M-eux Test

Test Automation using Eclipse

User Guide

Abstract

This Getting Started Guide describes how to use M-eux Test for testing mobile applications using Eclipse. This document is intended for Quality Assurance (QA) engineers and testers who wish to get acquainted with the functionalities of M-eux Test.

Part Number MT-UG-EC
Revision 3093, February 2014

© Copyright 2014 Jamo Solutions N.V. No parts of this document may be reproduced, stored in, or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Jamo Solutions.

This document is provided for informational purposes only. Jamo Solutions makes no warranties as to the information in this document. The information contained herein is subject to change without notice.

All trademarks are the properties of their respective owners

Contents

Chapter 1: Introduction

Welcome to the user's guide for the Eclipse part of 'M-eux Test'. The product 'M-eux Test' uses Eclipse to create and maintain automation scripts for testing against mobile devices.

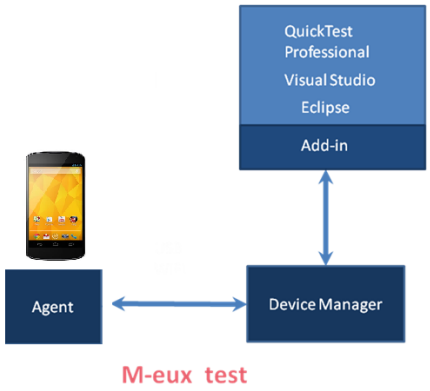
In order to get used to the Eclipse User Interface, please consult Oracle's provided help and documentation resources. This guide will focus on the additional functionality in order to create test cases against mobile devices.

We have made every attempt possible in making the instructions in this guide as clear as possible. However, we recognize that we are unable to cover everything in a single guide. Should you require further assistance, please do not hesitate to visit our www.jamosolutions.com website or to contact our support team at http://www.jamosolutions.com/member-area/support/.

Chapter 2: M-eux Test Overview

M-eux Test currently supports the following products:

- The mobile device agent running on a mobile device or emulator connected with a USB cable or WIFI to the PC.
- The DeviceManager application. This application is running on the PC and connects the mobile device with Eclipse.
- Eclipse for creating and maintaining the scripts.



- **Figure 1: M-eux Test Architecture**

2.1. The mobile device agent

Mobile devices can only connect to the device manager if the agent is installed on the device. The device needs to be connected to the PC, on which Eclipse and the device manager are installed, by a USB cable or a WIFI-connection.

2.2. The device manager application

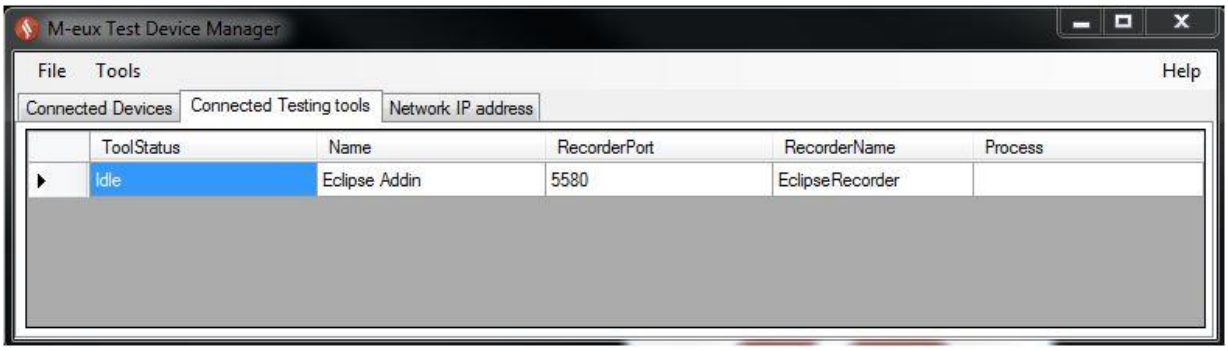
The device manager application manages the connection between the devices and Eclipse. The device manager application lists three tables:

- The connected devices
- The connected testing tools

2.2.1. The connected devices

This table lists all connected devices. The first column displays the name of the device. This name is used to identify the device. You have to make sure that if you connect multiple devices, that each device has a unique name.

2.2.2. The connected testing tools



This table lists all connected testing tools to the device manager. The table will display the connected Eclipse session and its status. The status can be ‘recording, ‘idle’ or ‘replaying’.

Chapter 3: Getting Started

The M-eux Test integration with Eclipse allows you to create and replay your scripts using Eclipse as your script editor. All the functionality of Eclipse / java will be at your disposal for creating your test scripts. The M-eux Test integration with Eclipse gives you a lot of freedom and flexibility in creating your test cases which will result in a rich testing experience.

M-eux Test supports Eclipse Galileo and above (32 bit version).

Chapter 4: Eclipse Plugins

M-eux provides two options to develop scripts from Eclipse –using the M-eux test Environment and Old Plugin using Java perspective.

We recommend to use the new M-eux test Environment as it is more user friendly and more features are provided but if in case you are already using the Old Plugin and comfortable with it then you can continue to use it.

Chapter 5 – Explains the usage of M-eux Test Environment (New M-eux Perspective) (Recommended)

Chapter 6 - Explains the usage of Old Plugin (Java Perspective)

Chapter 5: Overview of the M-eux Test Environment

5.1. Salient features -

Backward compatible-

M-eux Test Environment is designed in a way that the scripts created using old plugin are unaffected. These scripts can also make full advantage of the functionalities provided in the M-eux Test Environment.

Old plugin will be functioning too

If you are comfortable with old editor and plugin, feel free to switch to “Java Perspective” from eclipse and continue to work in the same way.

Projects View

This view will show you all the projects you have in workspace. Clicking on an m-eux project will show a test suite in Test Suite view and UserScript.java file will become visible. Also you can add new m-eux project by clicking on + button. You can connect to M-eux Device Manager by pushing Connect button.

Test Suite View (TS view)

For each m-eux project created this view will show a test suite by default. As for now only 1 test suite is allowed per project but, from next release multiple test suites will be allowed in one project.

Object Repository view (OR view)

Now you can see you Object Pool (or Object Repository) from the Object Repository view. Objects can be more intuitively understood by the images introduced for each type of object. Highlighting the objects from OR is also possible. You can see the objects getting highlighted in RDS. You can do a drag drop of Objects from this view to your script. (Explained later)

Descriptive Properties View (DP view)

This view will show the descriptive properties and their corresponding values for each object currently selected from OR view. Update to the descriptive properties is possible from this view. No need to go to the source of the Object pool.

Object Properties View (OP view)

This view will show the properties for each object currently selected from OR view. It is possible to drag and drop these properties to DP view. Once you drag a property from OP view and drop it to DP view, this drag dropped property will become a descriptive property for the object selected in OR view. You need to assign a value to this property in DP view. No need to go to the source of Object Pool.

Results View

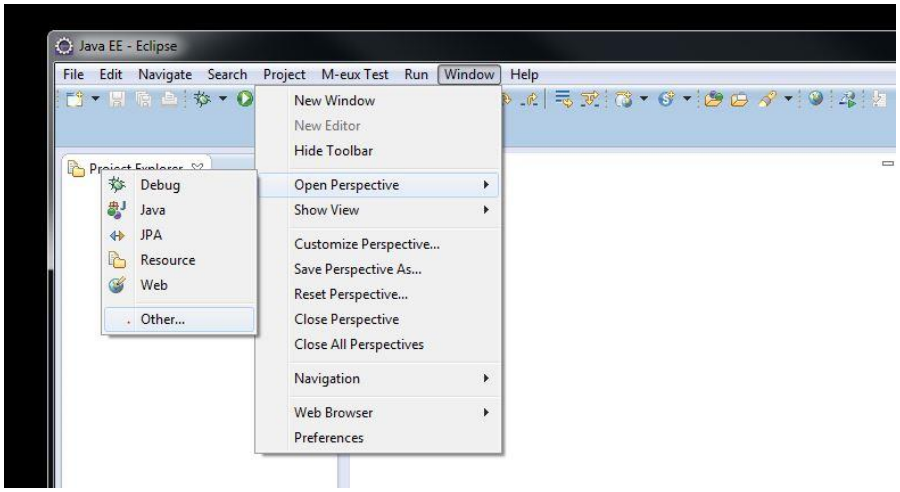
In this view you can see the report/s file generated for your test run.

The following section will give step by step instructions on how to create an M-eux Test Eclipse Project and give an overview of the project structure.

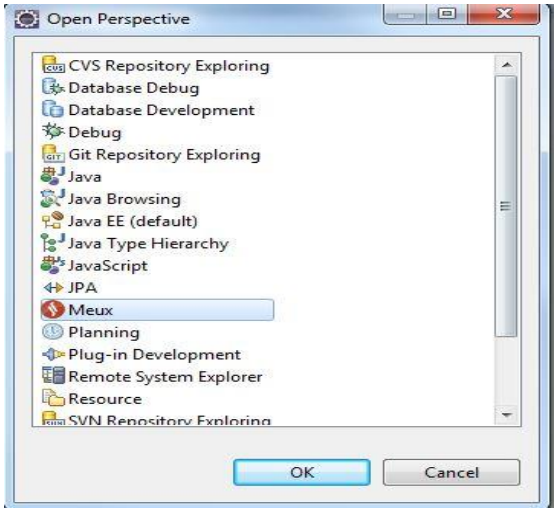
5.2. Launching the new M-eux perspective/plugin

1) Launch Eclipse with the "MeuxEclipseStarter" shortcut on the desktop.

2) Create a new Workspace.
3) Eclipse will automatically open the Meux perspective. If in case it does not opens by default the Meux perspective then you can follow additional steps below-
    a. Go to Windows->show Perspective->Other



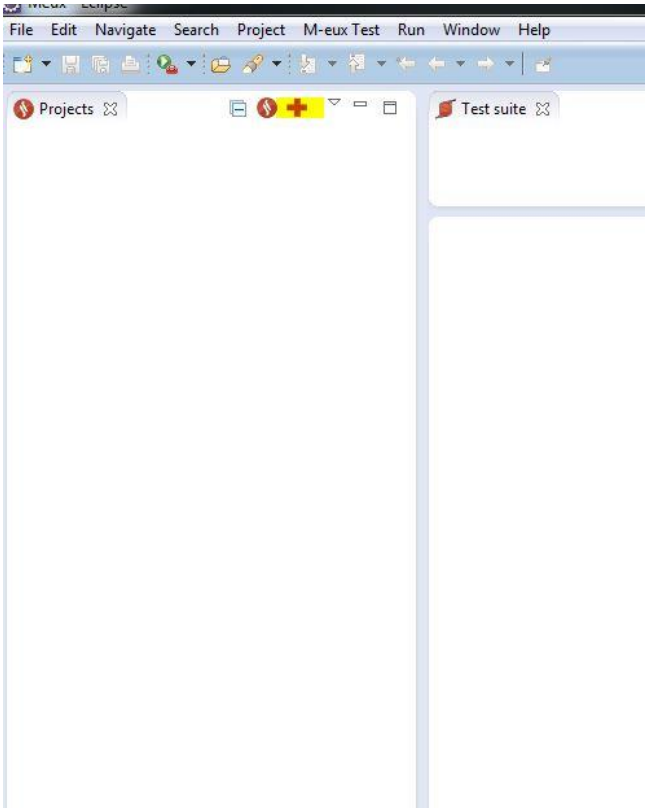
    b. Select Meux



c. Now you will see the M-eux Test perspective in eclipse.

