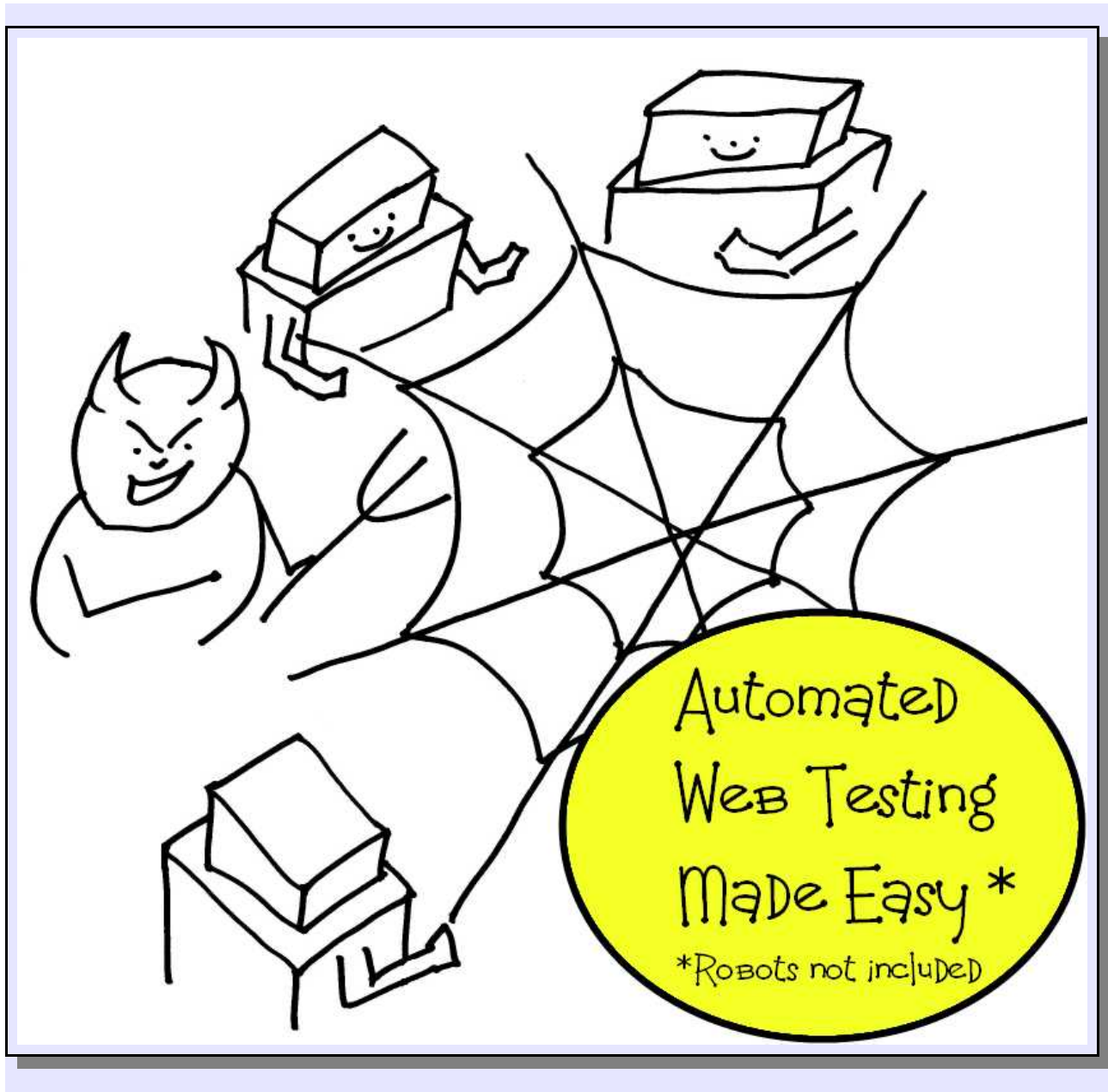


Selenium Simplified

A tutorial guide to using the Selenium API in Java with JUnit

Alan Richardson



Published by Compendium Developments

Copyright © 2010 by Alan Richardson

The right of Alan Richardson to be identified as the author of this work has been asserted by him in accordance with the Copyright, Design and Patents Act 1988.

The views expressed in this book are those of the author.

First published in Great Britain in 2010 by:

Compendium Developments

<http://www.compendiumdev.co.uk/selenium>

contact details:

alan@compendiumdev.co.uk

Author's Software Testing Blog: www.eviltester.com

Compendium Developments: www.compendiumdev.co.uk

Every effort has been made to ensure that the information contained in this book is accurate at the time of going to press, and the publishers and author cannot accept any responsibility for any errors or omissions, however caused. No responsibility for loss or damage occasioned by any person acting, or refraining from action, as a result of the material in this publication can be accepted by the editor, the publisher or the author.

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Design and Patents Act 1988; this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission of the publishers, or in the case of reprographic reproduction in accordance with the terms and licenses issued by the Copyright Licensing Agency, 90 Tottenham Court Road, London, W1T 4LP. Enquiries concerning reproduction outside these terms should be sent to the publishers.

Version @ 19/01/2011

2nd Printing

e-book ISBN : 978-0-9567332-0-7

paperback book ISBN : 978-0-9567332-1-4

Table of Contents

Introduction.....	9
Getting the Source-code.....	9
Windows, Linux or Mac.....	10
Acknowledgments.....	10
Chapter 1:Getting Started With The Tools.....	11
What Tools?.....	11
Selenium IDE.....	11
Java.....	11
Eclipse.....	12
Selenium-RC.....	12
Chapter 2:Selenium IDE Basics.....	13
Install Firefox.....	13
Install Selenium-IDE.....	13
Capture Play Back – Recording a Script with the IDE.....	14
Command, Target and Value.....	17
Save The Test.....	17
Further reading.....	17
Chapter 3:Install Java.....	19
Chapter 4:Install and Run Selenium-RC.....	20
Step One - Download & Install Selenium-RC.....	20
Overview of the Contents of the Selenium Archive.....	21
Step Two - Run Selenium-RC.....	22
Step Three – Stop the Selenium Server.....	23
From the command line.....	23
From a browser URL.....	24
Running an IDE Generated Test in Different Browsers.....	24
End Notes.....	26
Chapter 5:The Eclipse IDE.....	27
First - Install Eclipse.....	27
Second - Run Eclipse.....	28
Third - create a new Java project.....	31
Chapter 6:Create a JUnit test Using the JUnit export from Selenium-IDE.....	33
Introduction.....	33
Export an IDE script as a JUnit test.....	33
Load an existing script into the IDE.....	33
Change the format of the script to Java.....	34
Create a New Class in Eclipse.....	35
Copy & Paste the code from Selenium IDE into Eclipse.....	38
Resolve Import Errors.....	40
Add JUnit to the build path.....	42
Remove the unused import.....	44
Run the JUnit test.....	45
Allow it through the Firewall.....	45
Seeing the test running.....	46
Check the results.....	47
It went too quickly!.....	48

Run the test in debug mode.....	49
Create and Import some more IDE Converted tests.....	52
Exercises.....	55
Chapter 7:The Annotated Generated Test.....	56
Use Attach Source to see the Selenium Driver Source Code.....	56
Why should we do this?.....	59
MyFirstSeleniumTests.java Annotated.....	59
A little more about selenium-java-client-driver.jar.....	63
A little about JUnit.....	65
A look at SeleneseTestCase.java.....	65
Use the SeleneseTestCase Functionality.....	67
Using the IDE to add asserts and verifies.....	67
First record the setup steps.....	67
Add the Verify Statements.....	68
Add the Asserts.....	70
What is the difference between an Assert and a Verify?.....	71
How to find line numbers in Eclipse?.....	73
Creating Screenshots on Failure with SeleneseTestCase.....	74
Chapter 8:Let's get coding.....	77
A brief pause, because you have already learned some Java.....	77
What we will now learn.....	77
Create a New Test From Scratch.....	77
Our First Test with Annotations.....	78
Back to Our Selenium Test.....	80
Making Something Happen.....	82
Your Final Test Code.....	83
The DefaultSelenium constructor explained.....	84
Retrospectives.....	85
Well Done.....	85
Why did you do that?.....	85
Run Selenium-RC in a DOS window?.....	85
Start it easily.....	85
See the logging messages.....	85
Find The Window.....	86
Stop and Restart Selenium Easily.....	86
Final Notes.....	86
Back to Eclipse.....	87
Removing the Source Code Association.....	87
Chapter 9:Chapter - What if it all goes wrong?.....	88
Check that Selenium Server is running.....	88
Close any Selenium-RC windows.....	89
Stop and restart the Selenium-RC server.....	89
Remove any Java threads from the Debug perspective.....	89
Close any blocked browser windows.....	90
Firefox Update Dialog.....	90
Summary.....	91
Chapter 10:Essential Firefox Add-Ons.....	92
What is Firebug?.....	92
What is XPather?.....	92
Basic Firebug Usage.....	93

