Project Wonderland v0.5: Writing a Client or Server "Plugin"

Introduction

This tutorial describes how you can write a "plugin" for either the Project Wonderland client or server. The term "plugin" is often overloaded and considered synonymous with the term "module" in computing. In Project Wonderland, the two have different meanings:

- **Module:** A Wonderland module is a collection of code (including custom Cell types and plugins), art, scripts, WFS, and other resources that are installed into the Wonderland server
- **Plugin:** A piece of code that is run either on the Wonderland client or server during startup. A **plugin** is packaged into a Wonderland **module**

Plugins on either the client or server can be used for different purposes, as discussed below.

Writing a Client Plugin

A **ClientPlugin** executes code when the client is first initialized. Client-side plugins can be used, for example, to add items to the main menu. On the client, plugins can spawn threads too.

The client-side plugin API follows an initialize-activate-deactive-cleanup paradigm, as illustrated by Figure 1 below.

Figure 1: Wonderland connections

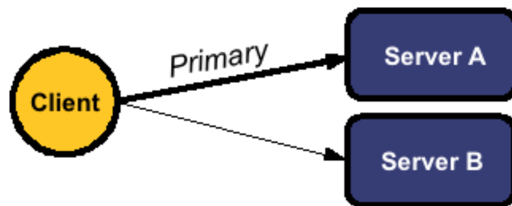

Figure 1 illustrates a client with *connections* to two servers. One of the connections is designated as the **primary** connection. In terms of the Wonderland 3D client, the primary connection represents the world your avatar is currently standing in. The Wonderland architecture provides for the ability for clients to establish connections with more than one server at a time. The secondary (i.e. non-primary) connections may be used, for example, to display a "portal" to the contents of these other worlds.

On the client, a connection to a Wonderland server is represented by the **ServerSessionManager** object. This object maintains a set of Wonderland **sessions** to each server, represented by **WonderlandSession** objects. You can think of a **WonderlandSession** as an individual communications channel to the server, and a **ServerSessionManager** object as a bundle of these channels. This tutorial only discusses the **ServerSessionManager** objects, and does not discuss **WonderlandSession** objects.

The life-cycle of Wonderland connections is as follows:

- **Initialize**: When a client first establishes a connection with a Wonderland server. During this phase, all of the needed client modules are downloaded from the server, if not already cached.
- **Activate**: When an existing Wonderland connection becomes primary.
- **Deactivate**: When an existing primary Wonderland connection is no longer the primary connection.
- **Cleanup**: When a client tears down an established connection with a Wonderland server.

The client-side plugin API mirrors the life-cycle of Wonderland connections. To define a plugin to run in each Wonderland client, extend the **BaseClientPlugin** abstract class (package **org.jdesktop.wonderland.client**) and annotate your Java class with the **@Plugin** annotation. The **BaseClientPlugin** class implements the **ClientPlugin** interface, but it is recommended that you extend **BaseClientPlugin** rather than implementing **ClientPlugin** yourself. All you need to do is simply compile your plugin into your module for it to be recognized by the system.

The **BaseClientPlugin** class contains four methods that you may override: **initialize()**, **activate()**, **deactivate()**, and **cleanup()**.

The following example client-side plugin adds an item to the "Tools" menu on the Wonderland client main menu bar (package name and import statements omitted). (**Note:** The following code has been copied and slightly modified from working examples, but has not specifically been tested itself)

In the **initialize()** method, a **JMenuItem** is created. The menu item is not added until the Wonderland connection becomes primary. It is important to invoke **super.initialize()** at the end of your **initialize()** method, otherwise **activate()** and **deactivate()** will never be called. A **ServerSessionManager** object is passed into the **initialize()** method that represents the connection to the server.

```
@Plugin
public class SampleClientPlugin extends BaseClientPlugin {
    private JMenuItem myMenuItem = null;

    @Override
    public void initialize(ServerSessionManager loginInfo) {
        myMenuItem = new JMenuItem("My Menu Item");
```

```
        myMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Do something here!
            }
        });
        super.initialize(loginInfo);
    }

    <... more methods here, see below ...>
}
```

When the Wonderland connection becomes primary, the menu item is added to the main menu, in the **activate()** method. You should wait until the **activate()** method to add menu items, rather than add menu items in **initialize()**.

```
    @Override
    protected void activate() {
        JmeClientMain.getFrame().addToToolMenu(myMenuItem);
    }
```

When the Wonderland connection is no longer primary, you should remove the menu items in **deactivate()**:

```
    @Override
    protected void deactivate() {
        JmeClientMain.getFrame().removeFromToolMenu(myMenuItem);
    }
```

Finally, when the Wonderland connection is torn-down with the Wonderland server, you can clean-up any object references if you wish. Strictly speaking, this is not necessary since Java will garbage collect **myMenuItem**, but is shown here as an example of the **cleanup()** method:

```
    @Override
    public void cleanup() {
        myMenuItem = null;
        super.cleanup();
    }
```

Writing a Server Plugin

A **ServerPlugin** executes code when the Darkstar server is first initialized. The most common example of a server plugin is one which registers a client-message handler with the Wonderland communication manager.

To define a plugin to run in each Wonderland server, implement the **ServerPlugin** (package **org.jdesktop.wonderland.server**) and annotate your Java class with the **@Plugin** annotation. The **ServerPlugin** interface contains a single method, **initialize()** that gets invoked when the server is first run. It takes no arguments.

For example: (**Note:** The following code has been copied and slightly modified from working examples, but has not specifically been tested itself)

```
@Plugin
public class SampleServerPlugin implements ServerPlugin {
    public void initialize() {
       // Register a communication handler, assumes AudioManagerConnectionHandler
       // is defined elsewhere.
       CommsManager cm = WonderlandContext.getCommsManager();
       cm.registerClientHandler(new AudioManagerConnectionHandler());
    }
}
```


Your use of this web site or any of its content or software indicates your agreement to be bound by these Terms of Participation.

Copyright © 1995-2006 Sun Microsystems, Inc.

O'REILLY COLLABNET
Powered by Sun Microsystems, Inc., O'Reilly and CollabNet

TWiki (TM) Copyright ©