OpenCMISS Programmer Documentation

OpenCMISS Programmer Documentation

Table of Contents

Chapter 1. Introduction

OpenCMISS is a project for the re-engineering of the CMISS computational engine.

1. CMISS

CMISS is an interactive computer program for Continuum Mechanics, Image analysis, Signal processing and System Identification. It provides a mathematical modelling environment that allows the application of finite element analysis, boundary element and collocation techniques to a variety of complex bioengineering problems.

CMISS consists of two main components, cmgui and cm. cmgui is a graphical front end with advanced 3D display and modelling capabilities. It is the open source project written in C/C++. cm is a computational backend that may be run remotely on powerful workstations or supercomputers. cm is developed under Fortran 77 and has some licensing limitations.

2. Shared Memory Computing vs Distributed Memory Computing

cm uses OpenMP for shared memory computing. A shared memory system is relatively easy to program since all processors share a single view of data and the communication between processors can be as fast as memory accesses to a shared location. However, the CPU-to-memory connection will eventually become a bottleneck.

Distributed memory systems have increased CPU and memory scalibility compared to shared memory systems, but are much harder to program. The MPI standard is used for message passing.

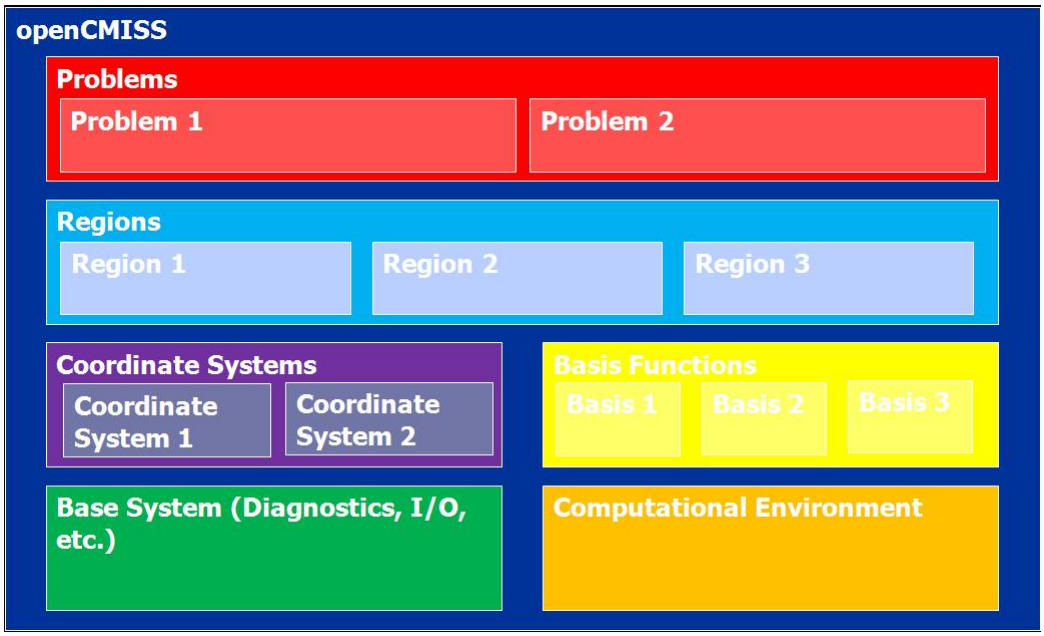
3. Objectives of OpenCMISS

OpenCMISS is re-engineering of CMISS cm component. It is intended to give a library based approach to enable use in multiple applications. It is developed under Fortan 95 and Modular, easily extendable and programmable. It will be integrated with MPI to provide modular, easily extendable and programmable code which is applicable for either MPI based distributed system, OpenMP based shared memory system and/or serial code. It will be open sourced.

Chapter 2. OpenCMISS Concepts

openCMISS has the following top level objects.

- Basis functions
- Coordinate systems
- Nodes
- Regions
- Problems
- Computational environments
- Base system(Diagnostics, I?O etc.)



1. Basis Functions

Basis functions are the key item to specify the field approximation/interpolation and the linking of nodes and elements to form a mesh. Currently, it has two types: Lagrange-Hermite tensor product and Simplex. Lagrange-Hermite tensor product can be further divided into linear to cubic lagrange, cubic and quadratic hermite. It can be arbitrarily collapsed (two or more nodes in the same location) in any one direction or in any two directions to give a degenerate basis. Simplex basic functions could contain line, triangular and tetrahedral elements. It could be linear, quadratic or cubic. Arbitrary Gaussian quadrature can integrate from 1st to 5th order (3rd order for lines at the moment). Can only have the same order in each direction at the moment. Specifying a basis function automatically generates all necessary line and face basis functions as sub-bases of the basis function.

Basis function has the following attributes:

- User number
- Global number
- Family number

- Finished tag
- Type
- Is Hermite
- Number of XI
- Number of XI coordinates
- ...

2. Coordinate Systems

Coordinate system is a system for assigning an n-tuple of numbers or scalars to each point in an n-dimensional space. It anchors the regions within the real world. Coordinate system can have different types such as:

- Rectangular cartesian
- Cylindrical polar
- Spherical polar
- Prolate spheroidal
- Oblate spheroidal

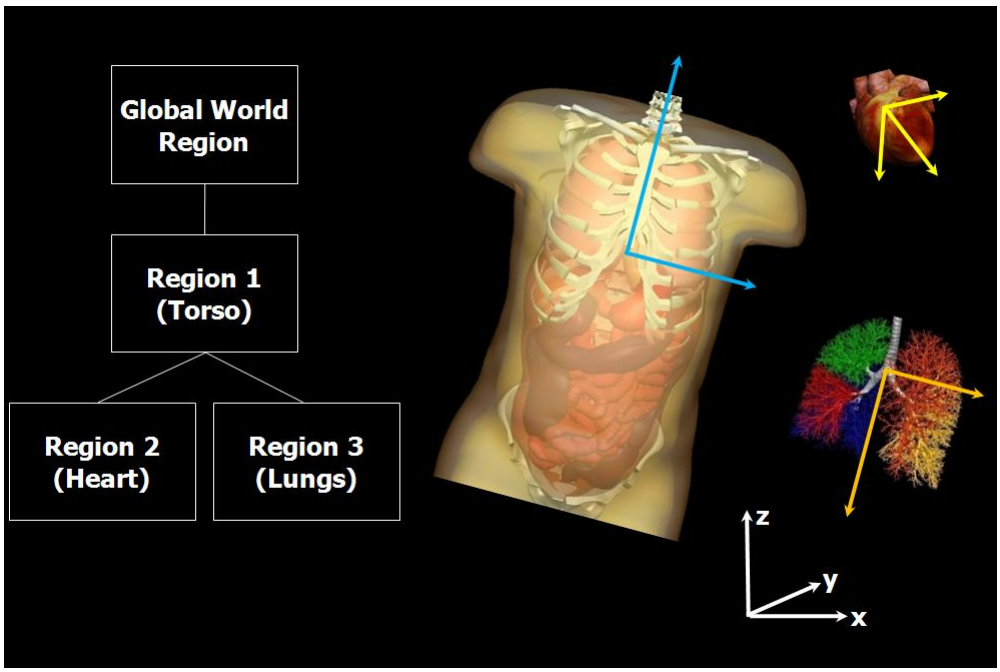
There is a global (world) coordinate system aligned with 3D rectangular cartesian space.

Coordinate system has the following attributes:

- User number
- Finished tag
- Type
- Number of dimensions
- Focus (for prolate-spheriodal system only)
- Origins
- Orientation

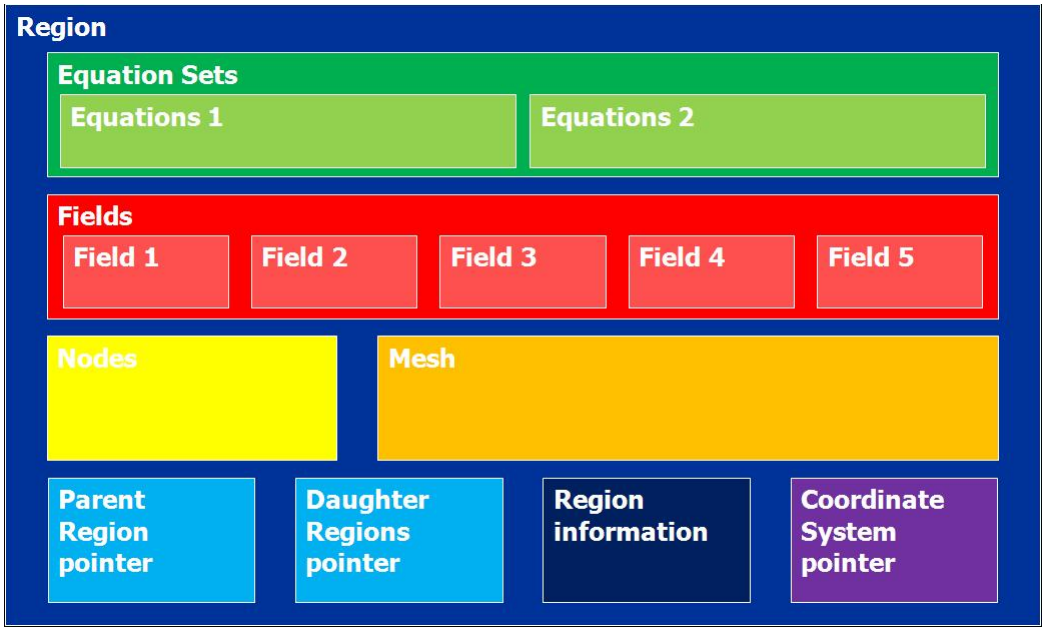
3. Regions

Regions are one of the primary objects in openCMISS. Regions are hierarchical in nature in that a region can have one parent region and a number of daughter sub-regions. Daughter regions are related in space to parent regions by an origin and an orientation of the regions coordinate system. Daughter regions may only have the same or fewer dimensions as the parent region. There is a global (world) region (number 0) that has the global (world) coordinate system.