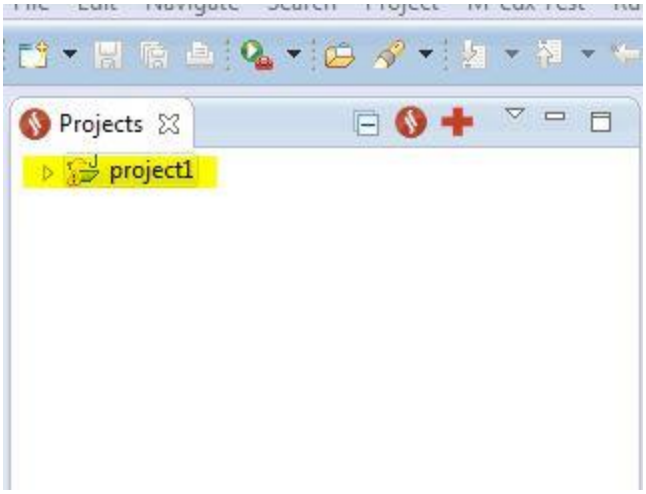
5.3. Creating a M-eux Test Eclipse Project

1) Click on the + sign (or  sign in Eclipse Helios and Galileo) in Project View



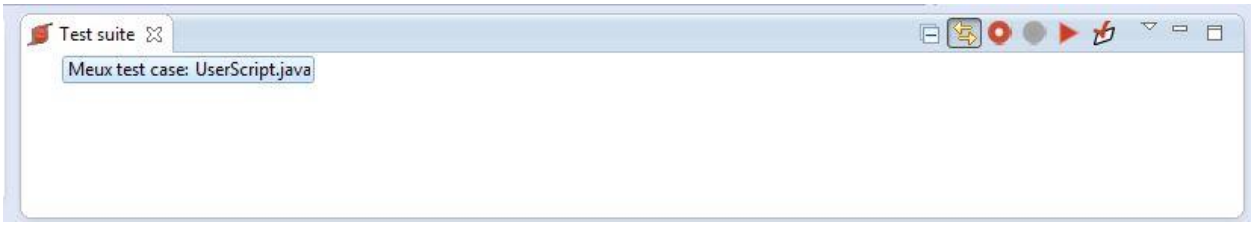
2) Enter a name for your project
3) Click on Finish button

4) A project will be added to the Project view



5.4. Create your first script using Recording feature

1) Once you have created a project, you can click on it. Test Suite view will show you the default test suite created for this project



2) Click on the default Test Suite- UserScript.java file will open in editor.

```
UserScript.java
package com.jamosolutions.project1;

import meuxplugin.meuxsystem.exceptions.MeuxException;

public class UserScript extends Script{

    private UserScript(){
        getMeuxReplaySettings().setDelay(0); // Delay before a method to run is deleg
        getMeuxReplaySettings().setFindObjectTimeOut(20000); // Maximum timeout for f
    }

    private UserScript(MeuxReplaySettings meuxReplaySettings){
        super(meuxReplaySettings);
    }

    private static UserScript instance;

    public static UserScript getInstance(){
        if (instance == null){
            instance = new UserScript();
        }
        return instance;
    }

    public void runCoreDelegate(ResultManager resultManager) throws MeuxException{
        this.setResultManager(resultManager);
        try {
            this.runCore();
        }finally{
            resultManager.AddResult("project1", resultManager.isGlobalSuccess());
            resultManager.setGlobalSuccess(true);
        }
    }

    public void runCore() throws MeuxException {

    }
}
```

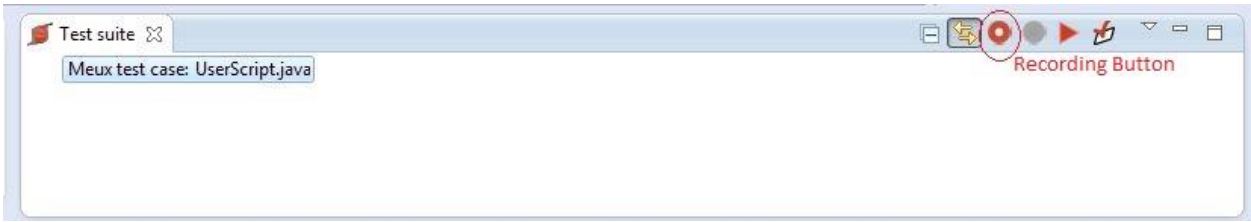
5.4.1. Recording a script

Recording is a tool that will help you in the creation of your test scripts. An alternative to recording is performing a learn GUI and typing out your script manually using Eclipse intellisense (Explained later).

1) Make sure the agent on the device is running and connected to the device manager

2) Make sure that App under test is made testable and launched.
3) Click on a project from Projects view
4) Select a Test suite from Test Suite view (by default a test suite is created for each project)

5) Press Recording button -



6) Perform actions on the device
7) The record statements will be generated in the runCore function of the UserScript.java Class.

```
public void runCore() throws MeuxException {
    androidphone.avw_apiDemos.avw_vG.select("App");
    androidphone.avw_apiDemos.avw_vG.select("Notification");

}
```

8) Press Stop recording to end the recording session

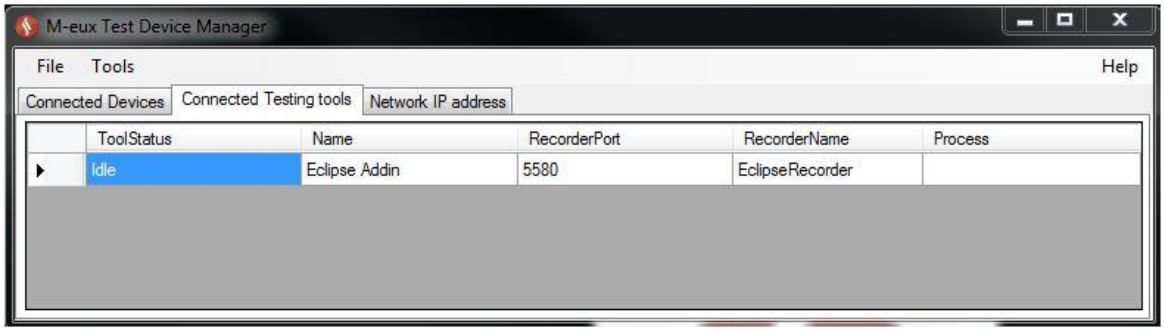
5.5. Create your script using Learn GUI feature

To add all the objects that are currently accessible on the mobile device to the object pool you can perform a learn GUI.

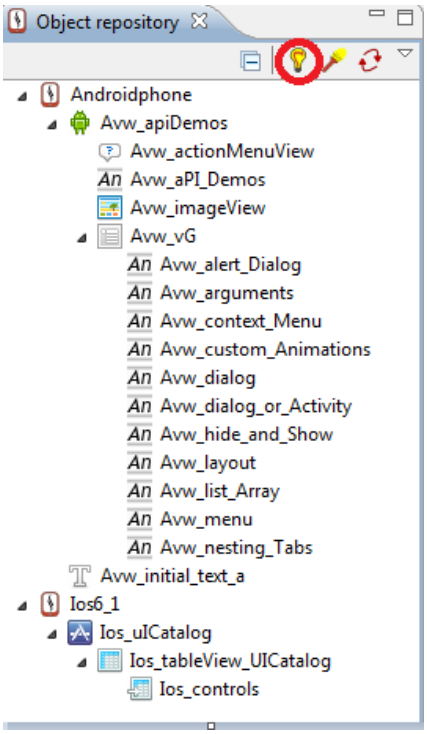
1) Make sure the agent on the device is running and connected to the device manager

2) Make sure that App under test is made testable and launched.

3) Make sure that Eclipse is connected to the device manager.



4) Select Learn GUI from the OR View

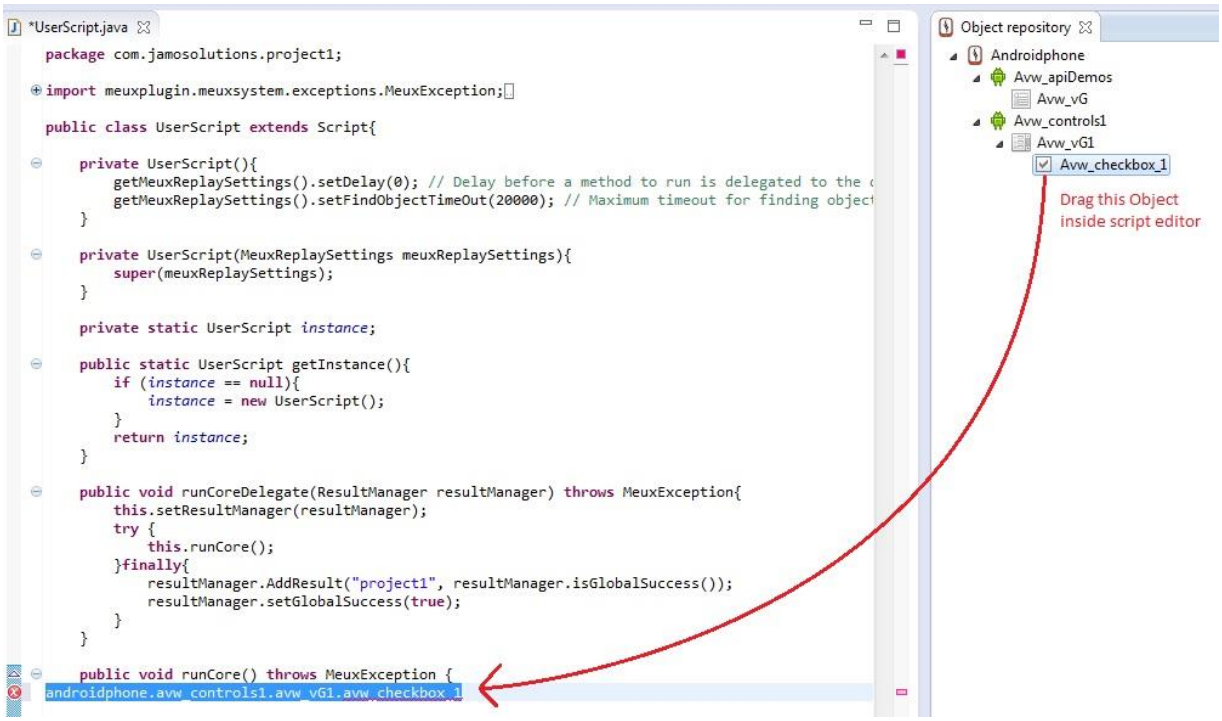


5)

a. If only one device is connected the objects will be added to the objectpool.

b. If multiple devices are connected you will get a dialog box inquiring which devices object you wish to learn.

6) Drag and drop of objects to the script editor for quick script creation.



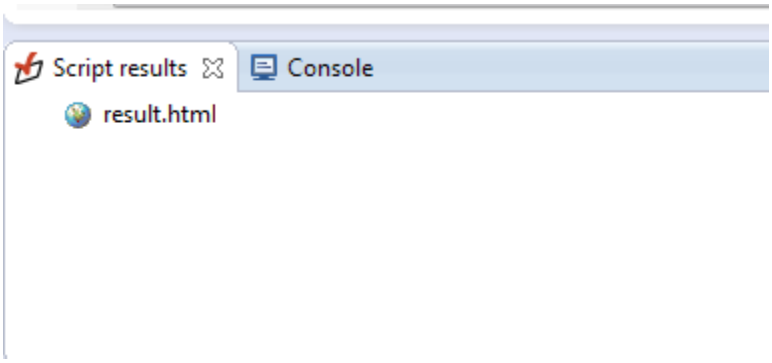
7) Use eclipse intelligence or the Meux function reference to invoke the right method suiting your test case. For ex-tap, doubletap, set, etc.

5.6. Replaying a Script

You can replay your script directly in Eclipse by clicking on the Play button.

Results file generated can viewed in Results View by clicking on Results button.

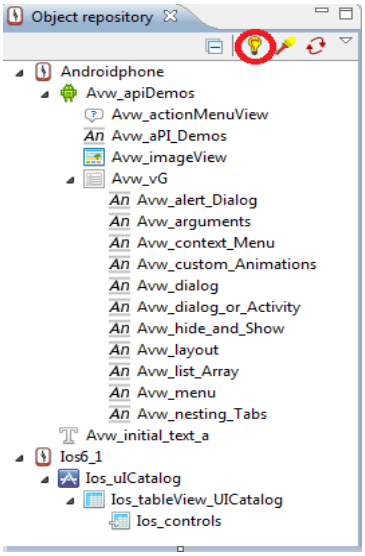
Results View-



5.7. Other Features and Manipulating Object Properties

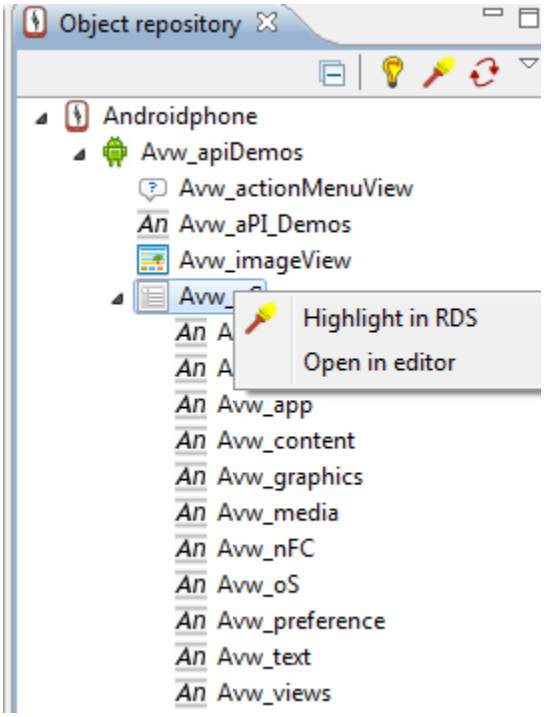
5.7.1. Object Repository View (OR View)

In this view you can see your Objects in Object repository.

