OWL Scripting

Class 1

Preliminary information about webdav repositories:

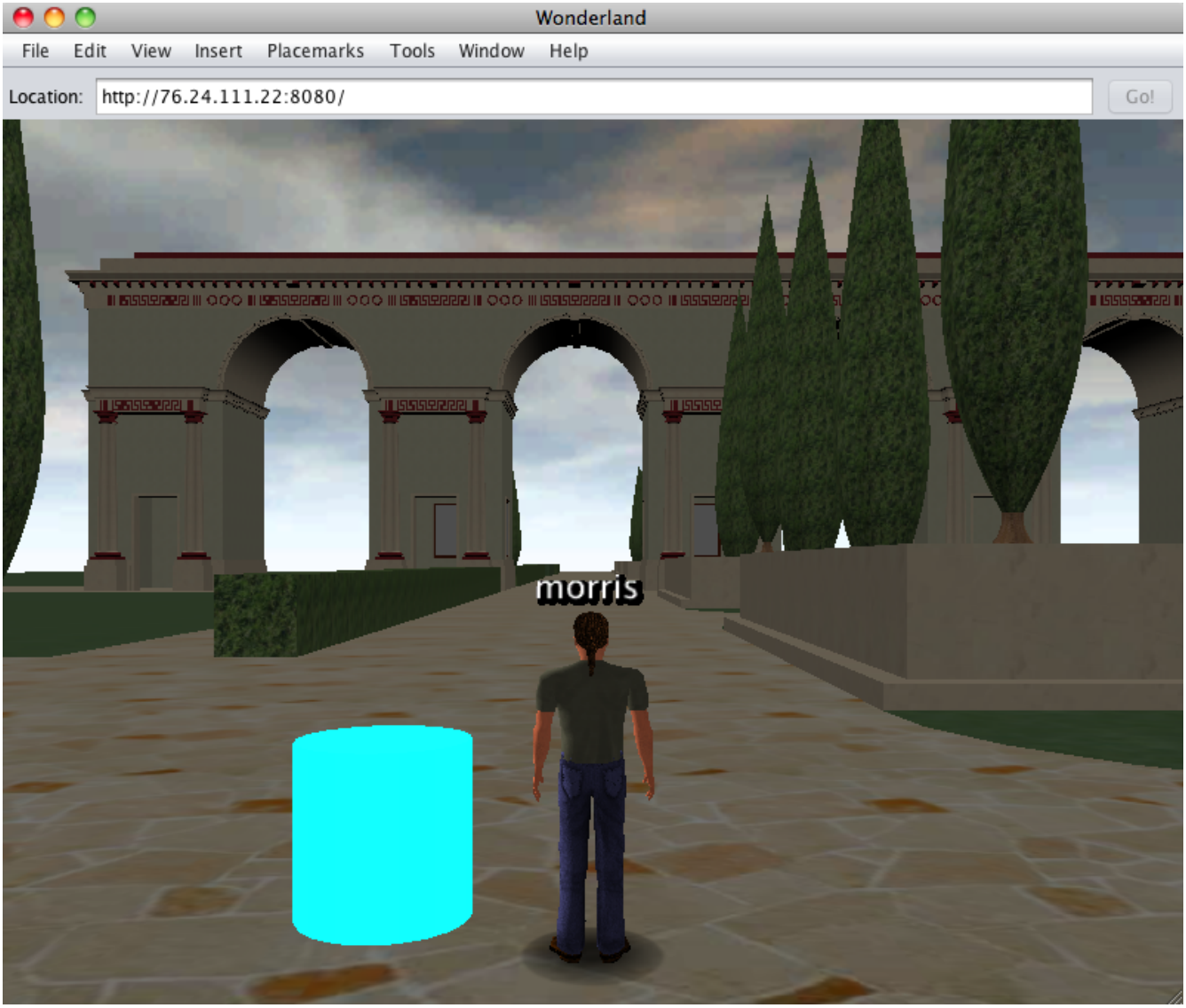
The scripting system uses the webdav content management system on the server to store the scripts. This system has two main areas for storing content, ie, user space and global space. The permission system for the content area allows any user access to her own storage area. The script storage area used in these scripting classes will be this user area since it is always available to the user. The other storage area is the global space and is only available to users who are members of the admin group and as such will only be available to a few users. The scripting editor can access both areas and as such is built to be useful for teaching purposes as well as for actual system development.

The storage area for the scripts uses the name of the model as part of the path to the specific script. In addition, the name of the specific script is the file name. Also, in the case of the user storage area, the user name (login name) is used as another part of the path. For example, the url for a sample global script would be: "http://99.99.99.99:8080//webdav/content/scripts/AniDoor4/startup.js" for a startup script. This would be the url for a sample user script: "http://99.99.99.99:8080//webdav/content/users/morris/scripts/testBlue/mouse1.js" for a left mouse button script. If desired these scripts can also be created and edited manually directly in the file space of OWL server.