Region has the following attributes

- User number
- Finished tag
- Label
- Number of sub(daughter) regions
- Coordinate system pointer
- Nodes
- Meshes
- Fields
- Equations
- Parent region pointer
- Daughter regions pointers



3.1. Nodes

There are three places storing nodal information. Nodes associated with region defines the nodes identification and the nodes geometric (initial) position.

Node has the following attributes

- User number
- Global number
- Label
- Initial Position

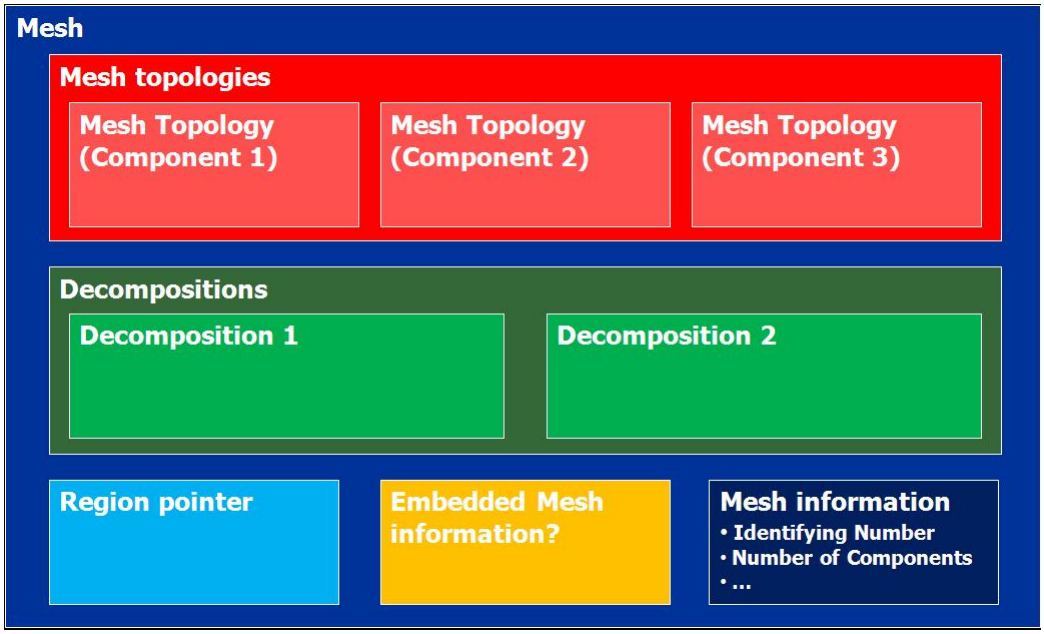
3.2. Meshes

Meshes are topological constructs within a region which fields use to define themselves. Meshes are made up of a number of mesh components. All mesh components in a mesh must “conform”, that is have the same number of elements, Xi directions etc.

Mesh has the following attributes:

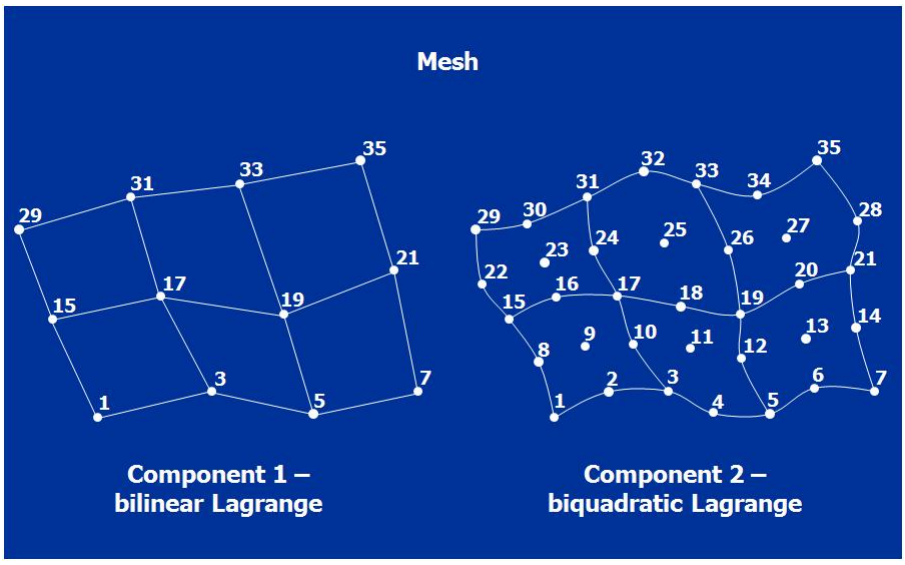
- User number
- Global number
- Finished tag
- Region pointer
- Number of dimensions
- Number of components
- Embedded flag
- Embedding mesh pointer

- Embedded meshes pointers
- Number of elements
- Number of faces
- Number of lines
- Mesh topology pointers
- Decomposition pointers



3.2.1. Mesh Topology

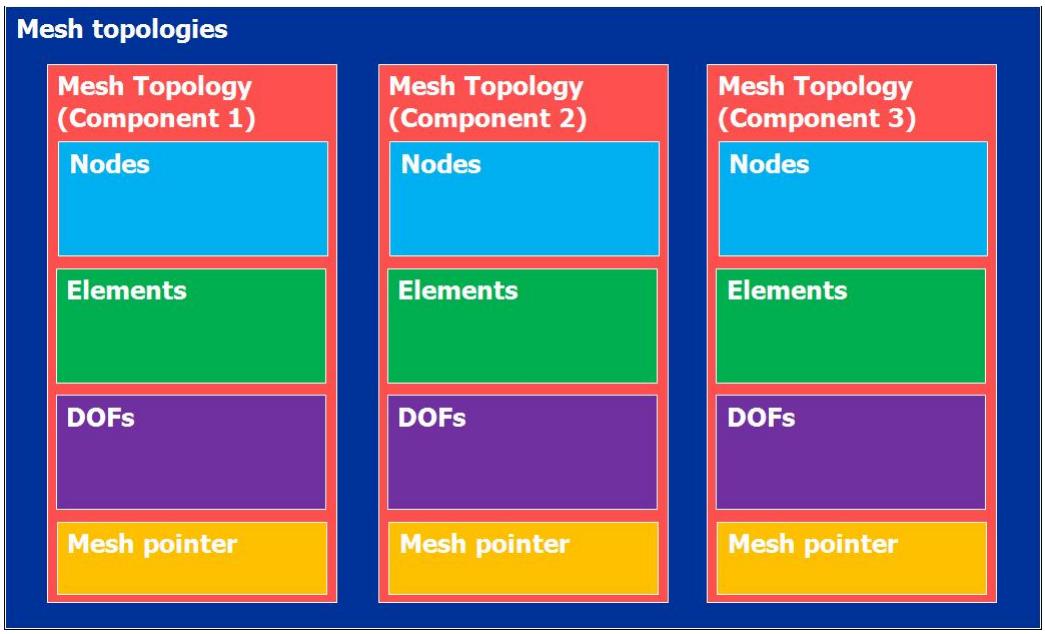
Mesh components (Topology) are made up from nodes, elements and basis functions. A new mesh component is required for each different form of interpolation e.g., one mesh component is bilinear Lagrange and another is biquadratic Lagrange.



