1 Overview – Objects in Open Wonderland

This document provides a **High Level** view of how to work with *objects* in Open Wonderland. We program new objects to be rendered in world by creating new ***Cells*** that will be packed for deployment in a ***module***. A *Cell* will have a representation in the client (by creating a cell that extends *Cell.java* or other similar classes) and a representation as a ***Managed Object*** (Darkstar terminology) in the server (by extending *CellMO.java*, or other similar classes).

There are many different types of *modules*, but this document describes just two; 3D objects created with the JMonkey Engine, and 2D applications created with Java Swing.

2 Working with Open Wonderland Modules

This section will discuss topics such as the main classes in a module, and the similarities and differences between JMonkey Engine modules and Swing modules.

Please note this is a ***very simplified view*** of how things are done, just pointing at classes but not at methods; to get the full picture have a look at the real code.

2.1 A JMonkey Engine module: Shape Cell

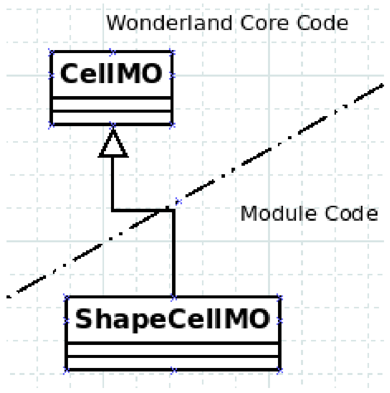
This section is based on the 4 part tutorial developing a new Cell.

A jME module is a module that will be represented in-world by a ***Cell*** created using the jME and MT-Game capabilities of Open wonderland. There is no need to understand all the implications of using these two APIs, but it would be good to have a look at the corresponding tutorials.

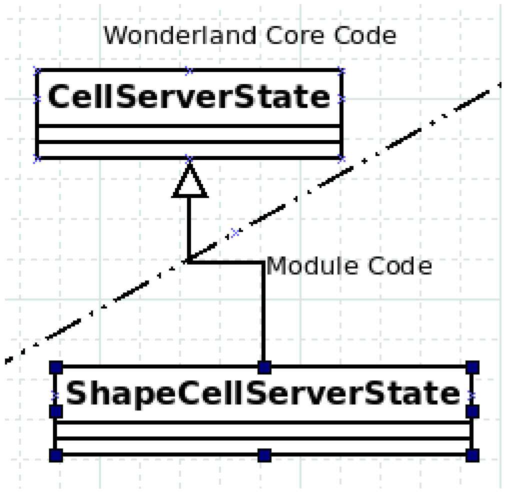
The tutorial divides the main classes into three packages, *Server*, *Common*, and *Client*:

Server Side

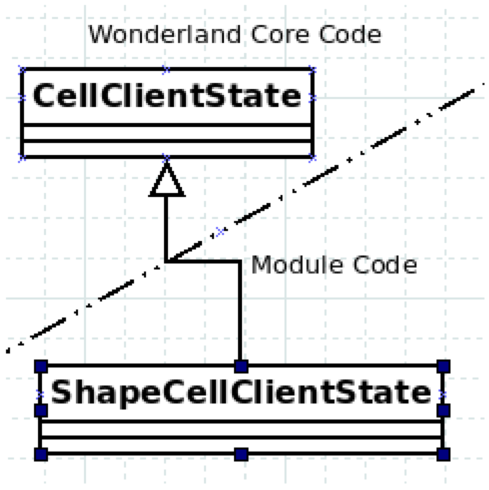
The ***ShapeCellMO*** is a *Managed Object* in the Darkstar gaming engine:



Common Package (needed in both server and client sides)
The main use of the ServerState is to persist an XML copy of the state of the Cell into the filesystem during snapshot creation. That is why, as we will see in the code, it is annotated using the JAXB library.