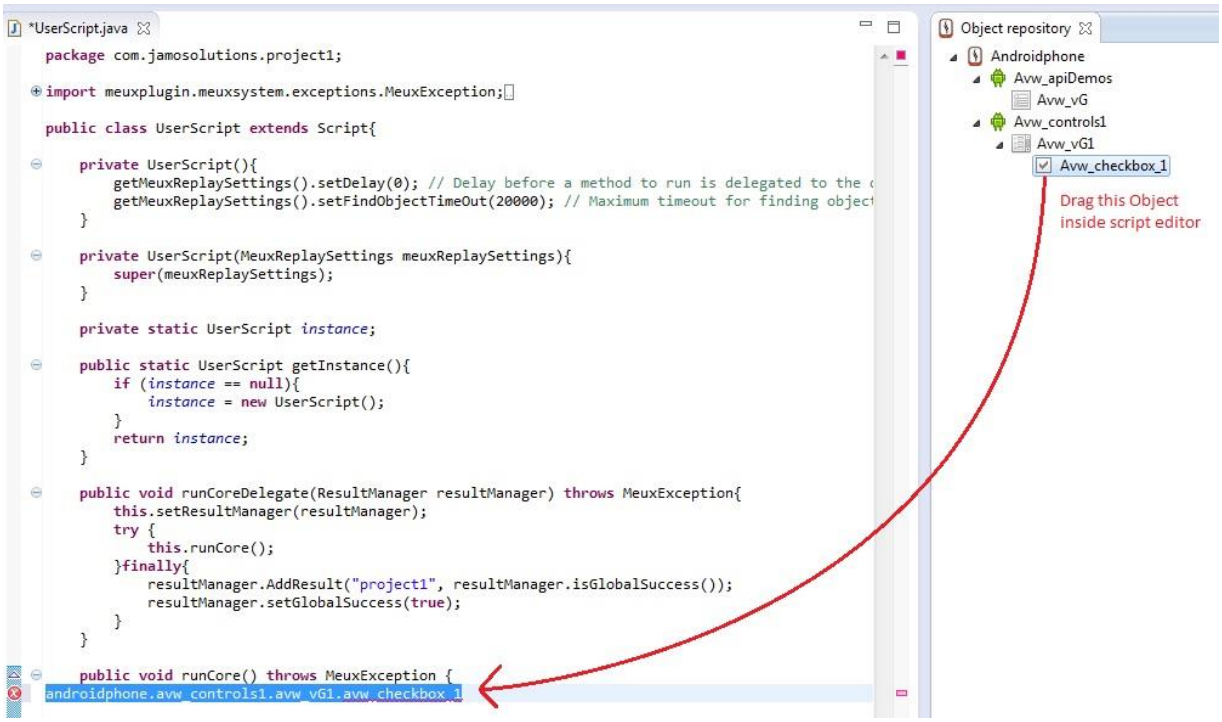


Functionalities-

1) Highlight the added object in the RDS.

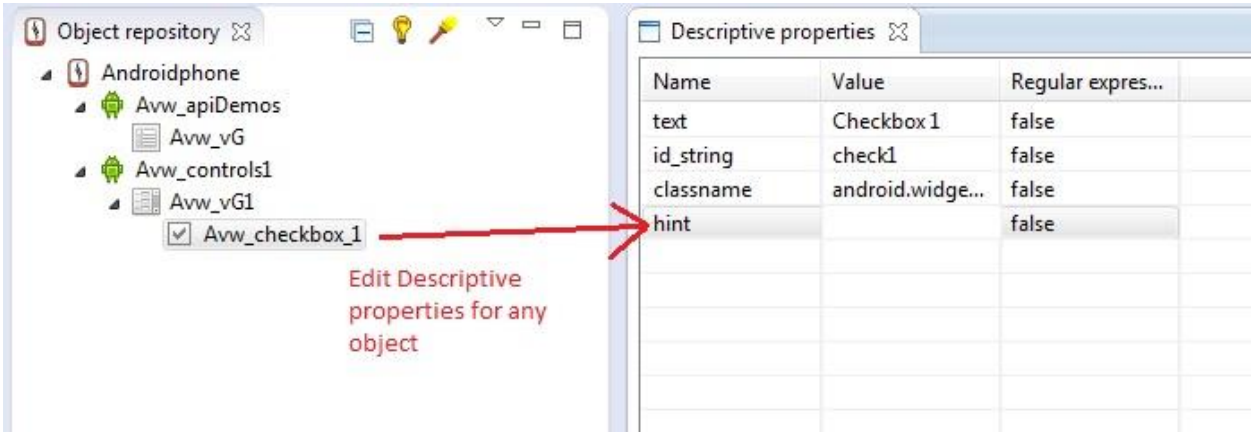


2) Open the Object Repository in editor using “Open in editor” option
    a. If you edit anything in the Object Repository from editor then to reflect those changes in OR view you need to press  .
3) Learn GUI- Objects from your app can be learnt directly using the  button
4) Drag and drop of objects to the script editor for quick script creation.



5.7.2. Descriptive Property View (DP View):

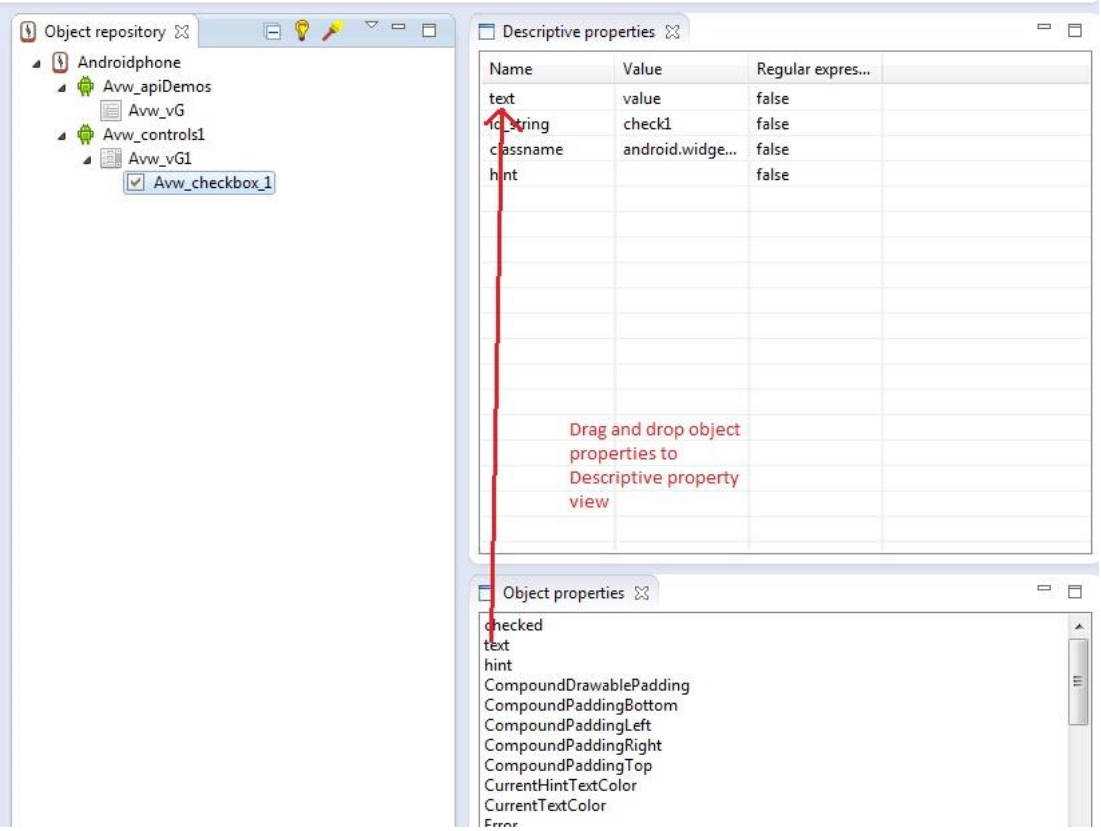
This view will get loaded with descriptive Property - Value pairs of the object selected in OR view. Descriptive properties can be edited for any object using this view. No need to manipulate the properties in the source file of the Object Repository.



5.7.3. Object Properties View (OP View):

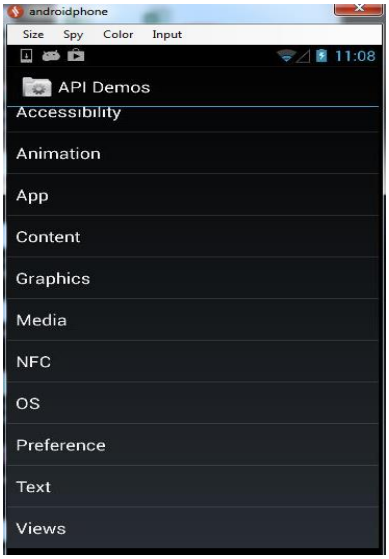
This view will show all the properties which are associated with the object selected in OR view. Using Drag and Drop feature you can make any property from this view a descriptive property. But you will

require to edit the "Value" of that property yourself.

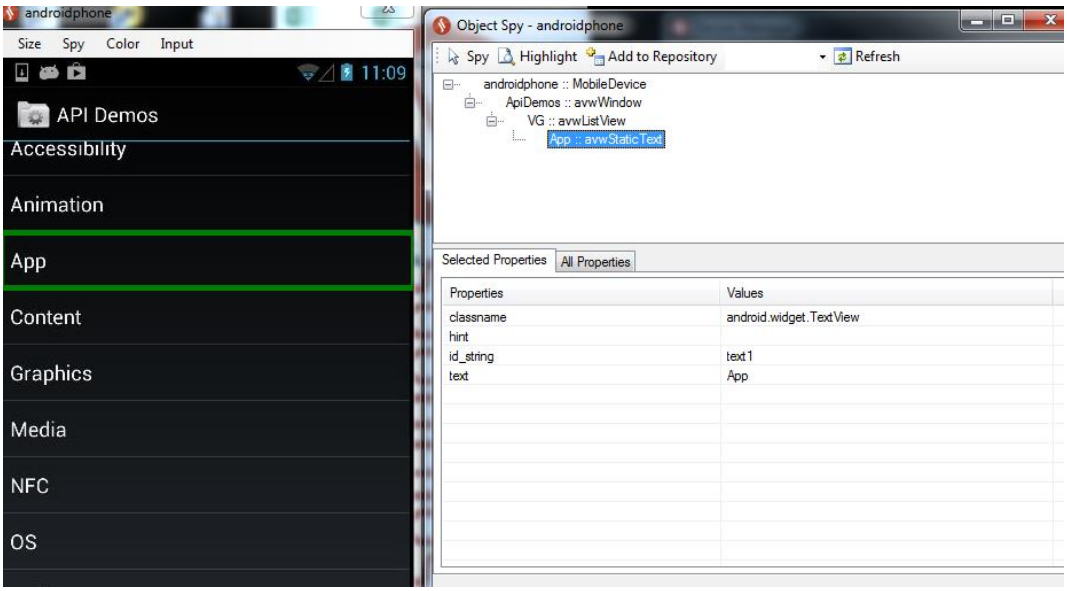


5.7.4. Adding Objects to Object Repository View (OR View) using Meux spy

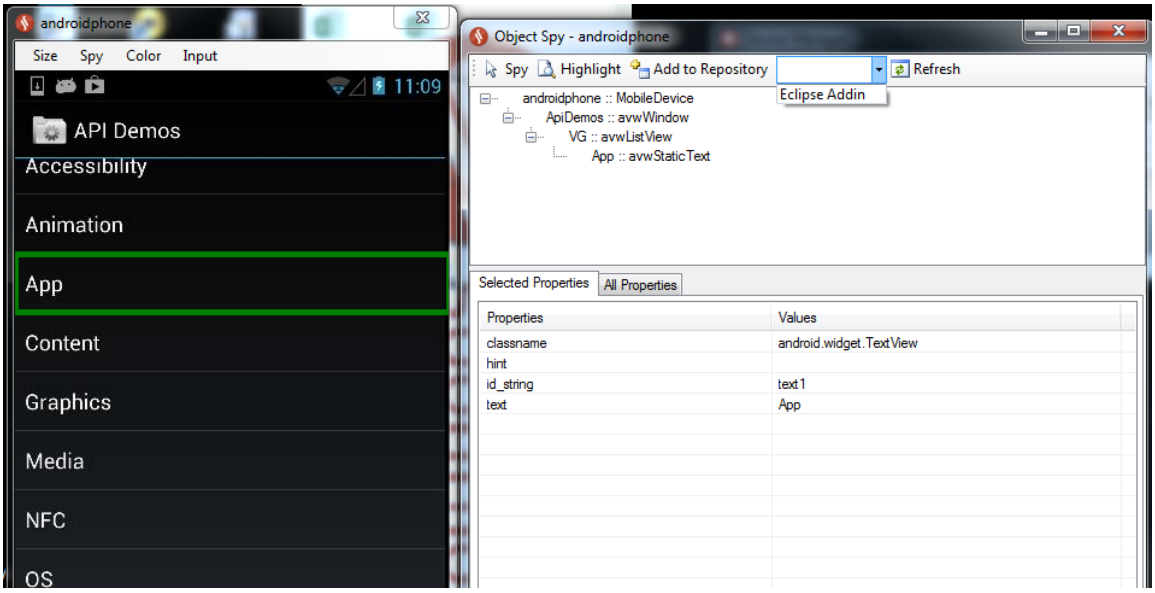
1) Open RDS and launch testable app
2) Select Spy option from the RDS



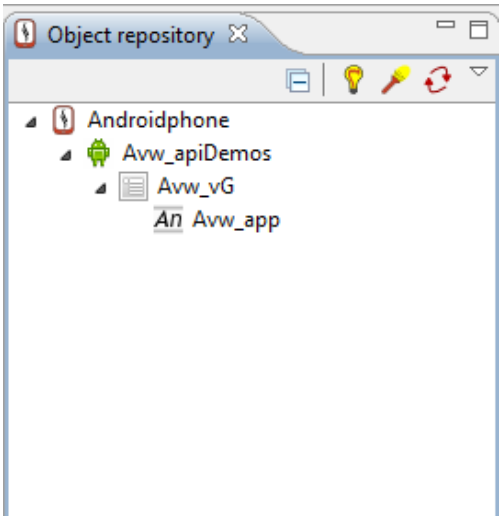
3) Move your mouse pointer on RDS and locate the Object in scope



4) Select Eclipse Add in from the drop down in spy window.



