Introduction to NetCDF4 binary file with Python, C++ and R

Bertrand Brelier

SciNet HPC Consortium
Compute Canada

March 12, 2014

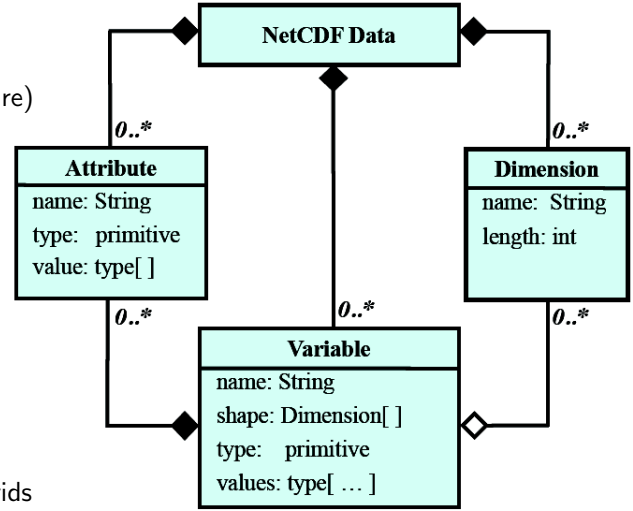
NetCDF

Network **C**ommon **D**ata **F**orm

- Data model for scientific data and metadata
  - Widely used in ocean, climate, atmospheric science
  - Used in some other disciplines: molecular dynamics, neuro imaging, fusion research
- File format for portable data
  - Array-oriented scientific data and metadata
  - NetCDF data is self-describing, portable, direct access, appendable, networkable, extensible, sharable, archivable
- Application programming interfaces (APIs)
  - C, Java, C++, Fortran (Developed and supported by UCAR / Unidata)
  - Python, Ruby, Perl, MATLAB, R (3rd party APIs)

NetCDF classic data model

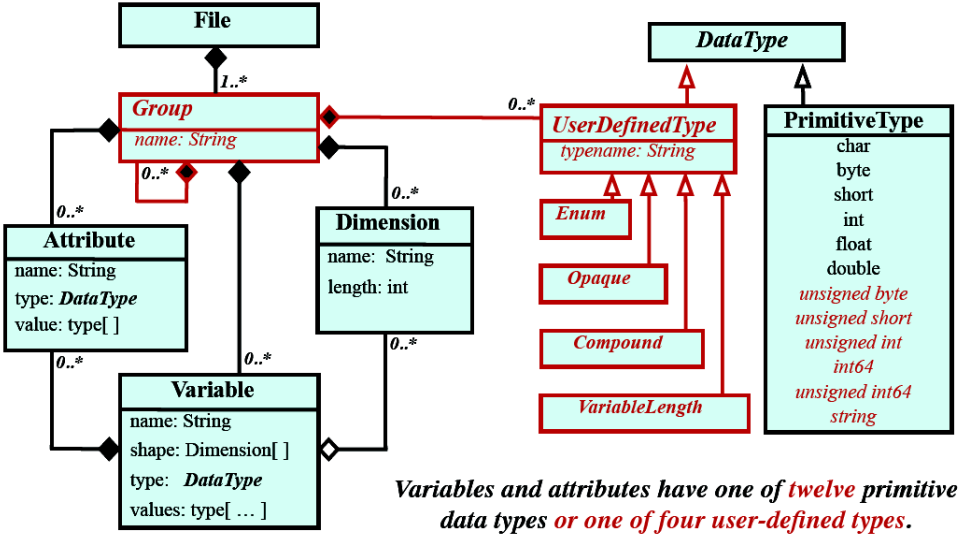
- NetCDF Data has :
  - Variables (eg temperature, pressure)
  - Attributes (eg units)
  - Dimensions (eg time)
- Each variable has
  - Name, shape, type, attributes
  - N-dimensional array of values
- Each attribute has
  - Name, type, value(s)
- Each dimension has
  - Name, length
- Variables may share dimensions
  - Represents shared coordinates, grids
- Variable and attribute values are of type
  - Numeric: 8-bit byte, 16-bit short, 32-bit int, 32- bit float, 64-bit double
  - Character: arrays of char for text