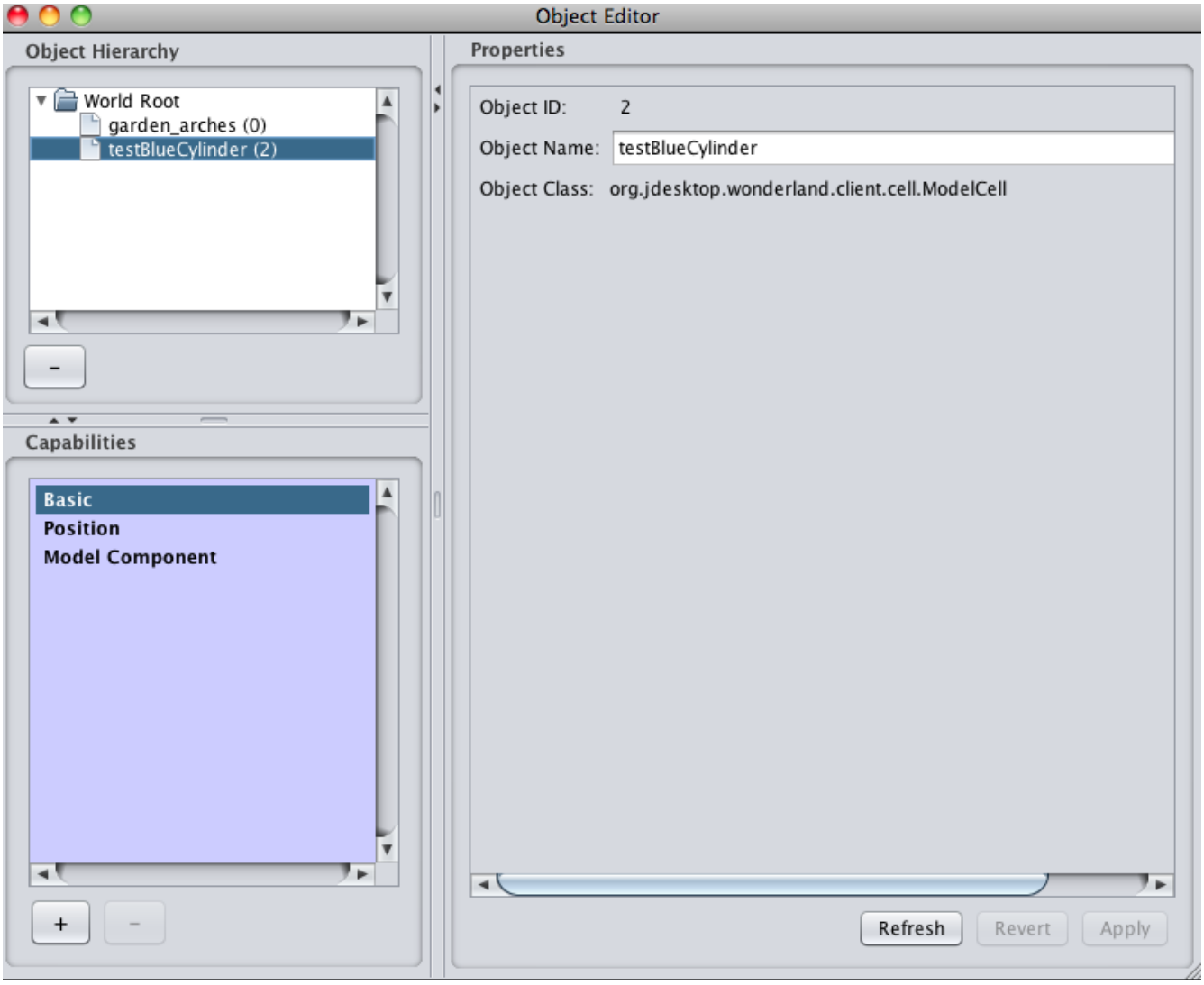
Start up Owl client.

Ensure that your java console is displayed.

Insert a model of some sort.



Right click on model to bring up properties panel.

