Introduction to the OpenSim API

Before you get started with Scripting or learning how to use the C++ API in the Developer's Guide, you should review this background information:

What is an API?

An **application programming interface** (**API**) is a particular set of rules ('code') and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the graphical user interface (GUI) facilitates interaction between humans and computers. Using the OpenSim API, other programs such as any C++ programming software or Matlab can access the OpenSim source code. In this way, OpenSim developers can make use of existing OpenSim code, to build on it and extend the available capabilities. For example, the API can be useful for creating Main programs to build and run models, or for creating new analyses, actuators (e.g. an altered muscle model), or controllers.

Why use the OpenSim API?

In order to augment existing models, incorporate custom components, write analyses and build custom scripts, it is necessary to have a general understanding of which classes and objects in OpenSim are responsible for which actions and behaviors. The accessible methods of all the public classes in OpenSim (that other programs/applications can call) define its Application Programming Interface, or API

To use the OpenSim API in both scripting and development reasons, its important to know the OpenSim API structure and its dependencies. OpenSim is written using object-oriented programming. It consists of a large set of classes. You can view all classes and their hierarchical structure available in OpenSim using **Doxygen**. This is an automatically generated overview of all available code from internal C++ files. Doxygen is your best resource to view all the different classes and functions that are available during scripting and development. Information on how to use OpenSim Doxygen can be found on the Guide to Using Doxygen page.

By interacting with the API, you can interface with OpenSim's libraries and use them in your scripting environment or in development. This could mean that you could use Matlab or Python to manipulate OpenSim objects and run methods (Scale, Inverse Kinematics) or you could use C++ language to build your own class that OpenSim can use, e.g. a new muscle model. It could also mean that you build a standalone (main) program that uses existing OpenSim classes to run custom simulations.

You can also convert a newly developed program in to a **plug-in**, a software component that adds specific abilities to the OpenSim application. Your plug-in will be a separate dynamically linked library (.dll on Windows), which you will be able to load into the GUI or into your main program or have it be used by different tools. An example of a plug-in is given in Adding New Functionality where an Analysis plug-in is made, and used from within the GUI.

Some C++ Basics

OpenSim is written in C++. C++ is a general-purpose programming language. It is regarded as an intermediate-level language, as it comprises a combination of both high-level language features (more abstract and conceptual) and low-level language features (directly interfacing with hardware). If you are not familiar with C++, it is advisable to learn some basic C++ programming language before starting the examples in the Developer's Guide. Understanding the basic concepts of C++ and object oriented programming will also help you master OpenSim Scripting. A good free, online course is available on http://www.cplusplus.com/doc/tutorial/. Some limited C++ basics are described here.

C++ code, in contrast to for example Matlab code, needs to be **compiled** or **built**, before it can be run. A **compiler** is a computer program (or set of programs) that transforms source code written in the programming language (the source language) into another computer language (the target language, often having a binary form known as object code). Visual Studio is an example of a compiler. The most common reason for wanting to transform source code is to create an executable program.

An **executable** (.exe) file is a program that causes a computer to perform indicated tasks according to encoded instructions (as opposed to a data file that must be parsed by a program to be meaningful). These instructions are typically machine code instructions for a physical CPU. Thus, when a C++ program is compiled, the user-readable programming language is transformed to computer-readable instructions.

OpenSim is a collection of executables and dynamically linked libraries (.dll). A **C++ library** is a collection of functions, constants, classes (see below) that are executable or callable at run-time and supply the full range functionality to the application (like OpenSim). Libraries enable the same implementation for a given functionality (like reading data from a file) to be supplied to any application that links to it (e.g,. the GUI, or the Matlab scripting interface) and is therefore more generally useful.

One of the main features of C++ is that it facilitates **object-oriented programming**. Rather than writing code in a potentially long sequence of executable lines (e.g., as in procedural languages such as Matlab), object-oriented programming allows the programmer to design applications from a point of view of communication between things, known as objects. The code consists of many **objects**, which all belong to certain **classes**. These classes define what objects can do and how they interact with each other, which defines the object's interface. This allows a greater reusability of code in a more logical and productive way. Some general descriptions of the differences between procedural and object oriented languages can be found here and here.

Structure of the OpenSim API

OpenSim is written using object-oriented programming. It consists of a large set of classes. For example, each muscle in an OpenSim model is an object (an instantiation) of a certain muscle model class, e.g., 'Thelen2003Muscle', which is a member (child) of its parent class 'ActivationFiberLengthMuscle, which in turn is a member of the more general class 'Muscle', which in turn is a member of the more general class 'Actuator', etc.

You can view all classes and their hierarchical structure available in OpenSim using **doxygen**. This is an automatically generated overview of all available code. You can access doxygen either through your OpenSim installation directory, in the folder: C:/OpenSim3.0/sdk/doc/OpenSimAPI.html. If you click on index.html, you will see an overview of the OpenSim hierarchy. As an example, you could click on ModelComponent, which will give you a list of model components, all members of the general class ModelComponent. If you then click on Force, you will get a list of available force generators, etc.