output of : ncdump Test.nc :

```
netcdf  Test {
dimensions:
        Dimension = 4 ;
        NRecords = UNLIMITED ; // (10 currently)
variables:
        float momentum(NRecords, Dimension) ;
        momentum:units = ``GeV'' ;
data:
        momentum =
   -24.68694, 22.93542, -48.75218, 59.2652,
   9.410192, 43.30603, 8.885784, 45.1999,
   -34.1302, -5.663451, 42.07206, 54.47121,
   12.25947, 26.26641, -3.21174, 29.1658,
   30.63048, -17.76378, 39.78646, 53.2621,
   42.81401, -36.93607, -4.968459, 56.76362,
   -32.08664, -33.3213, 44.05455, 63.88094,
   -13.81777, 1.84193, -15.19944, 20.6266,
   25.82092, -8.994961, -26.58193, 38.13579,
   -25.00047, -16.52758, 6.168945, 30.59985 ;
}
```

NetCDF-4 format



Variables and attributes have one of twelve primitive data types or one of four user-defined types.

NetCDF-4 format

- The opaque type is a type which is a collection of objects of a known size.Nothing is known to NetCDF about the contents of these blobs of data, except their size in bytes, and the name of the type.
- A variable length array is represented in C as a structure from HDF5, the nc_vlen_t structure. It contains a len member, which contains the length of that array, and a pointer to the array.
- A compound datatype is similar to a struct in C and contains a collection of one or more atomic or user-defined types.
  - It has a fixed total size.
  - It consists of zero or more named members that do not overlap with other members.
  - Each member has a name distinct from other members.
  - Each member has its own datatype.
  - Each member is referenced by an index number between zero and N-1, where N is the number of members in the compound datatype.
  - Each member has a fixed byte offset, which is the first byte (smallest byte address) of that member in the compound datatype.
  - In addition to other other user-defined data types or atomic datatypes, a member can be a small fixed-size array of any type with up to four fixed-size dimensions (not associated with named NetCDF dimensions).

SciNet compute • calcul CANADA

NetCDF-4 format

- Uses HDF5 as a storage layer
- Provides performance advantages of HDF5
  - Compression
  - Chunking
  - Parallel I/O
  - Efficient schema changes
- Useful for larger or more complex datasets
- Suitable for high-performance computing

SciNet compute • calcul CANADA

NetCDF-4 format : endianness

- NetCDF4 support little endian and big endian
- Default : NC_ENDIAN_NATIVE for native endianness
- can change the endianness of a variable with the function nc_def_var_endian : nc_def_var_endian(int ncid, int varid, int endian);
  - ncid : NetCDF ID, from a previous call to nc_open or nc_create.
  - varid : Variable ID.
  - endian :
    - NC_ENDIAN_NATIVE for native endianness.
    - NC_ENDIAN_LITTLE for little endian
    - NC_ENDIAN_BIG for big endian.

NetCDF compound variable

Will show an example to read and write a compound variable in :

- Python
- C++
- R

The compound variable will have 10 records with 8 members :

name	ADCcount	grid_i	grid_j	pressure	energy	idnumber	pressure2
16-character String	Unsigned short integer	32-bit integer	32-bit integer	float (single-precision)	double (double-precision)	Signed 64-bit integer	2-dim table of float (2*3)

