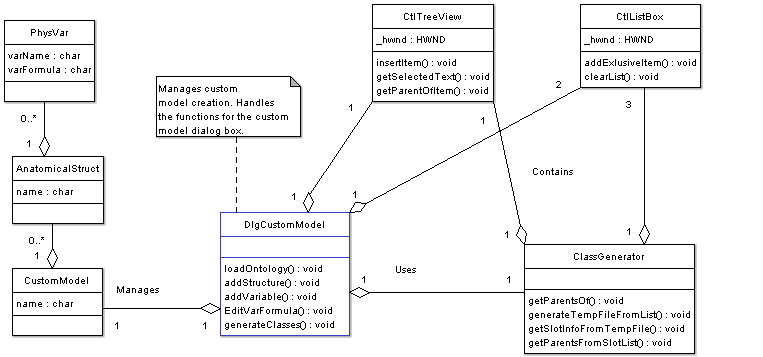
Documentation For “Custom Model” Classes

High Level Design

Robin Hu (Oct. 1, 2007)



(Figure 1: A UML diagram describing the relationships between classes involved in custom model creation.)

**1.1**

**Problem**: Find a way to allow the user to create a custom model in our program. The user’s model should have anatomical structures and physiological variables. The user should be able to select an anatomical structure from a list. The program can determine the selected anatomical structure’s parents and slot information. The program should also be able to generate the required C++ program files based on what the user selected (this is done by calling *ClassBuilder*).

**1.2**

**How is a custom model handled?**

The class called *DlgCustomModel* manages custom model creation through a user interface. The user interface has tools allowing the user to add/remove anatomical structures, add/remove physiological variables and edit physiological variable formulas. Those are managed through special methods in *DlgCustomModel*.

The custom model that the user creates is stored in memory. A classed called *CustomModel* stores this data. Each instance of *CustomModel*  has a list of *AnatomicalStruct* objects. Each *AnatomicalStruct* instance has a list of *PhysVar* objects. Lists are handled using the *ListContainer* class.

**1.3**

**Special User Interface Classes**

There are two special classes that help manage the user interface: *CtlListBox* and *CtlTreeView*. You would use these classes if you wanted to handle list boxes and tree lists (the classes hide the underlying Win32 function calls). For example, *CtlListBox* has methods that allow the programmer to insert and remove items from a list box. These two classes are used by *DlgCustomModel*.

**1.4**

**How to Find Class Information**

So the user selected an anatomical structure from the tree list. How does the program figure out its super-classes? What about its slots? The programmer would use the *ClassGenerator* class. *ClassGenerator* has a method that allows the programmer to find all the superclasses of a certain class. In order for that to work, the tree list window handle must be passed into the *ClassGenerator* instance. That’s because *ClassGenerator* actually uses the tree list to find information about superclasses. The superclasses are all inserted into a list box. In that sense, *ClassGenerator* actually uses Win32 windows to process data!

Suppose you now have a list box filled with class names (like all the superclasses of the anatomical structure “Lung”). You need to figure out all the aggregations and slot information for these classes. That information is stored in a “Protégé Instance File”. This file is so big, I had to break it up into about 30 smaller files. The classes in this file are stored alphabetically.

What I do when I need information about a certain class is, traverse the Protégé Instance File until I find the class I am looking for. I then copy the class information over to a temporary file (called **tempClasses.dat**). I do this for every single class I need. The temporary file can have dozens upon dozens of different classes.

When I am done creating the temporary file, I can call a method from *ClassBuilder* to process the data in the temporary file. *ClassBuilder* can then generate the actual C++ files using a combination of parsing routines.