>
</tr>
<tr>
<td>lat_bnds</td>
<td>109</td>
</tr>
<tr>
<td>LD_LIBRARY_PATH</td>
<td>9</td>
</tr>
<tr>
<td>Least Significant Digit</td>
<td>79</td>
</tr>
<tr>
<td>least_significant_digit</td>
<td>82</td>
</tr>
<tr>
<td>left hand casting</td>
<td>23, 122</td>
</tr>
<tr>
<td>Legendre polynomial</td>
<td>148</td>
</tr>
<tr>
<td>Lempel-Ziv deflation</td>
<td>88</td>
</tr>
<tr>
<td>Len Makin</td>
<td>226</td>
</tr>
<tr>
<td>lexer</td>
<td>116</td>
</tr>
<tr>
<td>‘ifg’</td>
<td>75</td>
</tr>
<tr>
<td>LFS</td>
<td>21, 40</td>
</tr>
<tr>
<td>LHS</td>
<td>122</td>
</tr>
<tr>
<td>libmco</td>
<td>6</td>
</tr>
<tr>
<td>libraries</td>
<td>9</td>
</tr>
<tr>
<td>linkers</td>
<td>30</td>
</tr>
<tr>
<td>Page</td>
<td>Entry</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>165</td>
<td>Linux</td>
</tr>
<tr>
<td>6</td>
<td>LLVM</td>
</tr>
<tr>
<td>163</td>
<td>ln</td>
</tr>
<tr>
<td>175</td>
<td>ln -s</td>
</tr>
<tr>
<td>163</td>
<td>log</td>
</tr>
<tr>
<td>163</td>
<td>log10</td>
</tr>
<tr>
<td>163</td>
<td>logarithm, base 10</td>
</tr>
<tr>
<td>163</td>
<td>logarithm, natural</td>
</tr>
<tr>
<td>109</td>
<td>lon_bnds</td>
</tr>
<tr>
<td>165</td>
<td>long double</td>
</tr>
<tr>
<td>27, 208</td>
<td>long options</td>
</tr>
<tr>
<td>237</td>
<td>long-term average</td>
</tr>
<tr>
<td>63</td>
<td>longitude</td>
</tr>
<tr>
<td>226</td>
<td>Lori Sentman</td>
</tr>
<tr>
<td>78</td>
<td>lossy compression</td>
</tr>
<tr>
<td>101</td>
<td>lrint()</td>
</tr>
<tr>
<td>101</td>
<td>lround()</td>
</tr>
<tr>
<td>79</td>
<td>LSD</td>
</tr>
<tr>
<td>226</td>
<td>Luk Claes</td>
</tr>
<tr>
<td>94</td>
<td>mabs</td>
</tr>
<tr>
<td>131</td>
<td>mabs()</td>
</tr>
<tr>
<td>6</td>
<td>Macintosh</td>
</tr>
<tr>
<td>6, 8, 33</td>
<td>Makefile</td>
</tr>
<tr>
<td>23</td>
<td>malloc()</td>
</tr>
<tr>
<td>106</td>
<td>mantissa</td>
</tr>
<tr>
<td>100</td>
<td>manual type conversion</td>
</tr>
<tr>
<td>75</td>
<td>’map_dm’</td>
</tr>
<tr>
<td>75</td>
<td>’map_lfp’</td>
</tr>
<tr>
<td>75</td>
<td>’map_nc4’</td>
</tr>
<tr>
<td>75</td>
<td>’map_prd’</td>
</tr>
<tr>
<td>75</td>
<td>’map_rdi’</td>
</tr>
<tr>
<td>75</td>
<td>’map_rew’</td>
</tr>
<tr>
<td>75</td>
<td>’map_scl’</td>
</tr>
<tr>
<td>75</td>
<td>’map_xst’</td>
</tr>
<tr>
<td>226</td>
<td>Mark Flanner</td>
</tr>
<tr>
<td>226</td>
<td>Markus Liebig</td>
</tr>
<tr>
<td>226</td>
<td>Martin Dix</td>
</tr>
<tr>
<td>227</td>
<td>Martin Otte</td>
</tr>
<tr>
<td>226</td>
<td>Martin Schmidt</td>
</tr>
<tr>
<td>51</td>
<td>Martin Schultz</td>
</tr>
<tr>
<td>142, 162</td>
<td>mask</td>
</tr>
<tr>
<td>220, 223</td>
<td>mask condition</td>
</tr>
<tr>
<td>219</td>
<td>masked average</td>
</tr>
<tr>
<td>30</td>
<td>Mass Store System</td>
</tr>
<tr>
<td>226</td>
<td>Matej Vela</td>
</tr>
<tr>
<td>163</td>
<td>mathematical functions</td>
</tr>
<tr>
<td>15</td>
<td>Matlab</td>
</tr>
<tr>
<td>94</td>
<td>max</td>
</tr>
<tr>
<td>131</td>
<td>max()</td>
</tr>
<tr>
<td>94</td>
<td>maximum</td>
</tr>
<tr>
<td>88</td>
<td>MD5 digest</td>
</tr>
<tr>
<td>94</td>
<td>mean</td>
</tr>
<tr>
<td>94</td>
<td>mebs</td>
</tr>
<tr>
<td>131</td>
<td>mebs()</td>
</tr>
<tr>
<td>22, 90</td>
<td>memory available</td>
</tr>
</tbody>
</table>

