Rochdf/phdf Users and Deverlopers Guide

Xiangmin Jiao

December 12, 2003

Contents

1 Users Guide

Rochdf and Rocphdf are service modules of Roccom. They perform HDF for visualization using Rocketeer and for checkpointing. Rochdf performs IO in serial, while Rocphdf performs IO in the background by forking a pthread. These two libraries have the same semantics but Rocphdf delivers better performance by allowing overlapping I/O with computation.

Both Rochdf and Rocphdf provide three main interface functions: write_attribute, read_attribute, and sync. The first two functions are self-explanatory. The third function requests the process to wait for write operations to finish (for Rocphdf). Section 3 explains the interfaces of these functions.

In general, Rochdf and Rocphdf should be used through Roccom. A typical application would first load either Rochdf or Rocphdf to Roccom by calling COM_load_module("Rochdf", "HDF") or COM_load_module("Rocphdf", "HDF"), which locates and invokes Rochdf_load_module() and Rocphdf_load_module(), respectively. Second, it need to obtain function handles to "HDF.write_attribute" and "HDF.read_attribute" using COM_get_function_handle, and then pass the function handles and arguments to these functions to COM_call_function to invoke their execution, where the argument *attr* should be replaced by an attribute handle obtained by calling COM_get_attribute_handle. The readers are referred to Roccom's Users Guide for details on these Roccom calls.

The following is a sample code fragment in C++ for using Rochdf through Roccom.

```
#include "roccom.h"

// Initialize Roccom and Rochdf
COM_init( &argc, &argv);
COM_load_module("Rochdf", "HDF");

... create a window named "str" with attribute "vel" and "disp" ...

// Obtain handles to the functions and attributes
```

```
int hdf_write = COM_get_function_handle( "HDF.write_attribute");
int hdf_read  = COM_get_function_handle( "HDF.read_attribute");
int str_all = COM_get_attribute_handle( "str.all");
int str_mesh = COM_get_attribute_handle( "str.mesh");
int str_velo = COM_get_attribute_handle( "str.velo");

// Write out the mesh data and all the attributes in window str
// (include vel and disp) into a file with prefix "str_fields".
// The material type of "str", the time stamp is "000".
COM_call_function( hdf_write, "str_fields",
                   str_all, "str", "000", "w");

// Write out the mesh data of window str into a file
// with prefix "str_mesh". The time stamp is "001".
COM_call_function( hdf_write, "str_mesh",
                   str_mesh, "str", "001", "w");

// Write out the attribute vel of window str into a file with prefix
//  "str_vel", which make reference to the mesh data output above.
COM_call_function( hdf_write, "str_vel",
                   str_vel, "str", "001", "w", "str_mesh");

...

// Read back in the attribute vel of window str.
COM_call_function( hdf_read, "str_vel",
                   str_vel, "str", "001", "str_mesh");

// Read back in the mesh data (only the coordinates; omit connectivities)
// of window str.
COM_call_function( hdf_read, "str_mesh",
                   str_mesh, "str", "001");

// Finalize Rochdf and Roccom
Rochdf_unload_module("HDF");
COM_finalize();
```

See hdftest1.C for a complete C++ sample code.

A sample Fortran 90 code fragment is given as follows.

```
#include "roccomf90.h"

! Initialize Roccom and Rochdf
CALL COM_init
CALL COM_load_module("Rocphdf", "HDF")

... create a window named "str" with attribute "vel" and "disp" ...

! Obtain handles to the functions and attributes
HDF_WRITE = COM_get_function_handle( "HDF.write_attribute")
HDF_READ  = COM_get_function_handle( "HDF.read_attribute")
STR_ALL = COM_get_attribute_handle( "str.all")
STR_MESH = COM_get_attribute_handle( "str.mesh")
STR_VELO = COM_get_attribute_handle( "str.velo")
```

```
! Write out the mesh data and all the attributes in window str
! (include vel and disp) into a file with prefix "str_fields".
! The material type of "str", the time stamp is "000".
CALL COM_call_function( HDF_WRITE, 5, "str_fields", &
     STR_ALL, "str", "000", "w")

! Write out the mesh data of window str into a file
! with prefix "str_mesh". The time stamp is "001".
CALL COM_call_function( HDF_WRITE, 5, "str_mesh", &
     STR_MESH, "str", "001", "w")

! Write out the attribute vel of window str into a file with prefix
!  "str_vel", which make reference to the mesh data output above.
CALL COM_call_function( HDF_WRITE, 6, "str_vel", &
     STR_VEL, "str", "001", "w", "str_mesh")

...

! Read back in the attribute vel of window str.
CALL COM_call_function( HDF_READ, 6, "str_vel", &
     STR_VEL, "str", "001", "w", "str_mesh")

! Read back in the mesh data of window str.
CALL COM_call_function( HDF_READ, 5, "str_mesh", &
     STR_MESH, "str", "001", "w")

! Finalize Rochdf and Roccom
CALL COM_unload_module("Rocphdf")
CALL COM_finalize
```

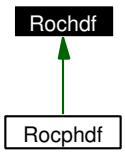
See hdftest2.f90 and modtest.f90 for a complete F90 sample code.

2 Class Documentation

2.1 Rochdf Class Reference

`#include <Rochdf.h>`

Inheritance diagram for Rochdf:



2.1.1 Detailed Description

Serial HDF IO for visualization (using Rocketeer) and checkpointing.

Examples:
hdftest2.f90.

Static Public Member Functions

User interface

- void write_attribute (const char *fname_pre, const COM::Attribute *attr, const char *material, const char *timelevel, const char *io_mode, const char *mfile_pre=NULL)
  Write an attribute or a collection of attributes into an HDF file, whose name is <fname_pre>_<dddd>.hdf, where <dddd> is the rank of the given MPI process.

- void read_attribute (const char *fname_pre, COM::Attribute *attr, const char *material, const char *timelevel, const char *mfile_pre=NULL)
  Read an attribute or a collection of attributes from an HDF file, whose name is <fname_pre>_<dddd>.hdf, where <dddd> is the rank of the given MPI process.

- void sync ()
  This function is supplied for generality. It does nothing for serial IO.

Static Protected Member Functions

Initialization and finalization

- void init (const std::string &mname)
  Initialize the module by registering it to Roccom with the given module name.

- void finalize (const std::string &mname)
  Finalize the module by deregistering it from Roccom.

IO helpers

- void io_attribute (const char *fname_pre, COM::Attribute *attr, const char *material, const char *timelevel, const char *mfile_pre, int mode, int rank=-1)
  Read/write an attribute from/to a given file.

Friends

Module loading and unloading

- void Rochdf_load_module (const char *name)
  Load the module Rochdf into Roccom using the given module name.
- void Rochdf_unload_module (const char *name)
  Unload the module Rochdf from Roccom.

2.1.2 Member Function Documentation

2.1.2.1 void Rochdf::write_attribute (const char * *fname_pre*, const COM::Attribute * *attr*, const char * *material*, const char * *timelevel*, const char * *io_mode*, const char * *mfile_pre* = NULL) [static]

Write an attribute or a collection of attributes into an HDF file, whose name is <fname_pre>_<dddd>.hdf, where <dddd> is the rank of the given MPI process.