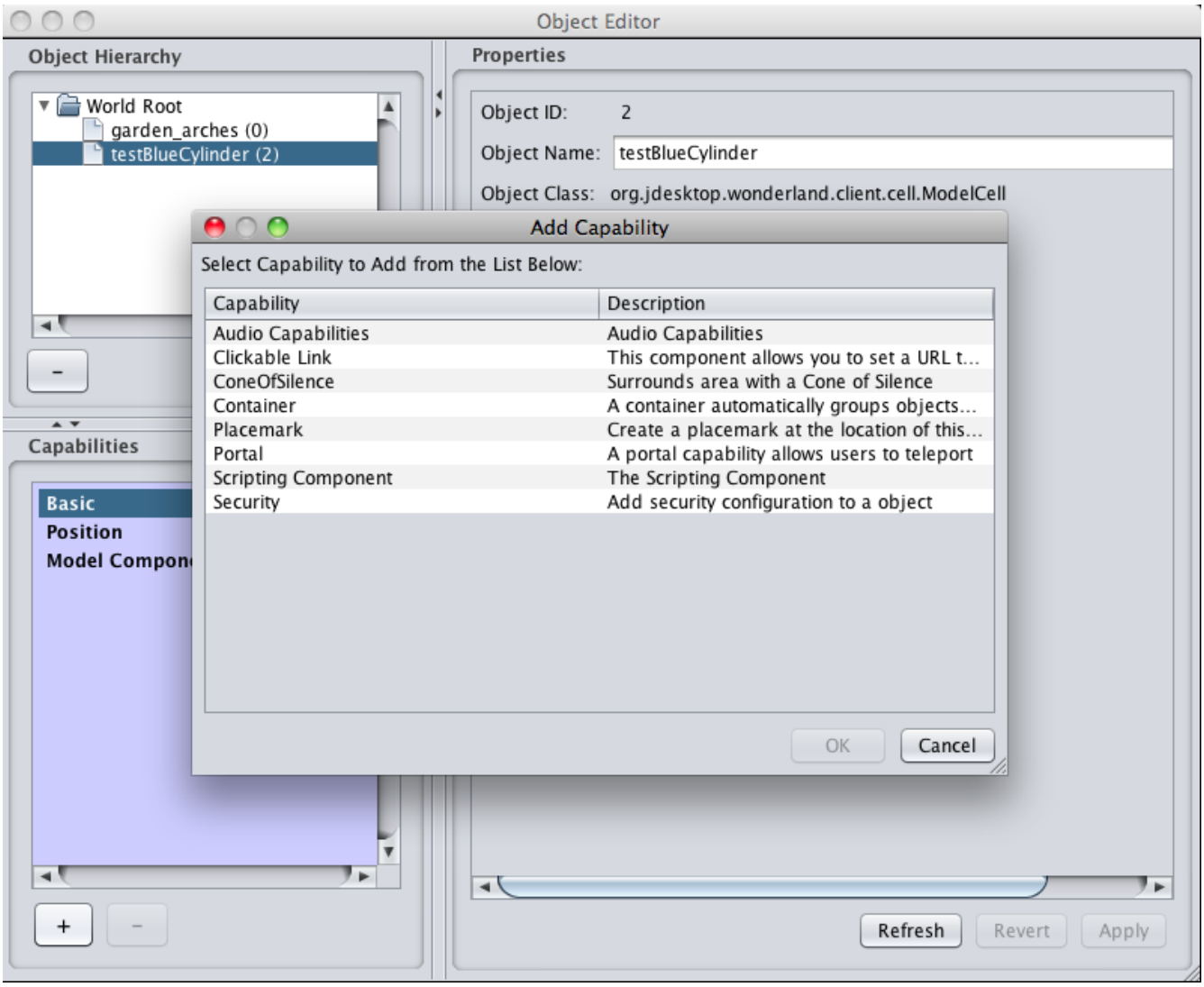


Change Object Name to a unique name with no spaces.

Click apply to apply name change.

Click on + at bottom left corner to bring up capabilities panel.

Click on Scripting Component to add scripting.



Click somewhere inside OWL client window.

Hover cursor over your object.

Press 'p' key.

Scripting panel will open.

Click 'Retrieve Properties'.

Object will fill in.

Type mouse1.js into the 'Script Name' field.

Click the 'User Scripts' box to select it.

Type this script into the script edit window.

// Test script for mouse1.js event
print(“This is a test mouse1.js script\r\n”);

After the script has been entered, click on 'Save Script'.

Click on 'Test Script'.

Enter ice.js in the Script Name field.

Enter the following into the script edit area.

// Test ice.js script.
print(“ICE message received - “ + ICEEventMessage + “\r\n”);

Click 'Save Script.

Enter mouse1s.js in the Script Name.

Enter this script.

```
//Test mouse1s.js script.
MyClass.watchMessage(300);
```

Click Save Script.

Enter mouse1.js in the Script Name field.

Click Retrieve Script.

Add the last line to the script as shown.