Mesh topology has the following attributes:

- Mesh component number

- Mesh pointer
- Nodes pointers
- Element pointers
- DOFs pointers

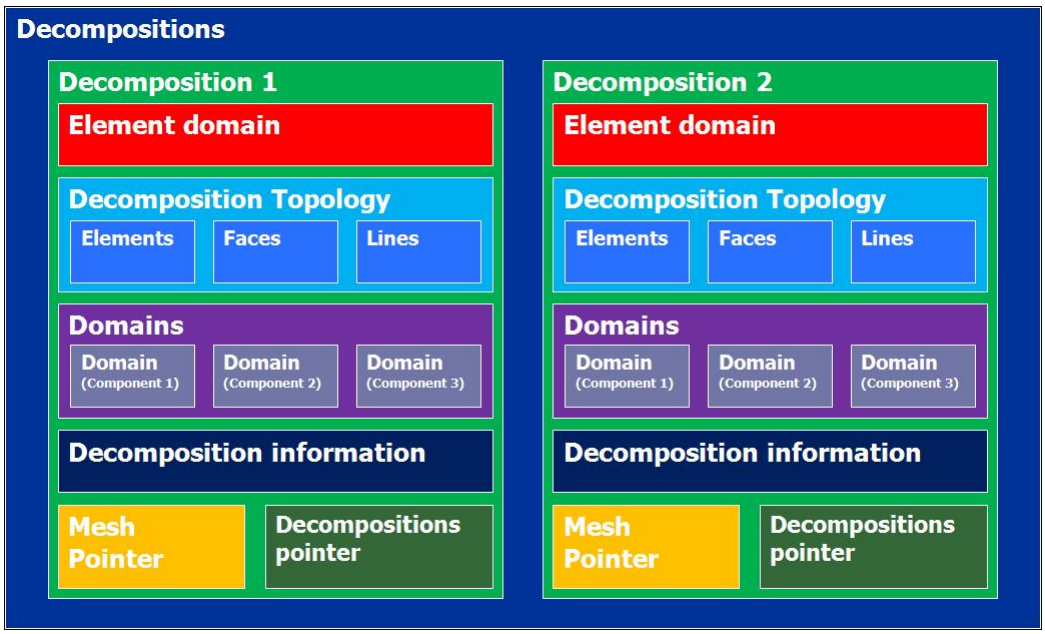


3.2.2. Decompositions

Mesh decomposition (partitioning) is used to split a computationally expensive mesh into smaller subdomains (parts) for parallel computing.

Decomposition has the following attributes

- User number
- Global number
- Finished tag
- Mesh pointer
- Mesh component number
- Decomposition type
- Number of domains
- Number of edge cut
- Element domain numbers
- Decomposition topology pointer
- Domains pointers(list of domain which has the same size as the number of components in the mesh)

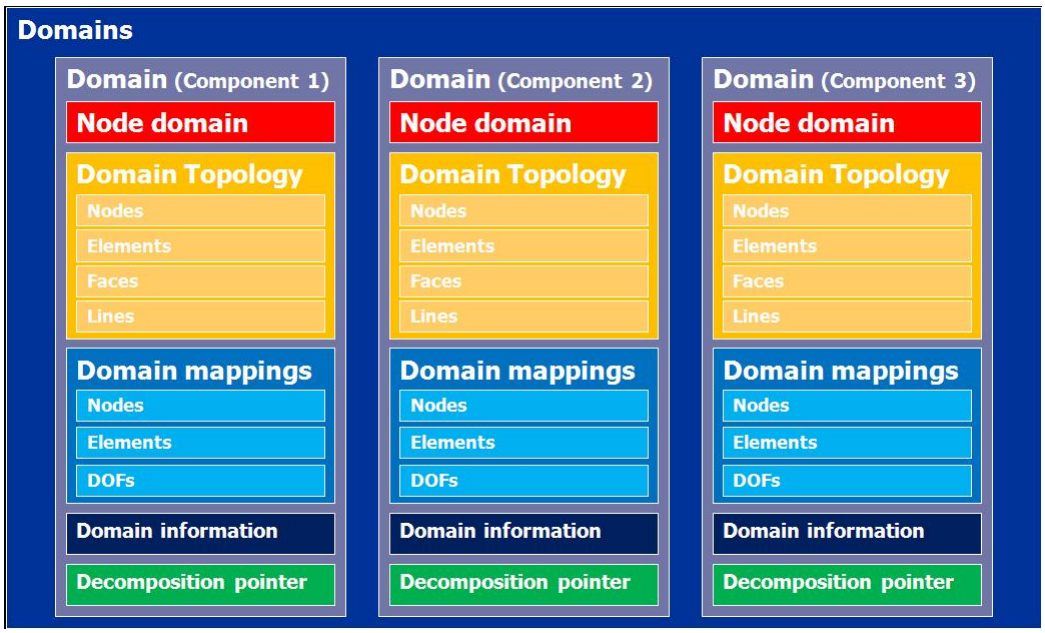


3.2.2.1. Domain

Each domain stores domain information for relevant mesh component.

The domain object contains the following attributes:

- Decomposition pointer
- Mesh pointer
- Mesh component number
- Region pointer
- Number of dimensions
- Node domain(The domain number that the np'th global node is in for the domain decomposition. Note: the domain numbers start at 0 and go up to the NUMBER_OF_DOMAINS-1)
- Domain mappings(for each mapped object e.g. nodes, elements, etc)
- Domain topology pointer(elements, nodes, DOFs)

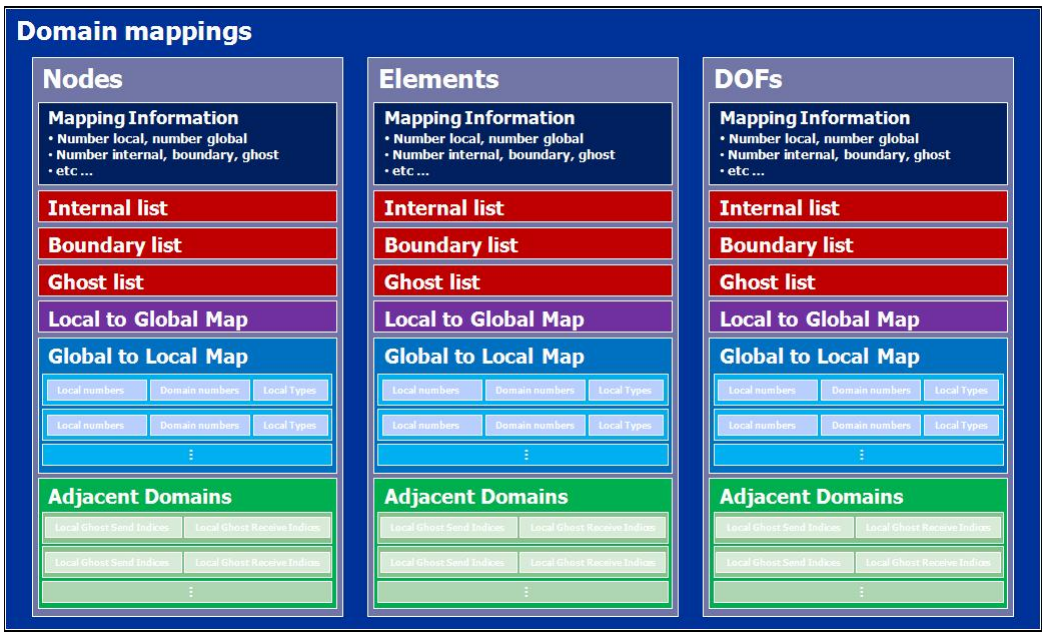


3.2.2.1.1. Domain Mappings

Stores information for each mapped object e.g. nodes, elements, etc.

The domain mapping contains the following attributes:

- Number of local
- Total number of local
- Numbers of domain local
- Number of global
- Number of domains
- Number of internal
- Internal list
- Number of boundary
- Boundary list
- Number of ghost
- Ghost list
- Local to global map
- Global to local map
- Number of adjacent domains
- Pointer to list of adjacent domains by domain number
- List of adjacent domains

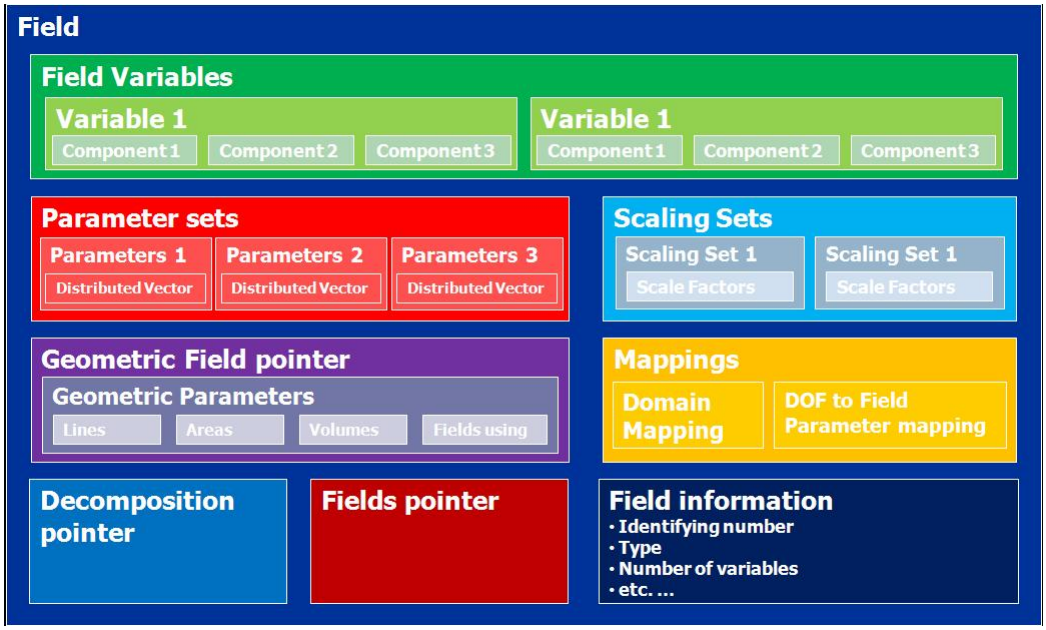


3.3. Fields

Fields are the central object for storing information and framing the problem. Fields have a number of field variables i.e., u, #u/#n, #u/#t, #2u/#t2. Each field variable has a number of components. A field is defined on a decomposed mesh. Each field variable component is defined on a decomposed mesh component.

