CombLayer Guide

Stuart Ansell

August 23, 2016

Contents

1 Introduction

CombLayer is designed to facilitate the rapid production of complex MCNP(X) models that depend on a long list of ranged variables and a number of module flags. It is also intended to help with placement of tallies, maintaining consistant material files and some variance reduction.

1.1 Coding Convensions

CombLayer has some coding convensions beyond the standard Scott Myers Efficient C++ convensions [**?**]. These are typcially there for two reasons (i) that in a model-build system, a rapid build time is essential since it is nearly impossible to have a sub-test framework for any component as the whole MCNP(X) model is required to check if is it valid, (ii) the code is intended to be used without complete understanding. Therefore as much as possible, each component is independent without code repetition. Back-references are to be minimized both in the run-time calling path and in the code build dependencies.

1.1.1 Include files

Include files (.h) are forbidden to include other files. This does several things (a) it reduces the *dependency hell* where it is almost impossible to find the definition of a function and what it depends on. (b) optimization of the include tree can be carried out and dependency continuously observered.

Namespaces are a good method of removing global name pollution but many other C++ programs allows *using namespace X*, this is almost 100% forbidden except in the test for that particular namespace unit. This also applies to boost, stl, tr1 etc, to which helps distinguish external functions and domains.

2 Layout

The basic program structure is given by figure 1. The main program structure is normally copied from an existing project and the areas are constructed by the user. It is normal flow is to to call functions that: (i) define new input options to enter parameters from the command line, (ii) variables that your project is going to use, (iii) build the geometry via a call to a makeProject functoin (iv) set up tallies (v) generate variance reduction.There are other ways to construct the system but this allows a degree of autonomy from tallies/variance reduction and producing an appropiate output.

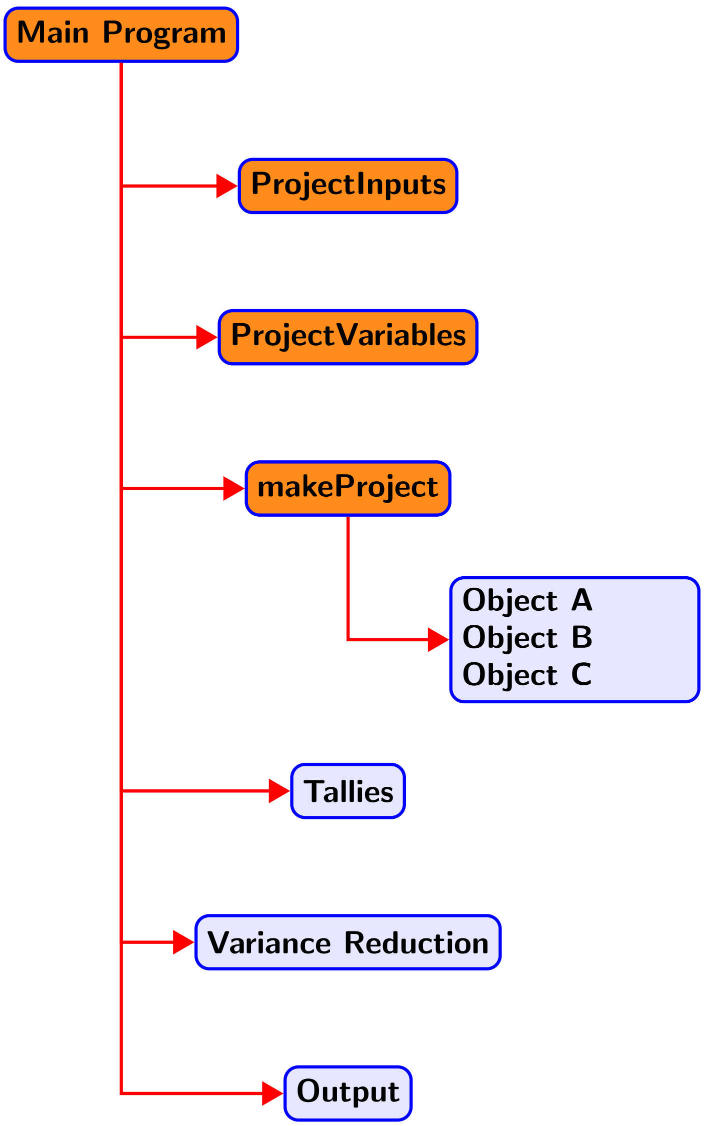

Figure 1: The main program calling sequence is shown. The parts in orange, are expected to be constructed by the user. Bespoke objects can be added for a project but it is not necessary.