//Test script for mouse1.js
print(“Test for mouse1.js script\r\n”);
MyClass.postMessageEvent(“Test ICE message”, 300);

Click Save Script.

Enter mouse1s.js in the Script Name

Click Retrieve Script.

Click Test Script.

Enter mouse1.js in the Script Name.

Click Retrieve Script.

Click Test Script.

What happened?

This clip from the java console shows typical trace after the mouse1.js test.

```
Enter getScriptIndex with script = mouse1.js
Inside - i = 0 - evName = mouse1.js
Exception in load propertiesjava.lang.NullPointerException
Scripts property = webdav
ScriptingComponent : Cell 2 : Start of executeScript - useGlobalScripts = false - userName = morris - cellOwner = morris
ScriptingComponent : Cell 2 : scriptPath = http://76.24.111.22:8080//webdav/content/users/morris/scripts/testBlueCylinder/mouse1.js
ScriptingComponent : Cell 2 : Script type = javascript
ScriptingComponent : Cell 2 : This script takes Compiled path
Script takes the webdav path
This is a test mouse1.js script
ScriptingComponent : Cell 2 : In postMessageEvent with payload = Test ICE message code = 300.0
ScriptingComponent : Cell 2 : In Intercell listener in commitEvent - payload = Test ICE message Code = 300
Exception in load propertiesjava.lang.NullPointerException
Scripts property = webdav
ScriptingComponent : Cell 2 : Start of executeScript - useGlobalScripts = false - userName = morris - cellOwner = morris
ScriptingComponent : Cell 2 : scriptPath = http://76.24.111.22:8080//webdav/content/users/morris/scripts/testBlueCylinder/ice.js
ScriptingComponent : Cell 2 : Script type = javascript
ScriptingComponent : Cell 2 : This script takes Compiled path
Script takes the webdav path
ICE message received - Test ICE message
```

Click with left mouse button directly on model.

Click on model of someone else.

What happened?

End of segment #1

Points covered in this segment

Two webdav repository areas for script storage.
Mouse events and mouse scripts.
ICE event and script.
Javascript syntax for interfacing commands.
MyClass.watchMessage(300);
MyClass.postMessageEvent(“message”, 300);

Enter mouse1c.js in the Script Name.

Enter the script as shown.

Click Save Script.

Click Test Script.

What happens?

```
// mouse1c.js
print("Hello from rotate test\r\n");

var v = new java.lang.Runnable()
   {
   run: function()
      {
      increment = (Math.PI/2)/90;
      for(i = 0; i <= 90; i++)
         {
         MyClass.rotateObject(0, 1, 0, -increment, 1);
         MyClass.mySleep(50);
         }
      }
   }

t = new java.lang.Thread(v);
t.start();
```

Change the '1' to a '0' in line

MyClass.rotateObject(0, 1, 0, -increment, 0);

Click Save Script.

Click Test Script.

What happens?

Do the objects of other clients rotate?

Using the properties panel, check for the x, y, z location of your object.

Add the 3 lines as shown. Use the x and z values from before for the setTranslation step.

```
// mouse1c.js
print("Hello from rotate test\r\n");

var v = new java.lang.Runnable()
   {
   run: function()
      {
      increment = (Math.PI/2)/90;
      for(i = 0; i <= 90; i++)
         {
         MyClass.rotateObject(0, 1, 0, -increment, 1);
         MyClass.moveObject(0, increment, 0, 0);
         MyClass.mySleep(50);
         }
      MyClass.mySleep(1000);
      MyClass.setTranslation(2, 1, 2, 0);
      }
   }

t = new java.lang.Thread(v);
t.start();
```

What would you expect this to change?

Click Save Script.