Create compound with Python

```
from netCDF4 import Dataset
from netCDF4 import chartostring, stringtoarr
import numpy

f = Dataset('particles.nc','w',format='NETCDF4')

size = 10

Particle = numpy.dtype([('name', 'S1', 16),                    # 16-character String
                        ('ADCcount',numpy.uint16),             # Unsigned short integer
                        ('grid_i',numpy.int32),                # 32-bit integer
                        ('grid_j',numpy.int32),                # 32-bit integer
                        ('pressure',numpy.float32),            # float  (single-precision)
                        ('energy',numpy.float64),              # double (double-precision)
                        ('idnumber',numpy.int64),              # Signed 64-bit integer
                        ('pressure2' , numpy.float32 , (2,3) ) # array of floats (single-preci
                        ])

Particle_t = f.createCompoundType(Particle,'Particle')

f.createDimension('NRecords',None)
v = f.createVariable('Data',Particle_t,'NRecords')
data = numpy.empty(size,Particle)
```

Create compound with Python

```
for i in xrange(10):
    data['name'][i] = stringtoarr('Particle: %6d' % (i),16)
    data['ADCcount'][i] = (i * 256) % (1 << 16)
    data['grid_i'][i] = i
    data['grid_j'][i] = 10 - i
    data['pressure'][i] = float(i*i)
    data['energy'][i] = float(data['pressure'][i] ** 4)
    data['idnumber'][i] = i * (2 ** 34)
    data['pressure2'][i] = [
        [0.5+float(i),1.5+float(i),2.5+float(i)],
        [-1.5+float(i),-2.5+float(i),-3.5+float(i)]]
#Fill data in File
v[:] = data

f.close()
```

Code works on gpc with the modules :

```
module load gcc/4.8.1 hdf5/187-v18-serial-gcc netcdf/4.1.3_hdf5_serial-gcc intel/14.0.0 python/2.7.2
```

Create compound with Python :

ncdump particles.nc :

```
netcdf particles {
types:
  compound Particle {
    char name(16) ;
    ushort ADCcount ;
    int grid_i ;
    int grid_j ;
    float pressure ;
    double energy ;
    int64 idnumber ;
    float pressure2(2, 3) ;
  }; // Particle
dimensions:
NRecords = UNLIMITED ; // (10 currently)
variables:
Particle Data(NRecords) ;
data:

 Data =
    {{"Particle:␣␣␣␣␣␣0"}, 0, 0, 10, 0, 0, 0, {0.5, 1.5, 2.5, -1.5, -2.5, -3.5}},
    {{"Particle:␣␣␣␣␣␣1"}, 256, 1, 9, 1, 1, 17179869184, {1.5, 2.5, 3.5, -0.5, -1.5, -2.5}},
    {{"Particle:␣␣␣␣␣␣2"}, 512, 2, 8, 4, 256, 34359738368, {2.5, 3.5, 4.5, 0.5, -0.5, -1.5}},
    {{"Particle:␣␣␣␣␣␣3"}, 768, 3, 7, 9, 6561, 51539607552, {3.5, 4.5, 5.5, 1.5, 0.5, -0.5}},
    {{"Particle:␣␣␣␣␣␣4"}, 1024, 4, 6, 16, 65536, 68719476736, {4.5, 5.5, 6.5, 2.5, 1.5, 0.5}},
    {{"Particle:␣␣␣␣␣␣5"}, 1280, 5, 5, 25, 390625, 85899345920, {5.5, 6.5, 7.5, 3.5, 2.5, 1.5}},
    {{"Particle:␣␣␣␣␣␣6"}, 1536, 6, 4, 36, 1679616, 103079215104, {6.5, 7.5, 8.5, 4.5, 3.5, 2.5}},
    {{"Particle:␣␣␣␣␣␣7"}, 1792, 7, 3, 49, 5764801, 120259084288, {7.5, 8.5, 9.5, 5.5, 4.5, 3.5}},
    {{"Particle:␣␣␣␣␣␣8"}, 2048, 8, 2, 64, 16777216, 137438953472, {8.5, 9.5, 10.5, 6.5, 5.5, 4.5}},
    {{"Particle:␣␣␣␣␣␣9"}, 2304, 9, 1, 81, 43046721, 154618822656, {9.5, 10.5, 11.5, 7.5, 6.5, 5.5}} ;
}
```