5) Click on Add to Repository button.
6) Object will get added in OR view



7) Now you can use this object in the script.

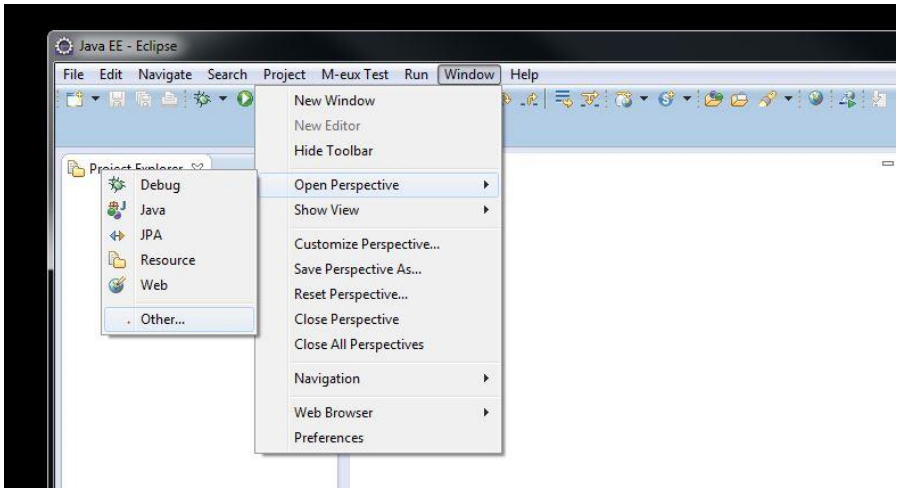
Chapter 6: Scripting with the Old M-eux Test Plugin – java perspective

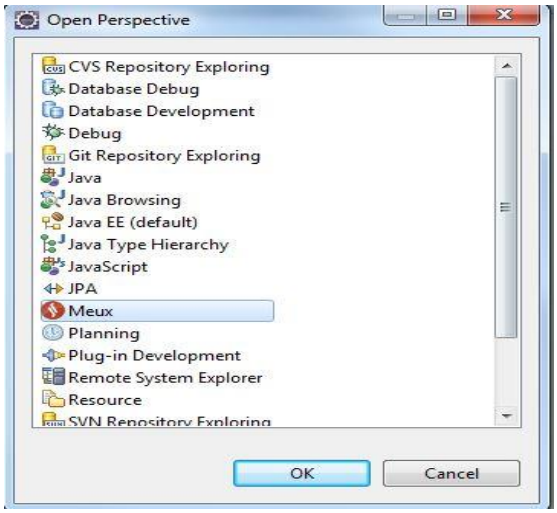
The following section will give step by step instructions on how to create a M-eux Test Eclipse Project and give an overview of the project structure.

6.1. Creating a M-eux Test Eclipse Project

4) Launch Eclipse with the "MeuxEclipseStarter" shortcut on the desktop.

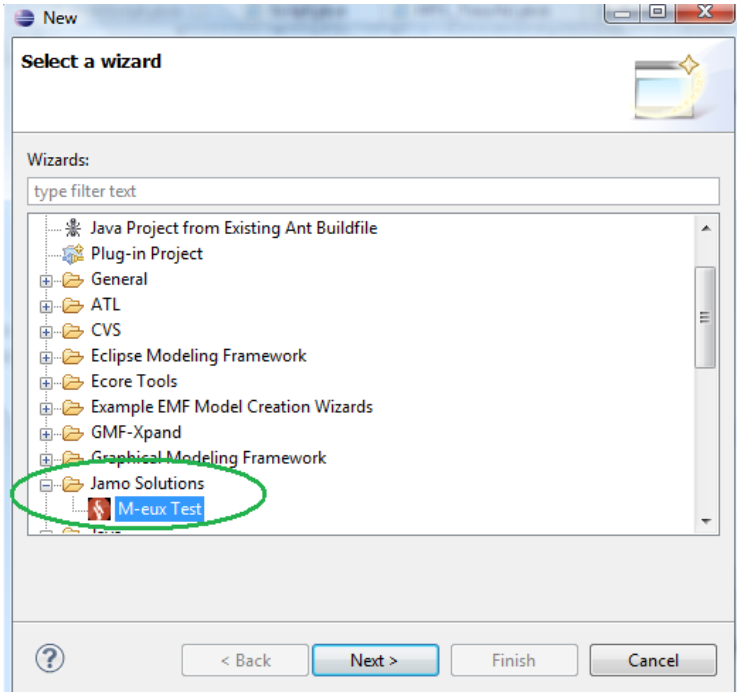
5) Create a new Workspace. (You can have multiple M-eux Eclipse project in your workspace.)
6) Meux perspective then you can follow additional steps below-
    d. Go to Windows->show Perspective->Other



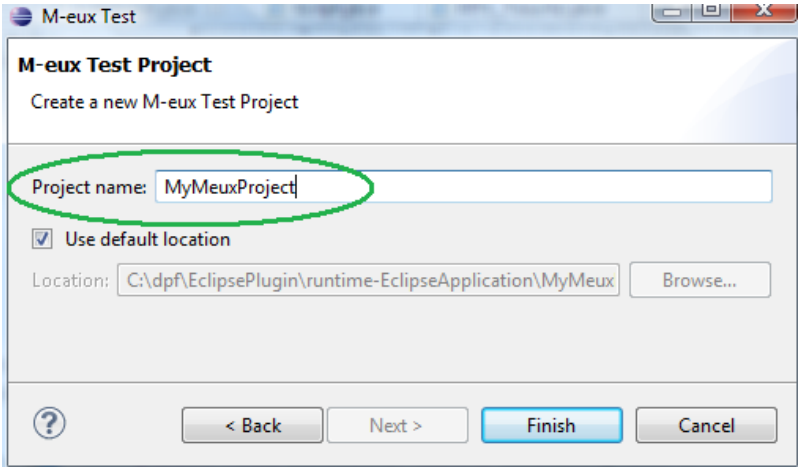


7) Select Java

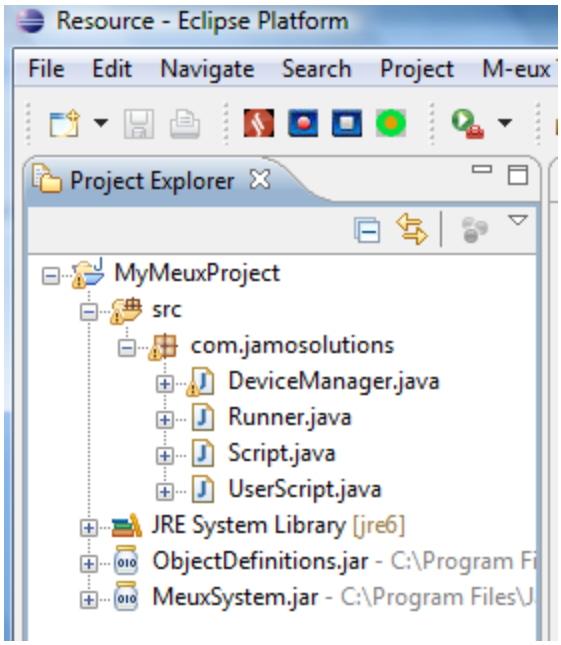
8) In Eclipse select File->New->Other
9) Select M-eux Test under Jamo Solutions.



10) Enter a name for your project



11) The result should look like this :



6.2. Project Structure

When you create a new Meux Eclipse Project there are 4 java classes in the com.jamosolutions package:

1. UserScript.java
2. Script.java
3. Runner.java
4. DeviceManager.java

1) **UserScript.java**

The UserScript.java class contains a private constructor and the runCore method.

The UserScript constructor :

```
private UserScript(){
    getMeuxReplaySettings().setDelay(0); // Delay before a method to run is delegated to the device
    getMeuxReplaySettings().setFindObjectTimeOut(20000); // Maximum timeout for finding objects (mil
}
```

In this constructor you can add a delay between execution statements. By doing so the script will halt for the specified period of time in milliseconds between each statement. The default delay is 0.
You can also change the “FindObjectTimeout”. The “FindObjectTimeout” is the time in milliseconds the script will look for the object. If the object is not found within the specified interval a MeuxObjectNotFound Exception will be generated. The default “FindObjectTimeout” value is 20000.

The runCore method

```
public void runCore() throws MeuxException {

}
```

In the runCore body your recorded statements will be generated and you can type your script.

```
public void runCore() throws MeuxException {
    hTC_P3470.hHTaskBar.tap("24", "7");

}
```

The runCoreDelegate method

```
public void runCoreDelegate(ResultManager resultManager) throws MeuxException{
    this.setResultManager(resultManager);
    try {
        this.runCore();
    }finally{
        resultManager.AddResult("test1", resultManager.isGlobalSuccess());
        resultManager.setGlobalSuccess(true);
    }
}
```

Call this method to run your test case from a different script. This can be useful when you’re making an execution plan.

For instance:

```
public void runCore() throws MeuxException {
    try {
        com.jamosolutions.test1.UserScript.getInstance().runCoreDelegate(getResultManager());
    } catch (Exception e) {
        e.printStackTrace();
    }
    try {
        com.jamosolutions.test2.UserScript.getInstance().runCoreDelegate(getResultManager());
    } catch (Exception e) {
        e.printStackTrace();
    }
```

2) **Script.java**
In the Script.java class the various MobileDevice objects and the DeviceManager object will be initialized and registered with the script. Initially it will look like this :

```
public abstract class Script extends AbstractScript {

    public Script(){
        this(new MeuxReplaySettings());
    }

    public Script(MeuxReplaySettings meuxReplaySettings) {
        super(meuxReplaySettings);
        register(deviceManager);
    }
    DeviceManager deviceManager = new DeviceManager(null);
}
```

After a recording session or a learn GUI it might look like this :

```
public abstract class Script extends AbstractScript {

    public Script(){
        this(new MeuxReplaySettings());
    }

    public Script(MeuxReplaySettings meuxReplaySettings) {
        super(meuxReplaySettings);
        register(deviceManager);
        register(hTC_P3470);
    }
    DeviceManager deviceManager = new DeviceManager(null);
    HTC_P3470 hTC_P3470 = new HTC_P3470(null);
}
```

3) **Runner.java**
The Runner.java class contains the main method.
4) **DeviceManager.java**
the DeviceManager.java class contains the DeviceManager definition.

From the 4 classes in the Meux project only the UserScript class and the ObjectPool (more on this later) should be edited by the user.

The Object Pool

Definition:

The object pool is the collection of Mobile Device classes that are added to the M-eux project when you

create your script. Each device will have its own java class that contains the known object definitions for that Mobile Device.

6.3. Example

The following example is created against a windows mobile device named HTC_P3470.

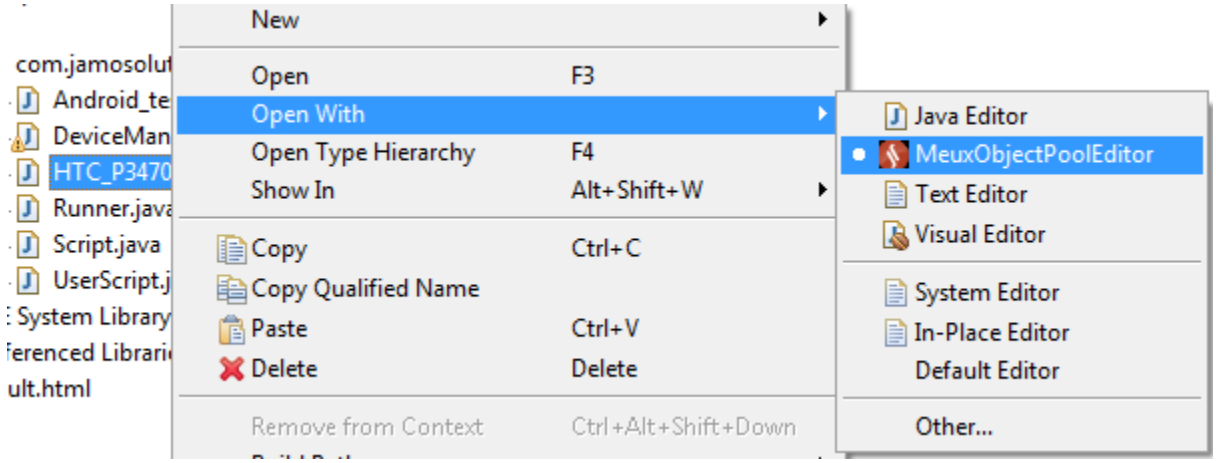
When recording a script: Statements are generated in the runCore method of UserScript.java

```
public void runCore() throws MeuxException {
    hTC_P3470.hHTaskBar.tap("24", "7");

}
```

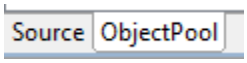
ObjectDefintions are generated in the ObjectPool to support the recorded statements.

