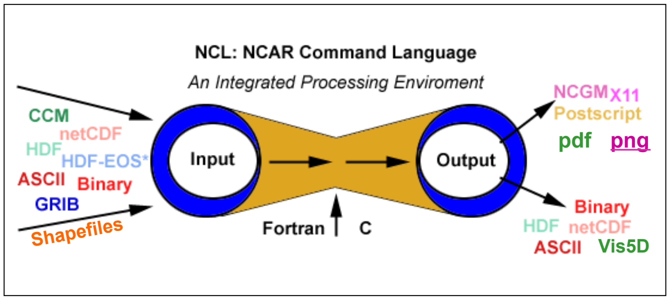
NCL File IO



Dennis Shea

National Center for Atmospheric Research

NCAR is sponsored by the National Science Foundation

I/O formats

- **Supported** formats **[need not know structure of file]**
  - **netCDF-3/4** [**net**work **C**ommon **D**ata **F**orm]
  - **HDF4/H5** [**H**ierarchical **D**ata **F**ormat (**S**cientific **D**ata **S**et only)]
  - **HDF-EOS** [**E**arth **O**bserving **S**ystem; HDF4 and HDF5]
  - **GRIB-1/2** [**Gr**id **in** **B**inary; WMO standard; NCEP, ECMWF]
  - **CCMHT** [**CCM** **H**istory **T**ape; COS blocked only; ccm2nc]
  - **Shapefile** [ESRI: geospatial vector data format GIS]
  - **6.1.0** ➔ near complete netCDF4, HDF5

- **Binary**
  - sequential [F: open(_,form="unformatted", access="sequential")]
  - flat/direct [F: open(_,form="unformatted",access=direct",recl=_)]
[C: write]

- **ASCII**
  - data organized in columns and rows
  - ‘complicated’ ascii formats: use NCL string (‘str_*’) functions
    - fortran or C to read; call from NCL

ncl_filedump

http://www.ncl.ucar.edu/Document/Tools/ncl_filedump.shtml

- **ncl_filedump [-c] [-v var1[,…]] [–h] file_name**
  - command line utility with **options**
  - provides textual overview of **any supported** file's contents
  - behavior analogous to Unidata's **ncdump -h**
  - **file_name** must have a file type suffix on command line
    - **.nc .grb .hdf .hdfeos .he5 .h5 .ccm .shp** *[case insensitive]*
    - suffix used as identifier only, actual file need not have

- **ncl_filedump file_name.[grb/nc/hdf/hdfeos]**
  - output can be sent to file or viewer via Unix redirection/ pipe

```
ncl_filedump foo.grb > foo.txt      [send to file]
ncl_filedump foo.hdf  | less        [send to viewer]
```

ncl_convert2nc

http://www.ncl.ucar.edu/Document/Tools/

- **ncl_convert2nc gribFile(s)** ***OPTIONS***
  - command line utility
  - converts GRIB/HDF/SHAPE file(s) to netCDF
  - output name same as input with **.nc** extension
- **ncl_convert2nc –h**
  - provides usage option information
- **ncl_convert2nc foo.grb**
  - will create **foo.nc**
- **ncl_convert2nc foo.hdf –L –nc4c –cl 1**
  - **-L** (files> 2GB)**;** **-nc4c** (netCDF4); **-cl 1** (compression lvl 1)

setfileoption

www.ncl.ucar.edu/Document/Functions/Built_in/setfileoption.shtml

- **allows user to specify file-format-specific options**
  - netCDF, GRIB and Binary options *[currently]*
- **sample usage of selected options**
  - writing netCDF
    - **setfileoption**(f, "DefineMode" ,True)
    - **setfileoption**("nc","Format","LargeFile")
    - **setfileoption**("nc","Format",”netCDF4Classic")
  - reading GRIB
    - **setfileoption**("grb" ,"ThinnedGridInterpolation", "cubic")
    - **setfileoption**("grb", "InitialTimeCoordinateType" \
      , "Numeric")
  - reading/writing Binary
    - **setfileoption**("bin", "ReadByteOrder", "LittleEndian")
    - **setfileoption**("bin", “WriteByteOrder", “BigEndian")

addfile (1 of 3)