### M
<table>
<thead>
<tr>
<th>Page</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>mabs</td>
</tr>
<tr>
<td>131</td>
<td>mabs()</td>
</tr>
<tr>
<td>6</td>
<td>Macintosh</td>
</tr>
<tr>
<td>6, 8, 33</td>
<td>Makefile</td>
</tr>
<tr>
<td>23</td>
<td>malloc()</td>
</tr>
<tr>
<td>106</td>
<td>mantissa</td>
</tr>
<tr>
<td>100</td>
<td>manual type conversion</td>
</tr>
<tr>
<td>75</td>
<td>’map_dm’</td>
</tr>
<tr>
<td>75</td>
<td>’map_lfp’</td>
</tr>
<tr>
<td>75</td>
<td>’map_nc4’</td>
</tr>
<tr>
<td>75</td>
<td>’map_prd’</td>
</tr>
<tr>
<td>75</td>
<td>’map_rdi’</td>
</tr>
<tr>
<td>75</td>
<td>’map_rew’</td>
</tr>
<tr>
<td>75</td>
<td>’map_scl’</td>
</tr>
<tr>
<td>75</td>
<td>’map_xst’</td>
</tr>
<tr>
<td>226</td>
<td>Mark Flanner</td>
</tr>
<tr>
<td>226</td>
<td>Markus Liebig</td>
</tr>
<tr>
<td>226</td>
<td>Martin Dix</td>
</tr>
<tr>
<td>227</td>
<td>Martin Otte</td>
</tr>
<tr>
<td>226</td>
<td>Martin Schmidt</td>
</tr>
<tr>
<td>51</td>
<td>Martin Schultz</td>
</tr>
<tr>
<td>142, 162</td>
<td>mask</td>
</tr>
<tr>
<td>220, 223</td>
<td>mask condition</td>
</tr>
<tr>
<td>219</td>
<td>masked average</td>
</tr>
<tr>
<td>30</td>
<td>Mass Store System</td>
</tr>
<tr>
<td>226</td>
<td>Matej Vela</td>
</tr>
<tr>
<td>163</td>
<td>mathematical functions</td>
</tr>
<tr>
<td>15</td>
<td>Matlab</td>
</tr>
<tr>
<td>94</td>
<td>max</td>
</tr>
<tr>
<td>131</td>
<td>max()</td>
</tr>
<tr>
<td>94</td>
<td>maximum</td>
</tr>
<tr>
<td>88</td>
<td>MD5 digest</td>
</tr>
<tr>
<td>94</td>
<td>mean</td>
</tr>
<tr>
<td>94</td>
<td>mebs</td>
</tr>
<tr>
<td>131</td>
<td>mebs()</td>
</tr>
<tr>
<td>22, 90</td>
<td>memory available</td>
</tr>
</tbody>
</table>