If the given attribute corresponds to Window.all, then this routine will dump all the attributes of a window associated with the panes, nodes, or elements (i.e., everything except for window attributes) into the HDF file. If it is some panel, nodal, or elemental attribute, it will dump the attribute into the file. In the above two cases, this routine will also dump the mesh data with the attributes if the argument mfile_pre is empty; otherwise, it will annotate the HDF file to refer to the mesh in the HDF file <mfile_pre>_<dddd>.hdf. If the attribute corresponds to Window.mesh, this routine will dump only the mesh (including nodal coordinates and element connectivity) into the file.

Parameters:

fname_pre the prefix of the HDF filename, which can contain the directory path of the file.

attr the attribute to be written.

material the material name for Rocketeer. Typically, it should be set to the window name.

timelevel a time stamp of the dataset.

io_mode indicates whether to overwrite (io_mode==”w”) or append (io_mode==”a”)

mfile_pre the prefix of the name of the HDF file that contains the mesh data of the given attribute. Set mfile_pre==NLLL (by default if the argument is missing) or mfile_pre==”” if the mesh should be written with the attribute.

2.1.2.2 void Rochdf::read_attribute (const char * *fname_pre*, COM::Attribute * *attr*, const char * *material*, const char * *timelevel*, const char * *mfile_pre* = NULL) [static]

Read an attribute or a collection of attributes from an HDF file, whose name is <fname_pre>_<dddd>.hdf, where <dddd> is the rank of the given MPI process.

The semantics of this routine and its parameters are symmetric with those of write_attribute.

Parameters:

fname_pre the prefix of the HDF filename, which can contain the directory path of the file.

attr the attribute to be read.

material the material name for Rocketeer. Typically, it should be set to the window name.

timelevel a time stamp of the dataset.

mfile_pre the prefix of the name of the HDF file that contains the mesh data of the given attribute. Set mfile_pre==NLLL (by default if the argument is missing) or mfile_pre=="" if the mesh should be read with the attribute.

See also:

write_attribute()

2.1.2.3 void Rochdf::sync () [static]

This function is supplied for generality. It does nothing for serial IO.

2.1.2.4 void Rochdf::init (const std::string & *mname*) [static, protected]

Initialize the module by registering it to Roccom with the given module name.

This function is called Rochdf_load_module.

See also:

Rochdf_load_module()

Reimplemented in Rocphdf.

2.1.2.5 void Rochdf::finalize (const std::string & *mname*) [static, protected]

Finalize the module by deregistering it from Roccom.

See also:
Rochdf_unload_module()

Reimplemented in Rocphdf.

2.1.2.6 void Rochdf::io_attribute (const char * *fname_pre*, COM::Attribute * *attr*, const char * *material*, const char * *timelevel*, const char * *mfile_pre*, int *mode*, int *rank* = -1) [static, protected]

Read/write an attribute from/to a given file.

The parameter mode indicates whether to read (if <0), overwrite (if 0), or append (if >0).

2.1.3 Friends And Related Function Documentation

2.1.3.1 void Rochdf_load_module (const char * *name*) [friend]

Load the module Rochdf into Roccom using the given module name.

This module provides three subroutines: "write_attribute", and "read_attribute", and "sync".

2.1.3.2 void Rochdf_unload_module (const char * *name*) [friend]

Unload the module Rochdf from Roccom.

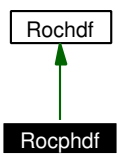
The documentation for this class was generated from the following files:

- Rochdf.h
- Rochdf.C

2.2 Rocphdf Class Reference

`#include <Rocphdf.h>`

Inheritance diagram for Rocphdf:



2.2.1 Detailed Description

Pthread background HDF IO for visualization (using Rocketeer) and checkpointing.

Static Public Member Functions

User interface

- void write_attribute (Rocphdf *ptr, const char *fname_pre, const COM::Attribute *attr, const char *material, const char *timelevel, const char *io_mode, const char *mfile_pre=NULL)
  This function has the same semantics as Rochdf::write_attribute, except that the data may not have been completely written into the HDF file when the call returns.

- void read_attribute (Rocphdf *ptr, const char *fname_pre, COM::Attribute *attr, const char *material, const char *timelevel, const char *mfile_pre=NULL)
  This function has the same semantics as Rochdf::read_attribute.

- void sync (Rocphdf *)
  It requests the process to wait for the completion of all pending write operations.

Static Protected Member Functions

Initialization and finalization

- void init (const std::string &mname)
  This function is called Rochdf_load_module.
  See also:
  Rochdf_load_module()

- void finalize (const std::string &mname)
  See also:
  Rochdf_unload_module()

IO helpers

- void output_entry (Rocphdf *rhdf)
  The pthread entry of the background IO thread.

Friends

Module loading and unloading

- void Rocphdf_load_module (const char *name)
  This module provides three subroutines: "write_attribute", and "read_attribute", and "sync".
- void Rocphdf_unload_module (const char *name)

2.2.2 Member Function Documentation

2.2.2.1 void Rocphdf::write_attribute (Rocphdf * *ptr*, const char * *fname_pre*, const COM::Attribute * *attr*, const char * *material*, const char * *timelevel*, const char * *io_mode*, const char * *mfile_pre* = NULL) [static]

This function has the same semantics as Rochdf::write_attribute, except that the data may not have been completely written into the HDF file when the call returns.

It takes one more argument than Rochdf::write_attribute, *ptr*, which is a pointer to a Rocphdf object.

See also:
Rochdf::write_attribute()

2.2.2.2 void Rocphdf::read_attribute (Rocphdf * *ptr*, const char * *fname_pre*, COM::Attribute * *attr*, const char * *material*, const char * *timelevel*, const char * *mfile_pre* = NULL) [static]

This function has the same semantics as Rochdf::read_attribute.

It takes one more argument than Rochdf::read_attribute, *ptr*, which is a pointer to a Rocphdf object.

See also:
Rochdf::read_attribute()

2.2.2.3 void Rocphdf::sync (Rocphdf *) [static]

It requests the process to wait for the completion of all pending write operations.

2.2.2.4 void Rocphdf::init (const std::string & *mname*) [static, protected]

This function is called Rochdf_load_module.

See also:
Rochdf_load_module()

See also:
Rocphdf_load_module()

Reimplemented from Rochdf.

2.2.2.5 void Rocphdf::finalize (const std::string & *mname*) [static, protected]

See also:
Rochdf_unload_module()

See also:
Rocphdf_unload_module()

Reimplemented from Rochdf.

2.2.2.6 void Rocphdf::output_entry (Rocphdf * *rhdf*) [static, protected]

The pthread entry of the background IO thread.

2.2.3 Friends And Related Function Documentation

2.2.3.1 void Rocphdf_load_module (const char * *name*) [friend]

This module provides three subroutines: "write_attribute", and "read_attribute", and "sync".

2.2.3.2 void Rocphdf_unload_module (const char * *name*) [friend]

The documentation for this class was generated from the following files:

- Rocphdf.h
- Rocphdf.C

3 File Documentation

3.1 Rochdf File List

Here is a list of all files with brief descriptions:

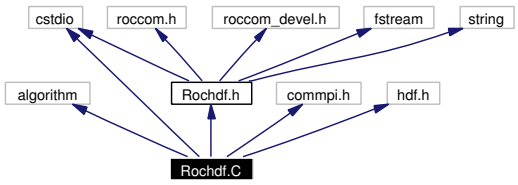
3.2 Rochdf.C File Reference

3.2.1 Detailed Description

Implementation of Rochdf for serial HDF IO.

```
#include <algorithm>
#include <cstdio>
#include "Rochdf.h"
#include "commpi.h"
#include <hdf.h>
```

Include dependency graph for Rochdf.C:



Functions

- void Rochdf_load_module (const char *name)

  Load the module Rochdf into Roccom using the given module name.