Read compound with Python :

```
from netCDF4 import Dataset
from netCDF4 import chartostring, stringtoarr
import numpy

f = Dataset('particles.nc','r',format='NETCDF4')
v = f.variables['Data']
datain = v[:] # read in all the data into a numpy structured array

Particle = numpy.dtype([('name', 'S1', 16),                    # 16-character String
                        ('ADCcount',numpy.uint16),             # Unsigned short integer
                        ('grid_i',numpy.int32),                # 32-bit integer
                        ('grid_j',numpy.int32),                # 32-bit integer
                        ('pressure',numpy.float32),            # float  (single-precision)
                        ('energy',numpy.float64),              # double (double-precision)
                        ('idnumber',numpy.int64),              # Signed 64-bit integer
                        ('pressure2' , numpy.float32 , (2,3) ) # array of floats (single-preci
                        ])
```

Read compound with Python :

```
Mydata = numpy.empty(datain.shape,Particle)

Mydata['name'] = datain['name'];
Mydata['ADCcount'] = datain['ADCcount'];
Mydata['grid_i'] = datain['grid_i'];
Mydata['grid_j'] = datain['grid_j'];
Mydata['pressure'] = datain['pressure'];
Mydata['energy'] = datain['energy'];
Mydata['idnumber'] = datain['idnumber'];
Mydata['pressure2'] = datain['pressure2'];

for i in xrange(int(str((datain.shape))[1:-2])):
    print "\""+str(chartostring(Mydata['name'][i]))+"\", "+str(Mydata['ADCcount'][i])+", "
    +str(Mydata['grid_i'][i])+", "+str(Mydata['grid_j'][i])+", "+str(Mydata['pressure'][i])+", "
    +str(Mydata['energy'][i])+", "+str(Mydata['idnumber'][i])
    print str(Mydata['pressure2'][i])

f.close()
```

gives this output :

```
"Particle:      0", 0, 0, 10, 0.0, 0.0, 0
[[ 0.5  1.5  2.5]
 [-1.5 -2.5 -3.5]]
"Particle:      1", 256, 1, 9, 1.0, 1.0, 17179869184
[[ 1.5  2.5  3.5]
 [-0.5 -1.5 -2.5]]
...
```

Create compound with C++ (1)

```
#include <iostream>
#include <netcdfcpp.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
using namespace std;

#define FILE_NAME "particles.nc"
#define SVC_REC "Particle"
#define DIM_LEN 10 //number of records in file
int main(void)
{
  typedef struct Particle
  {
    char   name[16];
    unsigned short int     ADCcount;
    int grid_i;
    int grid_j;
    float pressure;
    double energy ;
    long  idnumber;
    float pressure2[2][3];
  } Particle;

  //Definition of the data_in variable
  Particle data_in[DIM_LEN];
```

Create compound with C++ (2)

```
//Filling variable
for (int i=0; i<DIM_LEN; i++){
  char outString[sizeof(data_in[i].name)];
  sprintf(outString,"%s%d","Particle:      ",i);
  memcpy(data_in[i].name,outString,sizeof(data_in[i].name));;
  data_in[i].ADCcount=(i * 256) % 65536;
  data_in[i].grid_i=i;
  data_in[i].grid_j=10-i;
  data_in[i].pressure=float(i*i);
  data_in[i].energy=float(pow(data_in[i].pressure,4));
  data_in[i].idnumber=i * (pow(2,34));
  for (int k=0; k<3; k++){
    data_in[i].pressure2[0][k]=0.5+k+float(i);
    data_in[i].pressure2[1][k]=-1.5-k+float(i);
  }
 }

//Variables needed to create the NetCDF file :
int ncid, varid,mtypeid;
int dimid;
int dimids[] = {0}, fieldid;
char name[NC_MAX_NAME + 1];
size_t size;
size_t nfields;
```

