Get Involved

java-net Project
Request a Project
Project Help Wanted Ads
Publicize your Project
Submit Content

Get Informed

About java.net
Articles
Weblogs
News
Events
Also in Java Today
java.net Online Books
java.net Archives

Get Connected

java.net Forums
Wiki and Javapedia
People, Partners, and Jobs
Java User Groups
RSS Feeds

Search

Web and Projects: »
Online Books: »
Advanced Search

Home | Changes | Index | Search | Go

-

Project Wonderland v0.5: Applying textures to shapes (Part 3)

by Jordan Slott (jslott@dev.java.net)

Purpose

In this tutorial, you will extend the simple new Cell type you created in Part 1 and Part 2 of this tutorial series by texturing the basic geometric shape that is drawn in the Cell. You will learn how to package artwork in your modules and interact with the asset manager that comes part of the Project Wonderland client software to download the texture (a .png file).

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

You can find the entire source code for this module, including code for future tutorials in the "unstable" section of the Project Wonderland modules workspace. For instructions on downloading this workspace, see Download, Build and Deploy Project Wonderland v0.5 Modules.

Expected Duration: 30 minutes

Prerequisites

Before completing this tutorial, you should have already successfully completed Part 1 and Part 2 of this tutorial series. You will be extending the functionality you implemented there.

Artwork in Modules

In v0.5 of Project Wonderland, artwork may be packaged directly in a module. The artwork itself is deployed by a web server embedded in Project Wonderland. There is no need to setup a separate web server to publish art. In fact, developers can create modules that consist entirely of art to share with others.

By default, the Wonderland client downloads artwork from the Wonderland server. It is also possible to set up other web servers on the Internet to act as repositories for the artwork. The repository.xml file within the module defines the artwork repositories from where the Wonderland client should download the art. In this tutorial, however, you will not need to configure the repository.xml file since you'll be using the Wonderland server itself as the sole artwork repository.

In this tutorial, you'll be texturing your basic shape with a picture of a mountain landscape. Feel free to use your own image as a texture instead of this mountain landscape.

High-Level Design

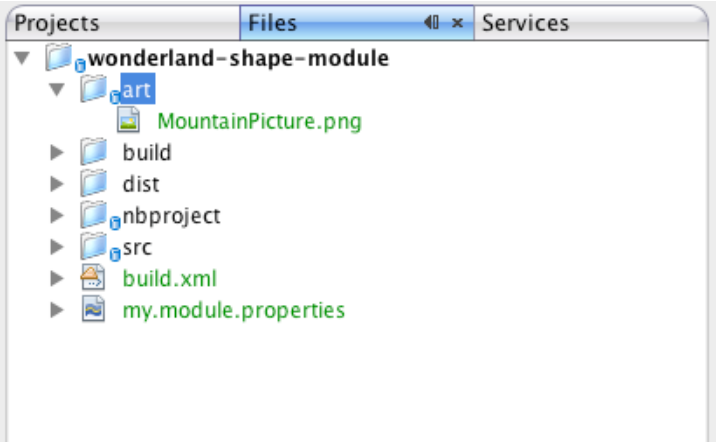
You will enhance the custom shape Cell to texture the sphere or box with an image from an artwork repository. The location of the texture image is given by a URI: this URI defines the module in which the artwork is located and its relative path within the module. This URI is specified in the Cell's factory class. The path of the texture is then communicated to the client-side Cell class that loads the image using the asset management facilities provided by Project Wonderland and then sets the shape's texture.

Packaging the Texture in the Shape module

First, you'll add the texture image to your Netbeans project. You may also accomplish these steps using the command-line too.

1. Download the texture image: MountainPicture.png. For now, save it anywhere on your local hard drive.
2. In Netbeans, click on the Files tab in the upper left-hand window to display a filesystem view of your Netbeans project.
3. Right-click on the **wonderland-shape-module** folder and select New -> Other... In the dialog box, under Categories, select Other and under File Types, select Folder. Click Next.
4. Name the new folder "art" and click Finish. You should see an art folder appear beneath the **wonderland-shape-module** folder.
5. Using your customary operating system file management tools, copy **MountainPicture.png** into the **art/** subdirectory beneath your module folder. (You need to do this outside of Netbeans).