- void Rochdf_unload_module (const char *name)

  Unload the module Rochdf from Roccom.

- std::vector< char > get_fname (const char *pre, int rank=-1, bool check=false)

  Get a file name by appending an underscore, a 4-digit rank id, and ".hdf" to the given prefix.

- void append_str (std::vector< char > &vec, const char *str)
- template<class T> const T * min_element__ (const T *begin, const int rank, const int shape[ ], const int ng1, const int ng2)

  Template implementation for determining the minimum entry in an array.

- template<class T> const T * max_element__ (const T *begin, const int rank, const int shape[ ], const int ng1, const int ng2)

  Template implementation for determining the maximum entry in an array.

- void hdf_error_message (const char *s, int i)

3.2.2 Function Documentation

3.2.2.1 void Rochdf_load_module (const char * *name*)

Load the module Rochdf into Roccom using the given module name.

This module provides three subroutines: "write_attribute", and "read_attribute", and "sync".

3.2.2.2 void Rochdf_unload_module (const char * *name*)

Unload the module Rochdf from Roccom.

3.2.2.3 std::vector<char> get_fname (const char * *pre*, int *rank* = -1, bool *check* = false) [static]

Get a file name by appending an underscore, a 4-digit rank id, and ".hdf" to the given prefix.

3.2.2.4 void append_str (std::vector< char > & *vec*, const char * *str*) [inline]

3.2.2.5 template<class T> const T* min_element__ (const T * *begin*, const int *rank*, const int *shape*[ ], const int *ng1*, const int *ng2*)

Template implementation for determining the minimum entry in an array.

The array can contain ghost layers and the entries in the ghost layers are omitted.

3.2.2.6 template<class T> const T* max_element__ (const T * *begin*, const int *rank*, const int *shape*[ ], const int *ng1*, const int *ng2*)

Template implementation for determining the maximum entry in an array.

The array can contain ghost layers and the entries in the ghost layers are omitted.

3.2.2.7 void hdf_error_message (const char * *s*, int *i*)

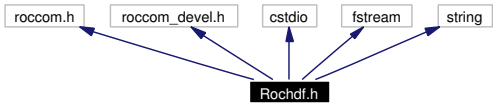
3.3 Rochdf.h File Reference

3.3.1 Detailed Description

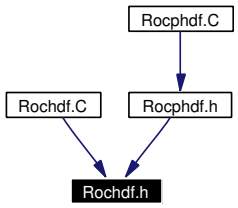
Serial HDF IO for visualization (using Rocketeer) and checkpointing.

```
#include "roccom.h"
#include "roccom_devel.h"
#include <cstdio>
#include <fstream>
#include <string>
```

Include dependency graph for Rochdf.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class Rochdf

  Serial HDF IO for visualization (using Rocketeer) and checkpointing.

Module loading and unloading

- void Rochdf_load_module (const char *name)

  Load the module Rochdf into Roccom using the given module name.

- void Rochdf_unload_module (const char *name)

  Unload the module Rochdf from Roccom.

3.3.2 Function Documentation

3.3.2.1 void Rochdf_load_module (const char * *name*)

Load the module Rochdf into Roccom using the given module name.

This module provides three subroutines: "write_attribute", and "read_attribute", and "sync".

3.3.2.2 void Rochdf_unload_module (const char * *name*)

Unload the module Rochdf from Roccom.

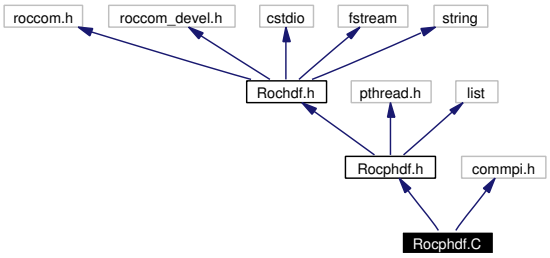
3.4 Rocphdf.C File Reference

3.4.1 Detailed Description

Wrapper for pthread-version of Rochdf.

```
#include "Rocphdf.h"
#include "commpi.h"
```

Include dependency graph for Rocphdf.C:



Compounds

- class Rocphdf::Write_Args

Functions

- void Rocphdf_load_module (const char *name)

  This module provides three subroutines: "write_attribute", and "read_attribute", and "sync".

- void Rocphdf_unload_module (const char *name)

Variables

- pthread_mutex_t mutex_io

3.4.2 Function Documentation

3.4.2.1 void Rocphdf_load_module (const char * *name*)

This module provides three subroutines: "write_attribute", and "read_attribute", and "sync".

3.4.2.2 void Rocphdf_unload_module (const char * *name*)

3.4.3 Variable Documentation

3.4.3.1 pthread_mutex_t mutex_io [static]

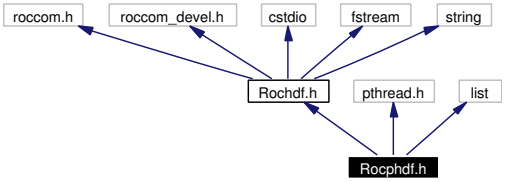
3.5 Rocphdf.h File Reference

3.5.1 Detailed Description

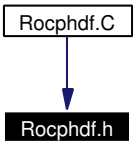
Pthread background IO of HDF files for visualization (using Rocketeer) and checkpointing.

```
#include "Rochdf.h"
#include <pthread.h>
#include <list>
```

Include dependency graph for Rocphdf.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class Rocphdf

  Pthread background HDF IO for visualization (using Rocketeer) and checkpointing.

Module loading and unloading

- void Rocphdf_load_module (const char *name)

  This module provides three subroutines: "write_attribute", and "read_attribute", and "sync".

- void Rocphdf_unload_module (const char *name)

3.5.2 Function Documentation

3.5.2.1 void Rocphdf_load_module (const char * *name*)

This module provides three subroutines: ”write_attribute”, and ”read_attribute”, and ”sync”.