- Opens a **supported** format
- Variables look like netCDF (Grib, HDF, HDF-EOS)

- **f** = **addfile** (**file_name.ext**, **status**)
  - **file_name** => any valid file name; string
  - **ext** => extension that identifies the type of file; string
    - netCDF: **"nc"** or **"cdf"** [read/write]
    - HDF: **"hdf"** , **"hdfeos"**, **"h5"**, **"he5"** [read/write]
    - GRIB: **"grb"** , **"grib"** [read only; GRIB1 or GRIB2]
    - CCMHT: **"ccm"** [read only]
    - SHAPE (GIS): **”shp"** [read only]
    - extension **not** required to be attached to file
  - **status** [read/write status] **"r"**, **"c"**, **"w"**
  - **f**
    - reference/pointer to a single file; any valid variable name
    - may have attributes (file attributes or global attributes)

http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclFormatSupport.shtml

addfile (2 of 3)

- **Examples: opening a single file**
  - fin = **addfile** ("0005-12.**nc**" , "**r**")
  - fout = **addfile** ("./ncOutput.**nc**" , "**c**")
  - fio = **addfile** ("/tmp/shea/sample.**hdf**" , "**w**")
  - g = **addfile** ("/dss/dsxxx/Y12345.**grb**", "**r**")
  - s = **addfile** ("foo.**shp**" , ”**r**")

- **Numerous functions to query contents of supported file**
  - getfilevarnames
  - getfilevardims
  - getfilevaratts
  - getfilevardimsizes
  - getfilevartypes
  - isfilevar
  - isfilevaratt
  - isfilevardim
  - isfilevarcoord

```
diri  = "/fs/cgd/data0/shea/GRIB/"
fili  = ”narr_2000121106”
fin   = addfile(diri+fili+".grb" , " r ")
```

```
varNames = getfilevarnames (fin)
if (isfilevarcoord(fin, "U", "lat") ) then
…
end if
```

Import Variable from Supported Fmt

u = f->U

- read variable and **all** meta data into memory **[structure]**
- no space allowed to left/right of **->** [fatal error]
- use **"$"** syntax to represent variable name if type string

```
f = addfile ("foo.grb", "r")
vNam = getfilevarnames (f)  ; all variables on file
        or
vNam = (/ "U", "V" /)         ; manually specify
do n=0,dimsizes(vNam)-1
    x = f->$vNam (n)$          ; $..$ substitute string
      .....
end do
```

u = (/ f->U /)

- read data values only and _FillValue attribute

Example: open, read, output netCDF

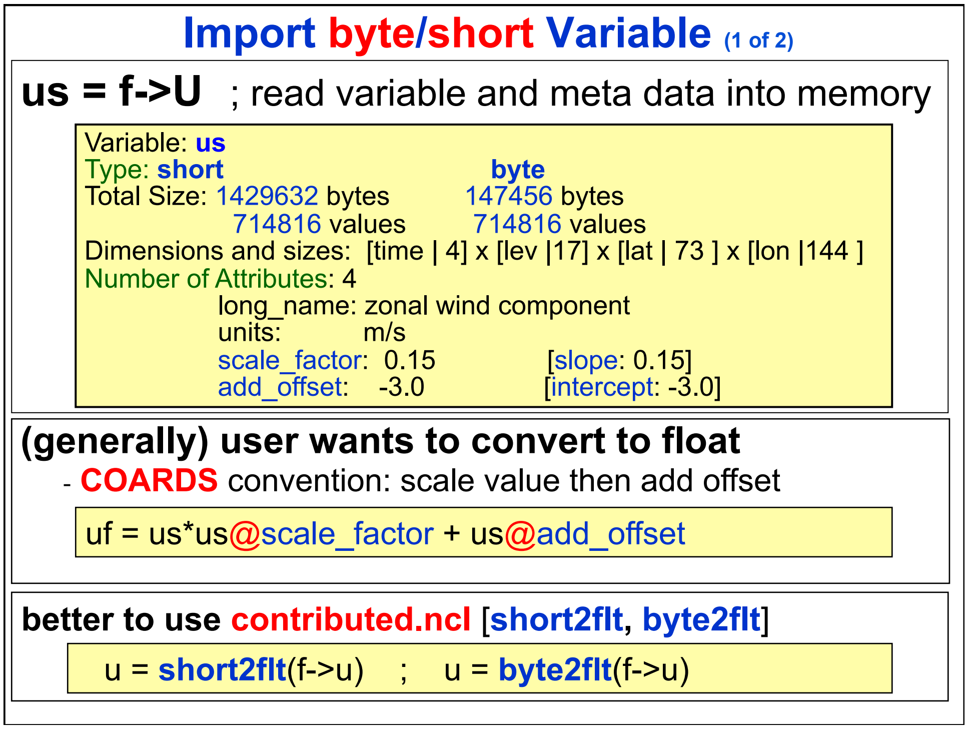
```
begin         ; optional
;-----------------------------------------------
  fin    = addfile ("in.nc, "r")      ; open file and read in data

  u      = fin->U                    ; import a variable

  fout   = addfile("out.nc" , "c") ; create reference to output file

  fout@title = "I/O Example 1"  ; add a global attribute to the file

;-----------------------------------------------
;Output variable u to netCDF file:    ncrcat –v U in.nc out.nc
;-----------------------------------------------
   fout->U2 = u                        ; Output variable u to nc file

end                                  ; only if begin is present
```