6.4. MeuxObjectPoolEditor

With the MeuxObjectPoolEditor the ObjectPool can be explored in Source and ObjectPool mode. To open a file with the MeuxObjectPoolEditor right click on the file and select “Open With”->”MeuxObjectPoolEditor”. If MeuxObjectPoolEditor isn’t in the list select “Other…” and look for it in the “Internal programs” list.

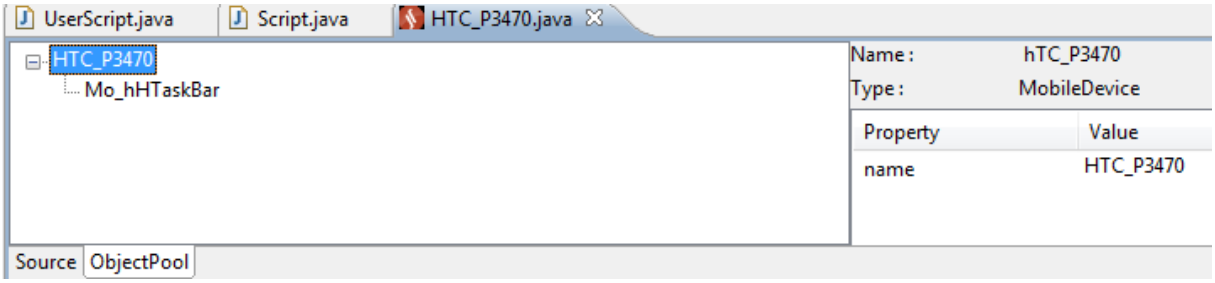


Switch between Soure and ObjectPool by toggling on the bottom.

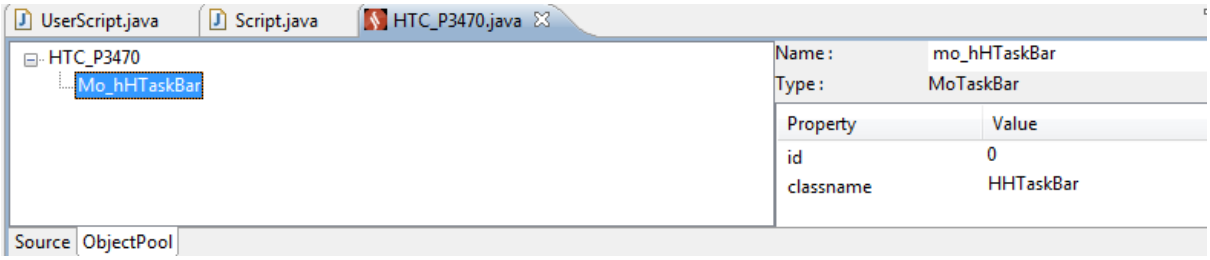


6.4.1. ObjectPool tab:

In the example a class HTC_P3470 was added of the type MobileDevice. This class has 1 property name and the value is set to “HTC_P3470” which is the name of the device.

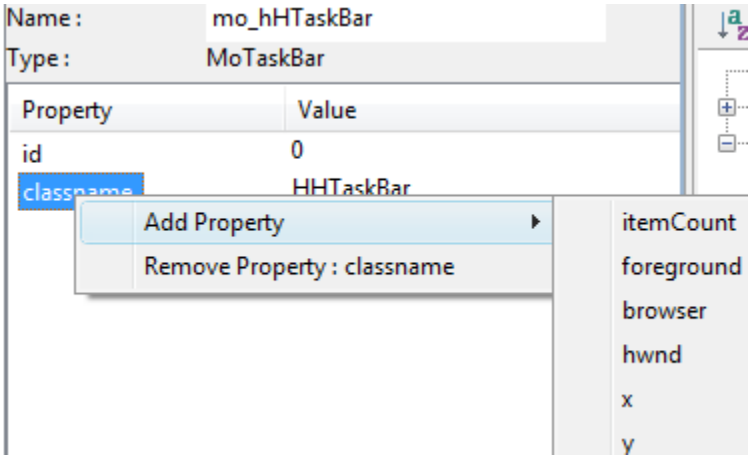


The HTC_P3470 object has 1 child object Mo_hHTaskBar of the type MoTaskBar which as 2 properties: id and classname with respective values “0” and “HHTaskBar”.



Add/Remove properties

In the properties table you can add properties by right clicking on one of the existing properties. You can remove properties by right clicking on the property you want to remove and selecting “Remove Property” from the menu or by pressing delete.



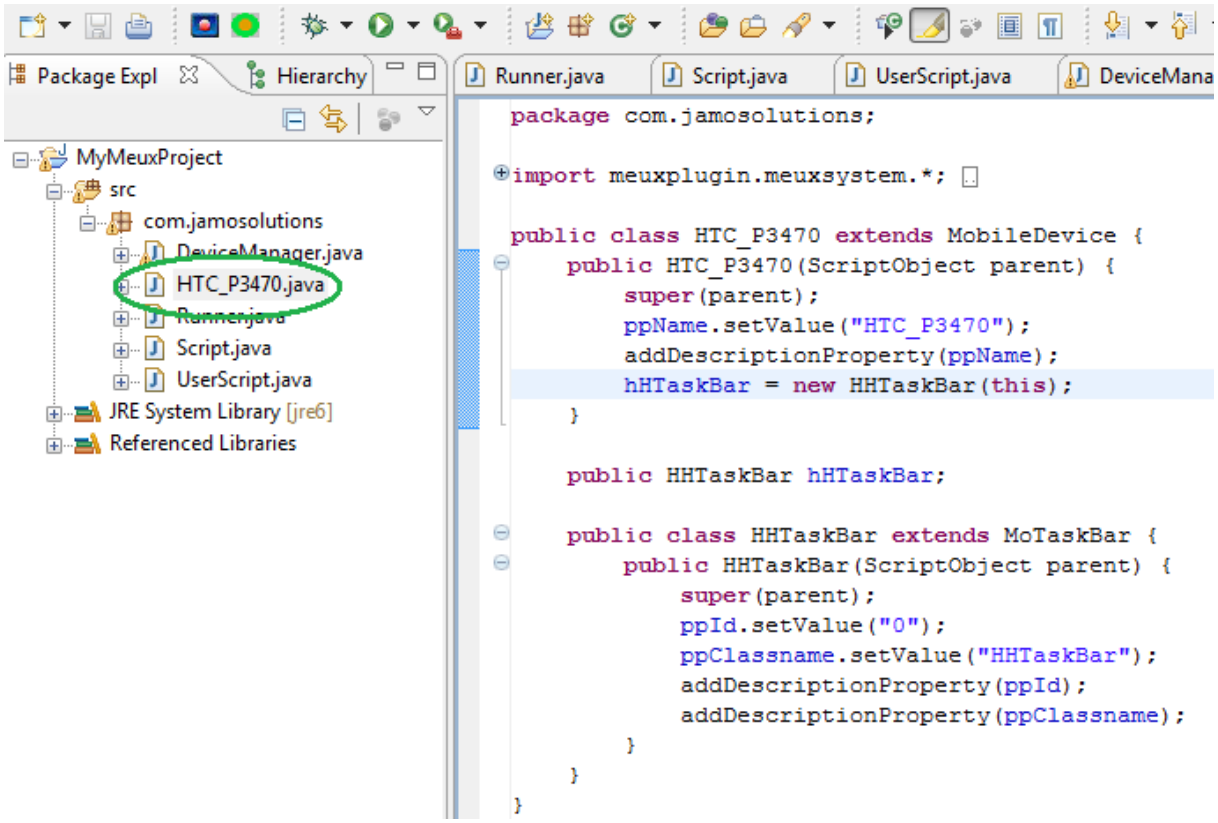
Remove objects

You can remove objects by selecting them from the tree and pressing delete.

Change Property values

Select the property value you want to change and type the new value in the edit field. Please note that you need to remove focus from the edit field for changes to take effect.

6.4.2. Source tab



In the example a class named HTC_P3470 was added of the type MobileDevice. This class has 1 property ppName and the value is set to "HTC_P3470" which is the name of the device.

```
public class HTC_P3470 extends MobileDevice {
    public HTC_P3470(ScriptObject parent) {
        super(parent);
        ppName.setValue("HTC_P3470");
        addDescriptionProperty(ppName);
        hHTaskBar = new HHTaskBar(this);
    }
```

The HTC_P3470 class has 1 inner class HHTaskBar of the type MoTaskBar which as 2 properties: ppID

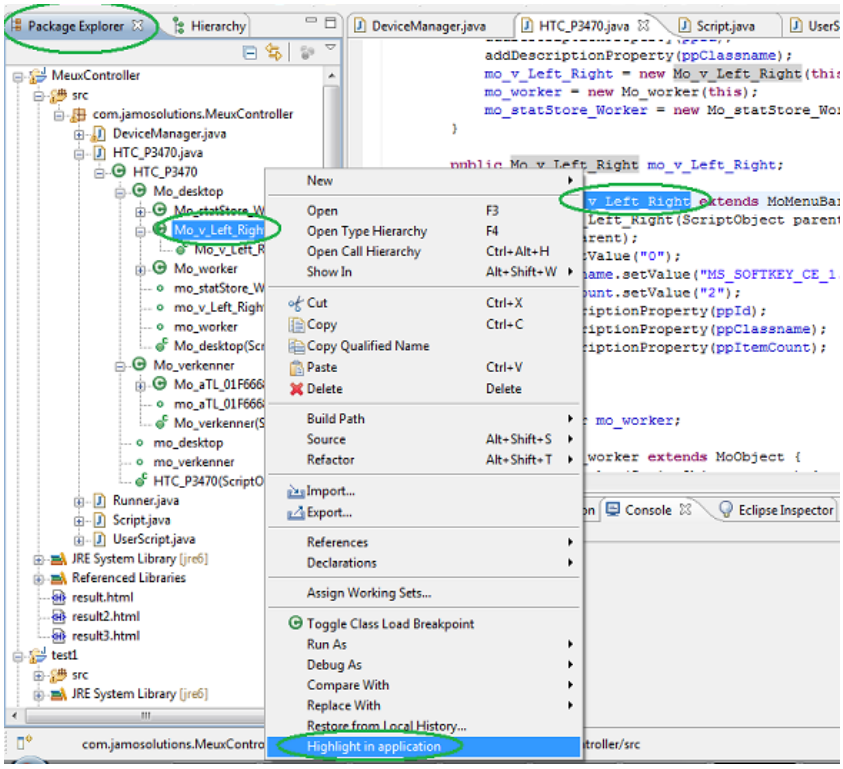
and ppClassname with respective values “0” and “HHTaskBar”.

```
public HHTaskBar hHTaskBar;

public class HHTaskBar extends MoTaskBar {
    public HHTaskBar(ScriptObject parent) {
        super(parent);
        ppId.setValue("0");
        ppClassname.setValue("HHTaskBar");
        addDescriptionProperty(ppId);
        addDescriptionProperty(ppClassname);
    }
}
```

6.5. Highlight in application

To help identify objects in the objectpool, the Highlight in application functionality has been added to the M-eux test plug-in. The highlight functionality is only available for devices that support RDS.

1. Select an object in the Eclipse package explorer.
2. Right click on the object
3. Select Highlight in application

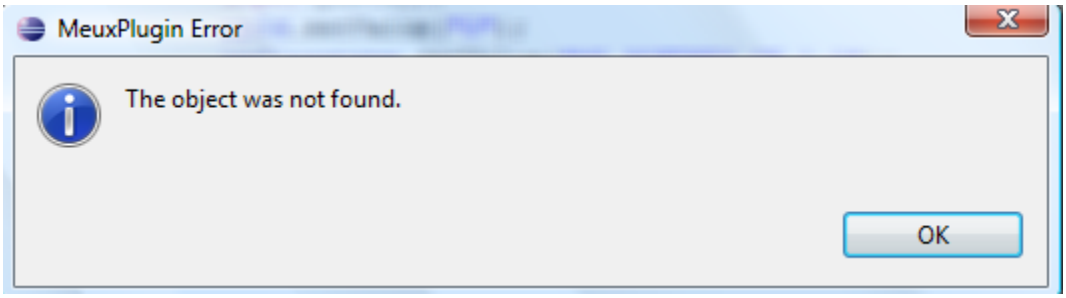


4. If the object is found in the application, the RDS will be activated and highlight the object on the display. (Highlighting in white the menu bar)

5. You can close the RDS by selecting Tools->Remote Device Screen->DeviceName

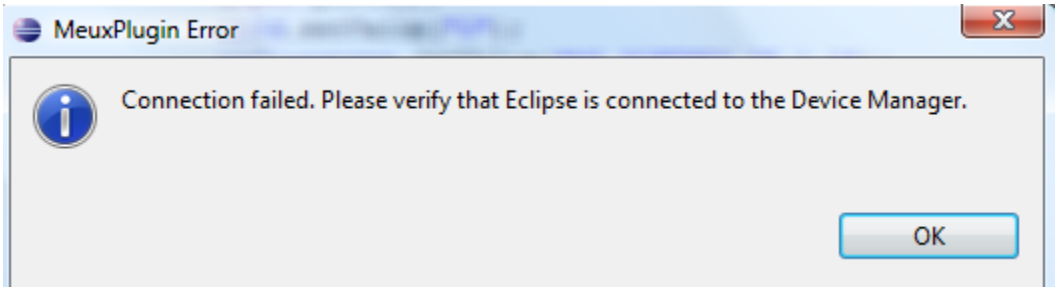
6.5.1. Object not found

A dialog indicating the object was not found in the application. Please verify if the device is listed in the "Connected Devices" tab of the device manager and that the object is visible on the screen.