Basic XPath Usage.....	95
Optimise XPath with XPather.....	96
Chapter 11:First Steps with Test Automation Thinking.....	99
Let's Automate Search.....	99
Start writing some tests.....	99
Make Selenium Type In Some Text.....	100
Start with a copy and Paste.....	100
Fix the duplicate method name.....	101
Add a Selenium Command.....	102
The “type” method.....	103
Find the locator with FireBug.....	103
A brief introduction to XPath.....	105
Add the locator into the code.....	106
Check that it works.....	106
Now make it Click on the Search Button.....	107
Selenium “Click” Method.....	107
Locate the button with FireBug.....	107
Create an XPath so that selenium can locate it.....	108
Add the XPath details to our source code.....	108
One last thing to do.....	108
Now check that it works.....	108
Quick Summary.....	108
But is it a test if we don't check any results?.....	109
But we can't control the data.....	109
Automate the Acceptance Criteria.....	110
Acceptance Criteria: Selenium Remote Control homepage listed in returned results.....	110
Look for text in the page.....	111
Use assertions to make the test fail on error.....	112
Debugging a Failing Test.....	113
Our final test with assertions.....	114
Scan the page source code.....	115
Look for a specific URL in the page.....	116
Most Robust Method.....	119
Acceptance Criteria: Page title has “Selenium-RC” in it.....	120
Acceptance Criteria: Search box on the page has Selenium-RC in it.....	120
One test to rule them all.....	120
Summary.....	121
Chapter 12:First Steps with Automation Refactored.....	123
Introducing Refactoring.....	123
Automatically refactoring the start up code into a new method.....	124
Automatically refactor the tear down code into a new method.....	126
JUnit @Before and @After	128
Refactoring Plan and Analysis.....	128
Remove Parameter from stopSeleniumAndCloseBrowser.....	129
Move setup code into a new method.....	130
Remove the setup code from each @Test method.....	131
Annotate stopSeleniumAndCloseBrowser with @After.....	131
Remove the tear down code from each @Test method.....	131
The Refactored Code.....	133
@BeforeClass & @AfterClass.....	134

Our refactored code.....	135
Automatically create JUnit Test Structure.....	136
Summary.....	138
Chapter 13:How do I upgrade my Selenium Server when a new version comes out?.....	139
First Download the New Version.....	139
Configure Eclipse Project.....	139
Remember to re-attach the source.....	141
Chapter 14:Chapter - Basic HTML Theory.....	142
Page Structure.....	142
Elements & Tags.....	142
Attributes.....	143
Expanded.....	143
Chapter 15:Basic XPath Theory.....	144
XPath Expressions.....	144
Node Types.....	144
Selections	145
Predicates.....	145
Advanced.....	146
Combining Match Queries.....	146
Wild Card Matches.....	146
Boolean Operators.....	147
XPath Functions.....	147
XPath optimisation.....	148
Use the ID.....	148
Use the attributes.....	148
Start at the first unique element.....	149
Selenium XPath Usage.....	149
Chapter 16:Basic CSS Selector Theory.....	150
Firefinder for Firebug.....	150
CSS Selector Expressions.....	151
Selenium & CSS.....	151
Selections.....	152
Direct Descendents and Absolute Paths.....	152
Sub Elements and Relative Paths.....	153
Attribute and Index matching.....	153
Attribute Matching.....	153
Special attribute selectors.....	153
Indexed Matching.....	154
Advanced.....	155
Combining Matches.....	155
Wild Card Matches.....	155
Attribute Substring matching.....	155
Boolean Operators.....	155
Sibling Combinators.....	156
Useful Links.....	156
Chapter 17:Learning The Selenium API.....	157
Chapter 18:Testing HTML Forms.....	158
An HTML Form.....	159
Submit a Form with Default Values.....	159
Commands Used.....	162

Submit Form without clicking button.....	162
Amend a Text, Password, TextArea or File Field.....	162
Text Fields.....	162
Type vs TypeKeys.....	163
Password Fields.....	163
TextArea Fields.....	163
File Field.....	164
Attachfile command.....	165
Amending Checkboxes and Radio Items.....	165
Click.....	165
Check	166
Uncheck.....	167
Amending Multiple Select Values.....	168
Amending Dropdown Elements.....	169
Amending Hidden Field Values.....	170
Checking the Values of the Input Fields.....	171
Text, Password, TextArea, File.....	171
CheckBox & Radio Items.....	171
Multiple Select & DropDown.....	172
getValue.....	173
isSomethingSelected.....	173
getSelectedId & getSelectedIds.....	173
getSelectedIndex & getSelectedIndexes.....	173
getSelectedLabel & getSelectedLabels.....	174
getSelectedValue & getSelectedValues.....	174
getSelectOptions.....	175
Put It All Together.....	175
Chapter 19:Testing Static HTML.....	177
A Basic Static HTML Test.....	177
getTitle – Check the page title.....	179
getText – Check the username.....	180
isTextPresent – check the 'heading'.....	182
isElementPresent – check comments and filename.....	182
getXPathCount – count the checkboxes entered.....	183
click – navigate to the form.....	185
The rest of the test.....	185
The full test and summary.....	186
getAttribute.....	187
assignId.....	187
getBodyText & getHtmlSource.....	188
goBack.....	189
highlight.....	189
Chapter 20:Using JavaScript with Selenium.....	191
A Basic JavaScript Enabled Page.....	191
The Class.....	192
FireEvent, isAlertPresent, and getAlert.....	193
getEval.....	194
RunScript.....	194
AddScript.....	195
Chapter 21:Start Selenium Programmatically.....	196

Add Selenium Server to the project.....	196
Start it in the code.....	197
The Start Up routine explored.....	198
BindException Handling.....	198
Stop Existing Server.....	199
Custom Remote Control Configurations.....	200
Chapter 22:Running Tests Outside Eclipse.....	202
What is Ant?.....	202
What is Hudson?.....	202
What is Subversion?.....	202
Chapter 23:Using Ant to Run Selenium Tests.....	203
Install Ant.....	203
Install Java JDK.....	203
Update the Environment Variables and Path.....	204
Check that Ant works.....	206
Check that Java works.....	206
Create a Lib Folder.....	207
Amend Eclipse Project Properties.....	207
Create an initial build.xml file.....	207
Running the Ant file.....	211
Add the Tests which require no server.....	213
Refactor The Build File.....	214
Summary.....	217
Additional Reading.....	217
Chapter 24:Using Hudson to Run Ant.....	219
Introduction.....	219
Install Hudson.....	219
Using Hudson without Version Control.....	222
Configure the Hudson Project.....	224
Scheduling the build automatically.....	227
Chapter 25:Adding Subversion to the mix.....	228
Chapter 26:Take Stock of Where we are.....	229
Plan Of Action.....	229
Chapter 27:Evolving A Selenium Manager Class.....	232
Chapter 28:Create Our First Page Objects.....	239
Create a SearchPageTests class.....	239
Add the SeleniumManager	240
Convert startSeleniumAndSearchForSeleniumRC.....	241
typeInASearchTermAndAssertThatHomePageTextIsPresent.....	245
typeInASearchTermAndAssertThatHomePageURLExists.....	246
typeInASearchTermAndCheckPageTitleHasSearchTermInIt.....	247
typeInASearchTermAndCheckSearchInputHasSearchTermInIt.....	247
After all that	248
SearchPageTests.java.....	248
SearchPage.java.....	249
Final Notes.....	249
Chapter 29:More Page Object Creation.....	250
The Tests.....	250
HTML_form_tests.java.....	250
test_submit_form_with_default_values	251

test_submit_form_without_clicking_submit	251
test_submit_form_with_new_username	252
test_submit_form_with_new_password.....	252
test_submit_form_with_new_password.....	252
test_submit_form_with_filename.....	252
test_submit_form_with_click_check_and_radio.....	253
test_submit_form_with_multiple_select_values.....	253
test_submit_form_with_dropdown_values.....	254
test_submit_form_with_hidden_field.....	254
test_check_text_entered_values.....	254
test_check_checkbox_values.....	255
test_check_radio_values.....	256
test_randomly_select_value_from_dropdown.....	257
Static_HTML_tests.java.....	258
JavaScript_With_Selenium_Tests.java.....	263
The Page Objects.....	264
BasicHTMLForm.java.....	264
Possible refactorings.....	266
HTMLFormResultsPage.java.....	268
Discussion of getFilename.....	269
BasicAjaxPage.java.....	270
Chapter 30:Page Object Models Summary.....	271
What is a Page Object Model?.....	271
Grow an abstraction layer.....	271
Refactor the Page Objects internally.....	271
Create Domain Level Abstractions.....	272
Lessons learned.....	272
Related Study.....	273
Chapter 31:Data Driven Tests in JUnit.....	274
Basic Data Driven Testing.....	274
JUnit Parameterized Class Runner And Constructor.....	275
Data supplied by an @Parameters annotated method.....	275
A Data Driven Test.....	276
Running a Data Driven Test.....	276
The full code for the test.....	277
Multiple Tests.....	278
Reading Data From Tab Delimited Files.....	278
General Hints.....	279
Chapter 32:Screen Capture on Failure.....	281
Chapter 33:Run the tests on multiple browsers.....	286
Browser Codes.....	286
Property Files.....	287
System Properties.....	289
Chapter 34:JavaScript And Dynamic HTML Testing Redux.....	291
Example Dynamic HTML Page.....	291
First Model the behaviour in the test.....	292
Code the check in the page object model.....	292
First understand the page.....	292
Summary.....	293
Chapter 35:Cookie Handling.....	294

