Project Wonderland (v0.5): Using Deployed Models in Cells

Introduction

In previous tutorials, you learned how to import 3D models via drag-and-drop or via the Import Tool (see Importing 3D Models for more details) and use the world assembly tools built into the Project Wonderland client to position the 3D model in the world (see Assembling Worlds for more details). Also in previous tutorials, you learned how to develop a new kind of Cell (see Developing a New Cell for more details) -- in these basic tutorials you drew a shape using the jMonkeyEngine API.

In this tutorial, you will learn how to incorporate deployed 3D models into Cells you have developed. You can find the entire source code for this module in the "unstable" section of the Project Wonderland modules workspace, under the **satellite-model-tutorial/** directory. For instructions on downloading this workspace, see Download, Build and Deploy Project Wonderland v0.5 Modules.

This tutorial is designed for Project Wonderland v0.5 User Preview 2.

Expected Duration: 45 minutes

Prerequisites

This tutorial is geared towards advanced Project Wonderland developers. Before completing this tutorial, you should have already completed the following:

- Download, Configure, Build and Run from the Wonderland v0.5 Source
- Download, Build and Deploy Project Wonderland v0.5 Modules
- Project Wonderland: Working with Modules
- Developing a New Cell: Part 2, Part 4
- Importing 3D Models

In these tutorials you learned how to download and compile the Wonderland source code, run the Wonderland server, compile an example module project, and install the module into your Wonderland server. You also learned how to develop a custom Cell and deploy 3D models to Project Wonderland.

This tutorial also assumes you have downloaded the Communications Satellite by Marian from the Google 3D Warehouse. It also assumes you have converted the SketchUp (skp) file into a Google Earth (kmz) file. For details, please see Importing 3D Models.

Deploying 3D Models

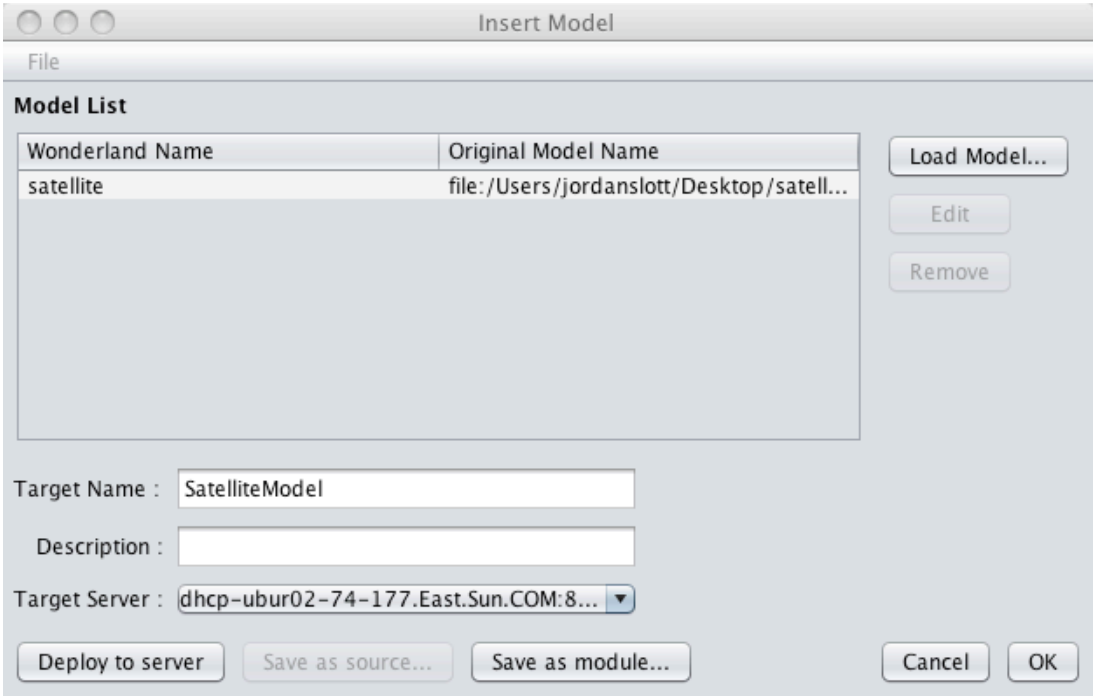
Before you may use a 3D model in Project Wonderland, you must first **deploy** it. The act of deploying a 3D model to Project Wonderland generates extra files that help describe the 3D model content and package all of the files together to use. When you dragged-and-dropped a 3D model into the Project Wonderland client or used the Model Import tool in Importing 3D Models, you were deploying a 3D model. In the latter case, the system packaged up the deployed model into a module and uploaded it to the server.