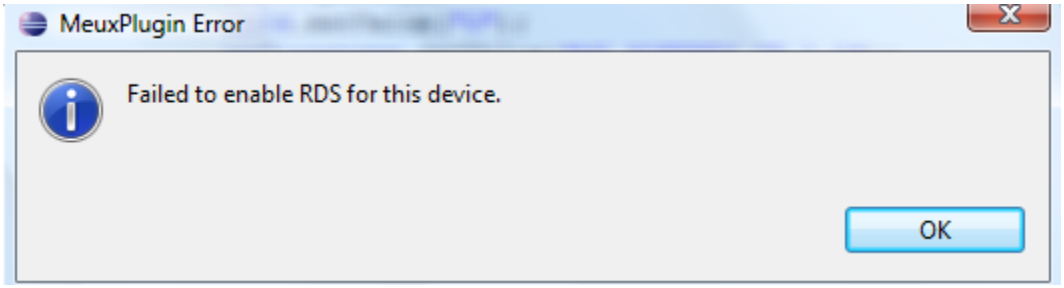
6.5.2. Connection failed

A dialog indicating that the device manager could not be contacted. Please verify if Eclipse is connected to the device manager in the "Connected Testing Tools" tab of the device manager.



6.6. Failed to enable RDS for this device

A dialog indicating that RDS was not enabled successfully.

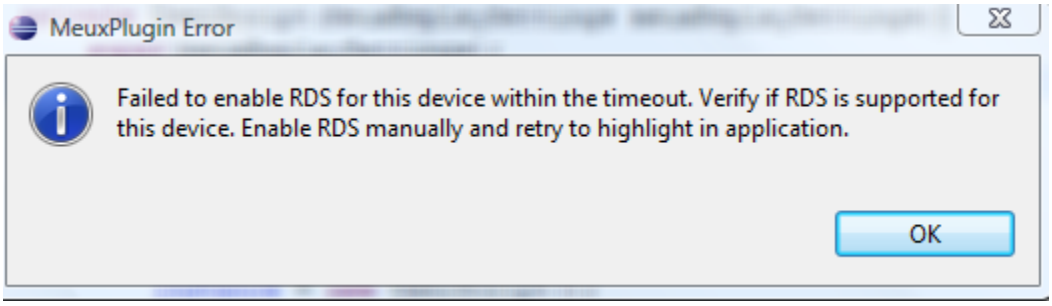


Please note that RDS is not supported for all device configurations. RDS is currently supported for:

5. Windows Mobile
6. Windows CE
7. Android Emulator
8. Android rooted devices

6.7. Failed to enable RDS for this device within the timeout.

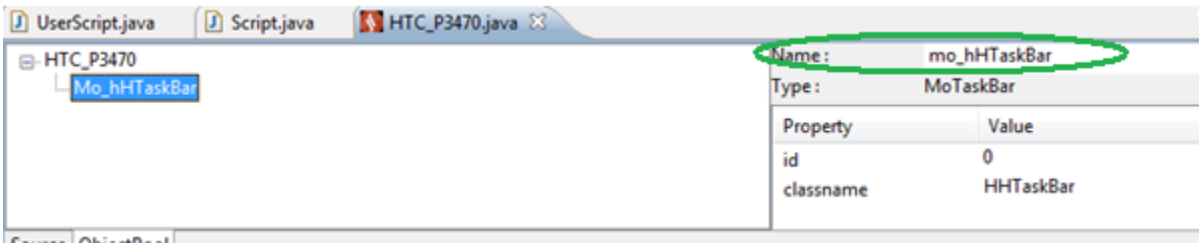
A dialog indicating RDS was not enabled within the timeout. The timeout is 20 seconds. To work around this problem you can enable RDS manually and then retry to highlight in application.



6.8. Changing names in the object pool (refactoring)

To make your script more readable it can be desirable to refactor some of the objectnames to names that are more meaningful for human users.

Open the file with MeuxObjectPoolEditor and change the name in the Name edit field. Please note that you need to remove focus from the edit field for changes to take effect.

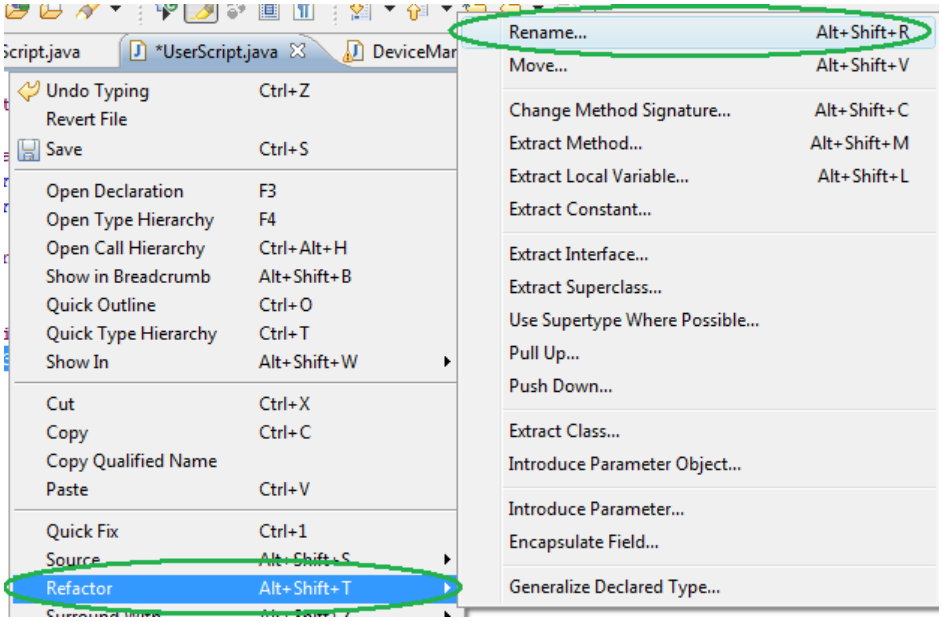


OR:

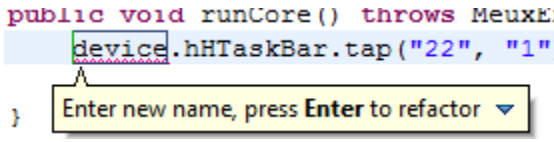
Select the name of the object you want to rename in UserScript.java or in your Object Pool

```
public void runCore() throws MeuxException {
    hTC_P3470.hHTaskBar.tap("22", "1");

}
```

Right click on it with the mouse and select Refactor->Rename



Enter the new name and press "Enter"



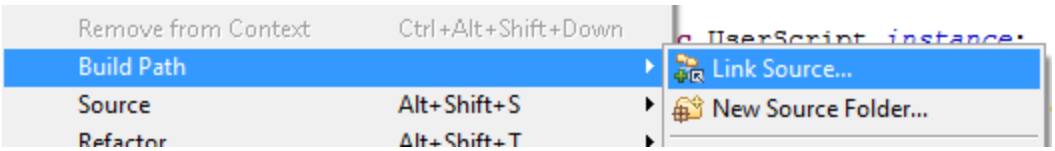
The result will look like :

(If you get an error after the refactoring you just need to save your UserScript.java)

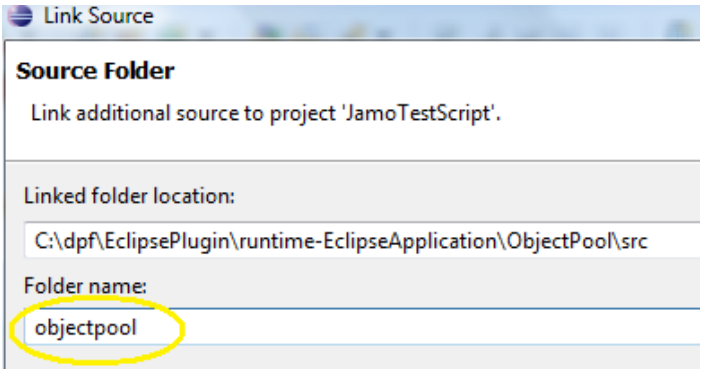
```
public void runCore() throws MeuxException {
    device.hHTaskBar.tap("22", "1");

}
```

6.9. Managing a central ObjectPool

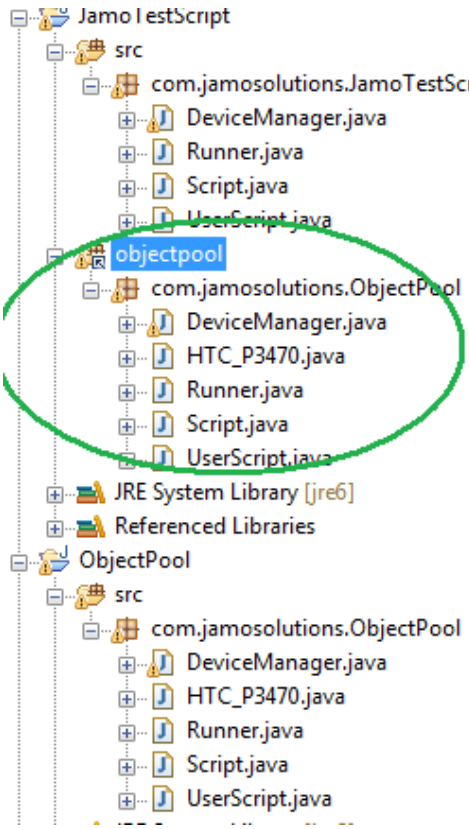
To manage a central objectpool for multiple testcases you will need to create at least 2 M-eux projects (1 TestScript project and 1 objectpool project). Link the objectpool source to the TestScript project. (Right click on the test project and selecting "Build Path->Link Source".)



The “Link Source” dialog will open:



Enter the physical location of the object pool’s src directory and type as folder name objectpool. It’s important you name the folder objectpool.

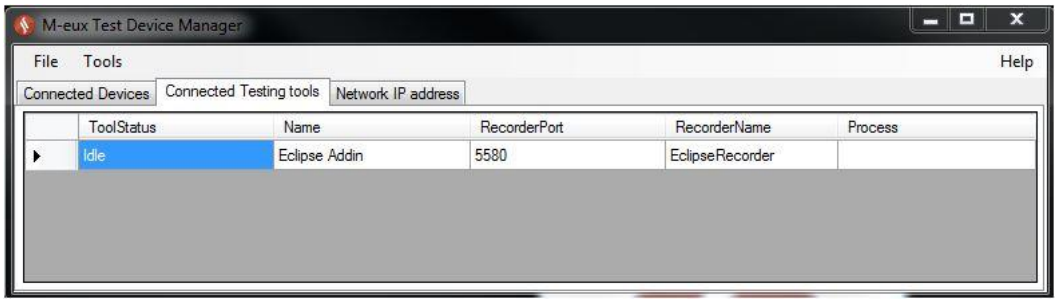


From this point on when you perform a learn GUI for the TestScript project the object definitions will be placed in the ObjectPool project. Record statements will go in the UserScript of the TestScript.

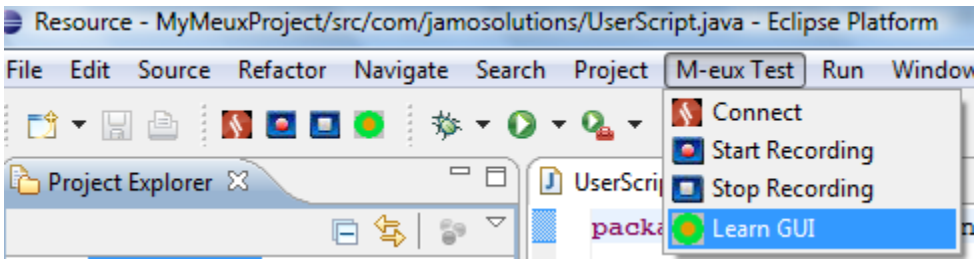
6.10. Learn GUI

To add all the objects that are currently accessible on the mobile device to the object pool you can perform a learn GUI.

8) Make sure that Eclipse is connected to the device manager.