Cookies in the example application.....	294
How to see cookies on a web page?.....	294
Firefox by Default.....	294
Firecookie - Firebug Plugin.....	297
Selenium Cookie Commands.....	298
Example Code For Cookie Testing.....	299
Basic Class.....	299
Check Cookie Creation using deleteAllVisibleCookies and isCookiePresent.....	299
Get Cookie values using getCookie and getCookieByName.....	300
getCookie.....	300
getCookieByName.....	301
deleteCookie.....	302
createCookie.....	303
Chapter 36:The Future of Selenium.....	305
If you want to experiment with 2.0.....	306
Chapter 37:Structuring the tests and code.....	308
Java project folder and package structures.....	308
Changing what goes into Version Control.....	311
Configuring Source-code Folders in Eclipse.....	311
Add libraries.....	313
Re-organise the Classes and Packages.....	313
Share the Server among tests and create a base test class.....	315
Share the Server using a Singleton.....	315
Create a Base Class.....	316
Get Ant Working.....	318
Make sure all tests are valid tests.....	320
Create end to end tests where we assert after executing functionality.....	320
Have all tests pass on multiple browsers.....	320
Locators as constants or methods.....	321
Locators as constants.....	321
Locators as methods.....	322
Get Hudson Working.....	323
Speed up the test execution with a suite.....	327
Running Different Suites for Different Browsers.....	331
Nested Suites.....	331
Browser Specific Groups.....	332
Create a conditional Ant File.....	333
Get Selenium 2.0 working.....	334
Ways of using what we have built.....	335
Exercises For the reader:.....	336
Chapter 38:In Closing.....	337
Additional Reading.....	337
Appendix A - Selenium API MindMap.....	339
Appendix B - Playing along at home.....	340

Introduction

For some people, seeing some Java code for Selenium-RC on the web, does not help as they don't know how to use it. They don't know:

- where to put the code,
- what a package is,
- how to run the code,
- how to install an IDE

So in this guide I present, in simple steps and supported by screenshots, the fundamental knowledge required to program Selenium tests in Java.

Once you have the basics, we move on to exploring common Selenium API commands and how to automate HTML pages, Forms and JavaScript enabled web applications.

I assume no previous knowledge of Java, or Automated Testing.

This book covers Selenium 1.0.3 the mature and stable version of Selenium. The Selenium 1.0.3 branch is likely to remain stable and static. The newer version of Selenium – Selenium 2.0 is still undergoing heavy development, so we cover this in the “The Future of Selenium” Chapter towards the end of the book.

Be assured that Selenium 1.0.3 is more than capable of automating your web applications. Indeed Selenium 2.0 supports the API used in Selenium 1.0.x, so everything you learn here, you can use in Selenium 2.0 when it matures.

I have chosen Java as the programming language because it is the language that the Selenium Server is written in (so by understanding Java you can understand Selenium more readily). It is also mature and very well supported by IDEs (Integrated Development Environments) which make coding simple for a beginner.

Getting the Source-code

This book takes a tutorial approach. So we present a lot of source-code. To save you having to type it all out – although there are benefits in doing this. You can download the source-code from:

- http://www.compendiumdev.co.uk/selenium/InitialSeleniumTests_JUNIT.zip
- <http://www.compendiumdev.co.uk/selenium/finalChapterSource.zip>

Follow the instructions in “Appendix B - Playing along at home” on page 340 to install the source-code.

This way, if you are following the explanations and typing in the code, should you get stuck, you can compare your source-code to that in the 'official' archive.

For Chapter 37 “Structuring the tests and code” on page 308, where we make our testing 'production' quality, the source-code can be downloaded from the public subversion repository

- <http://svn2.xp-dev.com/svn/seleniumsimplified>

This is the same code provided in finalChapterSource.zip listed above.

This is hosted on a free subversion server provided by <http://www.xp-dev.com/>

Windows, Linux or Mac

This book takes a Windows centric view of the world. So the main text assumes you are working on a Windows system and all screen shots have been taken on a Windows system.

This was purely to limit the install explanations for software. All the tools listed are available for Linux and Mac systems: Eclipse, Firefox, the various Firefox plug-ins, Selenium.

If you visit the various tool websites and follow the install instructions for your particular platform then you should still be able to follow along with the text and instructions.

Acknowledgments

No book has the exclusive influence of the Author. Normally in this section the author would give thanks to the editors, publishers and technical reviewers.

No technical reviewers were involved in this book. This book was forged through the usage of real readers. Readers like yourself that started with little knowledge of how to write program at all. Readers who can now use both Java and Selenium to include automation in their testing process.

In this case I give thanks to my early readers and guinea pigs.

This book grew from a 2 year period of working on an almost daily basis with Selenium on production test automation. So I need to thank the Channel 4 test team that worked with me during that time as we experimented with different ways of automating. Extra special thanks go to Tony Bruce and Michael Kiely, who participated in my Selenium training sessions based on this book. Their feedback led to immediate improvements and scope changes.

And a big thank you to the people who bought the beta e-book version. The early feedback and questions as they learned Selenium working through the book helped me tremendously. I was amazed at the speed of progress they made learning Selenium and found it hard to work through the later, more advanced, sections of the book fast enough to satisfy their learning needs. From this group I need to specifically thank those readers who submitted questions and comments that directly impacted the structure and content of this book: Alex Crookes, William Dutton, Gopi Ganesasundaram, Jay Gehlot, Ard-Jan Glas, Sunny Jain, Anthony Kearns, Felipe Knorr Kuhn, Richard Li, Michael McCollum, Krishnan Mahadevan, Kaushik Mahata, Nga Nguyen, Dennis Pham, Mahmoud Passikhani, Adrian Rapan, Doug Seward, Rajat Sud, Manupriya Vijay.

And most importantly...

Love and special thanks to my family:

Billie and Keeran

Chapter 1: Getting Started With The Tools

What Tools?

This chapter will introduce you to the tools we are about to install and use:

- Selenium IDE
- Java
- Eclipse
- Selenium-RC

Each tool has links to the official documentation so that you can learn more from the most authoritative sources.

Selenium IDE

The Selenium IDE is an add-on to Firefox.

At a basic level it allows us to:

- Record user actions when browsing in Firefox
- Replay recorded scripts
- Convert recorded scripts into programming languages such as Java, Ruby, and more
- Add verification and synchronisation steps to the script during the recording process

The IDE provides excellent support for writing automated test scripts in Selenium and gets better with every release. In this text we will focus on its use as an aid to writing automated test scripts in Java, rather than as a test tool in its own right.

For more information on Selenium IDE visit:

- Official documentation - http://seleniumhq.org/docs/03_selenium_ide.html
- Official Selenium IDE site - <http://seleniumhq.org/projects/ide/>

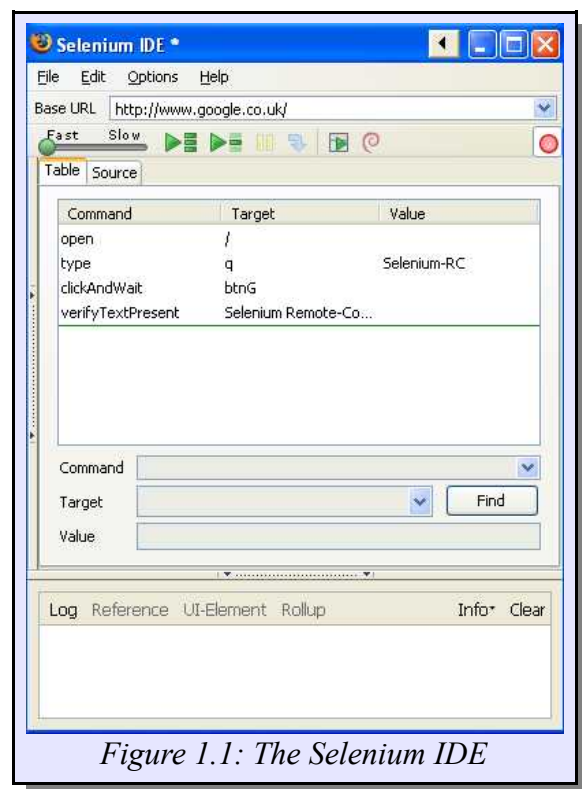


Figure 1.1: The Selenium IDE

Java

Java is a programming language created by Sun. The Java that we will be using in this text is very simple. If your work environment uses a different language e.g. Ruby, .Net, Python. Chances are that you will be able to use Selenium-RC in that language.

For more information on Java visit:

- Official Java Home page - <http://www.java.com>
- Official developers page - <http://java.sun.com>
- Java Community page - <http://theserverside.com>

Eclipse

Eclipse is a free and open-source IDE which supports many programming languages.

Out of the box it provides many useful support aids for beginning java programmers:

- wizards for code creation
- code completion
- automatically fix common coding errors

It is also simple to install, incredibly popular among developers, and has a lot of additional plugins that you can use to increase your productivity.

For more information on Eclipse visit:

- Official Home Page - <http://www.eclipse.org>
- Free Video Tutorials - <http://eclipsetutorial.sourceforge.net/index.html>

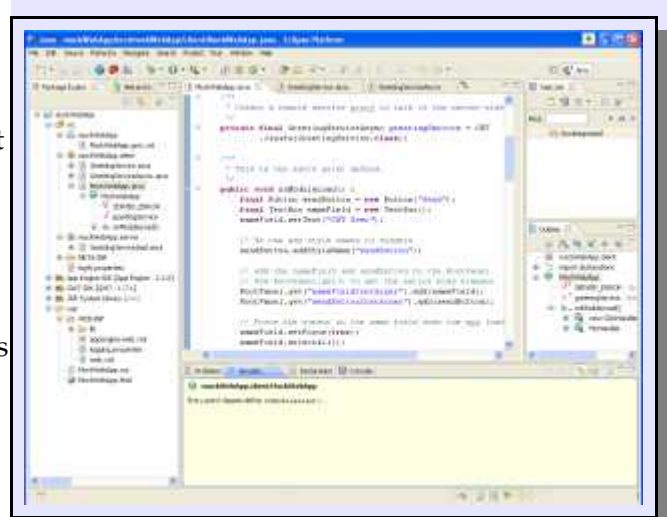


Figure 1.2: The Eclipse IDE

Selenium-RC

Selenium Remote Control is the server version of Selenium. You write your tests using a programming language and client library. Your tests issue commands which the client library sends to the server. The server then 'runs' your actions for you in the browser and reports the results back to your client.