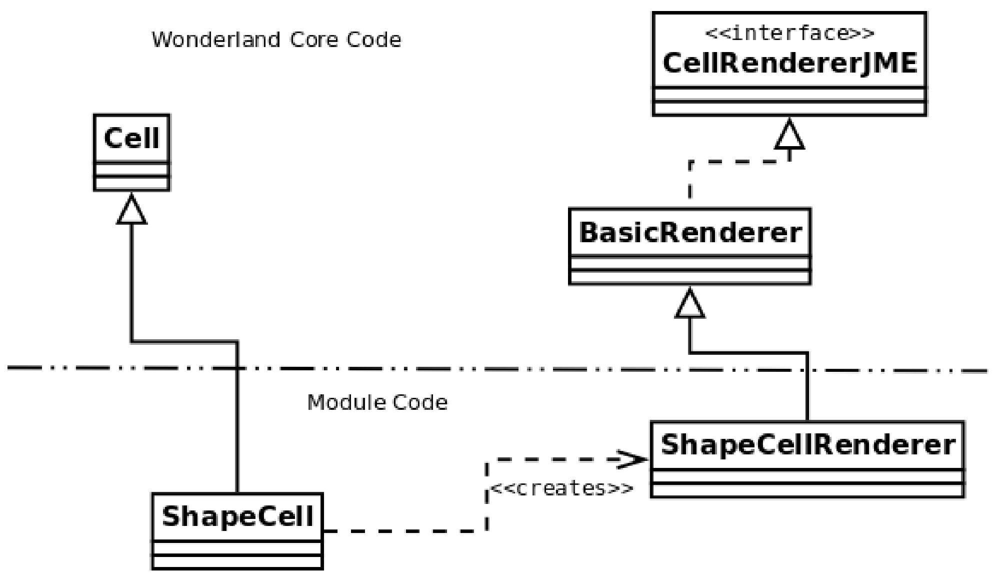


The main use of the ClientState is that this class is sent (in serialised form) from the server to the client when the server needs to communicate with the client.



In part 4 of the tutorial, we can see one more class in this package, a Message class. The basic need for messages in this context is the following: Imagine that you change the shape of the Cell you are seeing (by clicking on it). The fact that you clicked on the cell will have to be communicated to the Server. The Server then will send a message back to **all** the clients connected so that the shape is updated in all clients. If you don't send messages, the other clients won't know what happened, and the shape will only be updated in your own client (see video: http://www.youtube.com/watch?v=yJe_kaHnfDQ).

Client Side



The ShapeCell is the entry point for the object representation.
The ShapeCellRenderer uses jME + MT-Game code and dependencies..

Additional Classes:
The tutorial also shows a class implementing the CellFactorySPI. That class is needed if you want to show the module in a palette so that you can insert it inworld. Other ways of doing this could be by creating a plugin(we will mention them later) that can add a menu to the OWL client.

What else is in the tutorials?
Part 2 describes mouse input → pure Swing in a wonderland context.

Part 3 adds an /art folder with a texture. The ant file (build.xml) has to be modified so that it bundles the new folder in the resulting jar.

Part 4 deals with messages between server and clients, and we will see it in code.

2.2 A Swing module

The main differences with a jME module happens on the client side, to program the client representation of the Cell.

Server and common packages have the same classes and structure as in the previous module (except for the fact that this modules does not use messages).