Field can contains the following attributes:

- User number
- Global number
- Finished tag
- Region pointer
- Type(Geometric, Fibre, General, Material, Source)
- Dependent type(Independent, Dependent)
- Dimension
- Decomposition pointer
- Number of variables
- Variables
- Scalings sets
- Mappings(DOF->Field parameters)
- Parameter sets(distributed vectors)
- Geometric field pointer
- Geomatric field parameters
- Create values cache

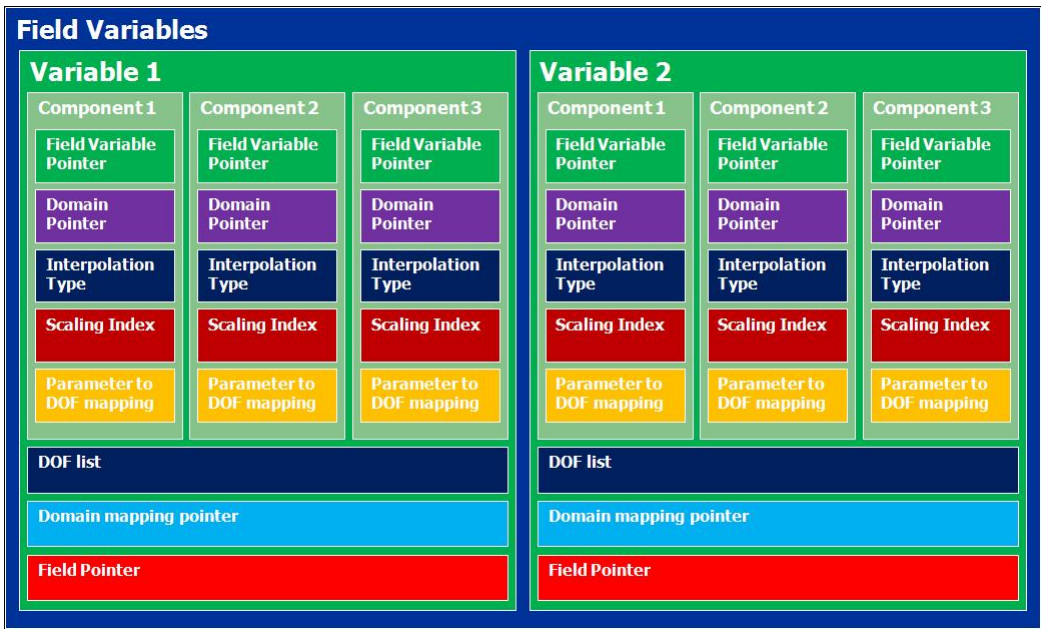


3.3.1. Field variable

Field variable stores variables for the field such as dependent variables. For example, in the Laplace's equation(FEM), it stores two variables: u and #u/#n. Each field variable has a number of components.

Field variable has the following attributes:

- Variable number
- Variable type
- Field pointer
- Region pointer
- Max number of interpolation parameters
- Number of DOFs
- Total number of DOFs
- Global DOF List
- Domain mapping pointer
- Number of components
- Components



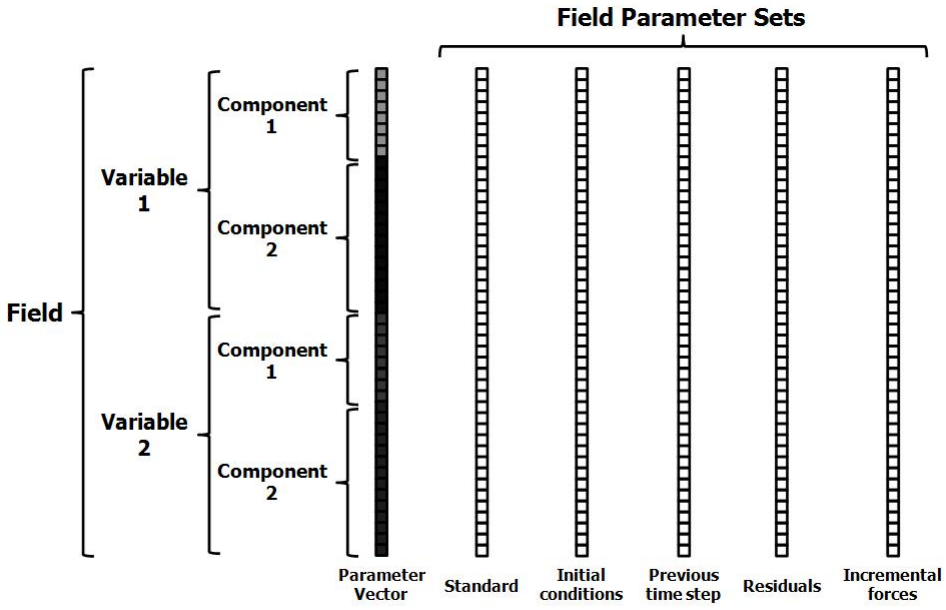
3.3.1.1. Field Variable Component

Field Variable Component has the following attributes:

- Component number
- Variable pointer
- Field pointer
- Interpolation type
- Mesh component number
- Scaling index
- Domain pointer
- Max number of interpolation parameters
- Mappings(Field paramters->DOF)

3.3.2. Parameter set

Parameter set stores values for each field variable component.



Parameter set has the following Attributes:

- Set index
- Set type
- Parameters pointer

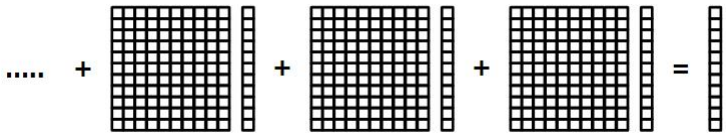
3.4. Equations Sets

Equations sets are aimed to have multiple classes, e.g. Elasticity, Fluid mechanics, Electromagnetics, General field problems, Fitting, Optimisation. Different equations are within each class, e.g. Bidomain, Navier-stokes etc. Each equation can use different solution techniques, e.g. FEM, BEM, FD, GFEM. The equation set is associated with a region and is built using the fields defined on the region.

The numerical methods are used which will result in a discretised matrix-vector form of the governing equations. openCMISS is designed to generate equations sets with a number of "equations" matrices.

e.g, damped mass spring system
$M\ddot{u} + C\dot{u} + Ku = f$

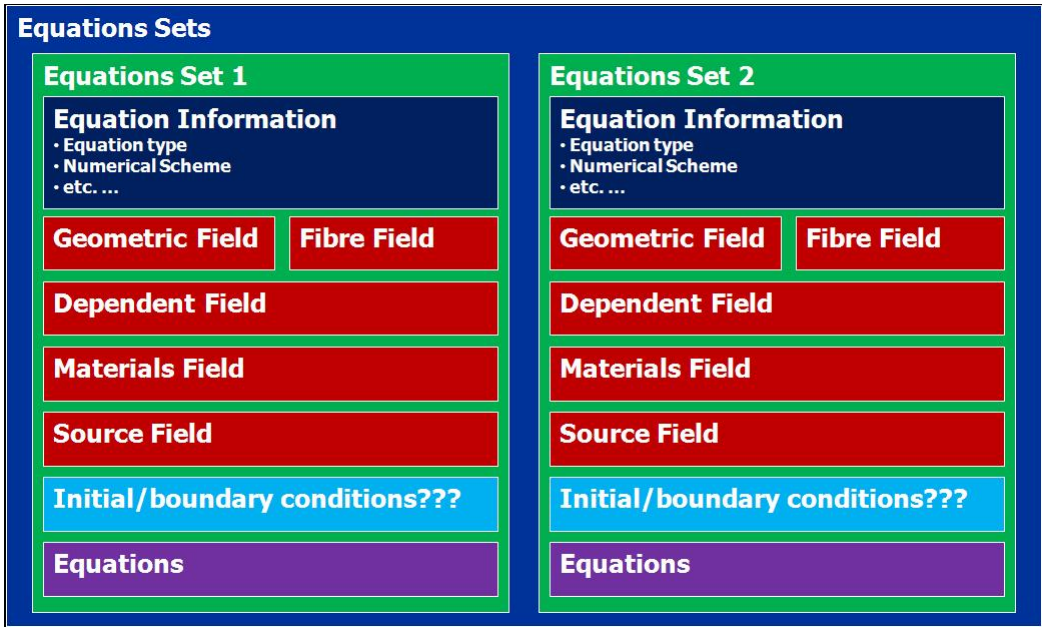
will be represented as:



Equations Set has the following attributes:

- User number
- Global number
- Finished tag
- Region pointer

- Class identifier
- Type identifier
- Sub type identifier
- Linearity type(?)
- Time dependence type(?)
- Solution method
- Geometry (fibre?) field pointer
- Materials field pointer
- Source field pointer
- Dependent field pointer
- Analytic info pointer(Analytic info stored in dependent field currently)
- Fixed conditions
- Equations pointer