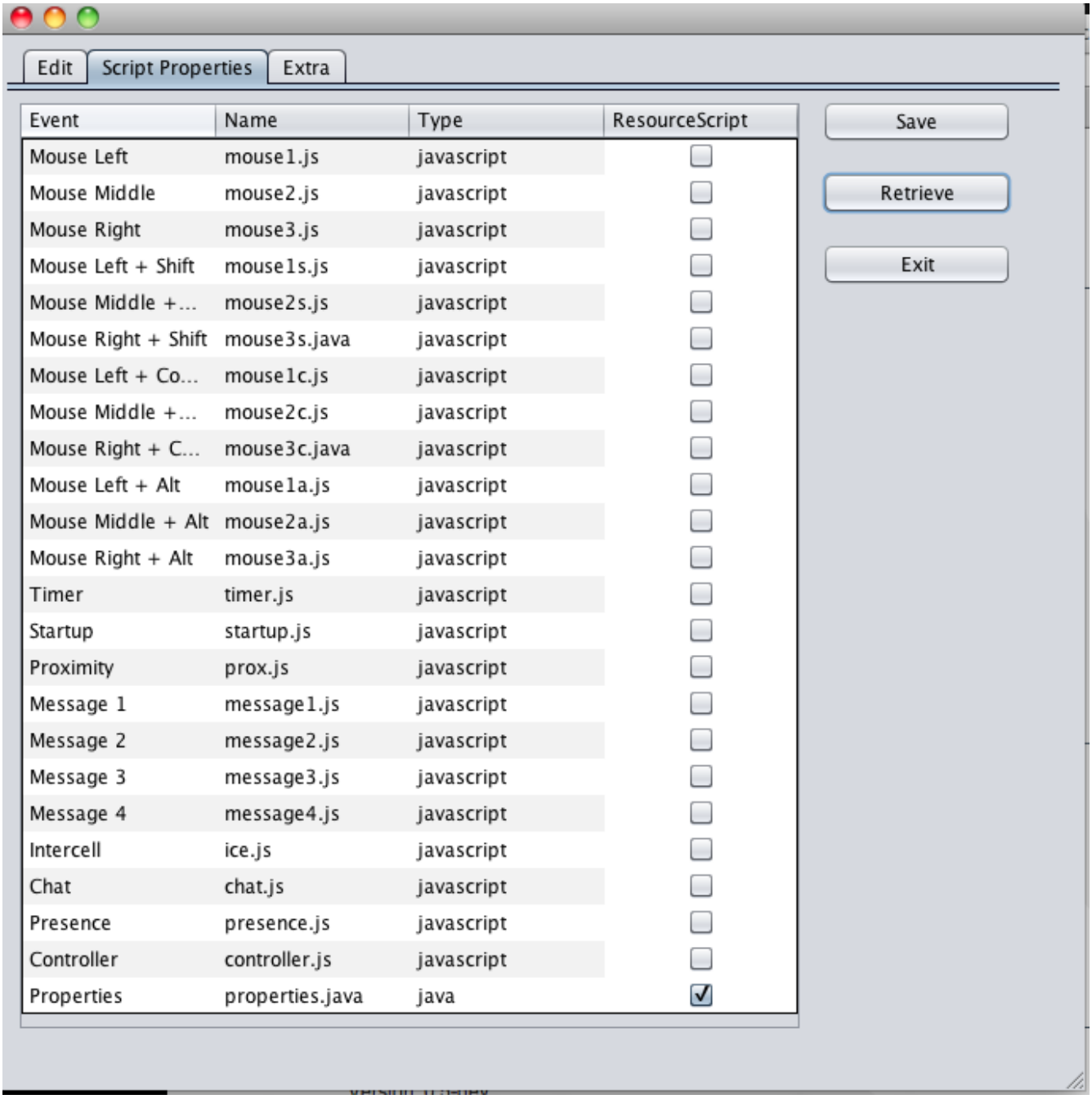
Click Test Script.

Click on the Script Properties tab.

This is the list of event → script relationships.

Note that the last one (properties) has the ResourceScript box checked.