3.5.2.2 void Rocphdf_unload_module (const char * *name*)

4 Example Documentation

4.1 hdftest1.C

A complete sample C++ program for using Rochdf and Roccom.

```
// $Id: hdftest1.C,v 1.15 2003/11/24 21:38:23 jiao Exp $

// An example that uses Roccom API for data registration and uses Rochdf
//    for writing structured and unstructured meshes into Rocketeer-
//    compatible HDF files and reading them back in.

// Author: Xiangmin Jiao
// Initial Date  : May 11, 2001
// Last modified : Jan. 31, 2002

#include "roccom.h"
#include <iostream>
#include <cstdlib>
#include <cassert>
#include "commpi.h"

// ==== Routines for creating mesh information
void init_structured_mesh( double coors[][3], int nrow, int ncol,
                           int rank, int nproc, int check=1) {
  // consider the processors as a 2*(nproc/2) grid
  int proc_col=nproc;
  if (nproc%2 ==0) {
    proc_col=nproc/2;
  }
  else {
    proc_col=nproc;
  }
  int row=rank/proc_col, col=rank%proc_col;

  const int width=300/(nrow-1), length=300/(ncol-1);
```

```
  int x0 = col*300, y0=row*300, y1=y0;
  for (int i=0; i<nrow; ++i) {
    int x1 = x0;
    for (int j=0; j<ncol; ++j) {
      if ( check<0)
        assert( coors[i*ncol+j][0]==x1 && coors[i*ncol+j][1]==y1 &&
                coors[i*ncol+j][2]==0.5);
      else {
        coors[i*ncol+j][0]=x1*check;
        coors[i*ncol+j][1]=y1*check;
        coors[i*ncol+j][2]=0.5;
      }
      x1 += length;
    }
    y1 += width;
  }
}

void init_unstructured_mesh( double coors[][3], int elmts[][4], int nrow,
                             int ncol, int rank, int nproc, int check=1) {
  // consider the processors as a 2*(nproc/2) grid
  int proc_col=nproc;
  if (nproc%2 ==0) {
    proc_col=nproc/2;
  }
  else {
    proc_col=nproc;
  }

  int row=rank/proc_col, col=rank%proc_col;

  const int width=300/(nrow-1), length=300/(ncol-1);

  int x0 = col*300, y0=row*300, y1=y0;
  for (int i=0; i<nrow; ++i) {
    int x1 = x0;
    for (int j=0; j<ncol; ++j) {
      if ( check<0)
        assert( coors[i*ncol+j][0]==x1 && coors[i*ncol+j][1]==y1 &&
                coors[i*ncol+j][2]==0.0);
      else {
        coors[i*ncol+j][0]=x1*check;
        coors[i*ncol+j][1]=y1*check;
        coors[i*ncol+j][2]=0.0;
      }
      x1 += length;
    }
    y1 += width;
  }

  // computes elmts
  for (int i=0; i<nrow-1; ++i) {
    for (int j=0; j<ncol-1; ++j) {
      if ( check<0) {
        assert( elmts[i*(ncol-1)+j][0]==i*ncol+j+1 &&
                elmts[i*(ncol-1)+j][1]==i*ncol+j+ncol+1 &&
                elmts[i*(ncol-1)+j][2]==i*ncol+j+ncol+2 &&
```

```
                elmts[i*(ncol-1)+j][3]==i*ncol+j+2);
      }
      else if (check>0) {
        elmts[i*(ncol-1)+j][0]=check*i*ncol+j+1;
        elmts[i*(ncol-1)+j][1]=check*i*ncol+j+ncol+1;
        elmts[i*(ncol-1)+j][2]=check*i*ncol+j+ncol+2;
        elmts[i*(ncol-1)+j][3]=check*i*ncol+j+2;
      }
    }
  }
}

void initialize_data( const char *wname, const char *attr, int check=1) {

  int npanes;

  COM_get_window_npanes( wname, &npanes);

  std::vector<int> pane_ids(npanes);
  std::vector<int> nnodes(npanes);
  std::vector<int> nelems(npanes);

  COM_get_window_panes( wname, &pane_ids[0], &npanes);
  COM_get_window_nnodes( wname, &nnodes[0], &npanes);
  COM_get_window_nelems( wname, &nelems[0], &npanes);

  std::string wa(wname); wa.append("."); wa.append(attr);

  char loc, unit[10];
  int type, dim2, ulen=9;
  COM_get_attribute_info( wa.c_str(), &loc, &type, &dim2, unit, &ulen);

  if ( dim2 > 0) {
    for ( int i=0; i<npanes; ++i) {
      void *ptr;
      int   size;
      COM_get_attribute_addr( wa.c_str(), pane_ids[i], &ptr);
      if ( loc == 'n') size = nnodes[i]; else size = nelems[i];
      for ( int j=0; j<size; ++j)
        for ( int k=0; k<dim2; ++k)
          if ( check<0) assert( ((double*)ptr)[j*dim2+k] == j);
          else ((double*)ptr)[j*dim2+k] = j*check;
    }
  }
  else {
    for ( int i=0; i<npanes; ++i) {
      void *ptr;
      int   size;
      COM_get_attribute_addr( wa.c_str(), pane_ids[i], &ptr);
      if ( loc == 'n') size = nnodes[i]; else size = nelems[i];

      for ( int k=0; k<-dim2; ++k)
        for ( int j=0; j<size; ++j)
          if ( check<0) assert( ((double*)ptr)[k*size+j] == j);
          else ((double*)ptr)[k*size+j] = j*check;
    }
  }
```

```
}

int main(int argc, char *argv[]) {
  const int nproc=4;
  const int nrow=40, ncol=30;
  const int num_nodes=nrow*ncol;
  const int num_elmts=(nrow-1)*(ncol-1);

  double  coors_s[nproc][num_nodes][3];
  double  coors_f[nproc][nrow*ncol][3];
  int     elmts[nproc][num_elmts][4];
  int     one=1, zero=0;

  std::cout << "sizeof(long int) is " << sizeof(long int)
            << " and sizeof(void*) is " << sizeof(void*) << std::endl;

  std::cout << "Initializing Roccom and Rochdf" << std::endl;
  COM_init( &argc, &argv);
  int vb = (argc>2) ? std::atoi(argv[2]) : 1;
  COM_set_verbose(vb );

  std::string modname = ( argc>1 && std::string(argv[1]) == "-p") ?
    "Rocphdf": "Rochdf";

  std::cout << "Loading " << modname << std::endl;
  COM_load_module( modname.c_str(), "HDF");

  std::cout << "Creating window \"unstr\"" << std::endl;
  COM_create_window("unstr_mesh");
  COM_new_attribute("unstr_mesh.empty", 'e', COM_DOUBLE, 1, "m/s");
  COM_init_mesh( "unstr_mesh.unit", 0, "meter", 5);

  for ( int pid=0; pid<nproc; ++pid) {
    init_unstructured_mesh( coors_s[pid], elmts[pid],
                            nrow, ncol, pid, nproc);
    if ( pid%2==0) {
      COM_init_mesh( "unstr_mesh.nc", pid+1, &coors_s[pid][0][0], num_nodes);
    }
    else {
      COM_init_mesh( "unstr_mesh.n-c", pid+1, (void*)NULL, num_nodes);
      COM_init_attribute_strided( "unstr_mesh.1-nc", pid+1,
                                  &coors_s[pid][0][0], 3);
      COM_init_attribute_strided( "unstr_mesh.2-nc", pid+1,
                                  &coors_s[pid][0][1], 3);
      COM_init_attribute_strided( "unstr_mesh.3-nc", pid+1,
                                  &coors_s[pid][0][2], 3);
    }
    COM_init_mesh( "unstr_mesh.q4", pid+1, &elmts[pid][0][0], num_elmts);
    COM_init_attribute( "unstr_mesh.empty", pid+1, NULL);
  }
  COM_window_init_done( "unstr_mesh");

  // Use the mesh and attributes from unstr_mesh
  COM_create_window( "unstr");
  COM_use_mesh( "unstr", "unstr_mesh");

  COM_new_attribute("unstr.velo", 'n', COM_DOUBLE, 3, "m/s");
```

```