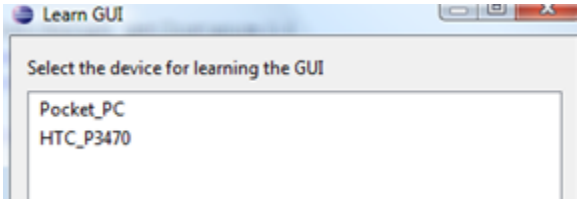


9) Select Learn GUI from the M-eux Test Menu

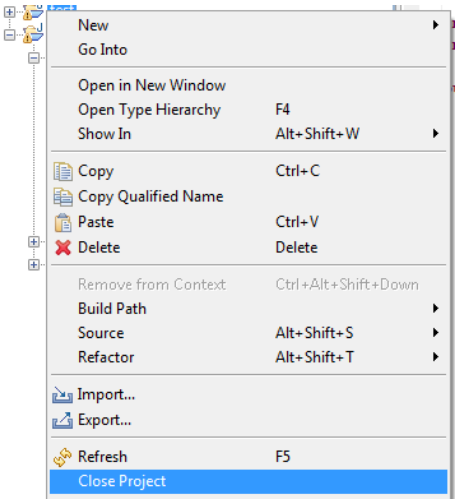


10)

a. If only one device is connected the objects will be added to the objectpool.
b. If multiple devices are connected you will get a dialog box inquiring which devices object you wish to learn.

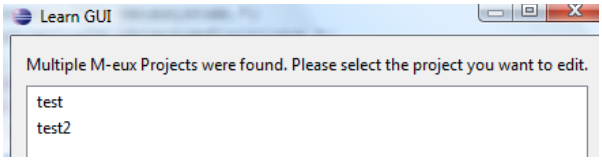


c. You can close projects you don't want to modify too avoid this dialog box. Right click on the project and select "Close Project".



11)

a. If only one M-eux Eclipse project is open the objects will be added to the objectpool
b. If multiple projects are open a dialog box will prompt for which project you wish to edit.

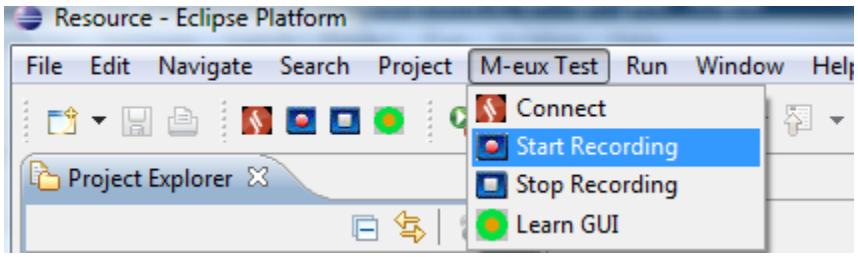


Chapter 7: Recording a script

Recording is a tool that will help you in the creation of your test scripts. An alternative to recording is performing a learn GUI and typing out your script manually using Eclipse intellisense.

9) Make sure the agent on the device is running and connected to the device manager

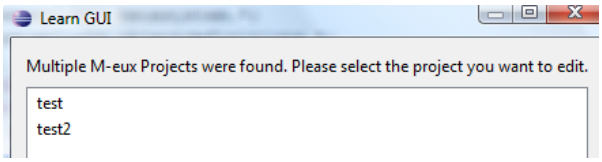
10) Press Start Recording from the Menu



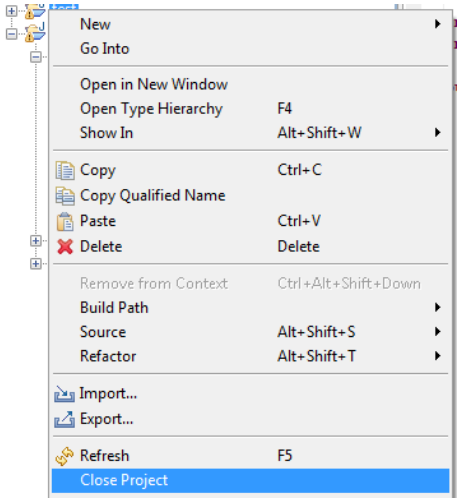
11)

a. If only one M-eux Eclipse project is open proceed to the next step.

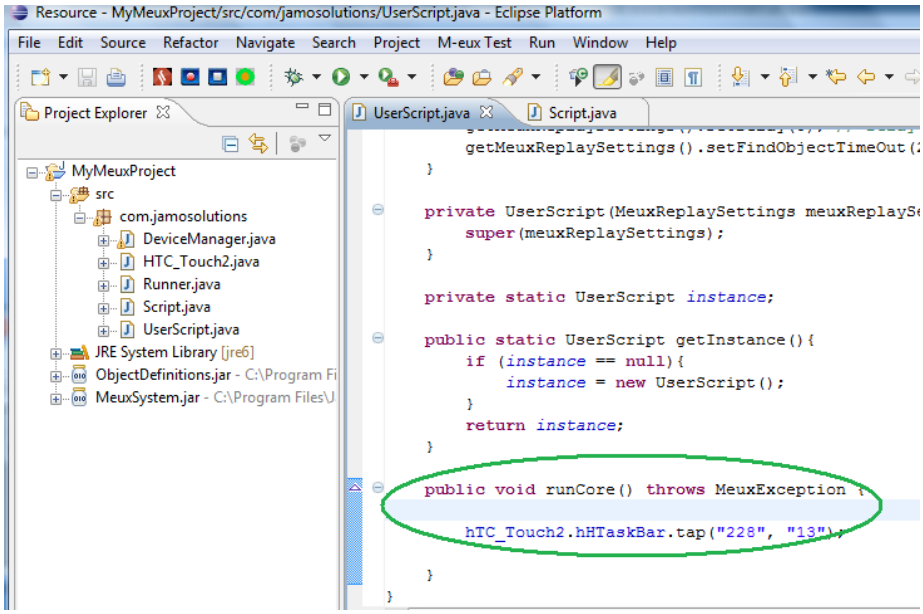
b. If multiple projects are open a dialog box will prompt for which project you wish to edit.



c. You can close projects you don't want to modify too avoid this dialog box. Right click on the project and select "Close Project".



12) Perform actions on the device
13) The record statements will be generated in the runCore function of the UserScript.java Class.



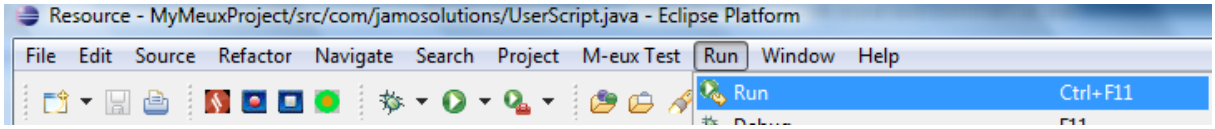
14) Press Stop recording to end the recording session

Chapter 8: Replaying a Script

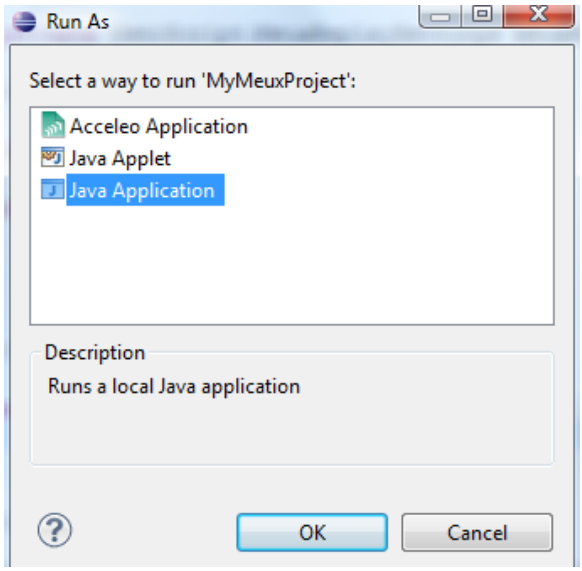
You can replay your script directly in Eclipse or with the command line.

8.1. Run the Eclipse test script from Eclispe IDE

1) Select Run->Run

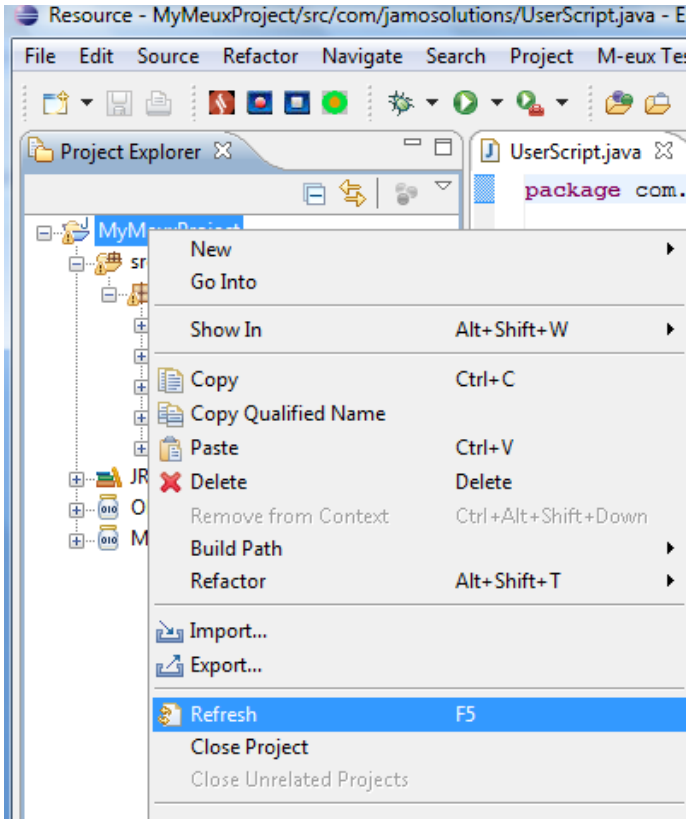


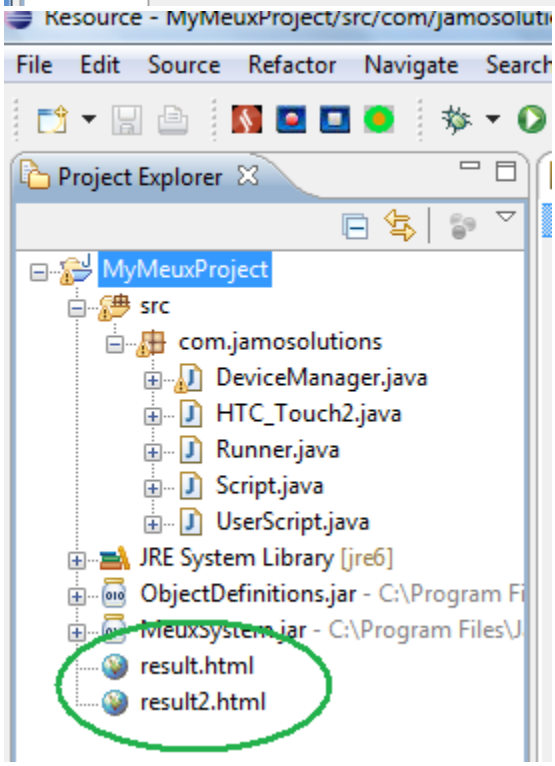
2) Select Run as Java Application



3) The results will saved in the project root folder as result.html

4) Select Refresh to view the results in the Eclipse project explorer

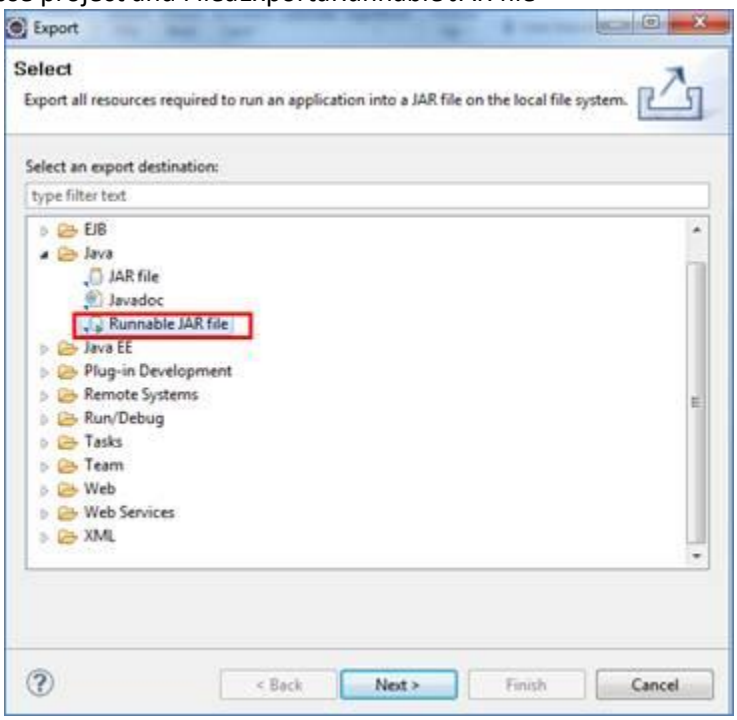




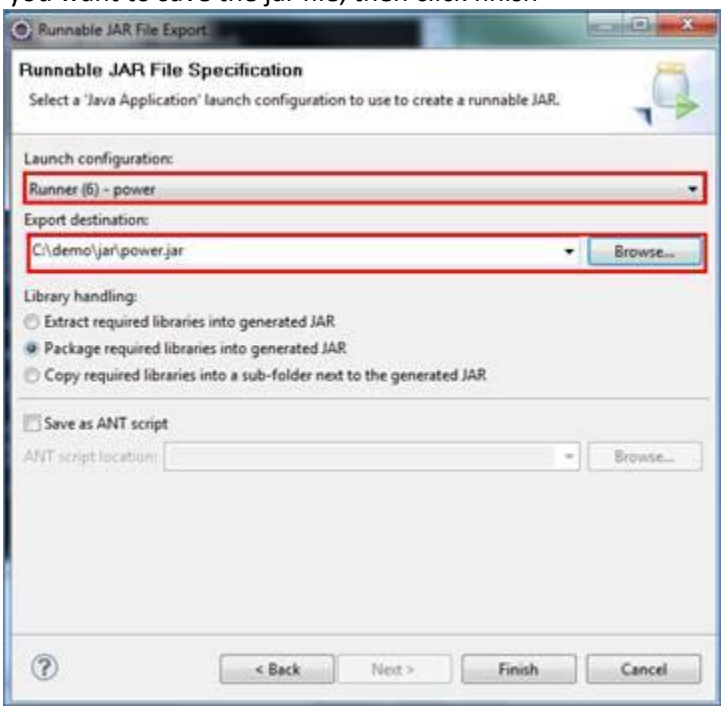
8.2. Run the Eclipse script from the command line

Generate jar file for your test project:

a. Go to your eclipse project and FileàExportàRunnable JAR file



b. Specify the project name which you want to generate jar file in the workspace and the directory where you want to save the jar file, then click finish



c. Now you already have the jar file generated, you can open the cmd window and navigate to the location where you save your jar file and execute the following command:

   “C:\Program Files (x86)\Java\jre6\bin\java.exe” –jar power.jar
   Of course change the power.jar to your own jar file.