Note: This method of netCDF creation has simple syntax. It can be slow but is commonly used.

Example: query file, system commands

```
;-----------------------------------------------------------------------
; open file, create array of variable names, # of names
;-----------------------------------------------------------------------
       fin      = addfile ("./in.nc", "r")
       vars    = (/"U", "V", "T" /)          ; manual specification
       nvars = dimsizes (vars)           ; nvars = 3
;-----------------------------------------------------------------------
; use system to remove output file before creation
;-----------------------------------------------------------------------
       system("/bin/rm –f   out.nc")
       fout    = addfile("out.nc" , "c")
;-----------------------------------------------------------------------
; loop, query if variable on the file, then output to netCDF
;-----------------------------------------------------------------------
       do n=0,nvars-1
              if (isfilevar(fin, vars(n)))  then
                     fout->$vars(n)$ = fin->$vars(n)$
              end if
       end do
```

```
ncrcat –v U,V,T   in.nc   out.nc
```



Import byte/short Variable (1 of 2)

us = f->U ; read variable and meta data into memory

```
Variable: us
Type: short                           byte
Total Size: 1429632 bytes          147456 bytes
             714816 values          714816 values
Dimensions and sizes:  [time | 4] x [lev |17] x [lat | 73 ] x [lon |144 ]
Number of Attributes: 4
               long_name: zonal wind component
               units:          m/s
               scale_factor:  0.15              [slope: 0.15]
               add_offset:    -3.0             [intercept: -3.0]
```

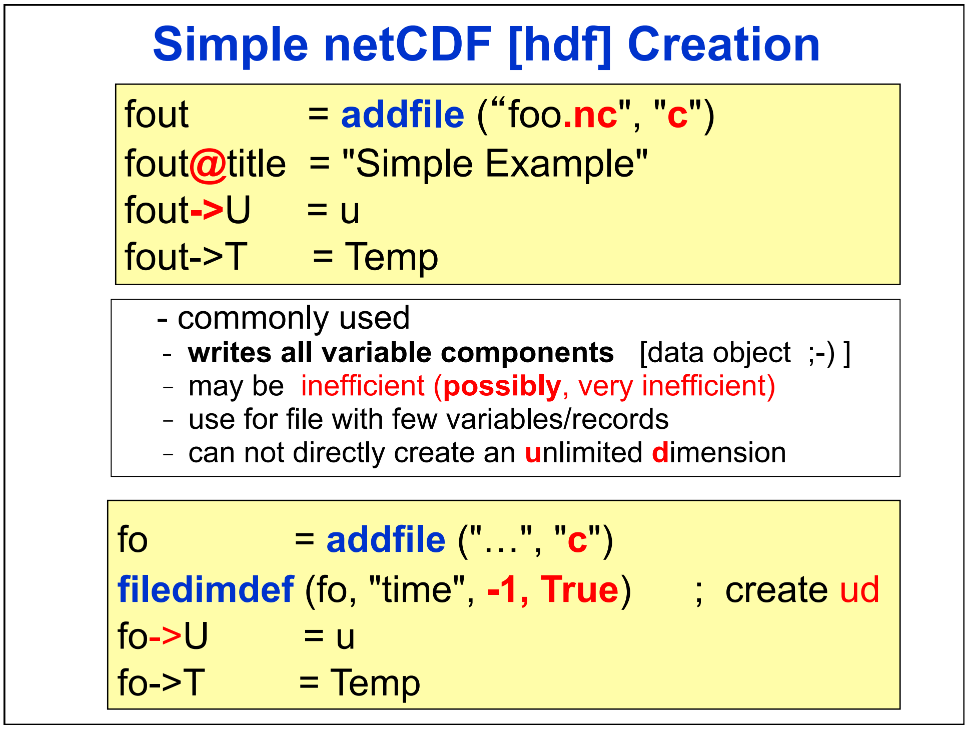
(generally) user wants to convert to float