COM_new_attribute("unstr.load", 'n', COM_DOUBLE, 3, "Newton");
COM_new_attribute("unstr.burnr", 'e', COM_DOUBLE, -3, "m/s");
COM_use_attribute( "unstr", "unstr_mesh.all");

// Allocate space for all the attributes in unstr
COM_allocate_attribute( "unstr.all");
COM_window_init_done( "unstr");

initialize_data( "unstr", "velo");
initialize_data( "unstr", "load");
initialize_data( "unstr", "burnr");

std::cout << "Creating window \"str\" by inheritance" << std::endl;
COM_create_window( "str_mesh");
COM_init_mesh("str_mesh.unit", 0, "meter", 5);
COM_new_attribute("str_mesh.coors", 'n', COM_DOUBLE, 3, "m");
COM_new_attribute("str_mesh.bcflag", 'p', COM_INT, 1, "");

for ( int pid=0; pid<nproc; ++pid) {
  init_structured_mesh(coors_f[pid], nrow, ncol, pid, nproc);

  COM_init_mesh("str_mesh.nc", pid+1, &coors_f[pid][0][0], nrow*ncol);
  int dims[2] = { ncol, nrow}; // Use Fortran convention
  COM_init_mesh("str_mesh.st", pid+1, &dims[0], 2);

  COM_init_attribute("str_mesh.coors", pid+1, &coors_f[pid][0][0]);
  COM_init_attribute("str_mesh.bcflag", pid+1, &one);
}

COM_init_mesh("str_mesh.nc", nproc+1, &coors_f[0][0][0], nrow*ncol);
int dims[2] = { ncol, nrow}; // Use Fortran convention
COM_init_mesh("str_mesh.st", nproc+1, &dims[0], 2);

COM_init_attribute("str_mesh.coors", nproc+1, &coors_f[0][0][0]);
COM_init_attribute("str_mesh.bcflag", nproc+1, &zero);

COM_new_attribute("str_mesh.load", 'n', COM_DOUBLE, 3, "Newton");
COM_new_attribute("str_mesh.burnr", 'e', COM_DOUBLE, -3, "m/s");

COM_allocate_attribute( "str_mesh.all");
COM_window_init_done( "str_mesh");

COM_create_window( "str");
// Use the mesh and attributes from str_mesh
COM_use_mesh_sub( "str", "str_mesh", "str_mesh.bcflag", 1, 0);

COM_use_attribute("str.nc", "str_mesh.coors");
COM_use_attribute("str.load", "str_mesh.load");
COM_use_attribute("str.burnr", "str_mesh.burnr");
COM_new_attribute("str.velo", 'n', COM_DOUBLE, 3, "m/s");

// Allocate space for all the attributes in str
COM_allocate_attribute( "str.all");
COM_window_init_done( "str");

initialize_data( "str", "velo");
initialize_data( "str", "load");
```

```
initialize_data( "str", "burnr");

// Tell Roccom to print out debugging messages and also
// profile the subroutines
COM_init_profiling( 1);
std::cout << "Writing all attributes of the structured mesh to "
          << "file \"str_fields.hdf\"" << std::endl;
int HDF_write = COM_get_function_handle( "HDF.write_attribute");
int HDF_read = COM_get_function_handle( "HDF.read_attribute");
int STR_all = COM_get_attribute_handle( "str.all");
COM_call_function( HDF_write, "str_fields",
                   &STR_all, "str", "000", "w");
for ( int pid=0; pid<nproc; ++pid)
  init_structured_mesh(coors_f[pid], nrow, ncol, pid, nproc, 0);

initialize_data( "str", "velo", 0);
initialize_data( "str", "load", 0);
initialize_data( "str", "burnr", 0);
COM_call_function( HDF_read, "str_fields", &STR_all, "str", "000");
for ( int pid=0; pid<nproc; ++pid)
  init_structured_mesh(coors_f[pid], nrow, ncol, pid, nproc, -1);

initialize_data( "str", "velo", -1);
initialize_data( "str", "load", -1);
initialize_data( "str", "burnr", -1);

std::cout << "Writing mesh of \"unstr\" to file \"unstr_mesh.hdf\""
          << std::endl;
// Omit the last argument to use default (NULL pointers)
int UNSTR_mesh = COM_get_attribute_handle( "unstr.mesh");
COM_call_function( HDF_write, "unstr_mesh",
                   &UNSTR_mesh, "unstr", "000", "w");
for ( int pid=0; pid<nproc; ++pid)
  init_unstructured_mesh( coors_s[pid], elmts[pid],
                          nrow, ncol, pid, nproc, 0);
COM_call_function( HDF_read, "unstr_mesh", &UNSTR_mesh, "unstr", "000");
// Check data
for ( int pid=0; pid<nproc; ++pid)
  init_unstructured_mesh( coors_s[pid], elmts[pid],
                          nrow, ncol, pid, nproc, -1);

std::cout << "Writing velocity of \"unstr\" to file \"unstr_velo.hdf\""
          << std::endl;
int UNSTR_velo = COM_get_attribute_handle( "unstr.velo");
COM_call_function( HDF_write, "unstr_velo",
                   &UNSTR_velo, "unstr", "000", "w", "unstr_mesh");
initialize_data( "unstr", "velo", 0);
COM_call_function( HDF_read, "unstr_velo",
                   &UNSTR_velo, "unstr", "000", "unstr_mesh");
initialize_data( "unstr", "velo", -1);
COM_call_function( HDF_write, "unstr_velo",
                   &UNSTR_velo, "unstr", "001", "a", "unstr_mesh");
initialize_data( "unstr", "velo", 0);
COM_call_function( HDF_read, "unstr_velo",
                   &UNSTR_velo, "unstr", "001", "unstr_mesh");
initialize_data( "unstr", "velo", -1);
```

```
  std::cout << "Writing loads of \"unstr\" to file \"unstr_load.hdf\""
            << std::endl;
  int UNSTR_load = COM_get_attribute_handle( "unstr.load");
  COM_call_function( HDF_write, "unstr_load",
                     &UNSTR_load, "unstr", "000", "w", "unstr_mesh");
  initialize_data( "unstr", "load", 0);
  COM_call_function( HDF_read, "unstr_load",
                     &UNSTR_load, "unstr", "000", "unstr_mesh");
  initialize_data( "unstr", "load", -1);
  COM_call_function( HDF_write, "unstr_load",
                     &UNSTR_load, "unstr", "001", "a", "unstr_mesh");
  initialize_data( "unstr", "load", 0);
  COM_call_function( HDF_read, "unstr_load",
                     &UNSTR_load, "unstr", "001", "unstr_mesh");
  initialize_data( "unstr", "load", -1);

  std::cout << "Writing burnr of \"unstr\" to file \"unstr_burnr.hdf\""
            << std::endl;
  int UNSTR_burnr = COM_get_attribute_handle( "unstr.burnr");
  COM_call_function( HDF_write, "unstr_burnr",
                     &UNSTR_burnr, "unstr", "000", "w", "unstr_mesh");
  initialize_data( "unstr", "burnr", 0);
  COM_call_function( HDF_read, "unstr_burnr",
                     &UNSTR_burnr, "unstr", "000", "unstr_mesh");
  initialize_data( "unstr", "burnr", -1);
  COM_call_function( HDF_write, "unstr_burnr",
                     &UNSTR_burnr, "unstr", "001", "a", "unstr_mesh");
  initialize_data( "unstr", "burnr", 0);
  COM_call_function( HDF_read, "unstr_burnr",
                     &UNSTR_burnr, "unstr", "001", "unstr_mesh");
  initialize_data( "unstr", "burnr", -1);

  std::cout << "Deleting windows" << std::endl;
  COM_delete_window( "str");
  COM_delete_window( "unstr");

  COM_print_profile("", ""); // Print profile data to standard output

  std::cout << "Finalizing Rochdf and Roccom" << std::endl;
  COM_finalize();
}
```