The Model Import tool also lets you download the deployed model packaged as a module to your local computer. Follow these steps to create and download a module with the Communications Satellite model, assuming you have saved it to your computer desktop as a file named **satellite.kmz**.

1. Select the Import -> Model menu item
2. Click the Load Model... button on the Insert Model dialog and select the **satellite.kmz** file on your computer desktop. Click Open
3. Click OK in the Model Import dialog to accept the position, rotation, and scale of the imported model
4. In the Target Name text field, enter SatelliteModel
5. Click the Save as module... button on the Insert Model dialog. Select the desired directory in which to save the module and click OK

Your Insert Model dialog should appear as follows:

Figure 1: Insert Model dialog populated with name of imported model



You should see a file named **SatelliteModel.jar** in the directory you have selected on your local computer. This is a Project Wonderland module that contains the deployed Communications Satellite model.

Delving into the Structure of Deployed Models

By now, you are at least somewhat familiar with the structure of a Project Wonderland module (see Working with Modules for more details). The following is a listing of the **SatelliteModel.jar** module you have just created. You can obtain the same listing by invoking **jar tvf SatelliteModel.jar** on the command-line.

```
     0 Wed Sep 23 12:50:28 EDT 2009 META-INF/
     2 Wed Sep 23 12:50:28 EDT 2009 META-INF/MANIFEST.MF
   240 Wed Sep 23 12:50:28 EDT 2009 module.xml
     0 Wed Sep 23 12:50:28 EDT 2009 art/
     0 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/
     0 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/images/
  2139 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/images/texture0.JPG
 95052 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/images/texture1.JPG
 96601 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/images/texture2.JPG
  2139 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/images/texture3.JPG
143064 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/images/texture4.JPG
 96601 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/images/texture5.JPG
147851 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/images/texture6.JPG
143064 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/images/texture7.JPG
     0 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/models/
138941 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/models/sat.dae.gz
   759 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/satellite.kmz.dep
   218 Wed Sep 23 12:50:28 EDT 2009 art/satellite.kmz/satellite.kmz.ldr
```

The Communications Satellite 3D model is packaged into the module under the **art/** directory. When Google Earth (kmz) files are deployed, all of its files (e.g. COLLADA(TM), textures) are stored beneath a directory named after the Google Earth file name. In this case, Project Wonderland creates a **satellite.kmz/** directory in which to keep all of the deployed files. Beneath the **satellite.kmz/** directory, the 3D model geometry is kept beneath the **models/** subdirectory (the system also compresses the COLLADA (dae) file using gzip to save both disk space and decrease download time to clients) and the textures are kept beneath the **images/** directory.

Project Wonderland also creates two additional XML files: a deployment file (**satellite.kmz.dep**) and a loader settings file (**satellite.kmz.ldr**), found beneath the **satellite.kmz/** directory.

The deployment file

The deployment file is the "entry point" to loading the 3D model into the system: it provides the necessary information to tell Project Wonderland where to find the 3D model geometry and also which "loader" to use to load the geometry. The following is the **satellite.kmz.dep** deployment file for the Communications Satellite model:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<deployed-model>
    <version>1</version>
    <modelURL>wla://SatelliteModel/satellite.kmz/models/sat.dae.gz</modelURL>
```

```
    <loaderDataURL>wla://SatelliteModel/satellite.kmz/satellite.kmz.ldr</loaderDataURL>
    <modelBGScale>
        <x>1.0</x>
        <y>1.0</y>
        <z>1.0</z>
    </modelBGScale>
    <modelBGTranslation>
        <x>-0.12887971</x>
        <y>-0.28338826</y>
        <z>0.0986658</z>
    </modelBGTranslation>
    <modelBGRotation>
        <x>-0.99999994</x>
        <y>0.0</y>
        <z>0.0</z>
        <angle>1.5707964</angle>
    </modelBGRotation>
    <modelLoaderClassname>org.jdesktop.wonderland.modules.kmzloader.client.KmzLoader</mo
</deployed-model>
```

While you do not need to have knowledge of the meanings of the information contained in this file, a short explanation is in order. The tag describes the URI of the 3D model geometry: the **wla** protocol indicates that the model is stored within a Project Wonderland module. The Model Import tool automatically generates the value of this field based upon the name of the module you entered in the **Target Name** field above. The tag describes the location of the loader file associated with the 3D model geometry file given in the tag. Loader files are described in more detail below. Finally, the tag gives the fully-qualified class name of the "loader" that knows how to load the particular 3D model format (i.e. Google Earth (kmz)). While it is beyond the scope of this tutorial, module developers may define new model loaders and deploy them to Project Wonderland to expand the 3D model formats it supports.

The loader file

Associated with each 3D model geometry file exists a loader file that provides additional information on how to load the 3D model. The following is the **satellite.kmz.ldr** loader file for the Communications Satellite model:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<loader-data>
    <version>1</version>
    <modelLoaderClassname>org.jdesktop.wonderland.modules.kmzloader.client.KmzLoader</mo
</loader-data>
```

Rendering a Deployed Model in a Custom Cell

There are two ways that you can use a deployed 3D model in a Cell that you have developed. One, you may use the **ModelRenderer** as your Cell Renderer, or two, you may load the 3D model geometry directly and add the jME **Node** into your scene graph.