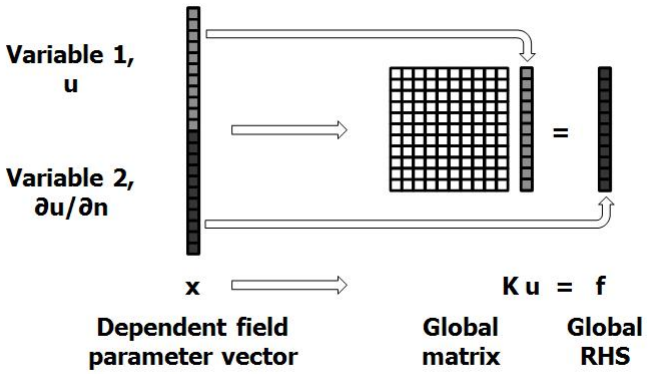


3.4.1. Equations

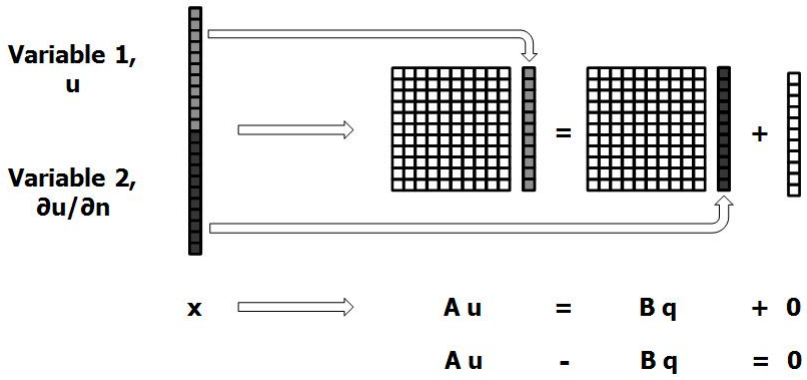
Equation holds the matrices and mapping information.

The Field variable to matrix mappings maps each field variable onto the equations matrices or RHS vector.

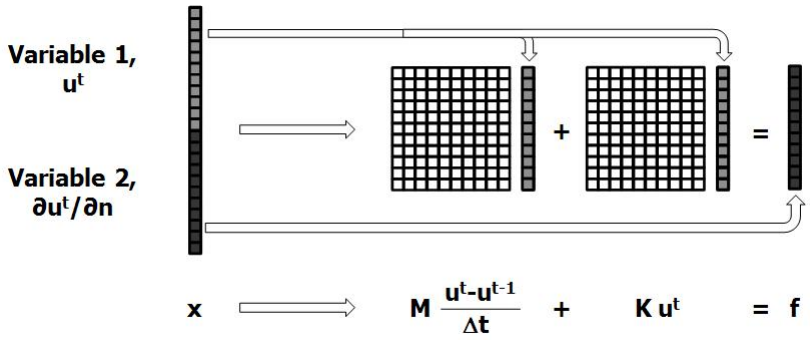
e.g. Laplace(FEM): 2 variables, 1 component



e.g. Laplace(BEM): 2 variables, 1 component



e.g. Heat equation(explicit time/FEM space): 2 variables, 1 component

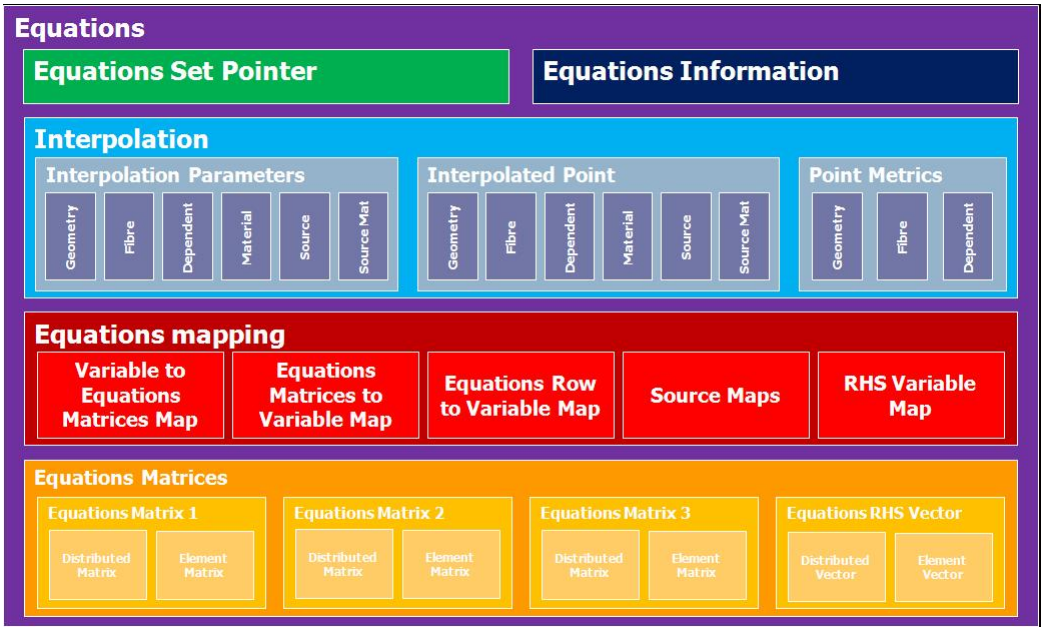


TODO matrix distribution

Equations has the following attributes:

- Equation set pointer
- Finished tag
- Output type
- Sparsity type
- Interpolation pointer
- Linear equation data pointer

- Nonlinear equation data pointer
- Time(non-static) data pointer
- Equations mapping pointer
- Equations Matrices

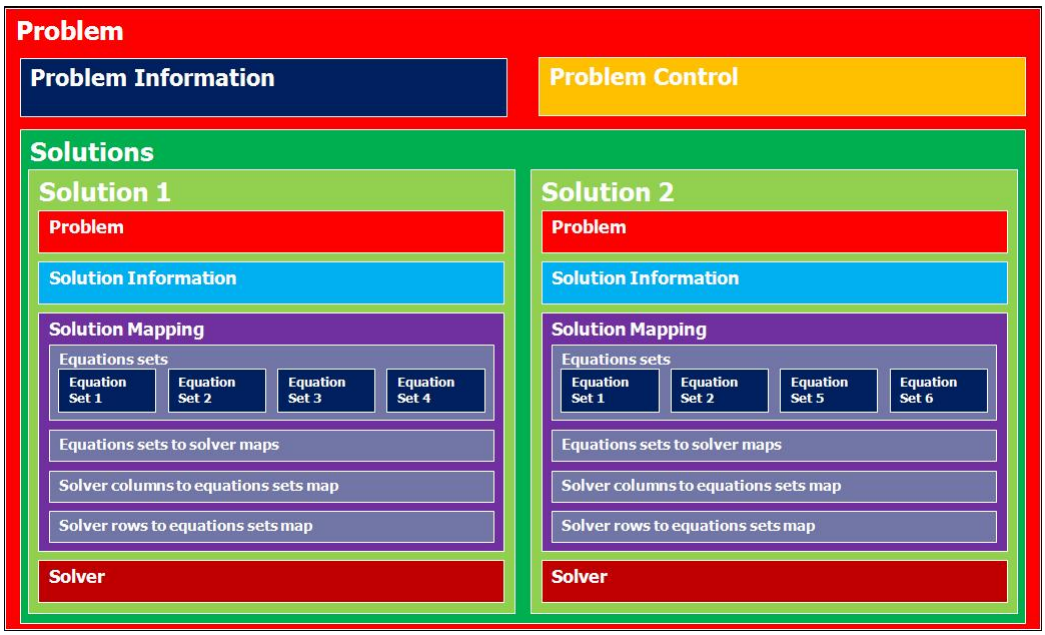


4. Problems

A problem has a number of solutions (each with their solver) inside a problem control loop. Problem associated with region via solution which maps to equations sets and hence links to region. Multiple problems can be in the same region, or multiple regions can woek to solve one problem.

Problem has the following attributes:

- User number
- Global number
- Finished tag
- Class
- Type
- Subtype
- Control pointer
- Number of solutions
- Soultions pointer



4.1. Solutions

Solution has the following attributes:

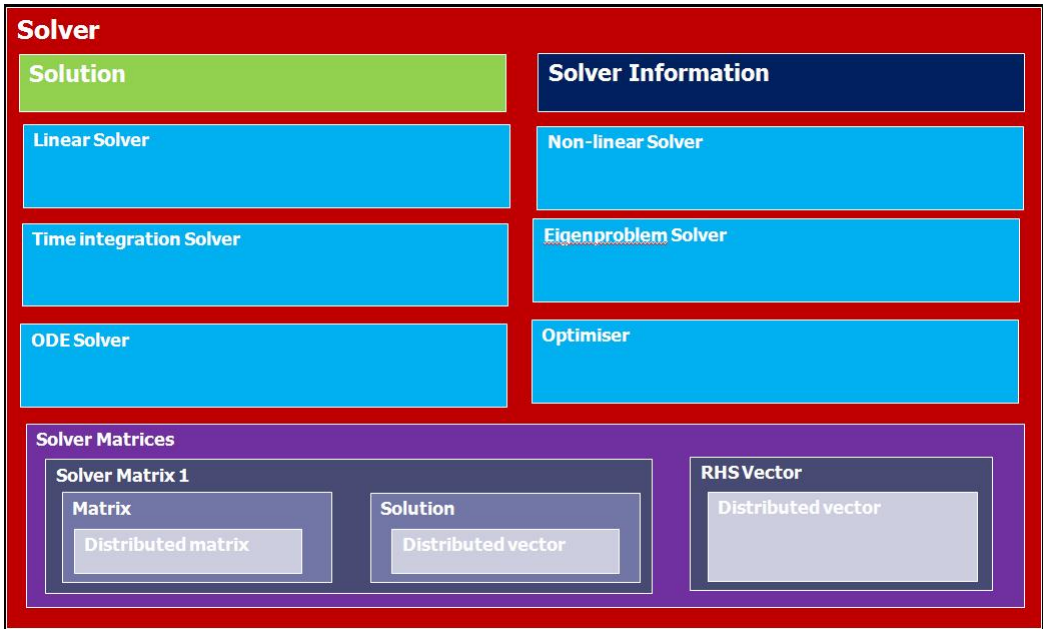
- Solution number
- Finished tag
- Linear solution data pointer
- Nonlinear solution data pointer
- Time (non-static) solution data pointer
- Equations set to add (the next equations set to add)
- Index of added equations set(the last successfully added equations set)
- Soultion mapping(which contains equations sets)
- Solver pointer