4.2 hdftest2.f90

A complete sample F90 program for using Rochdf and Roccom.

```
! $Id: modtest.f90,v 1.5 2003/12/03 22:20:21 jiao Exp $

!!! This is the Fortran version of hdftest1.C. It should generate identical
!!! HDF files as hdftest1 even when optimization is turned on.

MODULE ModTest

  IMPLICIT NONE
  INCLUDE  "roccomf90.h"
```

```
!!!  This is the data type for encapsulating global data
  TYPE myGlobal
!!!  Data arrays
     DOUBLE PRECISION, POINTER :: COORS_S(:,:,:), COORS_F(:,:,:,:)
     DOUBLE PRECISION, POINTER :: DATA(:,:,:)
     INTEGER, POINTER          :: ELMTS(:,:,:)
     INTEGER, POINTER          :: DIMS(:)

!!!  Function and attribute handles
     INTEGER :: HDF_WRITE, HDF_READ, STR_ALL, UNSTR_MESH
     INTEGER :: UNSTR_VELO, UNSTR_LOAD, UNSTR_BURNR, COOKIE
  END TYPE myGlobal

CONTAINS

  SUBROUTINE ASSOCIATE_POINTER( attr, ptr)
    TYPE(myGlobal), POINTER  :: attr, ptr

    ptr => attr
  END SUBROUTINE ASSOCIATE_POINTER

!!!=============================================================================
!!! A private function for initializing a structured mesh
  SUBROUTINE INIT_STRUCTURED_MESH( COORS, NROW, NCOL, RANK, NPROC)
    INTEGER, INTENT(IN) :: NROW, NCOL, RANK, NPROC
    DOUBLE PRECISION, INTENT(OUT) :: COORS(:,:,:)
    INTEGER             :: PROC_COL, ROW, COL, I, J
    INTEGER             :: WIDTH, LENGTH, X0, X1, Y0, Y1

!!! CONSIDER THE PROCESSORS AS A 2*(NPROC/2) GRID
    PROC_COL=NPROC

    IF ( MOD(NPROC, 2) ==0) THEN
       PROC_COL=NPROC/2
    ELSE
       PROC_COL=NPROC
    END IF

    ROW=RANK/PROC_COL
    COL=MOD(RANK, PROC_COL)

    WIDTH = 300 / (NROW-1)
    LENGTH = 300 / (NCOL-1)
    X0 = COL*300
    Y0 = ROW*300
    Y1 = Y0
    DO I = 1, NROW
       X1 = X0
       DO J = 1, NCOL
          COORS(1,J,I) = X1
          COORS(2,J,I) = Y1
          COORS(3,J,I) = 0.5
          X1 = X1 + LENGTH
       END DO
       Y1 = Y1 + WIDTH
    END DO
```

```
  END SUBROUTINE INIT_STRUCTURED_MESH

!!!========================================================================
!!! A private function for initializing an unstructured mesh
  SUBROUTINE INIT_UNSTRUCTURED_MESH( COORS, ELMTS, NROW, NCOL, RANK, NPROC)
    INTEGER, INTENT(IN)   :: NROW, NCOL, RANK, NPROC
    DOUBLE PRECISION, INTENT(OUT)   :: COORS(:,:)
    INTEGER, INTENT(OUT)  :: ELMTS(:,:)
    INTEGER               :: PROC_COL, ROW, COL, I, J
    INTEGER               :: WIDTH, LENGTH, X0, X1, Y0, Y1


    PROC_COL=NPROC

    IF ( MOD(NPROC, 2) ==0) THEN
       PROC_COL=NPROC/2
    ELSE
       PROC_COL=NPROC
    END IF

    ROW=RANK/PROC_COL
    COL=MOD(RANK, PROC_COL)

    WIDTH = 300 / (NROW-1)
    LENGTH = 300 / (NCOL-1)

    X0 = COL*300
    Y0 = ROW*300
    Y1 = Y0
    DO I = 0, NROW-1
       X1 = X0
       DO J = 1, NCOL
          COORS(1,I*NCOL+J) = X1
          COORS(2,I*NCOL+J) = Y1
          COORS(3,I*NCOL+J) = 0.0
          X1 = X1 + LENGTH
       END DO
       Y1 = Y1 + WIDTH
    END DO

    DO I=0, NROW-2
       DO J=1, NCOL-1
          ELMTS(1, I*(NCOL-1)+J)=I*NCOL+J
          ELMTS(2, I*(NCOL-1)+J)=I*NCOL+J+NCOL
          ELMTS(3, I*(NCOL-1)+J)=I*NCOL+J+NCOL+1
          ELMTS(4, I*(NCOL-1)+J)=I*NCOL+J+1
       END DO
    END DO
  END SUBROUTINE INIT_UNSTRUCTURED_MESH

!!!========================================================================
!!! A private function for initializing data
  SUBROUTINE INITIALIZE_DATA( WNAME, ATTR)
    CHARACTER(*), INTENT(IN)  :: WNAME, ATTR
    CHARACTER(LEN=32)  :: WA, UNIT
    CHARACTER(LEN=1)  :: LOC
    INTEGER       :: NPANES, TYPE, DIM2, ULEN, I, J, K, SIZE, STR
```

```
    INTEGER, ALLOCATABLE :: PANE_IDS(:), NELEMS(:), NNODES(:)
    DOUBLE PRECISION, POINTER :: PTR1D(:), PTR2D(:,:)

    CALL COM_get_window_npanes( WNAME, NPANES)
    ALLOCATE( PANE_IDS( NPANES))
    ALLOCATE( NELEMS( NPANES))
    ALLOCATE( NNODES( NPANES))
    CALL COM_get_window_panes( wname, PANE_IDS, NPANES)
    CALL COM_get_window_nnodes( wname, NNODES, NPANES)
    CALL COM_get_window_nelems( wname, NELEMS, NPANES)

    ULEN = 32
    CALL COM_get_attribute_info( WNAME//'.'//ATTR, LOC, TYPE, DIM2, UNIT, ULEN)

    DO I = 1, NPANES
       IF (LOC=='n') THEN
          SIZE = NNODES(I)
       ELSE
          SIZE = NELEMS(I)
       ENDIF
       IF (dim2 > 0) THEN
          CALL COM_get_attribute_addr( WNAME//'.'//ATTR, PANE_IDS(I), PTR2D)
          DO J = 1, SIZE
             PTR2D(1:DIM2, J) = J-1
          END DO
       ELSE
          DO K = 1, -DIM2
             WRITE (WA, '(A,A,I1,A)') WNAME, '.', K, '-'//ATTR
             CALL COM_get_attribute_addr_strided( TRIM(WA), &
                  PANE_IDS(I), PTR1D, STR)
             DO J = 1, SIZE
                PTR1D((J-1)*STR+1) = J-1
             END DO
          END DO
       END IF
    END DO


    DEALLOCATE( PANE_IDS)
    DEALLOCATE( NELEMS)
    DEALLOCATE( NNODES)

  END SUBROUTINE INITIALIZE_DATA

!!!=================================================================================
!!! Initizliation subroutine that allocates and initializes the pointer for
!!! global data, and create windows for the meshes in Roccom.
  SUBROUTINE INIT( g, name)
    TYPE(myGlobal), POINTER  :: g
    CHARACTER(*), INTENT(IN) :: name

    INTEGER :: PID, NPROC, NROW, NCOL