### N
<table>
<thead>
<tr>
<th>Page</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>memory leaks</td>
</tr>
<tr>
<td>22, 43, 90</td>
<td>memory requirements</td>
</tr>
<tr>
<td>17, 188</td>
<td>merging files</td>
</tr>
<tr>
<td>193</td>
<td>metadata</td>
</tr>
<tr>
<td>25</td>
<td>metadata optimization</td>
</tr>
<tr>
<td>183, 192</td>
<td>metadata, global</td>
</tr>
<tr>
<td>94</td>
<td>mibs</td>
</tr>
<tr>
<td>131</td>
<td>mibs()</td>
</tr>
<tr>
<td>51</td>
<td>Michael Decker</td>
</tr>
<tr>
<td>106</td>
<td>Michael Prather</td>
</tr>
<tr>
<td>226</td>
<td>Michael Schulz</td>
</tr>
<tr>
<td>6, 7</td>
<td>Microsoft</td>
</tr>
<tr>
<td>7</td>
<td>Microsoft Visual Studio</td>
</tr>
<tr>
<td>9</td>
<td>Mike Folk</td>
</tr>
<tr>
<td>226</td>
<td>Mike Page</td>
</tr>
<tr>
<td>71, 95, 168, 186, 193</td>
<td>missing values</td>
</tr>
<tr>
<td>129</td>
<td>missing values ncap2</td>
</tr>
<tr>
<td>71, 92, 217</td>
<td>missing_value</td>
</tr>
<tr>
<td>66, 69</td>
<td>MKS units</td>
</tr>
<tr>
<td>250, 253</td>
<td>MODIS</td>
</tr>
<tr>
<td>163</td>
<td>modulus</td>
</tr>
<tr>
<td>24</td>
<td>monotonic coordinates</td>
</tr>
<tr>
<td>229, 247</td>
<td>monthly average</td>
</tr>
<tr>
<td>229, 230</td>
<td>monthly data</td>
</tr>
<tr>
<td>48</td>
<td>move groups</td>
</tr>
<tr>
<td>59</td>
<td>MRO</td>
</tr>
<tr>
<td>60</td>
<td>MSA</td>
</tr>
<tr>
<td>109</td>
<td>msk_*</td>
</tr>
<tr>
<td>31</td>
<td>msrccp</td>
</tr>
<tr>
<td>35</td>
<td>msrcp</td>
</tr>
<tr>
<td>31</td>
<td>msread</td>
</tr>
<tr>
<td>30</td>
<td>MSS</td>
</tr>
<tr>
<td>22, 29, 30, 180, 183, 210, 212</td>
<td>multi-file operators</td>
</tr>
<tr>
<td>60</td>
<td>multi-hyperslab</td>
</tr>
<tr>
<td>59</td>
<td>Multi-Record Operator</td>
</tr>
<tr>
<td>163, 174, 187</td>
<td>multiplication</td>
</tr>
<tr>
<td>174</td>
<td>multiply</td>
</tr>
<tr>
<td>174, 185</td>
<td>multiplying data</td>
</tr>
<tr>
<td>60</td>
<td>multislab</td>
</tr>
<tr>
<td>41</td>
<td>mv</td>
</tr>
<tr>
<td>6</td>
<td>MVS</td>
</tr>
<tr>
<td>7</td>
<td>MVS</td>
</tr>
</tbody>
</table>

### Naked Characters
<table>
<thead>
<tr>
<th>Page</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>175</td>
<td>naked characters</td>
</tr>
</tbody>
</table>

### NaN
<table>
<thead>
<tr>
<th>Page</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>171</td>
<td>NaN</td>
</tr>
</tbody>
</table>

### NASA (North American Regional Reanalysis)
<table>
<thead>
<tr>
<th>Page</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>135</td>
<td>NASA</td>
</tr>
<tr>
<td>227</td>
<td>NASA EOSDIS</td>
</tr>
<tr>
<td>19</td>
<td>National Virtual Ocean Data System</td>
</tr>
<tr>
<td>33</td>
<td>nc_endef()</td>
</tr>
<tr>
<td>25</td>
<td>NC_BYTE</td>
</tr>
<tr>
<td>203</td>
<td>NC_CHAR</td>
</tr>
<tr>
<td>56, 176, 203</td>
<td>NC_CHAR</td>
</tr>
</tbody>
</table>
R

`rid` .............................................. 73
RAG .............................................. 182
RAM ............................................. 22, 90
RAM disks ...................................... 16, 90
RAM files ....................................... 16, 90
RAM variables ................................. 91, 134
ram_delete() .................................... 134
ram_write() ..................................... 134
random walk .................................... 163
rank ............................................. 118, 175, 176, 177, 219
rcp ............................................. 8, 30
RCS ............................................. 112
`rdi` ............................................. 75
re-dimension .................................... 202
re-order dimensions ......................... 202
record aggregation ............................ 182
record append .................................. 58
record average .................................. 210
record concatenation ........................... 212
record dimension ... 17, 54, 78, 179, 182, 184, 190,
191, 205, 209, 210, 212
record variable ................................ 54, 205
rectangular grids ................................ 142
recursion ........................................ 41
recursive ........................................ 41
regex ............................................ 44
regression ....................................... 240
regressions archive ......................... 14
regrid .......................................... 250
regular expressions .......................... 20, 28, 43, 162, 169, 172
remap ............................................ 137
Remik Ziemlinski ................................ 226
remote files .................................... 8, 30
rename groups .................................. 48
renaming attributes ......................... 214, 250
renaming dimensions ......................... 214, 250
renaming groups ................................ 214
renaming variables ............................. 214, 240, 250
reporting bugs .................................. 13
reshape variables .............................. 202
restrict ........................................... 7
reverse data .................................... 208
reverse dimensions ......................... 202, 207, 208
reverse() ....................................... 125
`rev` .............................................. 75
Rich Signell .................................... 226
rint .............................................. 163
rint() .......................................... 80
rms .............................................. 94
rmssdn .......................................... 94
rmssdn() ...................................... 131
root-mean-square .............................. 94
Rorik Peterson .................................. 225
round ............................................ 163
rounding ....................................... 78, 106
rounding functions ......................... 163
RPM ............................................. 12
running average .............................. 210
Russ Rew ...................................... 225, 226