4.1.1. Solvers

Solver has the following attributes:

- Solution pointer
- Finished tag
- Solve type
- Output type
- Sparsity type
- Linear solver pointer
- Non-linear solver pointer
- Time integrationn solver pointer

- Eigenproblem solver pointer
- Solver matrices



4.2. Control

Control has the following attributes:

- Problem pointer
- Finished tag
- Control type

Chapter 3. Obtaining the Code and Setting up the Development Environment

1. Obtaining the Code and Libraries

To obtain the openCMISS source you need to check it out from the subversion repository. There are two parts to openCMISS to obtain - openCMISS itself and the various libraries it needs.

In your root openCMISS directory, make the `opencmiss` and `opemcmissextras` directories.

1.1. Obtain the Code

The openCMISS repository is at https://opencmiss.svn.sourceforge.net/svnroot/opencmiss/cm

To check out the main trunk of openCMISS issue the following command in the `opencmiss` directory:
```
svn co https://opencmiss.svn.sourceforge.net/svnroot/opencmiss/cm/trunk cm
```

If you are not familar with subversion, have a look at here [http://svnbook.red-bean.com].

1.2. Obtain the Libraries

The openCMISS libraries repository is at http://www.physiome.ox.ac.uk/svn/opencmissextras/cm

To check out the main trunk of the various libraries required with openCMISS issue the following command in the `opencmissextras` directory:
```
svn co http://www.physiome.ox.ac.uk/svn/opencmissextras/cm/trunk/external/
architecture cm/external/architecture
```

where architecture is the appropriate architecture for the machine. Possible architectures are:

- i386-win32
- i386-win32-debug
- i686-linux
- i686-linux-debug
- x86_64-linux
- x86_64-linux-debug
- rs6000-32-aix
- rs6000-32-aix-debug

Currently, the svn repository for openCMISS libraries is down. An alternative location for the libraries within the ABI is on hpc. Go to `\\bioengsmb\cmiss\opencmissextras\cm` and copy the necessary files. The folder structure is the same as svn repository.

1.3. Makefile Structure

The top level makefile will eventually build a library. In the examples directory there are separate compilable "applications" with individual makefiles. However, the library stuff isn't there as we need to code the bindings.

2. Programmer documentation

This programmer documentation is written using DocBook [http://www.docbook.org/] with the source located in the `opencmiss/cm/doc/programmer_documentation/` folder. The source XML code can be transformed into either chunked or combined documents in PDF or HTML format. The `Makefile` can be used to perform the transformation using `make single` or `make chunk` in the above folder.

Note

PDFs can be generated using `make fo`

Note

Doxygen can be included ...

3. Project Set up

3.1. On AIX 5.3 (HPC)

3.1.1. Set environment

Set environment variable to point to openCMISS
`setenv OPENCMISS_ROOT <path to your opencmiss folder>` or `export OPENCMISS_ROOT=<path to your opencmiss folder>`.

Set environment variable to point to openCMISS-extras
`setenv OPENCMISSEXTRAS_ROOT <path to your opencmissextras folder>` or `export OPENCMISSEXTRAS_ROOT=<path to your opencmissextras folder>`

3.1.2. Set MPI

Create a file called `hostfile.list` in your home directory.

Inside the file, add several lines of “hpc.bioeng.auckland.ac.nz”

In `.rhost` file in the home direcotry, add “hpc <username>”

3.1.3. Compile

Change directory to opencmiss/cm

Change directory to examples/<example%gt;

Use `gmake`.

This should result in a binary that you can run in the `bin/rs6000-32-aix` folder.

3.2. On Ubuntu 8.04

3.2.1. Set environment

Set environment variable to point to opencmiss
`setenv OPENCMISS_ROOT <path to your opencmiss folder>` or `export OPENCMISS_ROOT=<path to your opencmiss folder>`.

Set environment variable to point to openCMISS-extras
`setenv OPENCMISSEXTRAS_ROOT <path to your opencmissextras folder>` or `export OPENCMISSEXTRAS_ROOT=<path to your opencmissextras folder>`

It is also helpful to add the following

```
setenv PATH ${OPENCMISSEXTRAS_ROOT}/cm/external/${archname}/bin:${PATH}

setenv PATH ${OPENCMISS_ROOT}/cm/bin/${archname}:${PATH}
```

where ${archname} is the appropriate architecture e.g., i686-linux, x86_64-linux.

If you are using totalview you will also need to add
`setenv LM_LICENSE_FILES <path-to-the-flex-directory>`

3.2.2. Install Compilers

Download Intel Fotran Compiler from here [http://www.intel.com/cd/software/products/asmo-na/eng/282048.htm].

Extract the file and follow the `install.htm` to install

3.2.3. Compile

To build an example project:
`make`

To run the example project:
`mpd & mpirun -n 2 path/to/the/execution/file`

To debug the project using TotalView:
`mpd & mpirun -tv 2 path/to/the/execution/file`

3.3. On Windows XP (Visual Studio 2005)

3.3.1. Install Compilers

Download Intel Fortran Compiler from here [http://www.intel.com/cd/software/products/asmo-na/eng/278834.htm].

Execute the exe file and follow the installation wizard.

3.3.2. Install MPI

Download MPICH2 from here. [http://www.mcs.anl.gov/research/projects/mpich2/]

You can either download the source archive and follow the `README.windows` file to install or download the installer to install.

Set bin folder to the path

To start the MPI, run `smpd -start` in command window.

NOTE: as from MPICH2 version 1.0.7 the library names have changed. `libmpich2` has now become `libmpi`!

3.3.3. Compile and Debug

Build the Fortran project under the debug mode and generate the `opencmisstest-debug.exe` file.

In the C Project (since the Fortran projects do not support MPI cluster debugger), configure the debugging properties according to this [http://download.microsoft.com/download/6/8/d/68d7d82b-e477-4699-b403-72be2e6218b1/CCS03DebugParallelAppsVS05.doc].

The MPIShim location is in the path similar to C:\Program Files\Microsoft Visual Studio 8\Common7\IDE \Remote Debugger\x86\mpishim.exe.

Debug the C project.

3.3.4. Run

To run the project in the command window:
mpiexec -n 2 -localroot <path to the execution file>

3.4. On Windows Vista (Visual Studio 2008)

3.4.1. Install Compilers

Download Intel Fortran Compiler from here [http://www.intel.com/cd/software/products/asmo-na/eng/278834.htm].

Execute the exe file and follow the installation wizard.

3.4.2. Install MPI

Before you install MPICH2 under Vista you must turn off User Account Control

1. Goto Start -> Control Panel
2. Double-click on User Accounts
3. Click "Turn User Account Control on or off"
4. Untick "Use User Account Control (UAC) to help protect your computer" and click OK
5. Restart your computer.

Download from here [http://www.mcs.anl.gov/research/projects/mpich2/]. Choose the Win32 IA32 (binary) option.

Run the downloaded .msi file. Follow all instructions and install "For everybody".

Once you have installed MPICH2 you can turn User Account Control back on. Follow the instructions above and in 4. tick the "Use User Account Control ...".

NOTE: as from MPICH2 version 1.0.7 the library names have changed. libmpich2 has now become libmpi!

3.4.3. Compile

For each example, go into the VisualopenCMISS_08 folder. Double click the VisualopenCMISS project solution file to lauch Visual Studio.

4. Libraries Build (Optional)

4.1. Compiling PETSc

Note this is assuming you have the Intel Fortran compiler version 10.1.024. Adjust the version string as necessary.

4.1.1. Step1: Linux Environment installation and Compiler Environment Set up (For Windows

Under windows system:

- Install Cygwin if you need to. Cywin can be found here [http://www.cygwin.com/]. Make sure you include the make and python modules when you install.
- Lauch a Command Prompt Window
- Run the ifortvars.bat batch file to setup your Intel Fortran environment. e.g., "C:\Program Files\Intel\Compiler\Fortran\10.1.024\IA32\Bin\ifortvars.bat"
- Run the Cygwin batch file to setup the unix environment e.g., "C:\Cygwin\Cygwin.bat"

N.B: PetSc uses X, so make sure in linux environment, libX11-dev package is installed. Also make sure blas and lapack (-dev) packages are installed.