Chapter 9: Using M-eux Test with JUnit Tests

You can use M-eux Test to test the UI of your application from within JUnit unit tests. This chapter describes how you can achieve this.

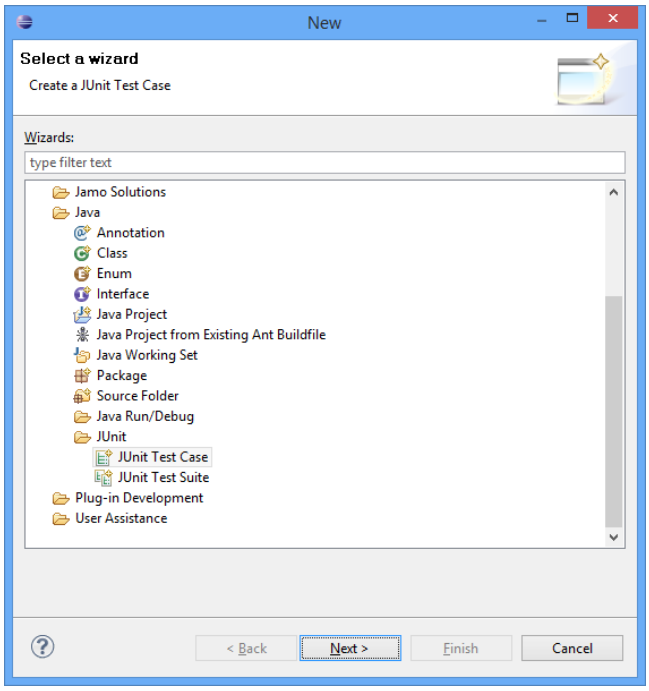
Before you get started, please note that you cannot record test scripts directly within JUnit tests.

9.1. Creating your JUnit Test Class

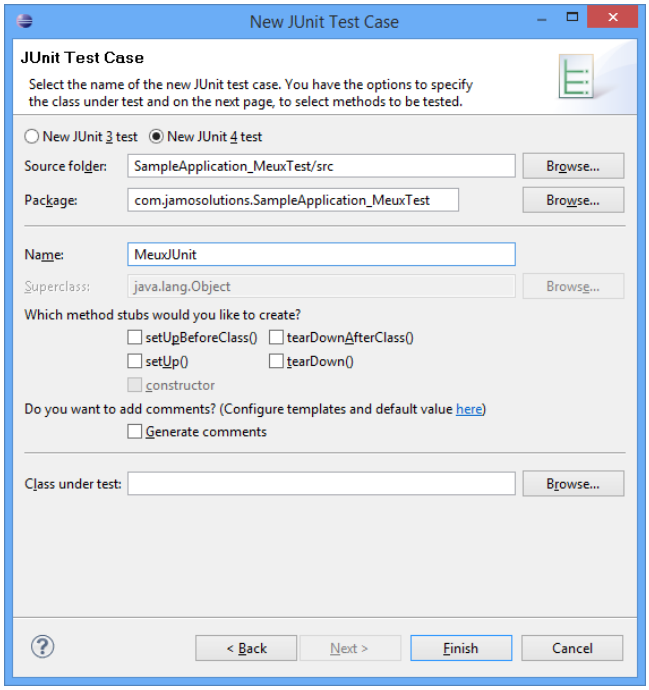
You can add JUnit tests to an existing M-eux Test project. This step assumes you already have a M-eux Test project and you already know how to record and replay test scripts from that project. For more information on M-eux Test project, please see section 6.1.

To add a JUnit Test to your test project:

1. Open your M-eux Test project in Eclipse.
2. Click **File**, **New** and **Other…**
3. In the **Wizards** tree, select **Java, JUnit** and **JUnit Test Case** and click **Next**



4. Select **New JUnit 4 test**, and accept the default values or provide your versions. Then, click **Finish**.



5. Add the following import statements to your JUnit class:

```
import java.io.File;

import junit.framework.Assert;

import meuxplugin.meuxsystem.ResultManager;
import meuxplugin.meuxsystem.exceptions.MeuxException;
import meuxplugin.meuxsystem.executer.HtmlResultManager;
import meuxplugin.meuxsystem.meuxdmrunnereclipse.MeuxDMRunnerEclipse;
```

6. Change the declaration of your JUnit class to inherit from the **Script** class:

```
public class MeuxJUnit extends Script {
```

7. Create the following constructor for your JUnit class. Replace MeuxJUnit with the name of your JUnit test class.

```
public MeuxJUnit(){
    MeuxDMRunnerEclipse meuxDMRunner = new MeuxDMRunnerEclipse();
    ResultManager r = uniqueFileOnCurrentDirectory("result");

    UserScript.setMeuxDMRunner(meuxDMRunner);
    this.setResultManager(r);
    this.run();

    getMeuxReplaySettings().setDelay(0); // Delay before a method to run is delegated
to the device manager  (milliseconds)
    getMeuxReplaySettings().setFindObjectTimeOut(20000); // Maximum timeout for
finding objects (milliseconds)
}
```

8. After the constructor, add the following two helper methods to your JUnit class.

```
public void runCore() throws MeuxException {
```

```
}

private static ResultManager uniqueFileOnCurrentDirectory(String fileName)
{
    File f = new File(".\\" + fileName + ".html");
    HtmlResultManager resultManager = new HtmlResultManager(f);
    return resultManager;
}
```

9.2. Add a Test to your Test Class

To add a test to your test class that can execute M-eux Test scripts, add a new method with the @Test decoration that throws a MeuxException, like this:

```
@Test
public void Test() throws MeuxException  {
}
```

9.3. Add test code to your Test class

You can use the following approaches for adding test code to your test classes:

- You can use the **Recording** functionality to generate the code in the runCore method of the UserScript class. Then, copy the code from the runCode method in the UserScript class to your test method in your test class. For more information on recording test scripts, see Chapter 7:: Recording a script.
- You can use the **Learn GUI** functionality to pre-populate the object pool. Then, you can write your test code directly against the Object Pool. For more information on Learn GUI, see Section 6.10: Learn GUI.

You can now use the standard JUnit features such as assertions to add validation logic to your test script. Below, you can find an example of a JUnit test written agains the UICatalog application on an iOS device using M-eux Test:

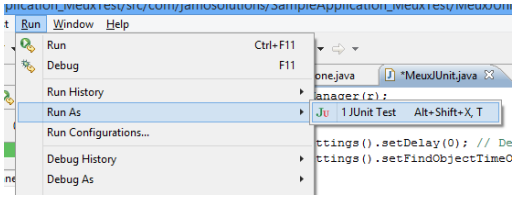
```
/*
 * This is a test for the 'Buttons' screen of the UICatalog sample application that ships
with
 * M-eux Test for iOS.
 * It contains a button, 'Gray', which, when clicked, changes a counter which keeps track
of how
 * often this button has been clicked (1, 2, 3,...).
 * This is displayed in a label which says 'Gray button clicked: N'
 *
 * This test simulates a click on the Gray button and ensures the counter increased by
exactely
 * one.
 */
@Test
public void Test() throws MeuxException  {
       String text;
```

```
	// This will hold the number of times the button was clicked when the test
started.
	int firstClickedCount;

	// This will hold the number of times the button was clicked when the test
finished.
	int secondClickedCount;

	// Get the text 'Gray button clicked: N' and derive firstClickedCount from there.
	text =
debby_s_iPhone.ios_uICatalog7_0.ios_tableView_Buttons.ios_v_gray_button_clicked1
	.ios_v_gray_button_clicked.getPPText().value;
	firstClickedCount = GetClickedCount(text);

	// Execute a click on the Gray button.
	debby_s_iPhone.ios_uICatalog7_0.ios_tableView_Buttons.ios_background_Image.ios_gra
y
	.push();

	// Get the text 'Gray button clicked: N' and derive firstClickedCount from there.
	text =
debby_s_iPhone.ios_uICatalog7_0.ios_tableView_Buttons.ios_v_gray_button_clicked1
	.ios_v_gray_button_clicked.getPPText().value;
	secondClickedCount = GetClickedCount(text);

	// Make sure secondClickedCount = firstClickedCount + 1
	Assert.assertEquals(firstClickedCount + 1, secondClickedCount);
}

/*
 * From the text 'Gray button clicked: N', gets the number N.
 */
private int GetClickedCount(String text)
{
	String clickedText = text.split(":")[1].trim();
	int clicked = Integer.parseInt(clickedText);
	return clicked;
}
```

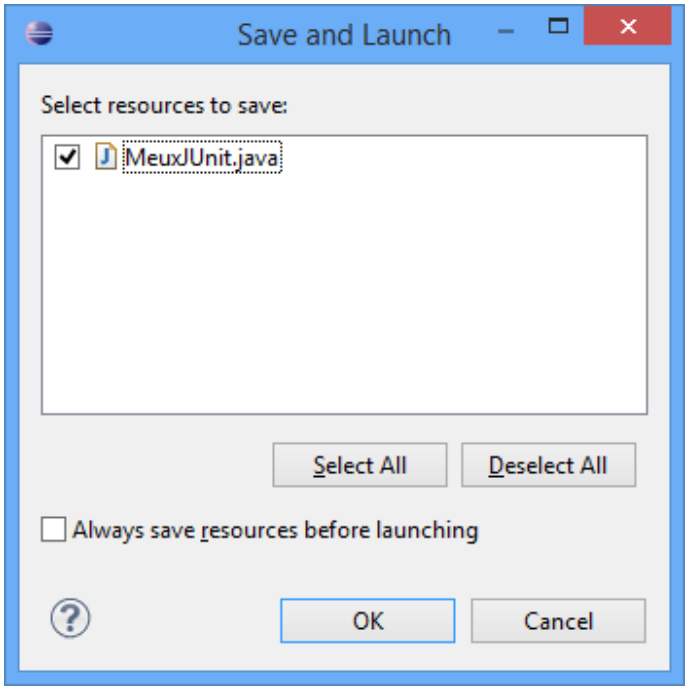
9.4. Execute your JUnit Test

To execute your test, please follow these steps:

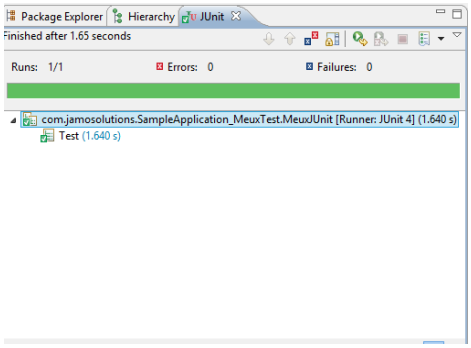
1. In Eclipse, click **M-eux Test** and **Connect**
2. In the source code explorer, navigate to the JUnit test you want to execute.
3. Click **Run**, **Run As** and select **JUnit Test**



4. Specify which tests you want to run and click **OK**



5. Once the test has executed successfully, you will see the test results in the left hand side of the screen. If the JUnit window does not appear, click **Window**, **Show View**, **Other** and select **Java**, **JUnit**.



9.5. Using the test set up and tear down scripts

You can use the test set up and tear down methods to perform actions such as starting and stopping your application.

Chapter 10: Troubleshooting

10.1. Starting eclipse from the command line

Start eclipse with the –vm option passing javaw.exe as argument:

```
C:\Program Files\eclipse>eclipse -vm "C:\Program Files\Java\jre6\bin\javaw.exe"
```

Chapter 11: Summary

Now you should be able to start with the testing on Windows Phone.

We wish you a good journey with your test adventure using M-eux Test tool. If you are facing any issues with our tool, or having trouble about making your test scripts for your test cases, you can always contact us at support@jamosolutions.com.