S

safeguards ...................................... 15, 214
scale_factor ................................... 92, 183, 202, 212
scale_format ................................... 24
scaling ........................................... 24
Scientific Data Operators ..................... 227
'scl' .......................................... 75
Scott Capps .................................... 226
scp .............................................. 8, 30
script file ....................................... 116
SDO ............................................. 227
seasonal average ............................... 229
security .......................................... 88
SEH ............................................. 227
semi-colon ....................................... 117
separator ....................................... 195
server .......................................... 21, 32, 35
Server-Side Distributed Data Reduction &
Analysis ........................................ 227
server-side processing ....................... 33, 227
set_miss() ....................................... 130
sh ................................................................ 33
significand ..................................... 80
sin ................................................. 163
sine function .................................... 163
single-precision ................................ 165
sinh .............................................. 163
siz() .............................................. 132
SLD (Swath-like Data) ......................... 135
SMP ............................................... 25
sort ............................................... 137
sort alphabetically ............................ 189, 198
source code ...................................... 5
spatial distribution ............................ 237
special attributes .............................. 190
special characters ............................. 171
speed ............................................. 9, 16, 22, 24, 25, 72, 168, 215
sqravg .......................................... 94
sqravg() ....................................... 131
sqrt .............................................. 94
sqrt ............................................. 163
square root function ........................... 163
SSDDRA ......................................... 227
SSH .............................................. 8, 35
sshort() ......................................... 133
standard deviation ........................... 94, 98, 240
standard input .................................. 20, 180, 183, 210, 212
standard_name ................................ 64
<table>
<thead>
<tr>
<th>Function/Command</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat() system call</td>
<td>90</td>
</tr>
<tr>
<td>statement</td>
<td>117</td>
</tr>
<tr>
<td>static linking</td>
<td>9</td>
</tr>
<tr>
<td>stdin</td>
<td>20, 109, 180, 183, 210, 212</td>
</tr>
<tr>
<td>Steve Emmerson</td>
<td>225</td>
</tr>
<tr>
<td>stride</td>
<td>56, 57, 61, 67, 210, 211, 212, 213</td>
</tr>
<tr>
<td>strings</td>
<td>171</td>
</tr>
<tr>
<td>Stu Muller</td>
<td>226</td>
</tr>
<tr>
<td>stub</td>
<td>32</td>
</tr>
<tr>
<td>sub-cycle</td>
<td>59</td>
</tr>
<tr>
<td>subcycle</td>
<td>59</td>
</tr>
<tr>
<td>subsetting</td>
<td>40, 45, 111, 198, 200</td>
</tr>
<tr>
<td>subtract</td>
<td>174</td>
</tr>
<tr>
<td>subtracting data</td>
<td>174</td>
</tr>
<tr>
<td>subtraction</td>
<td>163, 174</td>
</tr>
<tr>
<td>summary</td>
<td>3</td>
</tr>
<tr>
<td>Sun</td>
<td>6</td>
</tr>
<tr>
<td>swap space</td>
<td>21, 22, 90</td>
</tr>
<tr>
<td>Swath-like Data (SLD)</td>
<td>135</td>
</tr>
<tr>
<td>switches</td>
<td>27</td>
</tr>
<tr>
<td>symbolic links</td>
<td>8, 18, 21, 175, 203</td>
</tr>
<tr>
<td>synchronous file access</td>
<td>30</td>
</tr>
<tr>
<td>synonym</td>
<td>8</td>
</tr>
<tr>
<td>syntax</td>
<td>117</td>
</tr>
<tr>
<td>System calls</td>
<td>90</td>
</tr>
</tbody>
</table>