4.1.2. Step2: Compile PETSC

4.1.2.1. For Windows

- Change to the opencmissextras PETSc directory e.g., if opencmissextras root is E:\opencmissextras and we are compiling PETSC version petsc-2.3.3-p15 then "cd /cygwin/e/opencmissextras/cm/external/packages/PETSc/petsc-2.3.3-p15"
- If you have MPICH2 version 1.0.7 or greater edit the python/BuildSystem/config/packages/MPI.py file. Find the self.liblist_mpich line. After the line "['fmpich2.lib','mpich2.lib']," add the line "['fmpich2.lib','mpi.lib'],".
- `PETSC_DIR=/cygdrive/e/opencmissextras/cm/external/packages/PETSc/petsc-2.3.3-p15; export PETSC_DIR`
- For a debug install issue the following commands

```
PETSC_ARCH=cygwin-c-debug; export PETSC_ARCH

config/configure.py  --prefix=/cygdrive/e/opencmissextras/cm/external/i386-win32-debug    --with-shared=no    --with-cc='win32fe    cl'    --with-fc='win32fe    ifort'    --with-cxx='win32fe    cl'    --download-f-blas-lapack=1   LIBS=-L'/cygdrive/c/Program\   Files/Intel/Compiler/Fortran/10.1.024/IA32/Lib' --with-debugging=yes

PETSC_ARCH=cygwin-c-debug; export PETSC_ARCH
```

  For a non-debug install issue the following commands

```
PETSC_ARCH=cygwin-c-opt; export PETSC_ARCH

config/configure.py  --prefix=/cygdrive/e/opencmissextras/cm/external/i386-win32 --with-shared=no --with-cc='win32fe cl' --with-fc='win32fe ifort'  --with-cxx='win32fe  cl'  --download-f-blas-lapack=1  LIBS=-L'/cygdrive/c/Program\ Files/Intel/Compiler/Fortran/10.1.024/IA32/Lib' --with-debugging=no

PETSC_ARCH=cygwin-c-opt; export PETSC_ARCH
```

- `make -e all`
- `make -e install`

4.1.2.2. For AIX5.3

- Change to the opencmissextras PETSc directory. e.g /people/tyu011/workspace/opencmissextras/cm/external/packages/PETSc/petsc-2.3.3-p15
- `setenv PETSC_DIR /people/tyu011/workspace/opencmissextras/cm/external/packages/PETSc/petsc-2.3.3-p15`
- In `config` directory, copy the file `aix5.1.0.0.py` and rename it to `aix5.3.0.0.py`
- For a debug install issue the following commands

```
setenv PETSC_ARCH aix5.3.0.0

config/aix5.3.0.0.py                --prefix=/people/tyu011/workspace/opencmissextras/cm/external/rs6000-32-aix-debug --with-debugging=yes
```

```
setenv PETSC_ARCH aix5.3.0.0
```

For a non-debug install issue the following commands

```
setenv PETSC_ARCH aix5.3.0.0

config/aix5.3.0.0.py                --prefix=/people/tyu011/workspace/
opencmissextras/cm/external/rs6000-32-aix --with-debugging=no

setenv PETSC_ARCH aix5.3.0.0
```

- `make -e all`
- `make -e install`

Chapter 4. Object Interface

1. Objectives

- Simple interface for a variety of scripting and programming languages
- User friendly in that the library should sensibly set default parameters to minimise the amount of information the programmer/user has to send
- Should be extensible so that extra parameters can be added at a later stage without causing problems

2. General Rules

- Objects identified by a unique user number
- Objects are initialised with a `OBJECT_INITIALISE` call
- Objects are finalised with a `OBJECT_FINALISE` call
- Objects are created with a pair of `OBJECT_CREATE_START` and `OBJECT_CREATE_FINISH` calls
- Objects are destroyed with a `OBJECT_DESTROY` call
- Object parameters are set with a number of `OBJECT_PARAMETER_SET` calls between the START and FINISH calls
- START call initialises OBJECT and sets default parameters. SET calls modify default parameters. FINISH call finalises the object

3. Example(Basis functions)

```
!Initialise the basis functions.
CALL BASES_INITIALISE(ERR,ERROR,*999)
!Start the creation of a basis with a user number of 1.
!The default is tri-linear Lagrange
CALL BASIS_CREATE_START(1,BASIS,ERR,ERROR,*999)
!Set the number of xi directions to 2.
CALL BASIS_NUMBER_OF_XI_SET(BASIS,2,ERR,ERROR,*999)
!Set the interpolation to be cubic Hermite*quadratic Lagrange.
CALL BASIS_INTERPOLATION_XI_SET(BASIS, &
& (/CUBIC_HERMITE,QUADRATIC_LAGRANGE/),ERR,ERROR,*999)
!Finish the creation of the basis.
CALL BASIS_CREATE_FINISH(BASIS,ERR,ERROR,*999)
!Destroy the basis with the user number 1.
CALL BASIS_DESTROY(1,ERR,ERROR,*999)
!Finalise the basis functions.
CALL BASES_FINALISE(ERR,ERROR,*999)
```