2.1 Main

The main function for CombLayer follows a relatively linear template. Consider the example

```
int
main(int argc,char* argv[])
{
  int exitFlag(0);                // Value on exit
  ELog::RegMethod RControl("","main");
  mainSystem::activateLogging(RControl);
  std::string Oname;
  std::vector<std::string> Names;
  std::map<std::string,std::string> Values;

  // PROCESS INPUT:
  InputControl::mainVector(argc,argv,Names);
  mainSystem::inputParam IParam;
  createPipeInputs(IParam);

  Simulation* SimPtr=createSimulation(IParam,Names,Oname);
  if (!SimPtr) return -1;

  // The big variable setting
  setVariable::PipeVariables(SimPtr->getDataBase());
  InputModifications(SimPtr,IParam,Names);

  // Definitions section
  int MCIndex(0);
  const int multi=IParam.getValue<int>("multi");
  try
    {
      SimPtr->resetAll();

      pipeSystem::makePipe pipeObj;

      World::createOuterObjects(*SimPtr);
      pipeObj.build(SimPtr,IParam);
      SDef::sourceSelection(*SimPtr,IParam);

      SimPtr->removeComplements();
      SimPtr->removeDeadSurfaces(0);
      ModelSupport::setDefaultPhysics(*SimPtr,IParam);

      const int renumCellWork=tallySelection(*SimPtr,IParam);
      SimPtr->masterRotation();
      if (createVTK(IParam,SimPtr,Oname))
        {
          delete SimPtr;
          ModelSupport::objectRegister::Instance().reset();
          ModelSupport::surfIndex::Instance().reset();
          return 0;
        }

      if (IParam.flag("endf"))
        SimPtr->setENDF7();

      SimProcess::importanceSim(*SimPtr,IParam);
      SimProcess::inputPatternSim(*SimPtr,IParam); // energy cut etc

      if (renumCellWork)
        tallyRenumberWork(*SimPtr,IParam);
      tallyModification(*SimPtr,IParam);

```

```
      if (IParam.flag("cinder"))
        SimPtr->setForCinder();

      // Ensure we done loop
      do
        {
          SimProcess::writeIndexSim(*SimPtr,Oname,MCIndex);
          MCIndex++;
        }
      while(MCIndex<multi);

      exitFlag=SimProcess::processExitChecks(*SimPtr,IParam);
      ModelSupport::calcVolumes(SimPtr,IParam);
      ModelSupport::objectRegister::Instance().write("ObjectRegister.txt");
    }
  catch (ColErr::ExitAbort& EA)
    {
      if (!EA.pathFlag())
        ELog::EM<<"Exiting from "<<EA.what()<<ELog::endCrit;
      exitFlag=-2;
    }
  catch (ColErr::ExBase& A)
    {
      ELog::EM<<"EXCEPTION FAILURE :: "
              <<A.what()<<ELog::endCrit;
      exitFlag= -1;
    }
  delete SimPtr;
  ModelSupport::objectRegister::Instance().reset();
  ModelSupport::surfIndex::Instance().reset();

  return exitFlag;
}
\label{MainProg}
```

The Main program given in listing **??** highlights the areas that the user should be creating. The remainder of the main() function deals with trapping exceptions, loggin and building variance reduction and tallies into the model.

(i) **createPipeInputs** is a function to define which command line options [above the standard ones] this model should support. It doesn't do anything with them, just a list of options, number of arguements they can take and any default values that the options should take. All options defined here are access from the command line option with a - sign. E.g. *-r* as a renumber operation. In this form of the program, if the main program is run without any options, a list and very brief description of each option is shown (e.g. execute *./pipe*).

If no additional options are required, a call to *createInputs(IParam)* would be expected. Significant restructuring would need to take place to avoid that call.

(ii) setVariable::PipeVariables is the method that registers and sets a default value for all the variables that the model will use.

(iii) makePipe pipeObj and pipeObj.buid(SimPtr,IParam) are the main geometry building calls. Typically 100% of the geometry is built in this zone. It is not a place for tallies, variance reduction and other non-geometry items.

3 Installation

CombLayer is predominately written for the Linux platform using C++ compilers that support C++11 or greater. The code is available from
https://github.com/SAnsell/CombLayer,
either as a download of a zip file or by cloning/pulling the git repository.

3.1 Requirments

CombLayer needs to have the GNU Scietific Library [GSL] and the boost::regex system along with the STL libraries from your C++ compiler. The GSL can be avoided with the -NS flag in the getMk.pl and the CMake.pl script but some functionality will be lost, particularly in the choice of variance reduction methods.

Additionally, the primary build system uses cmake. There is another that just uses make but is significantly more time-consuming.

Functional documentation is supported using Doxygen and the construction of new cmake text files can be done via PERL scripts.

Currenly it is know that gcc version 4.6 and above can compile CombLayer as can clang (all tested versions). gcc 4.4 which is often the default on RedHat systems (2015) does not work.

3.2 Basic build method

If a clean directory is made and then the .zip file is uncompressed, the following commands should build a version of CombLayer.

```
./CMake.pl
cmake ./
make
```

This should make a number of executables, e.g. ess, simple, fullBuild etc. These can be used to make a simple model with commands like

```
./simple -r AA
```

This will produce an output file AA1.x which is a MCNP model.

4 Link system

CombLayers geometry is composed of a set of objects that have slightly stronger rules than a typical MCNPX model. Obviously any MCNPX model can be represented as a CombLayer model and in the extreme case that is done by defining one object to contain the MCNPX model. However, the little benefit would be derived from such an approach.

The basic geometric system is to build a number of geometric classes and construct the model by incorporating those into the desired configuration. Each geometric class is designed to be built and an arbitary position and rotation, be of an undetermined number, and interact with its surroundings in a well defined manor.