**T**

<table>
<thead>
<tr>
<th>Function/Command</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Takeshi Enomoto</td>
<td>226</td>
</tr>
<tr>
<td>tan</td>
<td>163</td>
</tr>
<tr>
<td>tanh</td>
<td>163</td>
</tr>
<tr>
<td>temporary files</td>
<td>15, 92</td>
</tr>
<tr>
<td>temporary output files</td>
<td>15, 92, 214</td>
</tr>
<tr>
<td>TkXinfo</td>
<td>5</td>
</tr>
<tr>
<td>thrnbr</td>
<td>26</td>
</tr>
<tr>
<td>threads</td>
<td>22, 23, 25</td>
</tr>
<tr>
<td>time</td>
<td>67, 112</td>
</tr>
<tr>
<td>time-averaging</td>
<td>163, 229, 230, 237, 240, 247</td>
</tr>
<tr>
<td>time_offset</td>
<td>112</td>
</tr>
<tr>
<td>timestamp</td>
<td>108</td>
</tr>
<tr>
<td>total</td>
<td>94</td>
</tr>
<tr>
<td>transpose</td>
<td>54, 205</td>
</tr>
<tr>
<td>TREFHT</td>
<td>105</td>
</tr>
<tr>
<td>trunc</td>
<td>163</td>
</tr>
<tr>
<td>trunc()</td>
<td>101</td>
</tr>
<tr>
<td>truncate (groups)</td>
<td>46</td>
</tr>
<tr>
<td>truncation function</td>
<td>163</td>
</tr>
<tr>
<td>truth condition</td>
<td>220, 223</td>
</tr>
<tr>
<td>ttl</td>
<td>94</td>
</tr>
<tr>
<td>ttl()</td>
<td>131</td>
</tr>
<tr>
<td>type conversion</td>
<td>99</td>
</tr>
<tr>
<td>type()</td>
<td>132</td>
</tr>
</tbody>
</table>

**U**

<table>
<thead>
<tr>
<th>Function/Command</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubyte()</td>
<td>133</td>
</tr>
<tr>
<td>UDUnits</td>
<td>6, 66, 109, 133</td>
</tr>
</tbody>
</table>

**V**

<table>
<thead>
<tr>
<th>Function/Command</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>value list</td>
<td>126</td>
</tr>
<tr>
<td>variable names</td>
<td>214</td>
</tr>
<tr>
<td>variables, appending</td>
<td>240</td>
</tr>
<tr>
<td>variance</td>
<td>94</td>
</tr>
<tr>
<td>version</td>
<td>112</td>
</tr>
<tr>
<td>Vista (Microsoft operating system)</td>
<td>7</td>
</tr>
</tbody>
</table>

**W**

<table>
<thead>
<tr>
<th>Function/Command</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weather and Research Forecast (WRF Model)</td>
<td>135</td>
</tr>
<tr>
<td>weighted average</td>
<td>219, 247</td>
</tr>
<tr>
<td>Wenshan Wang</td>
<td>227</td>
</tr>
<tr>
<td>wget</td>
<td>30</td>
</tr>
<tr>
<td>where()</td>
<td>134</td>
</tr>
<tr>
<td>while()</td>
<td>136</td>
</tr>
<tr>
<td>whitespace</td>
<td>67</td>
</tr>
<tr>
<td>wildcards</td>
<td>28, 43, 169, 172</td>
</tr>
<tr>
<td>WIN32</td>
<td>8</td>
</tr>
<tr>
<td>Windows</td>
<td>6, 7</td>
</tr>
<tr>
<td>wrapped coordinates</td>
<td>56, 63, 143, 201</td>
</tr>
<tr>
<td>wrapped filenames</td>
<td>29</td>
</tr>
<tr>
<td>WRF</td>
<td>144</td>
</tr>
<tr>
<td>WRF (Weather and Research Forecast Model)</td>
<td>135</td>
</tr>
<tr>
<td>WWW documentation</td>
<td>5</td>
</tr>
</tbody>
</table>

**X**

<table>
<thead>
<tr>
<th>Function/Command</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xargs</td>
<td>20, 30</td>
</tr>
<tr>
<td>xlc</td>
<td>6</td>
</tr>
<tr>
<td>xlC</td>
<td>6</td>
</tr>
<tr>
<td>XML</td>
<td>195</td>
</tr>
<tr>
<td>XP (Microsoft operating system)</td>
<td>7</td>
</tr>
<tr>
<td>'xpl'</td>
<td>73</td>
</tr>
<tr>
<td>'xst'</td>
<td>73, 75</td>
</tr>
</tbody>
</table>
Y
Yorick ........................................ 15, 24

Z
zlib .............................................. 78