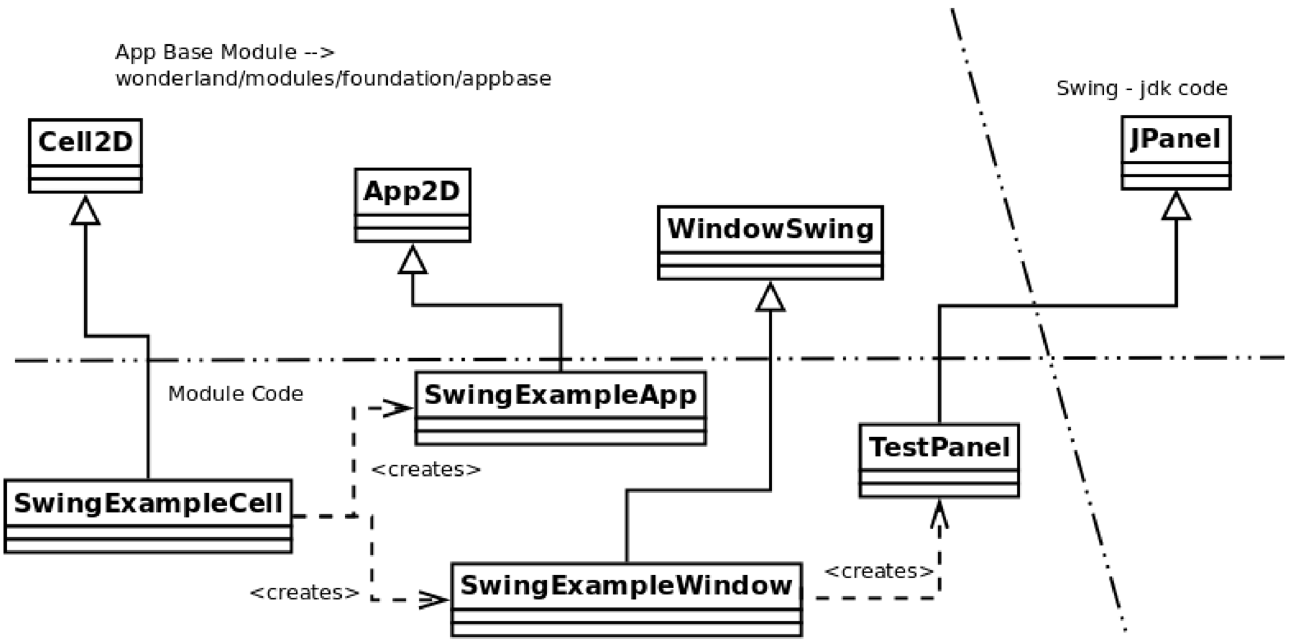
Client Side
Any swing module will depend on the appbase module that can be found in ~/wonderland/modules/foundation/appbase.

This is an extract from the Swing tutorial by Deron Johnson (https://docs.google.com/Doc?docid=0Adt0T7CUKzB9ZGNrODdxOXBfMjhncmZkM3doYg&hl=en&pli=1):

The Wonderland App Base is a module which allows 2D applications to run inside the 3D virtual world. It provides support for both Swing applications as well as shared, conventional applications (such as X11 apps). This document focuses on Swing applications. The App Base provides a set of utilities to Swing module developers. The most important of these is a class called WindowSwing, which is a 3D object. This object takes a lightweight Swing JComponent as an attribute. The image of this JComponent is continuously rendered into a texture map which is then mapped onto the 3D object, which is a flat rectangle. In this document, we will also refer to a WindowSwing as a swing window. Note that these swing windows are different from the "top level" Swing windows that you might use to display a JDialog or JOptionPane as a peer of the Wonderland client main window in the native window system. In this document a swing window is a WindowSwing which can actually be displayed with a 3D orientation in the 3D world.

A developer-written module which uses WindowSwing is called an app module. Specifically it is a Swing app module. A Swing app module must provide a cell which is a subclass of App2DCell. This is located in the package org.jdesktop.wonderland.modules.appbase.client.cell. In addition, the cell must create an app object which is an subclass of App2D. Once the app object has been created, one or more WindowSwings can be created. These windows are associated with, or added to, the app object. When the cell becomes visible to a user, the module developer should call setVisible on one or more of the windows.

Here is a depiction of the above explanation in the context of the Swing tutorial:



Do you need a factory in a Swing module? Only if you want it!

3 Other things that can go in a module

Plugins, context menu factories, mouse event listeners, components (a capability module), code to embed on glassfish (a web module), and a bunch of stuff more that I haven't discovered yet!