- **COARDS** convention: scale value then add offset

```
uf = us*us@scale_factor + us@add_offset
```

better to use contributed.ncl [short2flt, byte2flt]

```
u = short2flt(f->u)   ;   u = byte2flt(f->u)
```



Simple netCDF [hdf] Creation

```
fout          = addfile (“foo.nc", "c")
fout@title  = "Simple Example"
fout->U       = u
fout->T       = Temp
```

- commonly used
  - **writes all variable components**   [data object ;-)]
  - may be inefficient (**possibly**, very inefficient)
  - use for file with few variables/records
  - can not directly create an **u**nlimited **d**imension

```
fo              = addfile ("…", "c")
filedimdef (fo, "time", -1, True)      ;  create ud
fo->U          = u
fo->T          = Temp
```

Efficient netCDF Creation

- **requires ‘a priori’ definition of file contents**
  - must be done in other languages/tools also [F, C, IDL, ..]

- **NCL functions to predefine a netCDF/HDF file**:
  - **setfileoption**: specify entering define mode
  - **filevardef**: define name(s) of one or more variables
  - **filevarattdef**: copy attributes from a variable to one or more file variables
  - **filedimdef**: defines dimensions including unlimited
  - **fileattdef**: copy attributes from a variable to a file as global attributes

- Less tedious than other languages

Contents of a well written netCDF variable

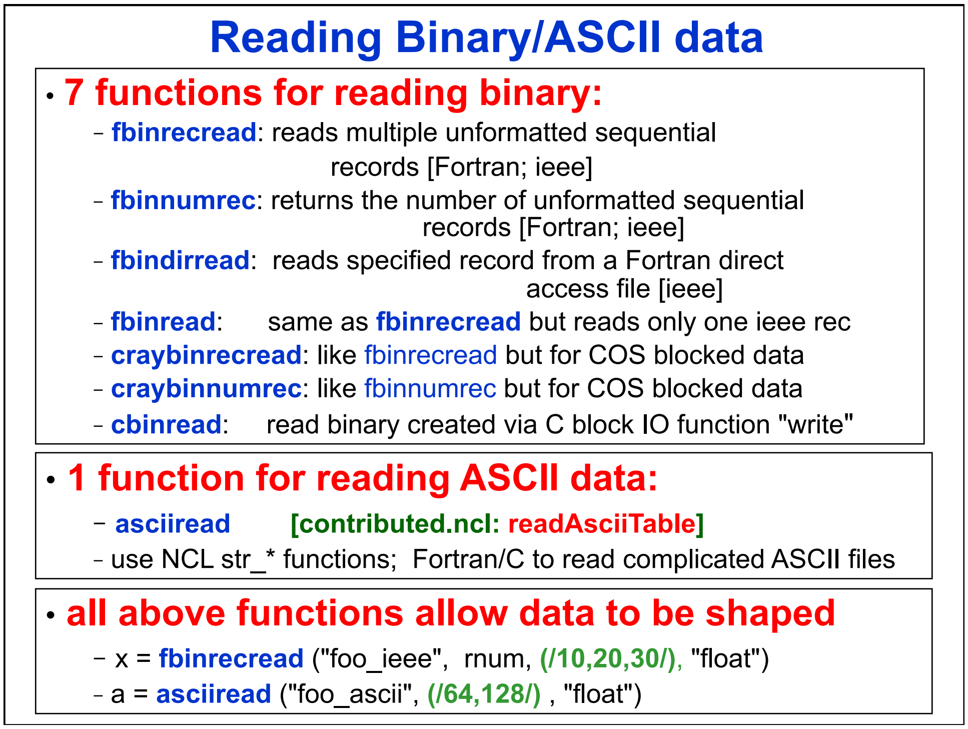
- **Variables**
  - **long_name***
  - **units***
  - **_FillValue** [if applicable]
  - missing_value [“]
  - named dimensions
  - coordinate variable(s)