    INTEGER :: NUM_NODES, NUM_ELMTS
    INTEGER, POINTER :: ELMTS(:,:)
    DOUBLE PRECISION, POINTER :: COORS_S(:,:), COORS_F(:,:,:)

    IF ( g%COOKIE /= 12345 .OR. name /= "test") THEN
```

```
      PRINT *, "Initialization routine received wrong cookie or name ", &
           g%cookie, name
      STOP
   END IF

   NPROC = 4
   NROW = 40
   NCOL = 30
   NUM_NODES = NROW*NCOL
   NUM_ELMTS = (NROW-1)*(NCOL-1)

   ALLOCATE( g%COORS_S(3, NUM_NODES, NPROC))
   ALLOCATE( g%ELMTS(4, NUM_ELMTS, NPROC))
   ALLOCATE( g%COORS_F(3, NCOL, NROW, NPROC))
   ALLOCATE( g%DIMS(2))

   ALLOCATE( g%DATA(6, NUM_NODES, NPROC))

!!! Create windows
   PRINT *, 'Creating window "unstr"'
   CALL COM_create_window("unstr")
   CALL COM_new_attribute("unstr.watt", 'w', COM_INTEGER, 0, "m/s")
   CALL COM_new_attribute("unstr.velo", 'n', COM_DOUBLE_PRECISION, 3, "m/s")
   CALL COM_new_attribute("unstr.load", 'n', COM_DOUBLE_PRECISION, -3, "Newton")
   CALL COM_new_attribute("unstr.burnr", 'e', COM_DOUBLE_PRECISION, -3, "m/s")
   CALL COM_init_mesh( "unstr.unit", 0, "meter", 5)

   DO PID = 1, NPROC
      COORS_S => g%COORS_S(:, :, PID)
      ELMTS => g%ELMTS(:, :, PID)
      CALL init_unstructured_mesh( COORS_S, ELMTS, NROW, NCOL, PID-1, NPROC)
      CALL COM_init_mesh( "unstr.nc", PID, COORS_S, NUM_NODES)
      CALL COM_init_mesh( "unstr.q4", PID, ELMTS, NUM_ELMTS)
      CALL COM_init_attribute_strided( "unstr.1-load", PID, &
           g%DATA( 1, 1, PID), 6)
      CALL COM_init_attribute_strided( "unstr.2-load", PID, &
           g%DATA( 2, 1, PID), 6)
      CALL COM_init_attribute_strided( "unstr.3-load", PID, &
           g%DATA( 3, 1, PID), 6)
      CALL COM_init_attribute_strided( "unstr.1-burnr", PID, &
           g%DATA( 4, 1, PID), 6)
      CALL COM_init_attribute_strided( "unstr.2-burnr", PID, &
           g%DATA( 5, 1, PID), 6)
      CALL COM_init_attribute_strided( "unstr.3-burnr", PID, &
           g%DATA( 6, 1, PID), 6)
   END DO

   CALL COM_allocate_attribute( "unstr.all")
   CALL COM_window_init_done( "unstr")

   CALL initialize_data( "unstr", "velo")
   CALL initialize_data( "unstr", "load")
   CALL initialize_data( "unstr", "burnr")

   PRINT *, 'Creating window "str" using inheritance'
   CALL COM_create_window( "str_mesh")
   CALL COM_init_mesh("str_mesh.unit", 0, "meter", 5)
```

```
    CALL COM_new_attribute("str_mesh.empty", 'e', COM_DOUBLE_PRECISION, 1, "m/s")

    DO PID = 1, NPROC
       COORS_F => g%COORS_F(:,:,:,PID)
       CALL init_structured_mesh( COORS_F, NROW, NCOL, PID-1, NPROC)
       CALL COM_init_mesh("str_mesh.nc", PID, COORS_F, NROW*NCOL)
       g%DIMS(1) = NCOL
       g%DIMS(2) = NROW
       CALL COM_init_mesh("str_mesh.st", PID, g%DIMS, 2)
       CALL COM_init_attribute( "str_mesh.empty", PID);
    END DO
    CALL COM_new_attribute("str_mesh.load", 'n', COM_DOUBLE_PRECISION, 3, "Newton")
    CALL COM_new_attribute("str_mesh.burnr", 'e', COM_DOUBLE_PRECISION, -3, "m/s")

    CALL COM_allocate_attribute( "str_mesh.all")
    CALL COM_window_init_done( "str_mesh")

    CALL COM_create_window( "str")
    CALL COM_use_mesh( "str", "str_mesh")
    CALL COM_use_attribute( "str", "str_mesh.all")
    CALL COM_new_attribute("str.velo", 'n', COM_DOUBLE_PRECISION, 3, "m/s")

    CALL COM_allocate_attribute( "str.all")
    CALL COM_window_init_done( "str")

    CALL initialize_data( "str", "velo")
    CALL initialize_data( "str", "load")
    CALL initialize_data( "str", "burnr")

!!! Initialize function and attribute handles
    g%HDF_WRITE = COM_get_function_handle( "HDF.write_attribute")
    g%HDF_READ  = COM_get_function_handle( "HDF.read_attribute")
    g%STR_ALL = COM_get_attribute_handle( "str.all")
    g%UNSTR_MESH = COM_get_attribute_handle( "unstr.mesh")
    g%UNSTR_VELO = COM_get_attribute_handle( "unstr.velo")
    g%UNSTR_LOAD = COM_get_attribute_handle( "unstr.load")
    g%UNSTR_BURNR = COM_get_attribute_handle( "unstr.burnr")

  END SUBROUTINE INIT

!!!=============================================================================
!!! Output windows using the handles stored in the global data.
  SUBROUTINE OUTPUT_DATA( g)
    TYPE(myGlobal), POINTER :: g

    PRINT *,'Writing all attributes of the structured mesh to file "str_fields.hdf"'
    CALL COM_call_function( g%HDF_WRITE, 5, "str_fields", &
         g%STR_ALL, "str", "000", "w")
    CALL COM_call_function( g%HDF_READ, 4, "str_fields", &
         g%STR_ALL, "str", "000")

    PRINT *, 'Writing mesh of "unstr" to file "unstr_mesh.hdf"'
    CALL COM_call_function( g%HDF_WRITE, 5, "unstr_mesh", &
         g%UNSTR_MESH, "unstr", "000", "w")
    CALL COM_call_function( g%HDF_READ, 4, "unstr_mesh", &
         g%UNSTR_MESH, "unstr", "000")
```

```
    PRINT *,'Writing velocity of "unstr" to file "unstr_velo.hdf"'
    CALL COM_call_function( g%HDF_WRITE, 6, "unstr_velo", &
        g%UNSTR_VELO, "unstr", "000", "w", "unstr_mesh")
    CALL COM_call_function( g%HDF_READ, 5, "unstr_velo", &
        g%UNSTR_VELO, "unstr", "000", "unstr_mesh")
    CALL COM_call_function( g%HDF_WRITE, 6, "unstr_velo", &
        g%UNSTR_VELO, "unstr", "001", "a", "unstr_mesh")
    CALL COM_call_function( g%HDF_READ, 5, "unstr_velo", &
        g%UNSTR_VELO, "unstr", "001", "unstr_mesh")