When finished, your Files tab should look like the following:



Next, you must modify the build script, **build.xml** to package the **art/** directory in with the module.

1. Within the Files tab, double-click on **build.xml** to begin editing it.
2. Locate the <module></module> tags within the file.
3. Insert the following right before the closing </module> tag: <art dir="${current.dir}/art"/>
4. Save the **build.xml** file.

Adding Attributes to the ShapeCellServerState Class

First, you need to make some minor enhancements to the ShapeCellServerState class--this class, as you may recall, represents the Cell server state. In ShapeCellServerState.java, add the following declarations near the top of the class:

```
    @XmlElement(name="texture-uri")
    private String textureURI = null;
```

Next, add the following setter and getter methods for this new property:

```
    public void setTextureURI(String textureURI) {
        this. textureURI = textureURI;
    }

    @XmlTransient public String getTextureURI() {
        return this. textureURI;
    }
```

Specifying the Texture URI in ShapeCellFactory

When you create your Cell via the Cell Palette you have to tell it the default texture image to use. In the **getDefaultCellServerState()** method in your ShapeCellFactory.java, add the following line after you create your server state object:

```
        state.setTextureURI("wla://shape-tutorial-module/MountainPicture.png");
```

This is a special URI defined by Project Wonderland to represent an art asset within a module. The URI is identified by the special 'wla' protocol (which stands for Wonderland Art). Immediately following the 'wla://' is the name of the module, in our case *shape*. Following the module name is the relative path of the texture image within the **art/** subdirectory of the module. Note that the **art/** directory itself is omitted from the asset URI.

In previous versions of Wonderland that did not support the Cell Palette, you needed to create your own hand-edited XML file in WFS. This is no longer necessary, but since the texture URI property appears in the Cell's server state object, it will get written out to WFS, and you can still edit the XML file on disk if you wish.

Adding the Texture URI Property to ShapeCellClientState

You will also need to add the texture URI property to the ShapeCellClientState class: this is how the URI is communicated from the Wonderland server to all Wonderland clients. The code to add is similar to the ShapeCellServerState class.

In ShapeCellClientState.java, add the following field declaration near the top of the class:

```
    private String textureURI = null;
```

Next, add the following setter and getter methods for this new property:

```
    public void setTextureURI(String textureURI) {
        this.textureURI = textureURI;
    }

    public String getTextureURI() {
        return this.textureURI;
    }
```

Modify the ShapeCellMO Class

Changes to the server-side ShapeCellMO.java class are relatively simple: you just need to store the **textureURI** property in a member variable and populate the ShapeCellClientState and ShapeCellServerState classes when needed. First, add a member variable near the top of the class to store the texture URI:

```
    private String textureURI = null;
```

Next, when you create a new ShapeCellClientState class in the **getClientState()** method, set the value of the texture URI property from this class. Your **getClientState()** method should include the following line right below where you set the shape type:

```
        ((ShapeCellClientState)cellClientState).setTextureURI(textureURI);
```

Also, when you create a new ShapeCellServerState class in the **getServerState()** method, include the following line right below the where you set the shape type:

```
        ((ShapeCellServerState)state).setTextureURI(textureURI);
```

Finally, for the ShapeCellMO class, you need to modify the **setServerState()** method to fetch the value of the **textureURI** property and set the member variable, **this.textureURI**. Simply add the following line after the **this.shapeType = ...** line:

```
    this.textureURI = ((ShapeCellServerState)state).getTextureURI();
```

Loading the Texture on the Client

Changes to ShapeCell

Your ShapeCell and ShapeCellRenderer classes that resides on the client-side needs modification--in fact, it's where all of the major changes go. In ShapeCell, you will need to access the texture URI property in the ShapeCellClientState class and store it for the renderer. Add the following declaration to the top of your ShapeCell class:

```
    private String textureURI = null;
```

Next, set this field to be the value accessed from the ShapeCellClientState class. In the **setClientState()** method, add the following line:

```
        this.textureURI = ((ShapeCellClientState)state).getTextureURI();
```

Finally, add a getter method so that the renderer can access this field:

```
    public String getTextureURI() {
        return this.textureURI;
    }
```