Consider: T(:)

```
T@long_name = "Temperature"
T@units  =  ”degC"
T@_FillValue  = 1.e+20
T@missing_value = T@_FillValue
T!0 = "time"
T&time = time
```

Result: T(time)

*__COARDS__ and **CF** conventions

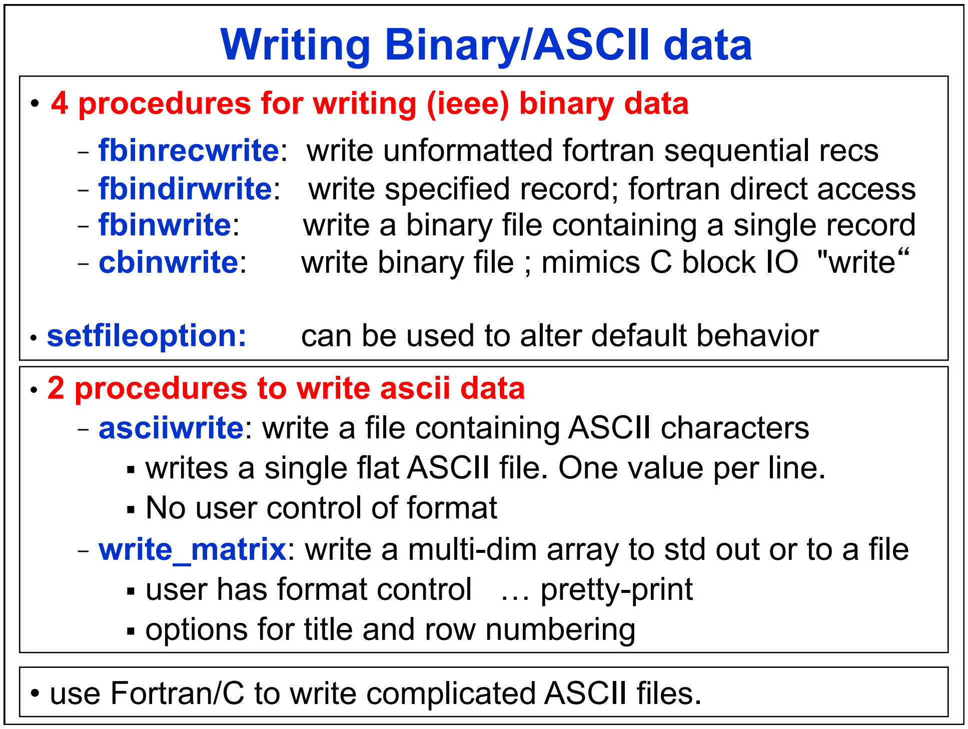


Reading Binary/ASCII data

- **7 functions for reading binary:**
  - **fbinrecread**: reads multiple unformatted sequential records [Fortran; ieee]
  - **fbinnumrec**: returns the number of unformatted sequential records [Fortran; ieee]
  - **fbindirread**: reads specified record from a Fortran direct access file [ieee]
  - **fbinread**: same as **fbinrecread** but reads only one ieee rec
  - **craybinrecread**: like fbinrecread but for COS blocked data
  - **craybinnumrec**: like fbinnumrec but for COS blocked data
  - **cbinread**: read binary created via C block IO function "write"

- **1 function for reading ASCII data:**
  - **asciiread** **[contributed.ncl: readAsciiTable]**
  - use NCL str_* functions; Fortran/C to read complicated ASCII files