Using Selenium-RC allows you to write Automated tests in any supported programming language.

Tests written in this way allow you to use standard programming practices to make them easy to maintain, robust and easy to collaborate on as a team.

For more information on Selenium-RC visit:

- Official Home Page - <http://seleniumhq.org/projects/remote-control/>
- Official Documentation - http://seleniumhq.org/docs/05_selenium_rc.html

Chapter 2: Selenium IDE Basics

Install Firefox

Because the Selenium IDE is a Firefox add-on we first have to download and install Firefox.

Visit <http://www.mozilla.com/firefox> and download and install the most recent version of Firefox.

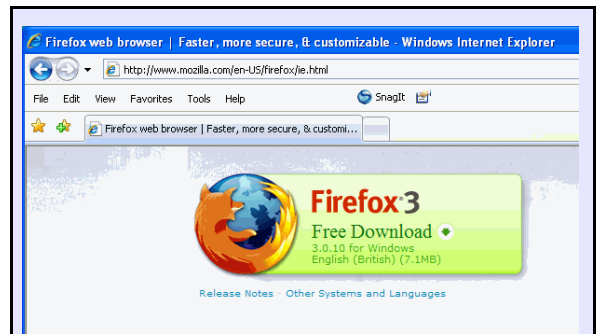


Figure 2.1: Download and Install Firefox

Install Selenium-IDE

We download and install the Selenium-IDE from within Firefox itself.

So start the Firefox browser.

Visit <http://seleniumhq.org/download/> and click on the Download link for the Selenium IDE.



Figure 2.2 : Download page and link for the IDE

Follow the series of dialogs that walk you through the install process, as illustrated below.

First you have to acknowledge that the add-on is safe to install.

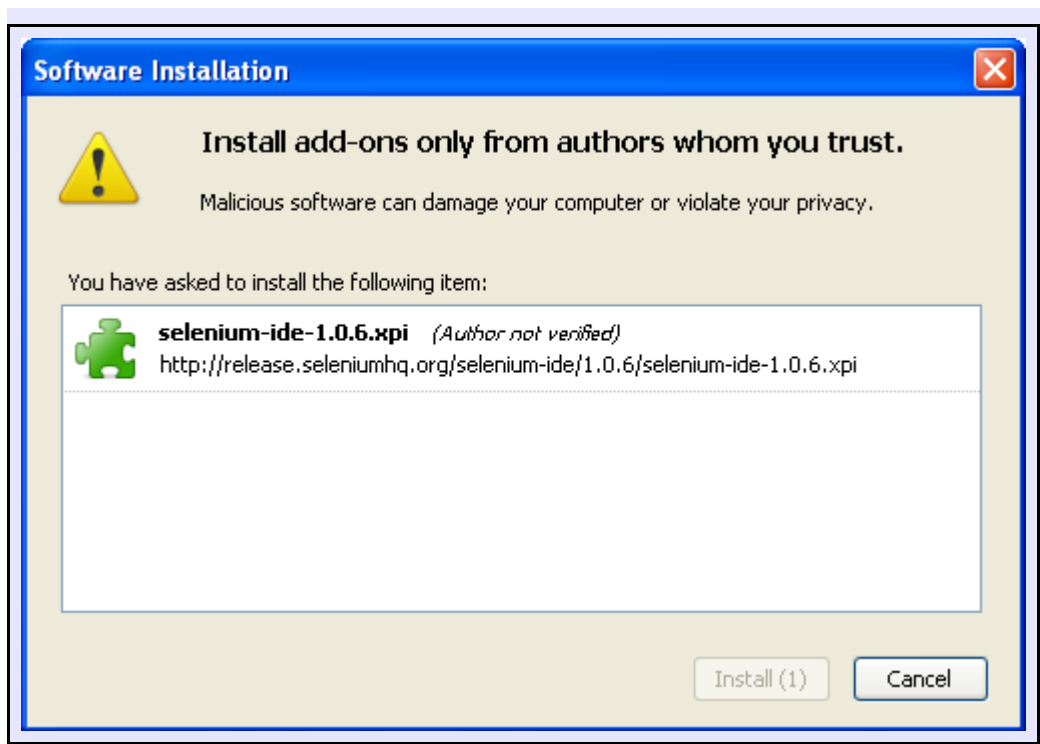


Figure 2.3 : Trust the Install of the Selenium IDE

Then after it has installed you will be prompted to restart Firefox, which you do by clicking the “Restart Firefox” button on the final Add-ons dialog.

When Firefox restarts, you should now have a new entry in your Tools menu for the Selenium-IDE.

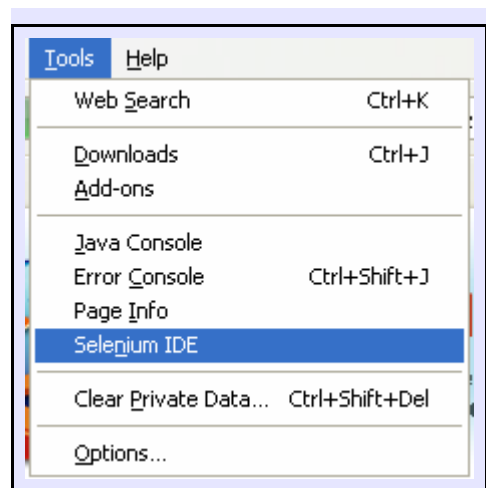


Figure 2.4: Start The Selenium-IDE

Capture Play Back – Recording a Script with the IDE

The Selenium IDE supports capture playback of test scripts by recording the actions you take when browsing web sites, and then replaying these in the browser.

So start the IDE by clicking on the “Selenium IDE” entry in the Tools Menu.

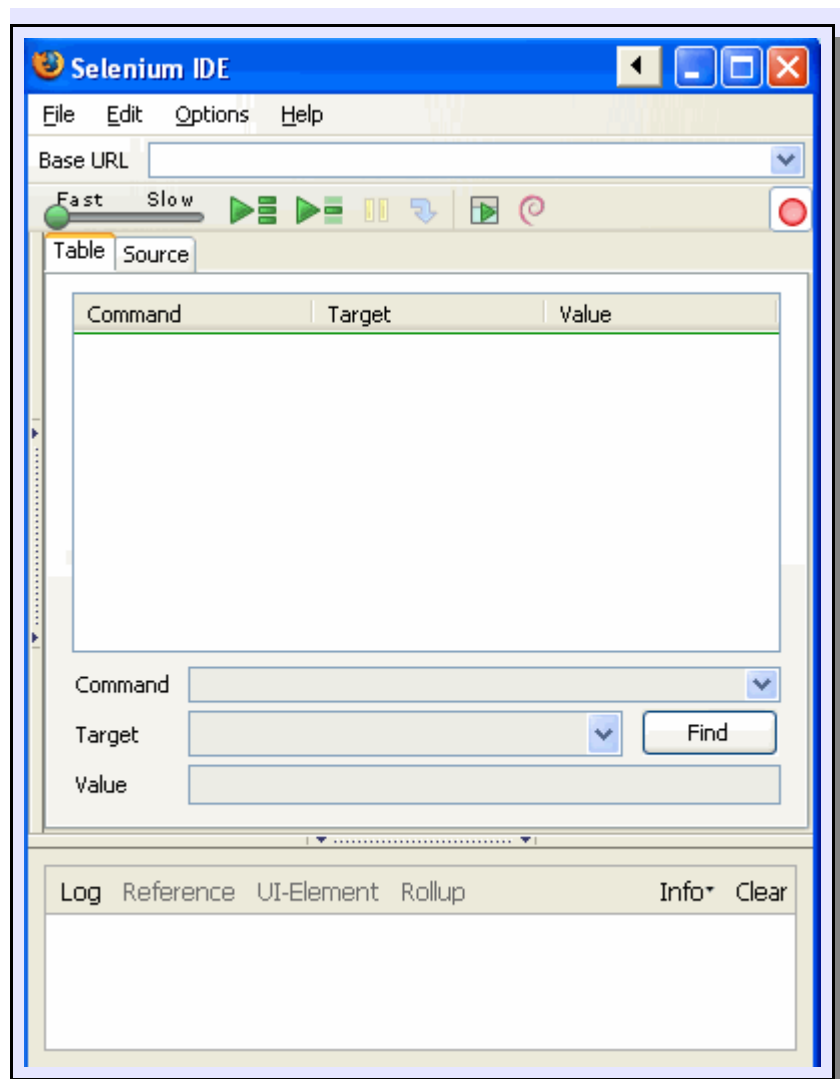


Figure 2.5 : Basic Selenium IDE

Hover the mouse over the recording button to check the recording status. If it is not recording then press the button to start recording your actions in Firefox.