Create compound with C++ (3)

```
//create the NetCDF file :
nc_create(FILE_NAME,NC_NETCDF4,&ncid);
nc_def_compound(ncid, sizeof(Particle), SVC_REC, &mtypeid);
nc_inq_compound(ncid, mtypeid, name, &size, &nfields);
nc_insert_compound(ncid, mtypeid, "ADCcount",
                   NC_COMPOUND_OFFSET(Particle, ADCcount), NC_SHORT);
nc_insert_compound(ncid, mtypeid, "grid_i",
                   NC_COMPOUND_OFFSET(Particle, grid_i), NC_INT);
nc_insert_compound(ncid, mtypeid, "grid_j",
                   NC_COMPOUND_OFFSET(Particle, grid_j), NC_INT);
nc_insert_compound(ncid, mtypeid, "pressure",
                   NC_COMPOUND_OFFSET(Particle, pressure), NC_FLOAT);
nc_insert_compound(ncid, mtypeid, "energy",
                   NC_COMPOUND_OFFSET(Particle, energy), NC_DOUBLE);
nc_insert_compound(ncid, mtypeid, "idnumber",
                   NC_COMPOUND_OFFSET(Particle, idnumber), NC_INT64);
int dim_sizes[] = {2,3};
nc_insert_array_compound(ncid, mtypeid, "pressure2",
                         NC_COMPOUND_OFFSET(Particle, pressure2), NC_FLOAT,2,dim_sizes);
int dim_sizes2[] = {16};
nc_insert_array_compound(ncid, mtypeid, "name",
                         NC_COMPOUND_OFFSET(Particle, name), NC_CHAR,1,dim_sizes2);
```

Create compound with C++ (4)

```
nc_def_dim(ncid, "NRecords", DIM_LEN, &dimid);
nc_def_var(ncid, "Data", mtypeid, 1, dimids, &varid);
nc_put_var(ncid, varid, data_in);
nc_close(ncid);

cout << "*** SUCCESS writting example file "<<FILE_NAME<<"!" << endl;
return 0;
}
```

```
module load gcc/4.8.1 hdf5/187-v18-serial-gcc netcdf/4.1.3_hdf5_serial-gcc
gcc -I$SCINET_NETCDF_INC Test.cpp -o Test -lnetcdf_c++
./Test

*** SUCCESS writting example file particles.nc!


ls -ltrh particles.nc
-rw-r--r-- 1 brelier scinet  6.2K Feb 20 12:02 particles.nc
```

Reading compound with C++

```
#include <iostream>
#include <netcdfcpp.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

using namespace std;

#define FILE_NAME "particles.nc"
#define DIM_LEN 10 //number of records in file
int main(void)
{
  typedef struct Particle
  {
    char   name[16];
    unsigned short int    ADCcount;
    int grid_i;
    int grid_j;
    float pressure;
    double energy ;
    long  idnumber;
    float pressure2[2][3];
  } Particle;

  //Definition of the data_in variable
  Particle data_in[DIM_LEN];
```

Reading compound with C++

```
//Initialization of the variable
for (int i=0; i<DIM_LEN; i++){
  data_in[i].ADCcount=0;
  data_in[i].grid_i=0;
  data_in[i].grid_j=0;
  data_in[i].pressure=0.;
  data_in[i].energy=0.;
  for (int j=0; j<2; j++){
    for (int k=0; k<3; k++){
      data_in[i].pressure2[j][k]=0.;
    }
  }
}

//Variables needed to open the NetCDF file :
int ncid, typeidd, varid;
int dimid;
int dimids[] = {0}, fieldid;

nc_def_dim(ncid, "NRecords", DIM_LEN, &dimid);
nc_def_var(ncid, "Data", typeidd, 1, dimids, &varid);

//Open the NetCDF file :
//NC_NOWRITE for read only, NC_WRITE for read and write
if (nc_open(FILE_NAME, NC_NOWRITE, &ncid)) cout<<"ERROR"<<endl;
//Read the data and fill the variable data_in :
if (nc_get_var(ncid, varid, data_in)) cout<<"ERROR"<<endl;
```