- **all above functions allow data to be shaped**
  - x = **fbinrecread** ("foo_ieee", rnum, **(/10,20,30/)**, "float")
  - a = **asciiread** ("foo_ascii", **(/64,128/)** , "float")



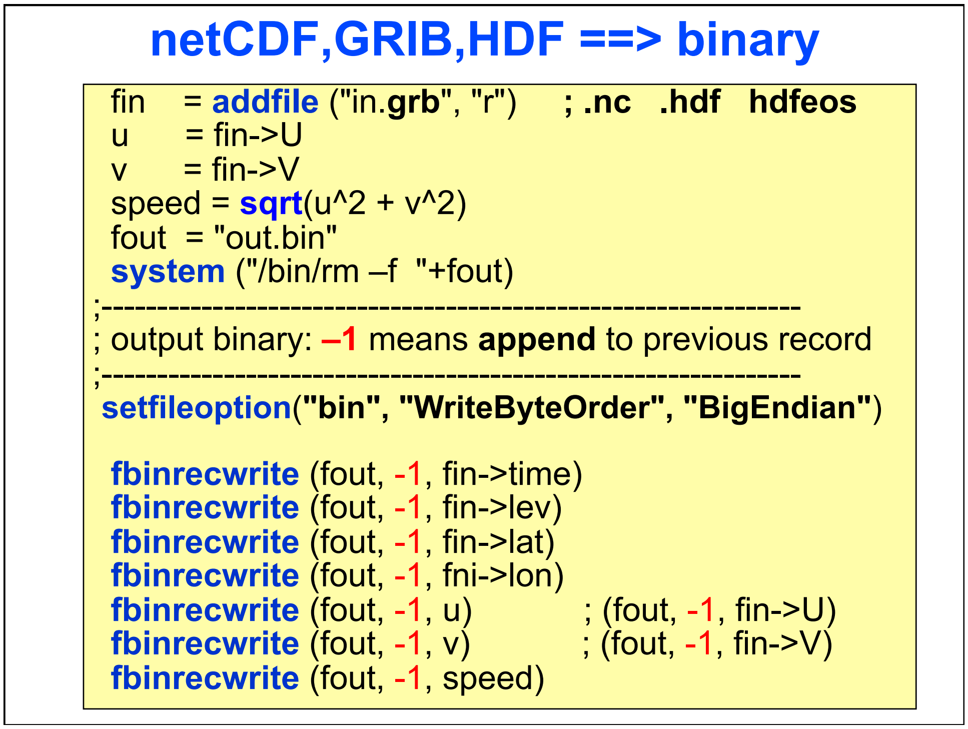
Writing Binary/ASCII data

- **4 procedures for writing (ieee) binary data**
  - **fbinrecwrite**: write unformatted fortran sequential recs
  - **fbindirwrite**: write specified record; fortran direct access
  - **fbinwrite**: write a binary file containing a single record
  - **cbinwrite**: write binary file ; mimics C block IO "write“