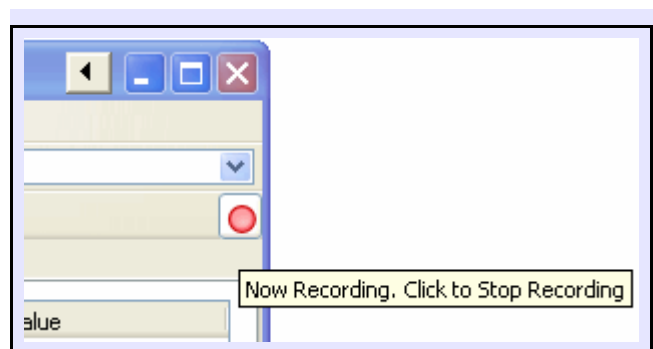


Figure 2.6: Now Recording in the IDE

Now perform the following actions in Firefox:

1. type <http://www.google.com> in the browser's URL field to visit Google.

2. Perform a search for “Selenium-RC”

You should see that the IDE has tracked your actions and set the “Base URL” field and created the necessary commands to repeat that script in Firefox.

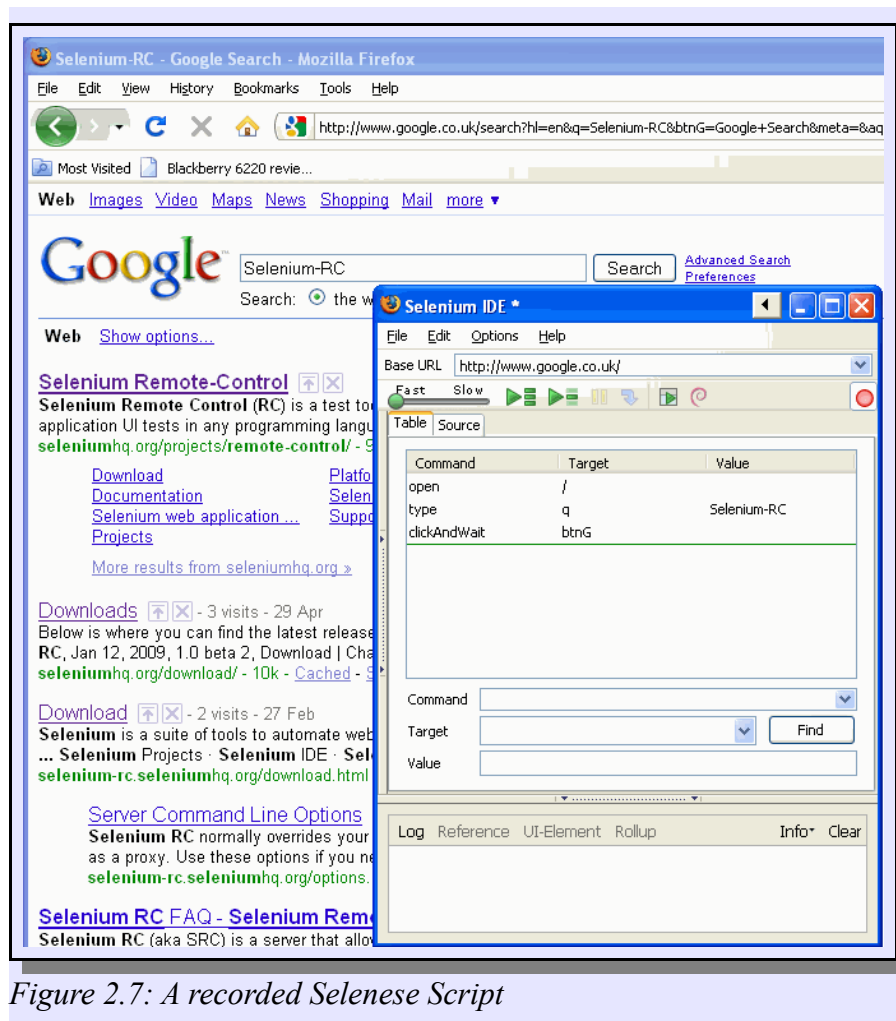


Figure 2.7: A recorded Selenese Script

Press the red recording button to stop the IDE recording any more actions. Then press the “Play Current Test Case” button to have the IDE replay your actions in the browser.



Figure 2.8: Play

You should have seen your actions repeated very quickly in Firefox. If so, then congratulations, you just recorded your first automated test script through the IDE.

Command, Target and Value

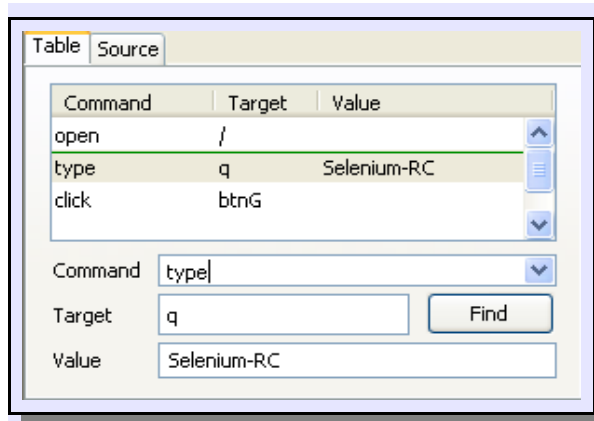


Figure 2.9: Edit any of the commands by selecting them in the table



Figure 2.10: Select Any of the Commands from the Command DropDown list

You can edit any of the recorded commands in the IDE.

Simply click on any of the commands in the table and you will see that the Command, Target and Value fields are filled in with the values on the table.

You can then use the drop down for “Command” to choose any of the commands available. When you do so, you will also see documentation about that command in the Reference pane.

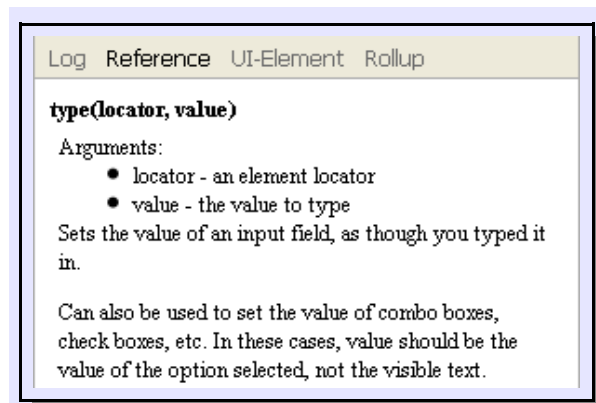


Figure 2.11: Learn what commands do in the Reference pane

Save The Test

Save this test to the disk for future use (in “Chapter 4 Install and Run Selenium-RC”).

Use the “File \ Save Test Case As ...” menu and save it somewhere you can find it again. I named mine “google_for_selenium_rc.htm”.

Further reading

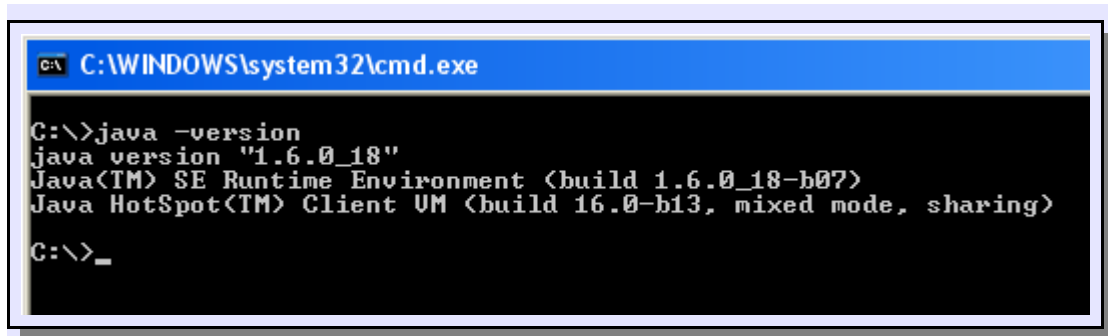
I won't really be covering Selenium-IDE in any depth in this book. So if you want more information you should read the following links:

- official documentation - http://seleniumhq.org/docs/03_selenium_ide.html
- Adam Goucher's posts on Selenium IDE - <http://adam.goucher.ca/?s=Selenium+IDE>
 - particularly the plugin posts - <http://adam.goucher.ca/?s=The+Selenium-IDE+1.x+plugin+API>
- Selenium IDE Extensions - <http://wiki.openqa.org/display/SIDE/Contributed+Extensions+and+Formats>
- The Firefox plugins search may reveal some plugins for Selenium IDE <https://addons.mozilla.org/en-US/firefox/search/?q=selenium>
- A lot of video tutorials for Selenium IDE get added to You Tube - http://www.youtube.com/results?search_query=selenium+ide

Chapter 3: Install Java

You can check if you have Java installed by:

- visiting <http://java.com> and using the “Do I have Java?” check <http://java.com/en/download/installed.jsp>
- running the command “java -version” from the command line

A screenshot of a Windows command prompt window. The title bar at the top is blue and contains the text 'C:\WINDOWS\system32\cmd.exe'. The command prompt itself has a black background with white text. The text shows the command 'C:\>java -version' being entered, followed by the output: 'java version "1.6.0_18"', 'Java(TM) SE Runtime Environment (build 1.6.0_18-b07)', and 'Java HotSpot(TM) Client VM (build 16.0-b13, mixed mode, sharing)'. The prompt ends with 'C:\>_'.

```
C:\>java -version
java version "1.6.0_18"
Java(TM) SE Runtime Environment (build 1.6.0_18-b07)
Java HotSpot(TM) Client VM (build 16.0-b13, mixed mode, sharing)
C:\>_
```

Figure 3.1 : Running “java -version” from the command line

If you don't have Java installed then visit <http://java.com> and follow the Download links to install it on your machine.

Chapter 4: Install and Run Selenium-RC

Before we can actually write any tests, we install Selenium-RC. "Install" is really the wrong word when all we will do is download and extract the Selenium-RC archive to a directory.