Reading compound with C++

```
  for (int i=0; i<DIM_LEN; i++){
    std::cout<<"ADCcount = "<<data_in[i].ADCcount
             <<" ,idnumber = "<<data_in[i].idnumber
             <<" ,grid_i = "<<data_in[i].grid_i
             <<" ,grid_j = "<<data_in[i].grid_j
             <<" ,pressure = "<<data_in[i].pressure
             <<" ,energy = "<<data_in[i].energy
             <<" ,name = "<<data_in[i].name
             <<std::endl;
    for(int j=0;j<2;j++){
      std::cout<<data_in[i].pressure2[j][0]<<" "<<data_in[i].pressure2[j][1]<<" "<<data_in[i].pre
    }
  }
  cout << "*** SUCCESS reading example file "<<FILE_NAME<<"!" << endl;
  return 0;
}
```

```
module load gcc/4.8.1 hdf5/187-v18-serial-gcc netcdf/4.1.3_hdf5_serial-gcc
gcc -I$SCINET_NETCDF_INC Test.cpp -o Test -lnetcdf_c++
./Test

ADCcount = 0 ,idnumber = 0 ,grid_i = 0 ,grid_j = 10 ,pressure = 0 ,energy = 0 ,name = Particle:      0
0.5 1.5 2.5
-1.5 -2.5 -3.5
...
```

NetCDF and R

RNetCDF: R Interface to NetCDF Datasets
provides an interface to Unidata's NetCDF library functions (version 3)
http://cran.r-project.org/web/packages/RNetCDF/index.html

Presently developping an interface to NetCDF version 4

```
module load intel/14.0.1 R/3.0.1 gcc/4.8.1 hdf5/1811-v18-serial-gcc netcdf/4.2.1.1_serial-gcc udunits/2.1.11

R CMD INSTALL --configure-args="--with-netcdf-include='$SCINET_NETCDF_INC' --with-netcdf-lib='$SCINET_NETCDF_LIB'
 --with-udunits-include='$SCINET_UDUNITS_INC' --with-udunits-lib='$SCINET_UDUNITS_LIB' " RNetCDF_2.0.tar.gz
```

Add-on-packages can be written in C.

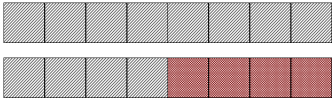
Issues with interface in C

- As we saw in the C++ examples, struct used to define the compound variable.
- Issue arrised when I tried to generalize the interface to a generic compound type :
- For a specific compound size, I did not want to code every possible combination of variables :
  - example with size = 8 bytes : 1 double, 2 floats, 2 int, 1float + 1int, 8 characters ...
- solution : use union in C
- unions allow to share same memory for several variables.

```
struct s1
{
  union{
    int i[2];
    float f[2];
    double d[1];
    short myShort[4];
    unsigned short myUnSignedShort[4];
    unsigned int myUnSignedInt[2];
    long mylong[1];
    unsigned long myUnSignedlong[1];
    char mybyte[8];
  }u;
};
struct s1 data;
data.u.d[0]=5.;
```

1 double :

with 1 int :



```
s1 Data;
Data.u.d[0]=5.;
std::cout<<Data.u.d[0]<<std::endl;
Data.u.i[1]=5;
std::cout<<Data.u.d[0]<<std::endl;
```

prints :