- **setfileoption:** can be used to alter default behavior

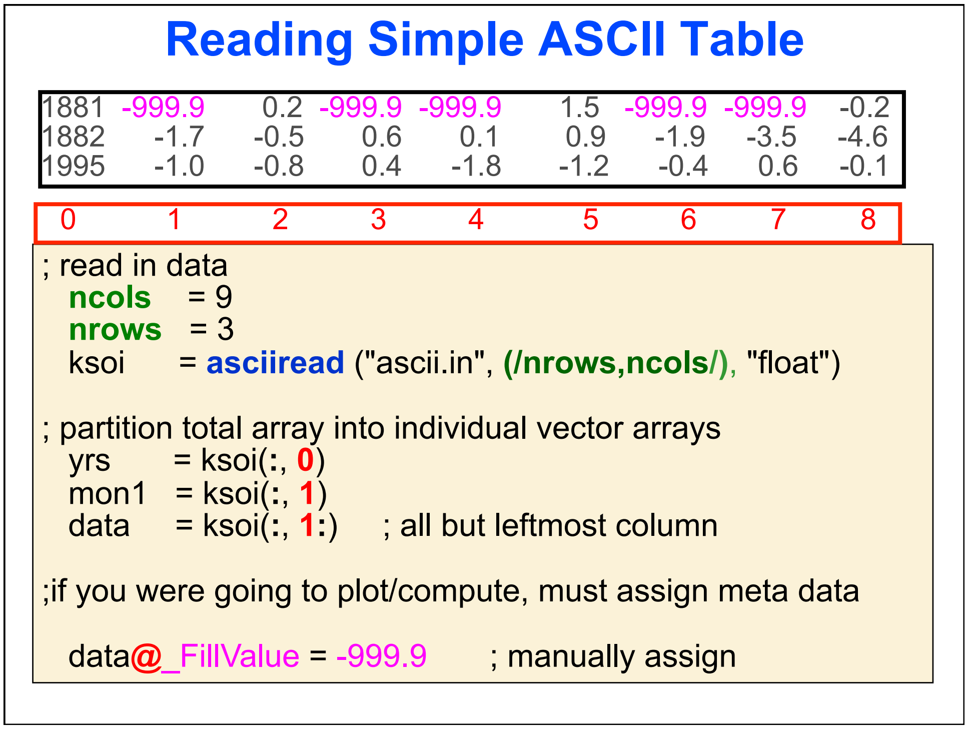
- **2 procedures to write ascii data**
  - **asciiwrite**: write a file containing ASCII characters
    - writes a single flat ASCII file. One value per line.
    - No user control of format
  - **write_matrix**: write a multi-dim array to std out or to a file
    - user has format control … pretty-print
    - options for title and row numbering

- use Fortran/C to write complicated ASCII files.



netCDF,GRIB,HDF ==> binary

```
 fin    = addfile ("in.grb", "r")     ; .nc   .hdf   hdfeos
 u      = fin->U
 v      = fin->V
 speed = sqrt(u^2 + v^2)
 fout  = "out.bin"
 system ("/bin/rm –f  "+fout)
;--------------------------------------------------------------
; output binary: –1 means append to previous record
;--------------------------------------------------------------
 setfileoption("bin", "WriteByteOrder", "BigEndian")

 fbinrecwrite (fout, -1, fin->time)
 fbinrecwrite (fout, -1, fin->lev)
 fbinrecwrite (fout, -1, fin->lat)
 fbinrecwrite (fout, -1, fni->lon)
 fbinrecwrite (fout, -1, u)            ; (fout, -1, fin->U)
 fbinrecwrite (fout, -1, v)            ; (fout, -1, fin->V)
 fbinrecwrite (fout, -1, speed)
```



Reading Simple ASCII Table

1881	-999.9	0.2	-999.9	-999.9	1.5	-999.9	-999.9	-0.2
1882	-1.7	-0.5	0.6	0.1	0.9	-1.9	-3.5	-4.6
1995	-1.0	-0.8	0.4	-1.8	-1.2	-0.4	0.6	-0.1
0	1	2	3	4	5	6	7	8

```
; read in data
   ncols    = 9
   nrows   = 3
   ksoi      = asciiread ("ascii.in", (/nrows,ncols/), "float")

; partition total array into individual vector arrays
   yrs       = ksoi(:, 0)
   mon1    = ksoi(:, 1)
   data      = ksoi(:, 1:)      ; all but leftmost column

;if you were going to plot/compute, must assign meta data

   data@_FillValue = -999.9        ; manually assign
```

Read ASCII Table with Header

```
          Jan-to-Aug Southern Oscillation Index 1881-1995
Year    Jan     Feb     Mar     Apr     May      Jun     Jul     Aug
1881  -999.9  -999.9  -999.9  -999.9  -999.9  -999.9  999.9  -999.9
1882    -1.7    -0.5     0.6     0.1     0.9     -1.9    -3.5    -4.6
1995    -1.0    -0.8     0.4    -1.8    -1.2     -0.4     0.6    -0.1
```

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"

  ncols   = 9
  nhead   = 2                ; number of lines to skip
  ksoi    = readAsciiTable ("ascii.in", ncols, "float", nhead)
  yrs     = ksoi(: , 0)
  col1    = ksoi(: , 1)
  data    = ksoi(: , 1:)    ; all but leftmost column
  data@_FillValue = -999.9
`
```

Last argument could be string:

```
  ksoi    = readAsciiTable ("ascii.in", ncols, "float", "Year")
```

write_matrix(x[*][*], fmt, opt)

- **pretty-print 2D array to standard out**
  - integer, float, double
  - user format control (fmt)
  - if not 2D use T=**onedtond**(**ndtooned**(TT) , (/N,M/))
  - T(7,5): **write_matrix** (T, "5f7.2", False)

```
  4.35     4.39    0.27   -3.35   -6.90
  4.36     4.66    3.77   -1.66    4.06
  9.73    -5.84    0.89    8.46   10.39
  4.91     4.59   -3.09    7.55    4.56
   17      3.68    5.08    0.14   -5.63
 -0.63    -4.12   -2.51    1.76   -1.43
 -4.29     0.07    5.85    0.87    8.65
```

- **create an ASCII file**

```
opt          = True
opt@fout = "foo.ascii"             ; file name
write_matrix (T, "5f7.2", opt)
```

Importing Multiple Supported Files

- **systemfunc:** returns info from unix/linux
  - fnames = **systemfunc** ("**ls** reAnal*")
    - fpath = **systemfunc**("**ls** /mydata/reAnal*") ; full path
    - fils   = **systemfunc**("**cd** "+path+ " **; ls** reAnal*")

      where: path = "/my/data/"
- **manually**
  - fnames = (/ "file1" , "file2", ... /)

```
path        = “/data0/shea/"
fnames   = (/ "reAnal1", "reAnal2", "reAnal3", "reAnal4"/)
nfiles     = dimsizes(fnames)                    ; nfiles = 4
do nf =0,nfiles-1
    f = addfile (path+fnames(nf)+".grb", "r")
    ……
end do
```

addfiles (1 of 2)

- span **multiple supported** files

- **q** = **addfiles** (**fNames**, "r")
  - **fNames** is a 1D array of file names (strings)
  - can be used for **any supported format**
  - technically, "q" is a variable of type **list**

```
T = q[:]->T              ; [:] read all files
```

- read T [with meta data] from each file in list ‘q’
- T must exist in each file and be same shape [rank]
- a **list** is used to sequence results of **addfiles**
- normal file variable selection is used with "**[…]**"

```
lat = q[0]->lat       ; [0] read from first file
Z  = q[2:6:2]->Z   ; extract Z only from files 2,4,6
```

addfiles (2 of 2)

- 2 options on variable merging
  - ListSetType (a, "cat")   [default; "cat" => concatenation]
  - ListSetType (a, "join")

- when to use "cat" and "join" [rule of thumb]
  - cat: continuous record
  - join: creating ensembles
    - a record dimension will be added

netCDF Operator (NCO): **cat ➔ncrcat**   **join ➔ ncecat**

Example: Read "T" across 5 files ["cat"]

[Each file has 12 months]

```
fils   = systemfunc ("ls  ./ann*.nc")
f       = addfiles (fils, "r")
ListSetType(f, "cat")           ; not necessary [default]
T     = f[:]->T                        ; read T from all files
printVarSummary(T)
```

```
Variable: T
Type: float
Total Size:  5529600 bytes
                  1382400 values
Attributes: 2
      units:          K
      long_name: temp
Number of Dimensions: 4
Dimensions and sizes: [time|60] x [lev|5] x [lat | 48]  x  [lon | 96]
Coordinates:
time: [2349 … 4123]      lat:   [-87.159..87.159]
lev:   [85000 … 25000]  lon:   [0..356.25]
```

addfiles: option ["join"]

```
fils  = systemfunc ("ls ./ann*.nc")
f      = addfiles (fils, "r")
ListSetType (f, "join")
T    = f[:]->T
printVarSummary (T)
```

```
Variable: T
Type: float
Total Size:  5529600 bytes
             1382400 values
Attributes: 2
   units:          K
   long_name: temperature
Number of Dimensions: 5
Dim/sizes:  [case | 5] x  [time|12]  x  [lev|5]  x  [lat | 48]  x  [lon | 96]
Coordinates:
time: [2349 … 2683]      lat:    [-87.159..87.159]
lev:   [85000 … 25000]   lon:   [0..356.25]
```