Step One - Download & Install Selenium-RC

Download Selenium RC from the [Selenium download page](http://seleniumhq.org/download/). (<http://seleniumhq.org/download/>)



Figure 4.1 : Download Selenium RC from SeleniumHQ

I saved it to c:\selenium

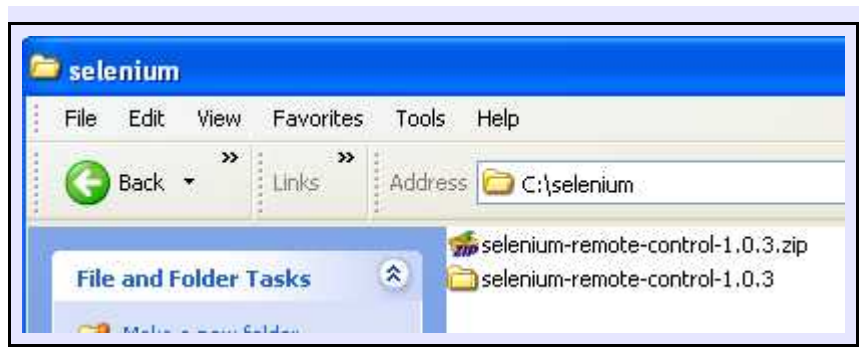


Figure 4.2 : Unarchive the saved distribution into a directory

Then I unarchived this into a folder of the same name as the the archive using an unarchiver (I use [IZArc](http://www.izarc.org/) <http://www.izarc.org/>).

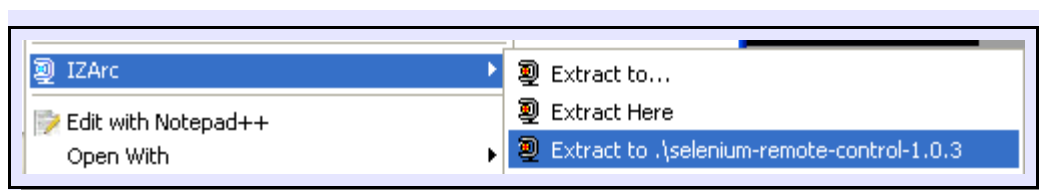


Figure 4.3 : Extracting to a folder with same name as archive

So now I have a folder with the Selenium files:

- C:\selenium\selenium-remote-control-1.0.3

Overview of the Contents of the Selenium Archive

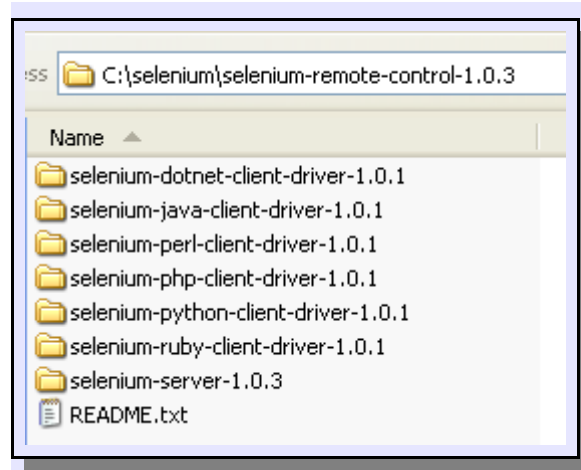


Figure 4.4 : Contents of Selenium RC

There are two main folder types here:

- specific language client drivers
- selenium server

Each of the supported languages has a client driver. You can see from the list of folders that the Selenium distribution supports:

- Microsoft .Net
- Java
- Perl
- PHP
- Python
- Ruby

The client drivers are what we use in our tests to communicate with Selenium-RC. We use the client drivers because our tests act as clients, which communicate with the server, in order to execute commands through the browser.

The client drivers contain the source code, tests, documentation and libraries that we need to work with the Selenium tests.

The selenium server directory has the documentation, source code, tests and executables for working with the Selenium server.

We will examine the contents of the archive in more detail in future sections but this high level overview should give you enough of a basic understanding to let us continue with the tutorial.

Step Two - Run Selenium-RC

I have the "[Open Command Window Here](#)" Microsoft [Powertoy for XP](#) installed. I recommend you install this very helpful tool, if you use XP. In Vista this functionality is built in and you can browse to the folder in Windows Explorer, hold down the Shift key, right-click on the folder and choose "Open Command Window Here".

- <http://download.microsoft.com/download/whistler/Install/2/WXP/EN-US/CmdHerePowertoySetup.exe>
- <http://www.microsoft.com/windowsxp/Downloads/powertoys/Xppowertoys.msp>

Note:

If you don't have "open command window here" installed then you need to get a command window open and type:

```
cd c:\selenium\selenium-remote-control-1.0.3\selenium-server-1.0.3
```

Hint: Try pressing 'tab' as you do this to automatically complete folder details.

So I right click on the "Selenium-server-1.0.3" folder and choose "Open Command Window Here" from the context menu.

Note:

I also use the open-source Console (<http://sourceforge.net/projects/console/>) to help me as I can have multiple console windows open in a single application.

I give the command window a new title so I can find it easily by issuing the command "Title SELENIUM SERVER".

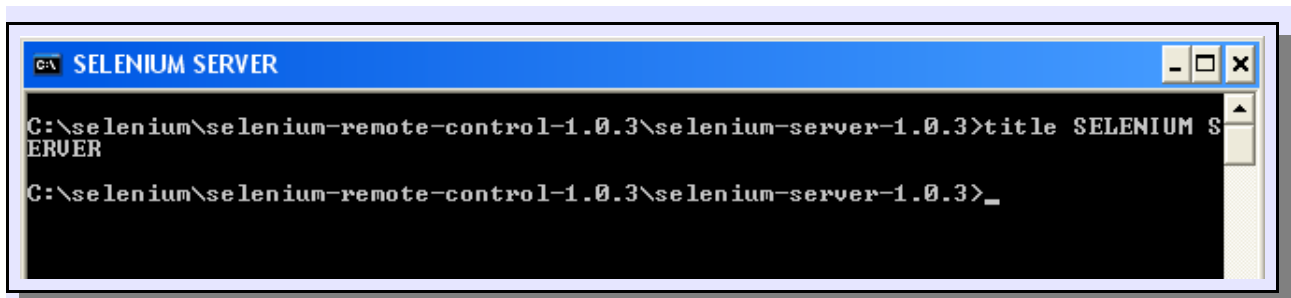
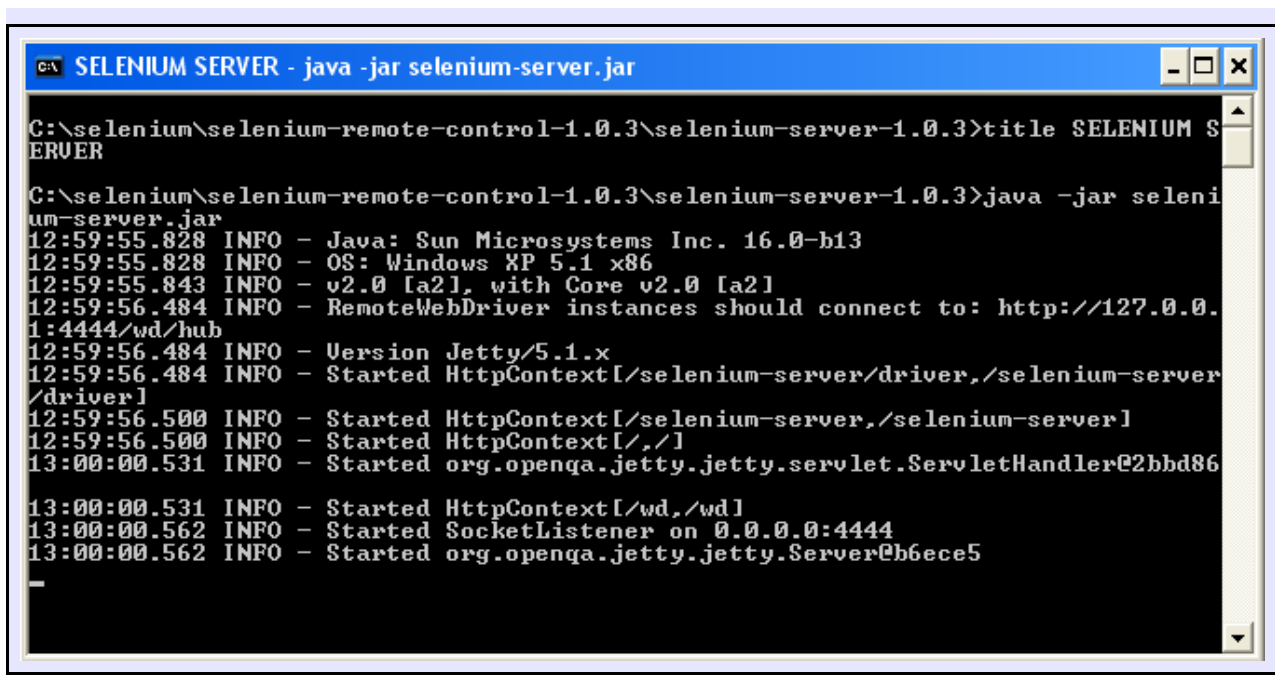


Figure 4.5 : An easy to identify Selenium Server command window

This makes it much easier to find when I Alt-Tab or look for it on the taskbar.