Using the ModelRenderer class

The **satellite-module-tutorial/** module in the **unstable/** section of the wonderland-modules workspace is an example of using the **ModelRenderer** class. This module simply defines a custom Cell type that renders a model. It stores the URI of the model in the server state of the Cell and passed to the client and stored in the client-side Cell class (**SatelliteCell**) as the **modelURI** variable. For example, the following excerpt from **SatelliteCellFactory** sets the URI to that of the satellite model you just imported. The URI refers to the satellite model you deployed above, using the deployment (dep) file.

```
    public <T extends CellServerState> T getDefaultCellServerState(Properties props) {
        SatelliteCellServerState state = new SatelliteCellServerState();
        state.setModelURI("wla://satellite-model-tutorial/satellite.kmz/satellite.kmz.de
        return (T)state;
    }
```

Note that the module does not include the imported 3D model, you will have to copy over the files yourself. To do this, first expand the **SatelliteModel.jar** file using the **jar xvf SatelliteModel.jar** command. Then, copy the **satellite.kmz/** directory (see the listing of **SatelliteModel.jar** above) beneath the **art/** directory of the **satellite-module-tutorial/** module. Note that the **art/** directory is bundled into the module's jar file by the line in your module's **build.xml** file.

Also note that you need to change the module name of the deployed 3D model. To do this, edit the **satellite.kmz.dep** file and replace the two instances of the string **SatelliteModel** with **satellite-module-tutorial**. This is necessary since you deployed the 3D model to a module named **SatelliteModel**, but you wish to use it in a module named **satellite-module-tutorial**.

To use the **ModelRenderer** class, implement the **createCellRenderer()** in **SatelliteCell** class as follows:

```
    @Override
    protected CellRenderer createCellRenderer(RendererType rendererType) {
```

```
        if (rendererType == RendererType.RENDERER_JME) {
            try {
                LoaderManager manager = LoaderManager.getLoaderManager();
                URL url = AssetUtils.getAssetURL(modelURI, this);
                DeployedModel dm = manager.getLoaderFromDeployment(url);
                return new ModelRenderer(this, dm);
            } catch (MalformedURLException ex) {
                logger.log(Level.SEVERE, null, ex);
            } catch (IOException e) {
                logger.log(Level.SEVERE, null, e);
            }
        }
        return super.createCellRenderer(rendererType);
    }
```

Before you can use the 3D model's URI, you must first invoke **AssetUtils.getAssetURL()** to transform it into a URL. The **getAssetURL()** method also adds information to the URL that describes the primary server to which your client is connected.

After you compile the **satellite-module-tutorial/** module, install it into your running Wonderland server, and start the client, you should see the Cell appear in the palette:

Figure 2: Satellite Cell in the Palette

When you create this Cell in the world, your client should look something like:

Figure 3: Satellite Cell in the world (click on the image for a full-sized version)

Loading the deployed model geometry

If you want more control over placing your deployed model into the scene graph (e.g. if you wish to load multiple 3D models and add them to your Cell's scene graph), you can write your own Cell Renderer (as done in the Developing a New Cell tutorials) and then attach the 3D model to the scene graph. The following code returns the jME **Node** object representing the scene graph of your deployed 3D model:

```
Node node = dm.getModelLoader().loadDeployedModel(dm, null);
```

Other examples

The Project Wonderland source code contains several other examples of programmatically loading 3D models. All of these examples are part of the core software, found in the **wonderland** workspace. The **phone** module (under **modules/tools/phone/**) and the **microphone** module (under **modules/tools/microphone/**) are examples of using the **ModelRenderer** class. (The former subclasses **ModelRenderer**).

The **portal** module (under **modules/world/portal/**) is another usage of 3D model loading not previously discussed in this tutorial: it registers a Portal Cell in the palette, but never creates its own custom Portal Cell type. Rather, in **PortalCellFactory.java**, it returns the server state of the pre-defined **ModelCell** that displays the portal 3D model. This usage is perfect if you do not need any further custom behavior of a Cell type besides displaying a 3D model.