In object orientated programming, functional rules and properties are normally added to objects by inherritance. CombLayer follows that pattern. As such most geometry item classes inherit from base classes within the attachSystem namespace.

4.1 AttachSystem Namespace

The CombLayer system is built around the interaction of FixedComp units, ContainedComp units and LinkUnits. The use of these and their interactions are the basic geometric building tools. These object reside within the attachSystem namespace.

Almost any geometric item can be designated as a FixedComp object. This is done by public inherriting from directly from the FixedComp, or by inherriting from one of the more specialised attachSystem objects e.g. TwinComp or LinearComp.

4.2 FixedComp

The basic FixedComp object holds the origin and the orthoganal basis set (X/Y/Z) for the geometry item being built. In addition it holds a number of LinkUnits which provide information about the outer (and/or inner) surfaces and positions on the geometric item.

As with all Object-Orientated (OO) constructions their is an implicit contract that the inherited object should adhere to. This is normally expressed as the *Liskov Substitution principle*: This principle states that functions that use pointers/references to the base object must be able to use the objects of derived classes without knowing it. In this case, that means that modification of the origin or the basis set should not invalidate it and that the object should do the expected thing. E.g. if the origin is shifted by 10 cm in the X direction the object should move by 10cm in the X direction. It also means that the basis set must remain orthogonal at all time.

Other than providing an origin and an basis set, the FixedComp has a number of link points. The link points are there to define joining surfaces, points and directions. Each link point defines all three parts.

For example a cube might have 6 linkUnits, and each linkUnit would have a point at the centre of a face, a direction that is normal to the face pointing outwards and a surface definition that is the surface pointing outwards. [Note that in the case that the link points define an inner volume, for example in a vacuum vessel, then the surfaces/normals should point towards the centre.]

The actual link surface does not need to be a simple surface. In the case, that an external surface needs multiple surfaces to define the external contact these can be entered into a link-rule. For example, if the cube above was replaces with a box with two cylindrical surfaces the link surface would be defined as the out going cylinder intersection with a plane choosing the side.

In the case of an equiry for the linkSurface (e.g. to do an line intersection) then it is the first surface that takes presidence. However, all actions can be carried out on the link-rule including line intersections etc.

4.3 ContainedComp

The ContainedComp defined both the external and interal enclosed volume of the geometric item. It is most often used to exclude the item from a larger enclosing geometric object: e.g. A moderator will be excluded from a reflector, or it can be used to exclude a part of the geometric item from another geometric object. E.g. two pipes which overlap can have one exclude itself from the other.

In CombLayer, the ContainedComp are considered the primary geometric item, i.e. it is the ContainedComp that is removed from the other items. However, it is used in a two stage process wereby cells are registered to be updated by the ContainedComp at a later date. This was to allow forward dependency planning but has more or less been superseded by the attachControl system.

5 Model Runtime control

C++ programs start from the main() function and in CombLayer the runtime control has been keep mostly in the main() function. Clearly that could be further refactored out but CombLayer lacks the sophisticated top level type abstraction that is required to do this in a generic way, so copy/pasted structure is used with variance to the particular model required. The sole advantage of the abscence of a top level abstraction is that the user is the freedom in writing new objects which allows other programs to be incorporated by making their main function a minor function and directly calling.

The structure of two example main()s will be compared from the units that exist with the stanard CombLayer distribution. That is *bilbau.cxx* and *reactor.cxx*. These build the delft reactor model and the Biblau low energy spallation source.

First part of the code is along list of #include's. They are the mainly dependency list of the objects *Simulation, weightManager, and tallySelector.* This can and should be copied at will. Do not make an file with them all in [see 1.1.1].

At the end of the include section there is typically, one or two model specific includes. These normally include *makeXXX.h* file and anything that they directly depend on. In the case of bilbau it is just *makeBib.h* whilst for reactor it is both *makeDelft.h* and *ReactorGrid.h*.

5.1 makeModel

The makeModel object is the place that creates, initializes and manages inquires for the instances of all the geometric components. Primary objects need to be created and registered with the objectRegister **??**. The makeModel component is

Tallies are the fundamental reason for running MCNPX. However, the manor in which MCNPX specifies tallies is not compatible with a vairiable defined model because in most cases the required tally is relative to an object whose position is unknown.

This problem has been addressed by allowing most tallies to use the FixedComp link system.

5.2 Tally System

The tally system is accessed either by a simple command line menu system, or via an XML file. The command line help system is very primative but can remind the user of the bais

5.3 Point Tally

Point tallies are fundamentally a 3D vector in space. In CombLayer, there are three levels of position available: (a) Real MCNP(X) output position, (b) CombLayer master origin position before master rotation, and (c) relative position to an object. Both (a) and (c) are well supported, however, to do option (b) there needs to be some real care with the layout of the calling sequence in the main() function. The fullBuild.cxx example is a suitable option to follow, but checking will be needed.

5.3.1 Free Point tally

The simplest way to put a point into CombLayer is to use a free point.

```
./prog -T point free 'Vec3D(300.0,10.0,5.0)' Output
```