Now we run the selenium server by typing the command "java -jar selenium-server.jar"



```
C:\selenium\selenium-remote-control-1.0.3\selenium-server-1.0.3>title SELENIUM S
SERVER

C:\selenium\selenium-remote-control-1.0.3\selenium-server-1.0.3>java -jar seleni
um-server.jar
12:59:55.828 INFO - Java: Sun Microsystems Inc. 16.0-b13
12:59:55.828 INFO - OS: Windows XP 5.1 x86
12:59:55.843 INFO - v2.0 [a2], with Core v2.0 [a2]
12:59:56.484 INFO - RemoteWebDriver instances should connect to: http://127.0.0.
1:4444/wd/hub
12:59:56.484 INFO - Version Jetty/5.1.x
12:59:56.484 INFO - Started HttpContext[/selenium-server/driver,/selenium-server
/driver]
12:59:56.500 INFO - Started HttpContext[/selenium-server,/selenium-server]
12:59:56.500 INFO - Started HttpContext[/,/]
13:00:00.531 INFO - Started org.openqa.jetty.jetty.servlet.ServletHandler@2bbd86
13:00:00.531 INFO - Started HttpContext[/wd,/wd]
13:00:00.562 INFO - Started SocketListener on 0.0.0.0:4444
13:00:00.562 INFO - Started org.openqa.jetty.jetty.Server@b6ce5
```

Figure 4.6 : Start up messages from the Selenium Server

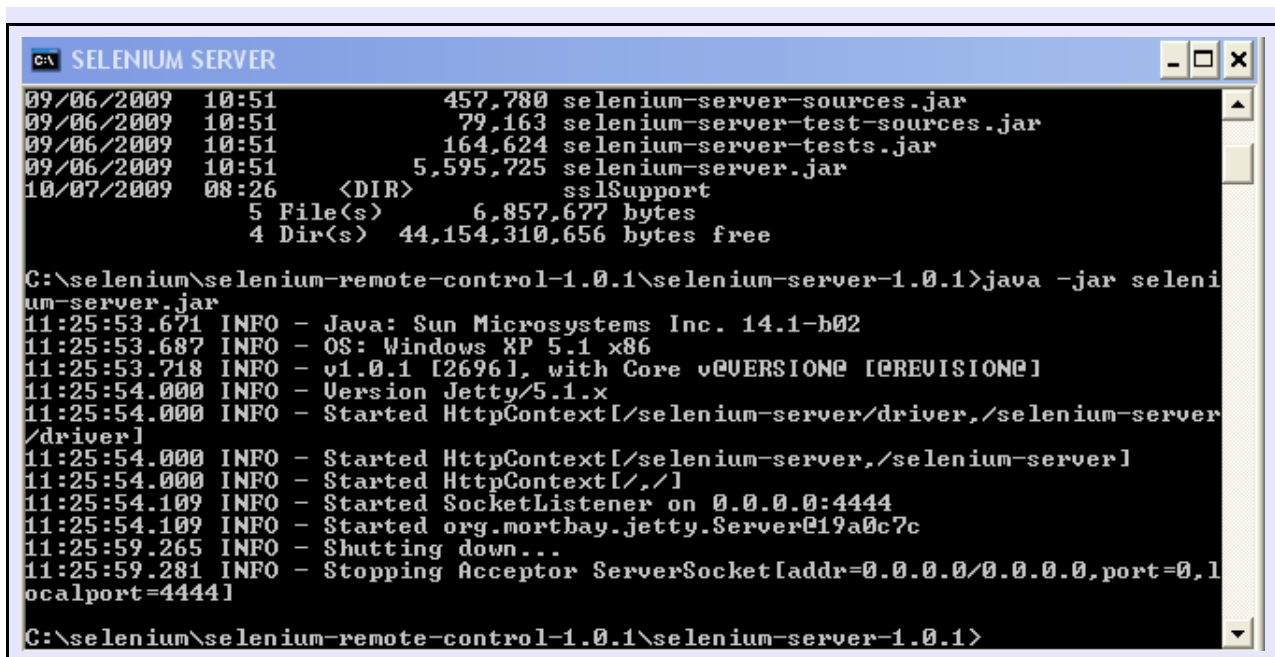
If you didn't see a screen like the above then you may need to install java.

Done - you have just installed and started the Selenium-RC server.

Step Three – Stop the Selenium Server

From the command line

Since you have started the Selenium server in a DOS window you can simply use “Ctrl+c” to stop it.



```
C:\selenium\selenium-remote-control-1.0.1\selenium-server-1.0.1>java -jar seleni
um-server.jar
11:25:53.671 INFO - Java: Sun Microsystems Inc. 14.1-b02
11:25:53.687 INFO - OS: Windows XP 5.1 x86
11:25:53.718 INFO - v1.0.1 [2696], with Core v@VERSION@ [PREVIOUS@]
11:25:54.000 INFO - Version Jetty/5.1.x
11:25:54.000 INFO - Started HttpContext[/selenium-server/driver,/selenium-server
/driver]
11:25:54.000 INFO - Started HttpContext[/selenium-server,/selenium-server]
11:25:54.000 INFO - Started HttpContext[/,/]
11:25:54.109 INFO - Started SocketListener on 0.0.0.0:4444
11:25:54.109 INFO - Started org.mortbay.jetty.Server@19a0c7c
11:25:59.265 INFO - Shutting down...
11:25:59.281 INFO - Stopping Acceptor ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,1
ocalport=4444]
C:\selenium\selenium-remote-control-1.0.1\selenium-server-1.0.1>
```

Figure 4.7 : After Ctrl+c the server shuts down

From a browser URL

Since selenium server operates a web server under Jetty it can receive commands via http through browsers. Issue the following to shutdown the server:

- <http://localhost:4444/selenium-server/driver/?cmd=shutDownSeleniumServer>

If you wanted to leave the server running and just close down the current Selenium session, you could issue the following command in a browser.

- <http://localhost:4444/selenium-server/driver/?cmd=shutDown>

Running an IDE Generated Test in Different Browsers

Although the IDE script we recorded in Chapter 2 was recorded in Firefox, and only Firefox has the IDE, we can run IDE recorded scripts in other browsers. This requires the use of Suites, and Selenium-RC.

Note: I do not recommend this approach to using Selenium. I only include it for completeness.

I have had issues getting this to work with Internet Explorer on Selenium 1.0.3 using *iexplore, *iehta, or *piiexplore. Other browser codes seem to work, but when using IE8 the Selenium server seems to throw an HTML suite exception. If this happens to you, and you consider it important, then you might need to drop down to an earlier version of Selenium-RC. Stick with the book though and it shouldn't be an issue as you will never use this method to run the tests.

http://groups.google.com/group/selenium-users/browse_thread/thread/8450ee42417806d0/3f32cadf651e57f8

You create a suite by creating an HTML file with links to the tests you want to create.

e.g suite.html

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>My Test Suite</title>
</head>
<body>
  <table cellpadding="1" cellspacing="1" border="1">
    <thead>
      <tr><td>A Suite of Selenese Tests</td></tr>
    </thead>
    <tbody>
      <tr><td><a href="google_for_selenium_rc.htm">Google For Selenium
RC</a></td></tr>
      <tr><td><a href="google_for_testing.htm">Google For Testing</a></td></tr>
      <tr><td><a href="visit_google_videos.htm">Visit Google Videos</a></td></tr>
    </tbody>
  </table>
</body>
</html>
```

For the above example you would have to have three recorded IDE scripts and they would all be saved in the same directory as the suite file.

You don't really need the header:

```

<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>My Test Suite</title>
</head>

```

or the cell padding `cellpadding="1" cellspacing="1" border="1"`

```

<html>
<body>
  <table>
    <thead>
      <tr><td>A Suite of Selenese Tests</td></tr>
    </thead>
    <tbody>
      <tr><td><a href="google_for_selenium_rc.htm">Google For Selenium
RC</a></td></tr>
      <tr><td><a href="google_for_testing.htm">Google For Testing</a></td></tr>
      <tr><td><a href="visit_google_videos.htm">Visit Google Videos</a></td></tr>
    </tbody>
  </table>
</body>
</html>

```

But since will typically create a test by copy and pasting, making it well typed and formatted may stand you in good stead for future versions of Selenium.)

Since you have Selenium-RC installed you can use the following command line syntax to run a suite of Selenese scripts in other browsers.