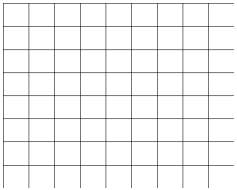
```
5
1.061e-313
```

Issues with interface in C

- Problem with struct : need to create one struct per struct size to have a generic interface.
- Other solution : allocate memory for the data and fill the compound variable by byte.

```
int DIM_LEN = INTEGER(Nrecords)[0];

char **data = (char **)malloc(sizeof(char*)*DIM_LEN);
data[0] = (char **)malloc(sizeof(char)*DIM_LEN * INTEGER(size)[0] );
for(int i=0;i<DIM_LEN;i++) {
  data[i]=&data[0][i*INTEGER(size)[0]];
 }
```



```
nc_put_var(mycid, myvarid,&data[0][0]);

free(data[0]);
free(data);
```

Create compound with R

```
Data <- data.frame(name=I(list()) , ADCcount=I(list())  , grid_i=I(list()) , grid_j=I(list()) ,pressure=I(list())
    ,energy=I(list()) ,idnumber=I(list()) , pressure2=I(list()) )
n<-10
for(i in 1:n){
Data[[i,'name']]=paste("Particle :     ", (i-1), sep = "")
Data[[i,'ADCcount']]=i*256
Data[[i,'grid_i']]=i+0
Data[[i,'grid_j']]=10-i
Data[[i,'pressure']]=i*i*1.
Data[[i,'energy']]=(i*i)^4
Data[[i,'idnumber']]=i*(2)^34
Data[[i,'pressure2']]=c(0.5+i,1.5+i,2.5+i,i-1.5,i-2.5,i-3.5)
}

library(RNetCDF)
nc <- create.nc("particles.nc","NC_NETCDF4")
mycompound <- compound.def.nc(nc,70,"Particle")
compound.inq.nc(nc,mycompound)
mycompound <- compound.insert.nc(nc,mycompound,"name","NC_CHAR",16)
mycompound <- compound.insert.nc(nc,mycompound,"ADCcount","NC_USHORT")
mycompound <- compound.insert.nc(nc,mycompound,"grid_i","NC_INT")
mycompound <- compound.insert.nc(nc,mycompound,"grid_j","NC_INT")
mycompound <- compound.insert.nc(nc,mycompound,"pressure","NC_FLOAT")
mycompound <- compound.insert.nc(nc,mycompound,"energy","NC_DOUBLE")
mycompound <- compound.insert.nc(nc,mycompound,"idnumber","NC_INT64")
mycompound <- compound.insert.nc(nc,mycompound,"pressure2","NC_FLOAT",2,3)

dim.def.nc(nc,"NRecords",10)
myvar <- var.def.nc(nc,mycompound,"Data",1)
compound.fill.nc(nc,mycompound,myvar,Data)
close.nc(nc)
```

Read compound in R

```
> library(RNetCDF)
> df <- get.compound.nc("particles.nc","NRecords","Data")
> df
                name ADCcount grid_i grid_j pressure    energy idnumber
1  Particle :     0      256      1      9        1         1       16
2  Particle :     1      512      2      8        4       256       32
3  Particle :     2      768      3      7        9      6561       48
4  Particle :     3     1024      4      6       16     65536       64
5  Particle :     4     1280      5      5       25    390625       80
6  Particle :     5     1536      6      4       36   1679616       96
7  Particle :     6     1792      7      3       49   5764801      112
8  Particle :     7     2048      8      2       64  16777216      128
9  Particle :     8     2304      9      1       81  43046721      144
10 Particle :     9     2560     10      0      100 100000000      160
```

Package is experimental : multidimension variables not working to read from NetCDF : issue is not reading the multivariable from NetCDF but to create a multidimension variable in the dataframe in the C interface.

Conclusion

- Presented a few examples to read and write a compound variable in NetCDF http://wiki.scinethpc.ca/wiki/index.php/NetCDF_table
- Many different types supported in NetCDF : int, float, double, enum ...
- NetCDF for I/O : http://wiki.scinethpc.ca/wiki/images/a/af/Netcdfhdf5.pdf
- Parallel netCDF: A Parallel I/O Library for NetCDF File Access. Compound variables (actually HDF5) isn't compatible with parallel-netcdf.
- parallel I/O facilities in NetCDF-4 : nc_open_par()