Project Wonderland v0.5: Adding Entries to the Context Menu

DRAFT: Under Construction

Introduction

This tutorial describes how to use the Context Menu in Project Wonderland. The Context Menu appears in your Project Wonderland client when you right-click over a Cell. The figure below shows what the Wonderland Context Menu looks like. The top-most item (in bold) gives the name of the Cell over which the right-click occurred.

The Context Menu in Wonderland is similar in idea to context menus in many other graphical applications. The Context Menu displays a list of menu items that can be selected. Some of these menu items always appear regardless of which Cell you click over; some menu items appear only for specific kinds of Cells. This tutorial describes how to add entries to the Wonderland Context Menu, both for all Cells and for specific Cells. You will extend the Shape Cell that you created in a previous set of tutorials.

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

In this tutorial you will add to the “shape” module. 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, Part 2, Part 3, and Part 4 of this tutorial series. You will be extending the functionality you implemented there.

Adding an Entry to the Context Menu for all Cells

To add an item to the context menu, you will define a new class `InfoContextMenuFactory` and place it in the `org.jdesktop.wonderland.modules.shape.client` package of the Shape Cell tutorial module. The context menu item you will add here will be visible for all Cells. Strictly speaking, it has nothing to do with the Shape Cell itself, but we are using the project infrastructure of this module for this new class.

1. Start the Netbeans IDE
2. Select File -> Open Project... from the Netbeans main menu. Navigate to your Shape Cell tutorial project, after you have completed Part 4 of that tutorial series
3. Right-click on the `org.desktop.wonderland.modules.shape.client` icon in the Projects view and select New -> Java

...
The Netbeans IDE should generate an empty class named InfoContextMenuFactory. All classes that insert items into the Context Menu that are not associated with any particular Cell type must do two things:

1. Have the @ContextMenuFactory Java annotation
2. Implement the ContextMenuFactorySPI interface

Perform the following steps in the source code for the InfoContextMenuFactory class:

1. Add the @ContextMenuFactory Java annotation right before the class definition (i.e. right before "public class InfoContextMenuItem ...")
2. Add implements ContextMenuFactorySPI to the end of your class definition line
3. Select Source -> Fix Imports... from the Netbeans IDE main menu
4. Next, click inside of the InfoContextMenuFactory class definition, then right-click and select Insert Code... from the context menu that appears. Select Implement Method from the menu that appears and check getContextMenuItems() in the dialog box and click Generate.

Your class definition should appear as follows:

```java
package org.jdesktop.wonderland.modules.shape.client;

import org.jdesktop.wonderland.client.contextmenu.ContextMenuItem;
import org.jdesktop.wonderland.client.contextmenu.annotation.ContextMenuFactory;
import org.jdesktop.wonderland.client.contextmenu.spi.ContextMenuFactorySPI;
import org.jdesktop.wonderland.client.scenemanager.event.ContextEvent;

@ContextMenuFactory
public class InfoContextMenuFactory implements ContextMenuFactorySPI {
    public ContextMenuItem[] getContextMenuItems(ContextEvent event) {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

When the client starts-up it queries all modules installed in the system for classes that are annotated with @ContextMenuFactory and implement the ContextMenuFactorySPI interface. Each factory class is then queried for Context Menu items when their getContextMenuItems() methods are invoked. This method returns an array of ContextMenuItem classes. The ContextMenuItem interface is found in the org.jdesktop.wonderland.client.contextmenu package.

Next, you will implement your getContextMenuItems() method by returning a single Context Menu item named "Info" that when selected will display some basic information about the Cell in a modal dialog box. Implement your getContextMenuItems() method as follows:

```java
public ContextMenuItem[] getContextMenuItems(ContextEvent event) {
    return new ContextMenuItem[] {
        new SimpleContextMenuItem("Info", new ContextMenuActionListener() {
            public void actionPerformed(ContextMenuItemEvent event) {
                Cell cell = event.getCell();
                final String msg = "Cell selected has ID " + cell.getCellID();
                SwingUtilities.invokeLater(new Runnable() {
                    public void run() {
                        JOptionPane.showMessageDialog(null, msg);
                    }
                });
            }
        });
    };
}
```

This method returns an array containing a single item: an instance of the SimpleContextMenuItem class (package org.jdesktop.wonderland.client.contextmenu). This class represents a "simple" Context Menu item that displays a label and an optional icon image. The SimpleContextMenuItem class also takes an instance of an object that implements the ContextMenuActionListener interface.

The ContextMenuActionListener interface has one method: actionPerformed(). The actionPerformed() method is invoked when the "Info" item is selected. Here, you simply display a modal dialog box with the Cell ID of the Cell over which the right-click happened.

To compile and run your module:

1. Run your Wonderland server via the "ant run-server" command where you have the Wonderland source code. Wait for the server to finish initializing
2. Do an "ant deploy" in your Shape Cell tutorial example
3. Start the Wonderland client.
4. Right-click over any Cell and select "Info".
Adding an Cell-Specific Context Menu Item

In addition to adding Context Menu items for all Cell types, you can also add Context Menu items that appear only for specific kinds of Cells. This is accomplished on the client-side Cell class itself. In this tutorial, you will modify the ShapeCell.java class to add a "Change Shape" Context Menu item that will change its shape.

To add a Cell-specific Context Menu item, you will add a ContextMenuComponent to your Cell class. In the four-part tutorial series, you were briefly introduced the notion of Cell Components. The ContextMenuComponent is another example of a Cell Component, much like ChannelComponent. Cell Components contains additional functionality that can be attached to any Cell.

Add the following definitions to the top of your ShapeCell.java file:

```java
@UsesCellComponent private ContextMenuComponent contextComp = null;
private ContextMenuFactorySPI menuFactory = null;
```

Also, add the following import statements:

```java
import org.jdesktop.wonderland.client.cell.annotation.UsesCellComponent;
import org.jdesktop.wonderland.client.contextmenu.cell.ContextMenuComponent;
import org.jdesktop.wonderland.client.contextmenu.spi.ContextMenuFactorySPI;
```

The @UsesCellComponent annotation is what adds the ContextMenuComponent to your ShapeCell class using a technique called "dependency injection". When an instance of your ShapeCell class is created on the client, the system looks for all @UsesCellComponent annotations and creates and adds those Cell Components to your Cell class. It also fills in your contextComp variable with the Cell Component object it creates. This dependency injection is guaranteed to happen before your setStatus() method is invoked, but no sooner. This means, for example, that you cannot rely on a valid value in contextComp until then; you cannot use this method in the ShapeCell constructor or in the setClientState() method, for example.

Next, you need to add the desired Context Menu items in your setStatus() method. Within the RENDERING clause of the switch statement, add the following code:

```java
if (menuFactory == null) {
    final MenuItemListener l = new MenuItemListener();
    menuFactory = new ContextMenuFactorySPI() {
        public ContextMenuItem[] getContextMenuItems(ContextEvent event) {
            return new ContextMenuItem[] {
                new SimpleContextMenuItem("Change Shape", l)
            };
        }
    };
    contextComp.addContextMenuFactory(menuFactory);
}
```

This creates a new instance of a factory that creates a single SimpleContextMenu and adds it to the ContextMenuComponent when the Cell becomes active. Next, add the following code within the INACTIVE clause of the switch
if (menuFactory != null) {
    contextComp.removeContextMenuFactory(menuFactory);
    menuFactory = null;
}

This removes the menu item when the Cell becomes inactive and is no longer loaded in memory. Strictly speaking, this bit of code has very little affect: if the Cell is not visible, then there is no way to activate its Context Menu. But it is also a good idea to "clean up" references on objects so that garbage collection has an easier time reclaiming unused objects.

The final step is to implement an inner class that implements the `ContextMenuActionListener` interface as follows:

```java
class MenuItemListener implements ContextMenuActionListener {
    public void actionPerformed(ContextMenuItemEvent event) {
        shapeType = (shapeType.equals("BOX") == true) ? "SPHERE" : "BOX";
        renderer.updateShape();

        ShapeCellChangeMessage msg = new ShapeCellChangeMessage(getCellID(), shapeType);
        sendCellMessage(msg);
    }
}
```

Much like when a single, left-mouse click happens on the Cell, the `actionPerformed()` method changes the shape of the object and sends a message to the server to inform all other clients of the shape change. Finally, make sure the following import statements appear in your `ShapeCell.java` file (although it is a good idea to have your IDE "fix" your import statements for you):

```java
import org.jdesktop.wonderland.client.contextmenu.ContextMenuItem;
import org.jdesktop.wonderland.client.contextmenu.ContextMenuItemEvent;
import org.jdesktop.wonderland.client.cell.annotation.UsesCellComponent;
import org.jdesktop.wonderland.client.contextmenu.ContextMenuActionListener;
import org.jdesktop.wonderland.client.contextmenu.cell.ContextMenuComponent;
import org.jdesktop.wonderland.client.contextmenu.spi.ContextMenuFactorySPI;
import org.jdesktop.wonderland.client.scenemanager.event.ContextEvent;
```

After you compile and deploy your Shape Cell tutorial module to the Project Wonderland server and re-run your client:

1. Select Tools -> Cell Palette... from the Wonderland main menu
2. Select "Shape Tutorial" from the list and click Create
3. Right-click over the shape that appears in front of you and select "Change Shape". The Shape Cell should change shape

The Context Menu for the Shape Cell should appear as follows. The "Change Shape" option appears at the bottom of the list.

![Figure 3. The Context Menu for the Shape Cell](image)

**Other Features**

There are two other features of the Context Menu not mentioned in this tutorial. First, a ContextMenuComponent can determine whether the "standard" Context Menu items (that is, those items not specific to a particular Cell) appear when the Context Menu is displayed for that Cell. The `setShowStandardMenuItems()` method takes a boolean value: if true (default) the standard Context
Menu item are displayed for that Cell; if false, the standard Context Menu items are not displayed for that Cell.

Second, modules may update the appearance of a Context Menu item after the factory’s `getContextMenuItem()` method is invoked. For example, suppose a module wishes to update some aspect of a Context Menu item as a result of a network call (perhaps to the Wonderland server). Since `getContextMenuItems()` should return quickly, the module can spawn a thread with the `SimpleContextMenuItem` it created and set its properties any time in the future. For example, it can change its label, its icon, or its enabled/disabled state. The thread then invokes the `fireMenuItemRepaintListeners()` method to tell the Context Menu to redraw the item.

**Conclusion**

In this tutorial you learned how to add entries to the Project Wonderland Context Menu applicable to all Cell types and specific to only a single Cell type.