    PRINT *, 'Writing loads of "unstr" to file "unstr_load.hdf"'
    CALL COM_call_function( g%HDF_WRITE, 6, "unstr_load", &
        g%UNSTR_LOAD, "unstr", "000", "w", "unstr_mesh")
    CALL COM_call_function( g%HDF_READ, 5, "unstr_load", &
        g%UNSTR_LOAD, "unstr", "000", "unstr_mesh")
    CALL COM_call_function( g%HDF_WRITE, 6, "unstr_load", &
        g%UNSTR_LOAD, "unstr", "001", "a", "unstr_mesh")
    CALL COM_call_function( g%HDF_READ, 5, "unstr_load", &
        g%UNSTR_LOAD, "unstr", "001", "unstr_mesh")

    PRINT *, 'Writing burnr of "unstr" to file "unstr_burnr.hdf"'
    CALL COM_call_function( g%HDF_WRITE, 6, "unstr_burnr", &
        g%UNSTR_BURNR, "unstr", "000", "w", "unstr_mesh")
    CALL COM_call_function( g%HDF_READ, 5, "unstr_burnr", &
        g%UNSTR_BURNR, "unstr", "000", "unstr_mesh")
    CALL COM_call_function( g%HDF_WRITE, 6, "unstr_burnr", &
        g%UNSTR_BURNR, "unstr", "001", "a", "unstr_mesh")
    CALL COM_call_function( g%HDF_READ, 5, "unstr_burnr", &
        g%UNSTR_BURNR, "unstr", "001", "unstr_mesh")

  END SUBROUTINE OUTPUT_DATA

!!!==========================================================================
!!! Deallocate global data
  SUBROUTINE FINALIZE( g)
    TYPE(myGlobal), POINTER :: g

    PRINT *, "Deleting windows"
    CALL COM_delete_window( "str")
    CALL COM_delete_window( "unstr")

    DEALLOCATE( g%COORS_S)
    DEALLOCATE( g%ELMTS)
    DEALLOCATE( g%COORS_F)
    DEALLOCATE( g%DIMS)

    DEALLOCATE( g%DATA)

  END SUBROUTINE FINALIZE

END MODULE ModTest


! $Id: hdftest2.f90,v 1.6 2003/12/07 04:07:33 jiao Exp $

!!! This is the wrapper for the Fortran version of hdftest. It has Roccom
!!! to maintain a reference to the global variables used in ModTest.
```

```
SUBROUTINE Glbtest_load_module( modName)
  USE  ModTest

  IMPLICIT NONE

  INTERFACE
     SUBROUTINE COM_set_pointer( attr, ptr, asso)
       USE ModTest
       CHARACTER(*), INTENT(IN) :: attr
       TYPE(myGlobal), POINTER  :: ptr
       EXTERNAL asso
     END SUBROUTINE COM_set_pointer
  END INTERFACE

  CHARACTER(*), INTENT(IN) :: modName
  TYPE(myGlobal), POINTER  :: ptr      ! Helper for nullifying the pointer
  INTEGER :: types(2)

  CALL COM_create_window( modName)
!!! Create an attribute for global data
  CALL COM_new_attribute( modName//".global", 'w', COM_F90POINTER, 1, '')
  CALL COM_allocate_attribute( modName//".global")

!!! The Fortran subroutines INIT, OUTPUT_DATA, FINALIZE takes a pointer
!!! which has the exact type as ptr.
  types(1) = COM_F90POINTER
  types(2) = COM_STRING
  CALL COM_init_member_function( modName//".initialize", INIT, &
       modName//".global", "bi", types(1))
  CALL COM_init_member_function( modName//".output", OUTPUT_DATA, &
       modName//".global", "b", types(1))
  CALL COM_init_member_function( modName//".finalize", FINALIZE, &
       modName//".global", "b", types(1))

  CALL COM_window_init_done( modName)

  ALLOCATE( ptr)
  ptr%COOKIE = 12345 ! For sanity check

!!!  This is essential for Roccom to nullify the pointer correctly.
  CALL COM_set_pointer( modName//".global", ptr, ASSOCIATE_POINTER)

END SUBROUTINE Glbtest_load_module

SUBROUTINE Glbtest_unload_module( modName)
  USE  ModTest

  IMPLICIT NONE

  INTERFACE
     SUBROUTINE COM_get_pointer( attr, ptr, asso)
       USE ModTest
       CHARACTER(*), INTENT(IN) :: attr
       TYPE(myGlobal), POINTER  :: ptr
       EXTERNAL asso
     END SUBROUTINE COM_get_pointer
  END INTERFACE
```

```
  CHARACTER(*), INTENT(IN) :: modName
  TYPE(myGlobal), POINTER  :: ptr      ! Helper for nullifying the pointer

  CALL COM_get_pointer( modName//".global", ptr, ASSOCIATE_POINTER)

  DEALLOCATE( ptr)
END SUBROUTINE Glbtest_unload_module


!!! This main driver does not directly use the module ModTest, but use
!!!     it through Roccom.
PROGRAM test2

  IMPLICIT NONE
  include "roccomf90.h"

!!! Function and attribute handles
  INTEGER :: initialize, output, finalize
  INTEGER, POINTER :: x(:)

  PRINT *, "Initializing Roccom and Rochdf"
  CALL COM_init

!!! Tell Roccom to print out debugging messages and also
!!! profile the subroutines
  CALL COM_set_verbose( 4);

  PRINT *, 'Size of F90 pointer is ', COM_get_sizeof_f90pointer(), ' bytes'

  CALL COM_load_module("Rocphdf", "HDF")
  CALL Glbtest_load_module( "test")

!!! Initialize the handles
  initialize = COM_get_function_handle( "test.initialize")
  output = COM_get_function_handle( "test.output")
  finalize = COM_get_function_handle( "test.finalize")

  CALL COM_init_profiling( 1);
!!! Call the functions through Roccom
  CALL COM_call_function( initialize, 1, "test")

  ALLOCATE(x(0))
  PRINT *, 'ASSOCIATED(x) is ', ASSOCIATED(x), &
       ' SIZE(x,1) is ', SIZE(X,1), ' UBOUND(x,1) is ', UBOUND(x,1)
  DEALLOCATE(x)
  CALL COM_get_attribute_addr( "unstr.watt", 0, x)
  PRINT *, 'ASSOCIATED(x) is ', ASSOCIATED(x), &
       ' SIZE(x,1) is ', SIZE(X,1), ' UBOUND(x,1) is ', UBOUND(x,1)

  CALL COM_call_function( output, 0)
  CALL COM_call_function( finalize, 0)

  CALL COM_print_profile( "","")  ! Print profile data to standard output
  PRINT *, "Finalizing Rochdf and Roccom"

  CALL Glbtest_unload_module( "test")
```

```
  CALL COM_finalize

END PROGRAM test2
```

Index