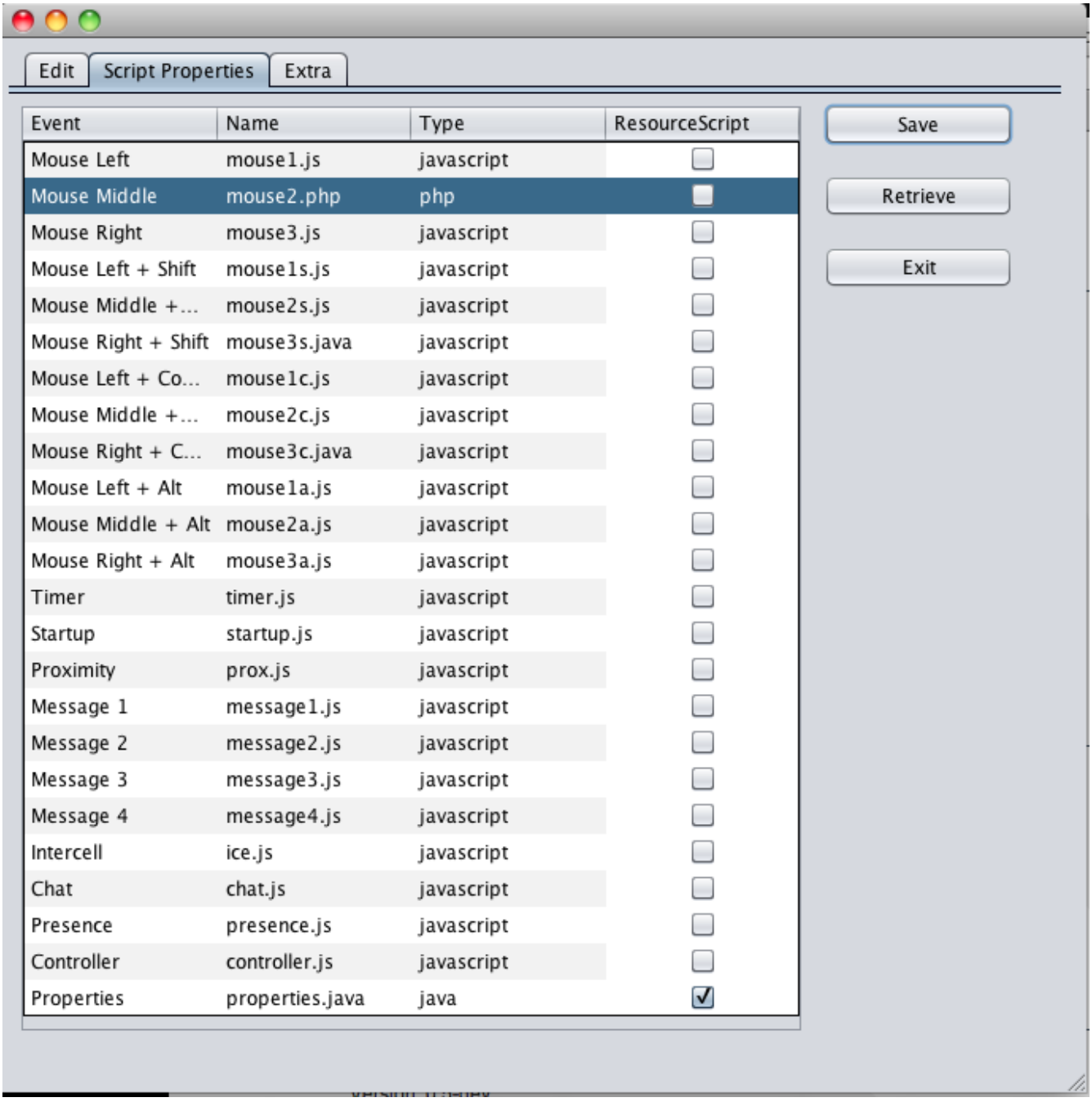
Script names and script types can be changed to allow other languages.



Change Name to mouse2.php for Mouse Middle.

Change Type to php for the same event.

Click the Save button.



Go back to the Edit tab.

Enter mouse2.php for Script Name

Enter the script as shown below.

```
<?php
$MyClass->testMethod(“This is a test”);
?>
```

Click Save Script.

Click Test Script.

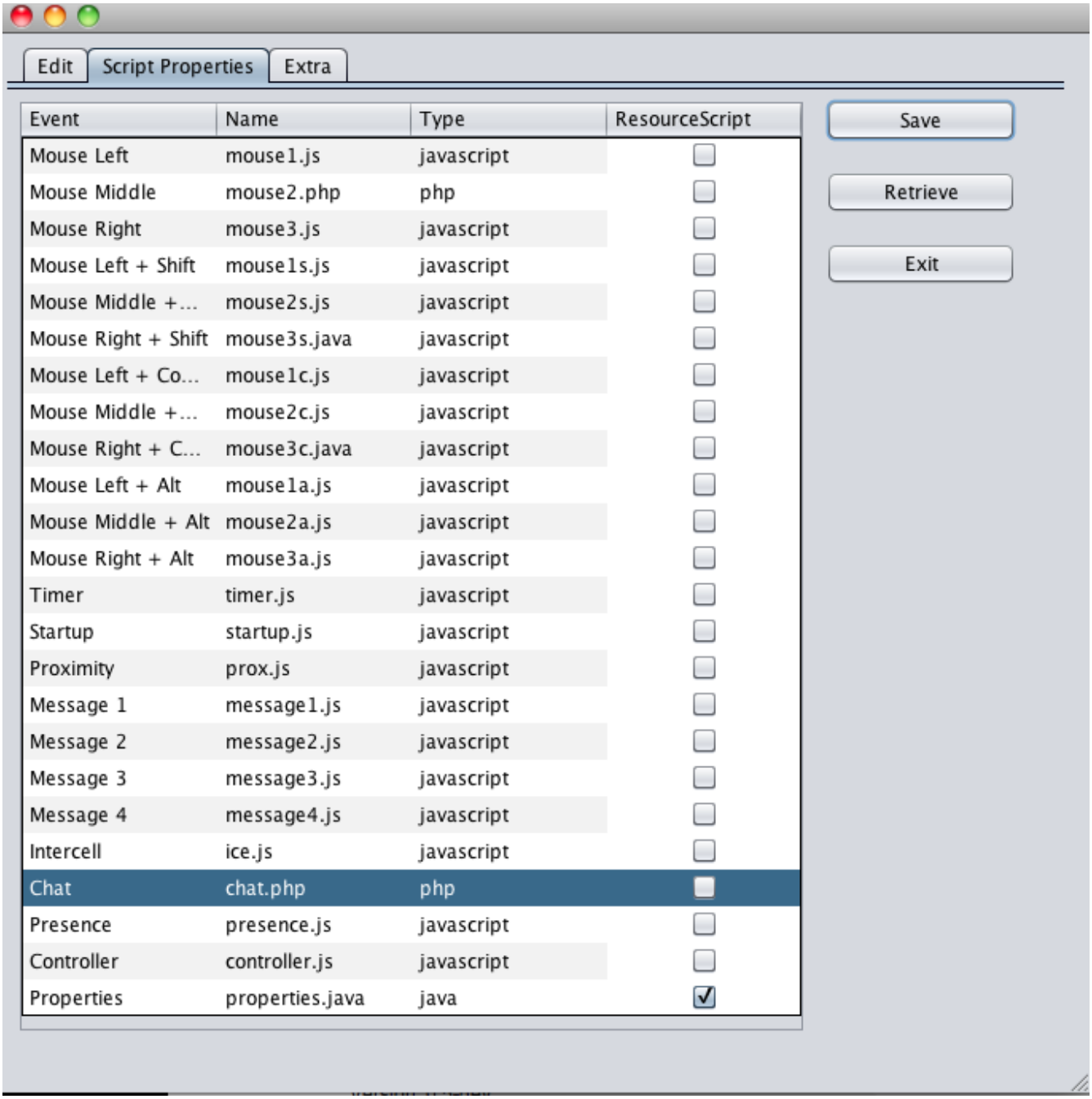
Click directly on the model with the middle mouse button.