Changes to ShapeCellRenderer

The ShapeCellRenderer class is where the texture actually gets loaded and rendered. First add some import statements that you will need later (although it is always a good idea to have your IDE 'fix' your import statements so they are correct):

```
import com.jme.image.Texture;
import com.jme.scene.state.RenderState.StateType;
import com.jme.scene.state.TextureState;
import com.jme.util.TextureManager;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.jdesktop.mtgame.RenderManager;
```

All of your changes will occur in the **createSceneGraph()** method, where you'll perform the following additional tasks:

1. Access the **textureURI** property from the ShapeCell class.
2. Load the texture using the Wonderland client-side asset manager.
3. Create instances of the jMonkeyEngine TextureState and Texture objects.
4. Set TextureState object on the jME Node object.

For the firsts step, you can add the following code beneath the initial set of field initialization lines:

```
        String textureURI = ((ShapeCell)cell).getTextureURI();
```

Next, load the texture using the Wonderland's asset manager. Project Wonderland has been designed to understand URIs with the 'wla' protocol. The 'wla' URI needs to translated into a URL, via the **BasicRenderer.getAssetURL()** method. Add the following lines at the end of your method right before the return statement:

```
        if (textureURI != null) {
            RenderManager rm = ClientContextJME.getWorldManager().getRenderManager();
            TextureState ts = (TextureState) rm.createRendererState(StateType.Texture);
            Texture t = null;
            try {
```

```
                URL url = getAssetURL(textureURI);
                t = TextureManager.loadTexture(url);
                t.setWrap(Texture.WrapMode.MirroredRepeat);
                t.setTranslation(new Vector3f());
                ts.setTexture(t);
                ts.setEnabled(true);
            } catch (MalformedURLException ex) {
                Logger.getLogger(ShapeCellRenderer.class.getName()).log(Level.SEVERE, nu
            }
            node.setRenderState(ts);
        }
```

Note that since the texture URI has a 'wla' protocol, Wonderland knows how to deal with it: the texture located within the module is downloaded from the web server embedded in Project Wonderland to serve art assets. The built-in asset manager within the Wonderland client automatically handles downloading and caching this asset for you. You do not need to interact with the asset manager directly. You do need to take one additional step to 'pre-process' the URI, however. This is the purpose of the call to the **getAssetURL()** method (defined by the BasicRenderer superclass). It creates and returns a proper URL object that you may use like any other URL in Java. The **getAssetURL()** method adds some important information that is needed by the system. You don't need to be concerned about the specifics, but in case you are interested, when you print out this URL, it should look something like this:

```
wla://shape-tutorial-module@<host>:<port>/MountainPicture.png
```

Let's explain the code above in a little more detail. First you create a TextureState object that will hold the texture and be attached to the node later:

```
            RenderManager rm = ClientContextJME.getWorldManager().getRenderManager();
            TextureState ts = (TextureState) rm.createRendererState(StateType.Texture);
```

Next, you create the Texture object and load the texture, with the following bit of code and set some of its attributes:

```
                t = TextureManager.loadTexture(url);
                t.setWrap(Texture.WrapMode.MirroredRepeat);
                t.setTranslation(new Vector3f());
```

The details of the jMonkeyEngine API are beyond the scope of this tutorial. For more information, please refer to the jMonkeyEngine Website.

Then, you associated the Texture with the TextureState object:

```
                ts.setTexture(t);
                ts.setEnabled(true);
```

Finally, you must set the TextureState on your Node object with the following:

```
        node.setRenderState(ts);
```

Running With Your New Textured Shape

You must now recompile your module and re-deploy it into your instance of Wonderland. Please follow the instructions in the previous tutorials on how to do this.

- Wonderland Web-Based Administration Guide
- Project Wonderland v0.5: Working with Modules
- Project Wonderland v0.5: Developing a New Cell (Part 1)

If you chose a Box to be displayed, your Wonderland world should look like this: (Click on the image to view a full-sized version)

Next Steps

In the next and final tutorial, you will learn how to synchronize the state of your shape Cell across many Wonderland clients.

Part 4 - Synchronizing the State of the Shape Cell Type