Chapter 5. Examples

1. Laplace equation

```
!Initialise CMISS.
CALL CMISS_INITIALISE(ERR,ERROR,*999)

!Start the creation a coordinate system (default 3D RC)
CALL COORDINATE_SYSTEM_CREATE_START(1,COORDINATE_SYSTEM,ERR,ERROR,*999)
!Set the coordinate system to be 2D
CALL COORDINATE_SYSTEM_DIMENSION_SET(COORDINATE_SYSTEM,2,ERR,ERROR,*999)
!Finish the creation a coordinate system
CALL COORDINATE_SYSTEM_CREATE_FINISH(COORDINATE_SYSTEM,ERR,ERROR,*999)

!Start the creation of a region with a coordinate system.
CALL REGION_CREATE_START(1,COORDINATE_SYSTEM,REGION,ERR,ERROR,*999)
!Finish the creation of a region
CALL REGION_CREATE_FINISH(REGION,ERR,ERROR,*999)

!Start the creation of a basis (default tri-linear Lagrange)
CALL BASIS_CREATE_START(1,BASIS,ERR,ERROR,*999)
!Set the number of Xi directions to 2 (bi-linear Lagrange)
CALL BASIS_NUMBER_XI_SET(BASIS,2,ERR,ERROR,*999)
!Finish the creation of a basis
CALL BASIS_CREATE_FINISH(BASIS,ERR,ERROR,*999)



!Start the creation of a generated mesh in the region
CALL GENERATED_MESH_CREATE_START(1,REGION,GENERATED_MESH,ERR,ERROR,*999)
!Set up a regular 100x100 mesh
CALL GENERATED_MESH_TYPE_SET(GENERATED_MESH,GENERATED_MESH_REGULAR_TYPE, &
    & ERR,ERROR,*999)
CALL GENERATED_MESH_BASIS_SET(GENERATED_MESH,BASIS,ERR,ERROR,*999)
CALL GENERATED_MESH_EXTENT_SET(GENERATED_MESH,(/2.0_DP,1.0_DP/),ERR,ERROR,*999)
CALL GENERATED_MESH_NUMBER_OF_ELEMENTS_SET(GENERATED_MESH,(/100,100/), &
    & ERR,ERROR,*999)
!Finish the creation of a generated mesh in the region
CALL GENERATED_MESH_CREATE_FINISH(GENERATED_MESH,MESH,ERR,ERROR,*999)

!Start the creation a decomposition on a mesh
CALL DECOMPOSITION_CREATE_START(1,MESH,DECOMPOSITION,ERR,ERROR,*999)
!Set the decomposition so that the domains are calculated
CALL DECOMPOSITION_TYPE_SET(DECOMPOSITION,DECOMPOSITION_CALCULATED_TYPE, &
    & ERR,ERROR,*999)
CALL DECOMPOSITION_NUMBER_OF_DOMAINS_SET(DECOMPOSITION,5,ERR,ERROR,*999)
!Finish the creation of decomposition
CALL DECOMPOSITION_CREATE_FINISH(DECOMPOSITION,ERR,ERROR,*999)



!Start the creation of a field on a region (default geometric)
CALL FIELD_CREATE_START(1,REGION,GEOMETRIC_FIELD,ERR,ERROR,*999)
!Set the decomposition the field will use
CALL FIELD_MESH_DECOMPOSITION_SET(GEOMETRIC_FIELD,DECOMPOSITION,ERR,ERROR,*999)
!Set mesh components each of the field variable components will use. NB. These
```

```
!are shown for example as each field variable component will default to the
!first mesh component.
CALL FIELD_COMPONENT_MESH_COMPONENT_SET(GEOMETRIC_FIELD, &
    & FIELD_STANDARD_VARIABLE_TYPE,1,1,ERR,ERROR,*999)
CALL FIELD_COMPONENT_MESH_COMPONENT_SET(GEOMETRIC_FIELD, &
    & FIELD_STANDARD_VARIABLE_TYPE,2,1,ERR,ERROR,*999)
!Finish the creation of a field
CALL FIELD_CREATE_FINISH(GEOMETRIC_FIELD,ERR,ERROR,*999)

!Set a field value in a parameter set
CALL FIELD_PARAMETER_SET_UPDATE_NODE(GEOMETRIC_FIELD, &
    & FIELD_VALUES_SET_TYPE,1,1,1,FIELD_STANDARD_VARIABLE_TYPE, &
    & 1.0_DP,ERR,ERROR,*999)
!Start the update of field values in a parameter set
CALL FIELD_PARAMETER_SET_UPDATE_START(GEOMETRIC_FIELD, &
    & FIELD_VALUES_SET_TYPE,ERR,ERROR,*999)
!!DO SOME CALCULATIONS
!Finish the update of field values in a parameter set
CALL FIELD_PARAMETER_SET_UPDATE_FINISH(GEOMETRIC_FIELD, &
    & FIELD_VALUES_SET_TYPE,ERR,ERROR,*999)



!Start the creation of an equations set on a region
CALL EQUATIONS_SET_CREATE_START(1,REGION,GEOMETRIC_FIELD,EQUATIONS_SET, &
    & ERR,ERROR,*999)
!Set the equations set to be standard Laplace#s equation problem
CALL EQUATIONS_SET_SPECIFICATION_SET(EQUATIONS_SET, &
    & EQUATIONS_SET_CLASSICAL_FIELD_CLASS,EQUATIONS_SET_LAPLACE_EQUATION_TYPE, &
    & EQUATIONS_SET_STANDARD_LAPLACE_SUBTYPE,ERR,ERROR,*999)
!Finish the creation of an equations set
CALL EQUATIONS_SET_CREATE_FINISH(EQUATIONS_SET,ERR,ERROR,*999)

!Start the creation of the equations set dependent field
CALL EQUATIONS_SET_DEPENDENT_CREATE_START(EQUATIONS_SET,ERR,ERROR,*999)
!Finish the create of the problems dependent field
CALL EQUATIONS_SET_DEPENDENT_CREATE_FINISH(EQUATIONS_SET,ERR,ERROR,*999)

!Start the creation of the equations set fixed conditions
CALL EQUATIONS_SET_FIXED_CONDITIONS_CREATE_START(EQUATIONS_SET,ERR,ERROR,*999)
!Set BC#s
CALL EQUATIONS_SET_FIXED_CONDITIONS_SET_DOF(EQUATIONS_SET,1, &
    & EQUATIONS_SET_FIXED_BOUNDARY_CONDITION,0.0_DP,ERR,ERROR,*999)
CALL EQUATIONS_SET_FIXED_CONDITIONS_SET_DOF(PROBLEM,10, &
    & EQUATIONS_SET_FIXED_BOUNDARY_CONDITION,1.0_DP,ERR,ERROR,*999)
!Finish the create of the equations set fixed conditions
CALL EQUATIONS_SET_FIXED_CONDITIONS_CREATE_FINISH(EQUATIONS_SET,ERR,ERROR,*999)



 !Start the creation of the equations for the equations set
CALL EQUATIONS_SET_EQUATIONS_CREATE_START(EQUATIONS_SET,ERR,ERROR,*999)
!Set the equations matrices sparsity type
CALL EQUATIONS_SET_SPARSITY_TYPE_SET(EQUATIONS_SET,EQUATIONS_SET_SPARSE_MATRICES, &
    & ERR,ERROR,*999)
!Set the equations output type
CALL EQUATIONS_SET_OUTPUT_TYPE_SET(EQUATIONS_SET,EQUATIONS_SET_TIMING_OUTPUT, &
    & ERR,ERROR,*999)
```

```
!Finish the creation of the equations for the equations set
CALL EQUATIONS_SET_EQUATIONS_CREATE_FINISH(EQUATIONS_SET,ERR,ERROR,*999)

!Start the creation of a problem
CALL PROBLEM_CREATE_START(1,PROBLEM,ERR,ERROR,*999)
!Set the problem set to be standard Laplace#s equation problem
CALL PROBLEM_SPECIFICATION_SET(PROBLEM,PROBLEM_CLASSICAL_FIELD_CLASS, &
    & PROBLEM_LAPLACE_EQUATION_TYPE,PROBLEM_STANDARD_LAPLACE_SUBTYPE,ERR,ERROR,*999)
!Finish the creation of a problem
CALL PROBLEM_CREATE_FINISH(PROBLEM,ERR,ERROR,*999)

!Start the creation of the control for a problem
CALL PROBLEM_CONTROL_CREATE_START(PROBLEM,ERR,ERROR,*999)
!Finish the creation of the control a problem
CALL PROBLEM_CONTROL_CREAT_CREATE_FINISH(PROBLEM,ERR,ERROR,*999)



 !Start the creation of the solutions for the problem
CALL PROBLEM_SOLUTIONS_CREATE_START(PROBLEM,ERR,ERROR,*999)
!Add in the equations set
CALL PROBLEM_SOLUTIONS_EQUATIONS_SET_ADD(PROBLEM,1,EQUATIONS_SET,ERR,ERROR,*999)
!Set the solutions output type
CALL PROBLEM_SOLUTIONS_OUTPUT_TYPE_SET(PROBLEM,1,PROBLEM_SOLUTION_MATRIX_OUTPUT, &
    & ERR,ERROR,*999)
!Finish the creation of the solutions for the problem
CALL PROBLEM_SOLUTIONS_CREATE_FINISH(PROBLEM,ERR,ERROR,*999)

!Start the creation of the solvers for the problem
CALL PROBLEM_SOLVER_CREATE_START(PROBLEM,ERR,ERROR,*999)
!Set the type of preconditioner
CALL PROBLEM_SOLVER_ITERATIVE_PRECONDITIONER_TYPE_SET(PROBLEM,1, &
    & PROBLEM_SOLVER_ITERATIVE_INCOMPLETE_LU_PRECONDITIONER,ERR,ERROR,*999)
!Set the number of iterations
CALL PROBLEM_SOLVER_ITERATIVE_MAX_ITERATIONS_SET(PROBLEM,1,100,ERR,ERROR,*999)
!Set the output type
CALL PROBLEM_SOlVER_OUTPUT_TYPE_SET(PROBLEM,1,PROBLEM_SOLVER_TIMING_OUTPUT, &
    & ERR,ERROR,*999)
!Finish the creation of the solvers for the problem
CALL PROBLEM_SOLVER_CREATE_FINISH(PROBLEM,ERR,ERROR,*999)

!Solve the problem
CALL PROBLEM_SOLVE(PROBLEM,ERR,ERROR,*999)

!Finalise CMISS.
CALL CMISS_FINALISE(ERR,ERROR,*999)
```

Chapter 6. Library Commands

Chapter 7. Coding Style

- Within a module all named constants and procedure names should be prefixed by a name indicating that module so as to maintain a namespace.
- All dynamic arrays should be `ALLOCATABLE` rather than `POINTER` unless full pointer functionality is required.
- A double space should be used for an indent. Tabs should not be used to indent code.
- All pointers should be checked to see if they are `ASSOCIATED` before de-referencing them.
- If there is just a single statement following an `IF` clause use the inline form of the `IF` statement and do not use `THEN` and `ENDIF`.
- There should be a space before the first continuation character and a space after the second continuation character when continuing lines.
- Use standard loop variable names e.g., `nn`, `component_idx`, when looping rather than temporary variable names.
- When using case statements put in all known values of the the case variable and use a `CALL FLAG_ERROR("Not implemented", ...` statement if the code for the case variable has yet to be coded.
- Use a `!================...` line between subroutines and functions
- For dummy array arguments the dimension qualifier should be with the array name i.e., use `INTEGER(INTG) :: FRED(N)` rather than `INTEGER(INTG), DIMENSION(N) :: FRED`.
- Code should be no more than 132 characters in a line. Use continuation &'s if need be. Add the space before/ after each & to distinguish with the complete line.
- No need to allocate local arrays if the size of array is known. e.g. `REAL(DP), ALLOCATABLE :: VALUE_BUFFER(:)` and then `ALLOCATE(VALUE_BUFFER(5),STAT=ERR)` can be replaced with `REAL(DP)::VALUE_BUFFER(5)`
- Use node hierarchy consistently in subroutine calls. i.e., derivative_number,node_number,component_number,variable_number instead of reverse order.
- Input arguments should be checked in a subroutine before being used.
- No space between IF and condition i.e. IF(rather than IF (.
- Use Fortran 2003 standard for get the arguments to a program than the non-standard GET_ARG
- Get routines
  - Use subroutines instead of function.
  - Return the result to the memory supplied by calling program. (i.e. using `INTENT(OUT)::A(:)` etc)
  - Indicate in the doxygen comment that an argument is changed inside the get routine.
  - Check that the size of the memory passed in is large enough to hold the data that it is going to store

The following styles are required by IBM Fortran Compiler:

- For `read/write/print` statement, avoid the comma before the data variable. For example, use `READ(FILE_ID, CHAR(DP_FMT), IOSTAT=IOS) REAL_DATA(1:LEN_OF_DATA)` instead of `READ(FILE_ID, CHAR(DP_FMT), IOSTAT=IOS), REAL_DATA(1:LEN_OF_DATA)`

- In `IF/WHILE` statement, if it checks whether a logical value is true or false, use `IF(SOMEVALUE)` or `IF(.NOT.SOMEVALUE)` instead of `IF(SOMEVALUE==.TRUE.)` or `IF(SOMEVALUE==.FALSE.)`
- For the complete array assignment, use `A=B` instead of `A(:)=B(:)`