Both the same?

Go to the Script Properties tab.

Change the Chat script Name to chat.php and the type to php as shown.

Click the Save button.



Go back to the Edit tab.

Enter chat.php in the Script Name.

Enter the script as shown.

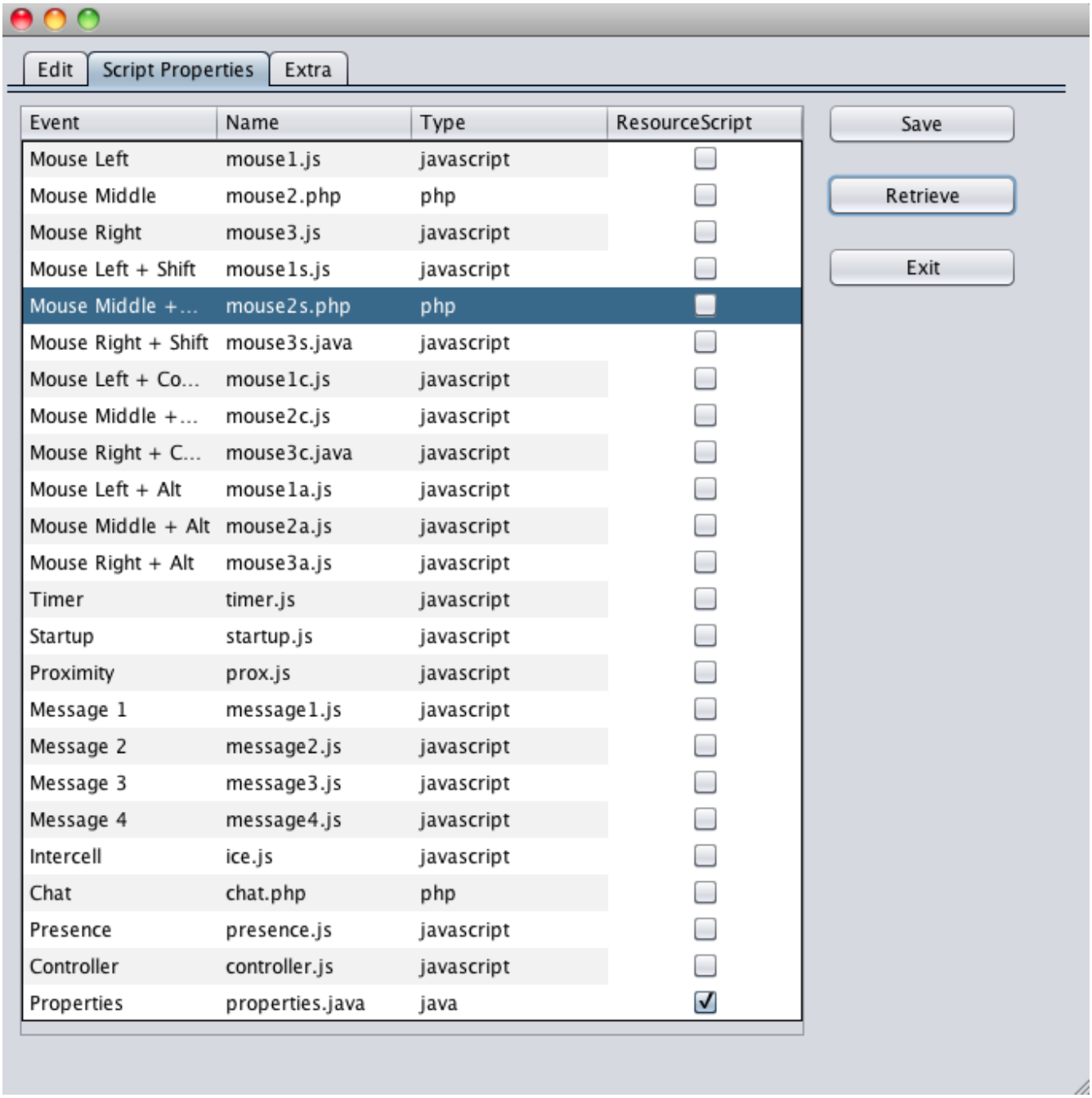
```
<?php
$MyClass->testMethod(“Chat message”);
$MyClass->testMethod(chatMessage);
$MyClass->testMethod(“from”);
$MyClass->testMethod(chatFrom);
$MyClass->testMethod(“to”);
$MyClass->testMethod(chatTo);
?>
```