Using C++, you can interface with OpenSim's libraries and use or add to them. This could mean that you can build your own class that OpenSim can use, e.g. a new muscle model, which could be a child of the more general class 'Muscle'. This means all the attributes of Muscle are inherited by your new class and therefore you only need to specify the new or different functionality that defines the behavior of your muscle class. It could also mean that you build a standalone (Main) program that uses existing OpenSim classes, for example to run a simulation.

What is an OpenSim "main" program?

A main program in C/C++ results in a standalone executable that you can run from a command prompt or by double-clicking in Windows. All C/C++ programs have a main() function, which can be as simple as printing "Hello World", or it can invoke several libraries to produce complex applications (e.g., Microsoft® Word and Excel). By including the OpenSim libraries, your main program can call the OpenSim API, and you may also include any other (C++) libraries that provide additional computational and/or visualization resources. Main programs are extremely flexible, but they are particularly useful for streamlining/automating processes independent of the GUI. For example, ik.exe, id.exe, and cmc.exe (available with the OpenSim distribution) are main programs that take setup files and perform tasks related to the OpenSim workflow. Alternatively, users have created their own main programs to systematically scale strengths of all muscles in a model, run forward simulations with their own controllers, perform design optimizations, etc. An advantage of a main program (compared to a plug-in) is that any classes you define in the project are immediately useable by your program. This can make prototyping and testing of your new component or analysis faster and easier without having to wrap, load, and call your plug-in from the GUI.

What is an OpenSim plug-in?

When creating a new component (e.g., a force controller) or a new analysis, you may want to include it in an existing model, run it with existing tools, and /or share your contribution with colleagues. An OpenSim plug-in is a way of packaging your code in a dynamically linked library (.dll on Windows) so that an existing OpenSim application can recognize it, load it, and make your code "runnable". An example of a plug-in is given in Adding New Functionality whe re an Analysis plug-in is made, and used from within the GUI.

Common Object Oriented Programming Terms

This table is a list of common terms used in object oriented programming.

Name	Description	Example
Class	Classes can be thought of as the instruction set to build an object. Classes represent generalizable features and behaviors (methods) of some type (e.g., forces, bodies, muscle, list of objects, analysis) and contain properties, attrib utes, or data members that are used to create an object of that type. An abstract Class defines the shared methods of many 'like' classes and can notbe instantiated to an object directly. Concrete Classes can be instantiated to objects and inherit the methods of parent abstract classes while defining their own unique methods.	To create a Rectus Femoris (RF) muscle object you could use the Thelen2003Muscle class. This class has a number of editable properties (e.g., maximum isometric force, tendon slack length), as well as a number of sub-objects such as Geometry Path. Once the properties inherited by this Thelen2003Muscle class have been set, an instance (object) of the class called 'Rectus Femoris' can be created.
Object	Objects are considered instances of classes. Objects contain both properties and methods (behaviors) inherited by their class. Classes can create objects that represent analyses (e.g., InverseKinematicsTool) and data lists (e.g., MarkerData) as well as physical systems.	Every muscle attached to the gait2392 model is an "object" (an instance of the Thelen2003Muscle class). The properties of each muscle can be individually specified.
Method	Methods define the behavior of an object when you run a program.	OpenSim::InverseKinematicsTool::run () executes the run method of the ***InverseKinematicsTool*** object, performing inverse kinematics according to the specified inputs.
Inheritance	Classes can inherit attributes and behavior from pre-existing classes called base classes, superclasses, or parent classes. The resulting classes are known as derived classes, subclasses, or child classes. The relationships of classes through inheritance give rise to a hierarchy.	Thelen2003Muscle (class) inherits the attributes and behaviors of its parent class 'ActivationFiberLengthMuscle', which in turn inherits the attributes and behaviors of the more general class 'Muscle', which inturn turn inherits the attributes and behaviors of the more general class 'Actuator', etc.
Handle	A handle is a reference to an object's resources that, once accessed, gives you the ability to read and write.	*ECRB = myModel.getMuscles().get("ECRB")* would give you a handle to the ECRB object that exists in the model. You could then edit the parameters of the *ECRB muscle (ECRB.setTendonSlackLength(0.22))*

ModelComponent	Model components are a specific type of Object that is found in OpenSim. As such, when talking about Components, all Components are Objects but not all Objects are Components. Components have a simplified interface that allows each component to connect and communicate with other components and the system as a whole. This specially designed communication layer affords users with the ability to build models much like building with lego blocks, stacking components in series or parallel.	Actuators, Bodies, Controllers, and Joints are allModelComponents. Analyses and Tools are not ModelComponents.

Programming Resources

Here are some useful things for developers on the programming side of things:

- http://stackoverflow.com/ - Q & A community for computer programming (and others)
- http://overapi.com/ - The programming cheat sheets (Including C++/Java/MATLAB/Python/NetBeans/etc)
- http://www.cplusplus.com/ - C++ Resource Network
- http://www.rapidee.com/en/download - Tool to streamline editing of your environment variables