Click Save Script.

Go to the Script Properties tab.

Change the Mouse middle + shift entries to mouse2s.php and php as shown.

Click the Save button.



Go back to the Edit tab.

Enter mouse2s.php in the Script Name.

Enter this script.

```
<?php
$MyClass->testMethod(“This is a chat test”);
$MyClass->getChat();
?>
```

Click Save Script.

Enter mouse2.php in the Script Name.

Click the Retrieve Script.

Add the $MyClass.sendChat … line to the script.

<?php
$MyClass->testMethod("This is a test");
$MyClass->sendChat("test chat message", "morris", "jagwire");
?>

Click Save Script.

Enter mouse2s.php in the Script Name.

Click Retrieve Script.

Click Test Script.

What does this accomplish?

Enter mouse2s.php in the Script Name.

Click Retrieve Script.

Click Test Script.

Do you see the trace from the chat.php script?

For the next steps, click directly on your model.

Click with the left mouse button.

Click with the left mouse button + shift on your model.

Click with the left mouse button + control on your model.

Click with the middle mouse button on your model.

Click with the middle mouse button + shift on your model.

Did all the mouse clicks cause the action that you would expect?

Click the exit button.

Oops! Did you press the 'x' corner or the red dot corner? If so, restart the client.

Insert a ScriptingPoster object into the world.

Hover the cursor over the poster and press the 'p' key.

Click the Retrieve Properties.

Enter mouse1.js in the Script Name.

Enter the following script.

```
//mouse1.js for poster
var text="<html>";
text=text+"<br>This is a test line</br>";
text=text+"<br>This is line 2</br>";
text=text+"</html>";
MyClass.executeAction("text", text);
```

Click Save Script.

Click Test Script.

Did the poster change.

Could you make the first model you worked with send data over to the poster to be displayed?

To make the poster receive info from another cell:

Tell the poster cell to listen for an ice message with an arbitrary code, 350 for example.

When data is received, have the poster cell put the data into the poster.

Modify one of the scripts for the other cell to send the desired data for the poster with a code 350.

Enter mouse1s.js in the Script Name field.

Enter the following script.

```
//mouse1c.js – ScriptingPoster
MyClass.establishIncomingSocket(400, 401, 2048);
```

Click on Save Script

What might that last command do?

Enter mouse1c.js in the Script Name field.

Enter the script below.

```
// mouse1c.js – ScriptingPoster
MyClass.watchMessage(400);
```

Click Save Script.

Click Test Script.

What does that script do?

Enter mouse1s.js in the Script Name field (again).

Click Retrieve Script.

Click Test Script.

Enter 'telnet localhost 2048' in a terminal window if you are on a system that has telnet.

The telnet should show connected and the java console should show connected.

Enter '00009abcd'.

The text abcd should be displayed in the poster.

The message as typed at the terminal is a 5 byte overall length and the payload.

It is also possible to initiate an outgoing socket connection with:

MyClass.establishSocket(code, error code, ip, port);