```

java -jar selenium-server.jar -htmlSuite "<insert_browser_name_here"
"<insert_root_domain_here" "<insert_full_path_to_suite_here>"
"<insert_a_full_path_to_store_reports_here"

```

e.g. To run it in Google Chrome

```

java -jar selenium-server.jar -htmlSuite "*googlechrome" "http://www.google.com"
"c:\ide_scripts\suite.htm" "c:\ide_scripts\results.htm"

```

To execute the above command you would need to run the command from the directory where selenium-server.jar is located, and your suite would need to be stored in [c:\ide scripts](#)

I have found that on windows I need to store the suite in the same folder as the test scripts and that I have to pass the full path for the suite file.

When you run the above command you will see the Selenese suite runner with the suite of tests in the right and it will run through and report the results.

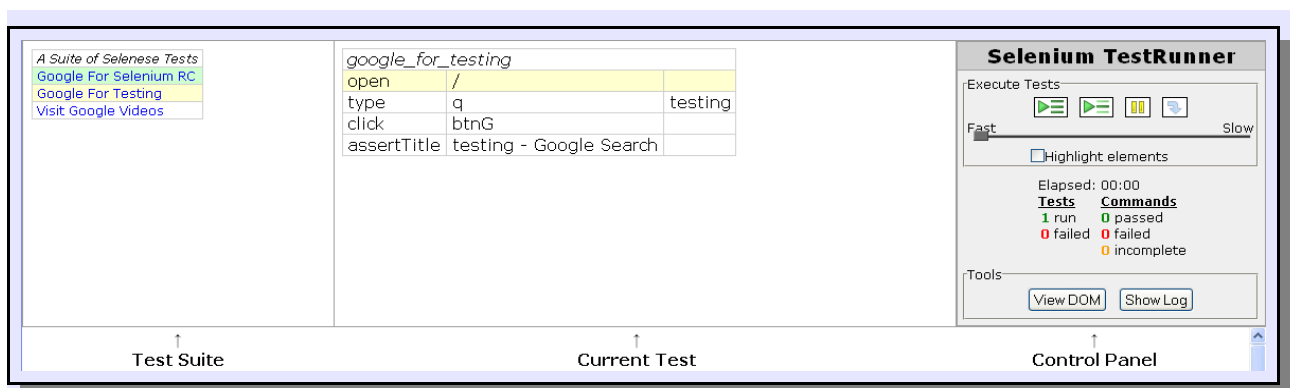


Figure 4.8: Running a suite of tests

Running the tests will result in the generation of an HTML file with the results using the path that you used in the command line.

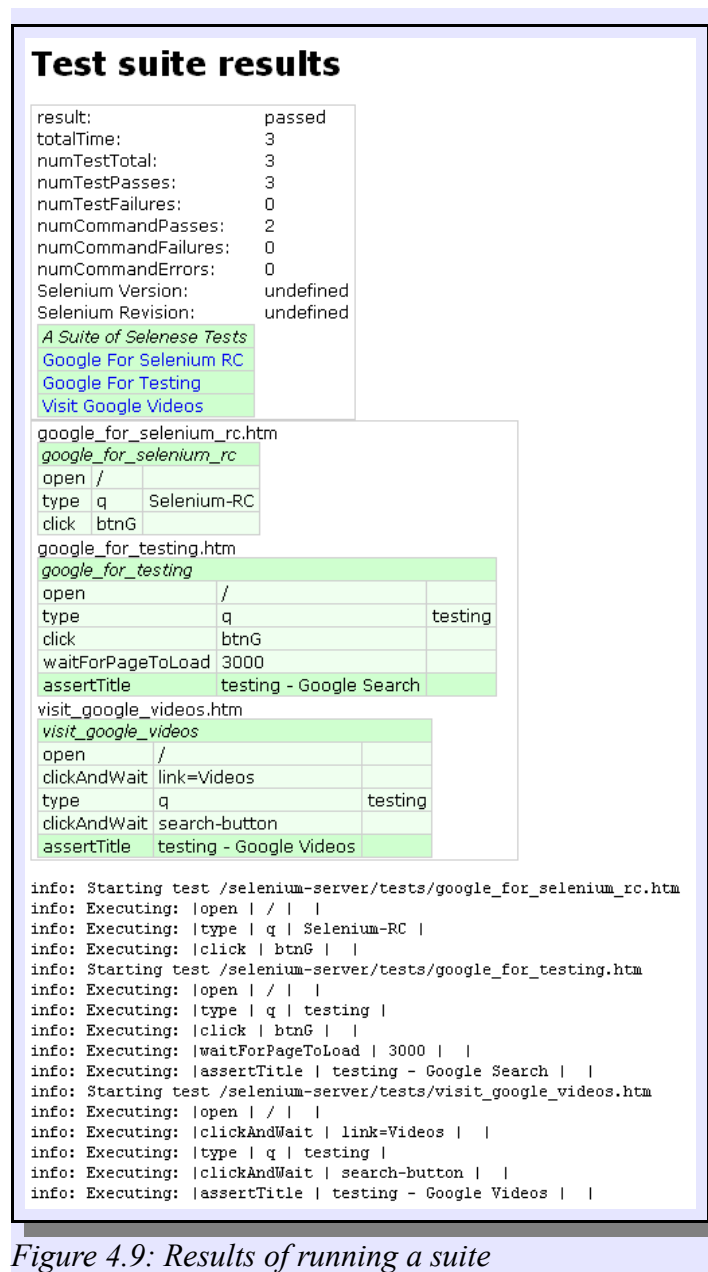


Figure 4.9: Results of running a suite

Try this after following the instructions on installing Java and Selenium-RC.

End Notes

There are a lot more functions available in the IDE and you could use it as a tool to manage an entire suite of test cases. But we are aiming to manage our test cases as code and in later sections we use the IDE to create our basic test cases and as a support tool for our automation.

Chapter 5: The Eclipse IDE

First - Install Eclipse

Visit <http://www.eclipse.org/> and click on the big download button.



Figure 5.1: Download Eclipse

Then download the “Eclipse IDE for Java Developers”



Figure 5.2: Download the appropriate IDE

Select a mirror from the list.



Figure 5.3: Choose the location to download from

Download the zip file and save it to a folder.

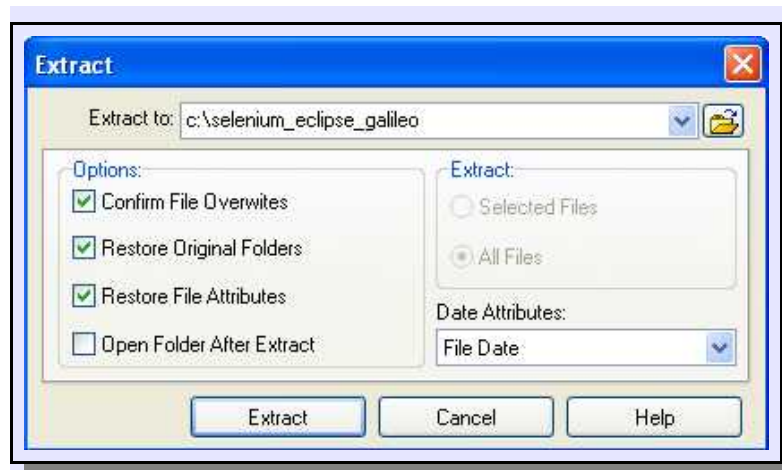


Figure 5.4: Extract Eclipse to the folder of your choice

Then unarchive this into a folder using your favourite zip unarchiver. I right click on the file and choose the unarchive option from the context menu to extract the contents of the archive to a folder – I chose “c:\selenium_eclipse_galileo” (at the time of writing the current version of Eclipse was named “Galileo”). If the version you installed has a different name then don't worry. Eclipse versions are backwardly compatible and improve with each release, so use the most up to date version of Eclipse.

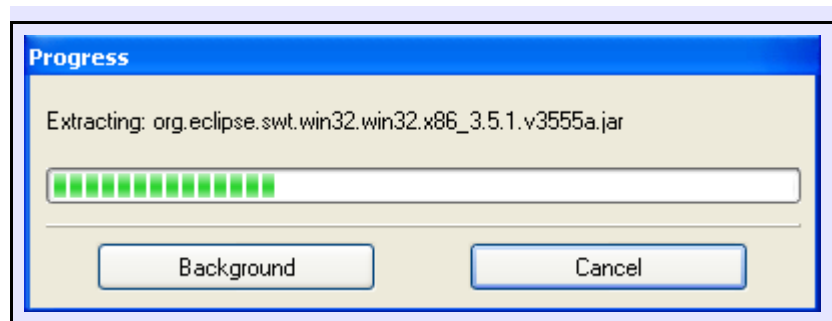


Figure 5.5: Extracting Eclipse

Second - Run Eclipse

Run Eclipse by double clicking on “eclipse.exe” in “c:\selenium_eclipse_galileo\eclipse”.

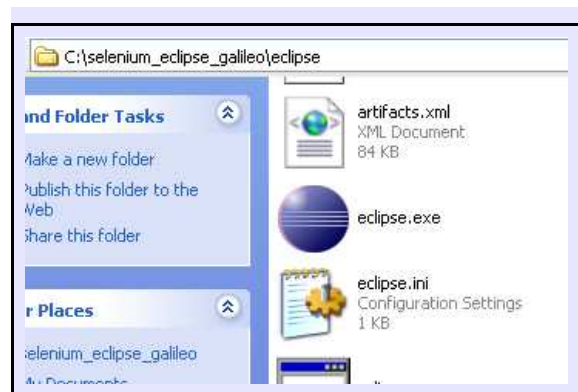


Figure 5.6: Run eclipse.exe

You will see the splash screen for Eclipse.

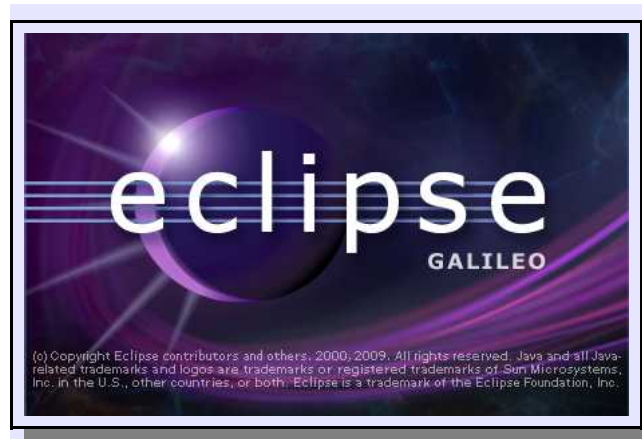


Figure 5.7: Eclipse Splash Screen

Choose a 'workspace' - where your project files will be saved. You can easily change this at a later date so choose a location that seems sensible for you. I have put it on the c:\ drive for clarity in the explanatory text and screen shots.

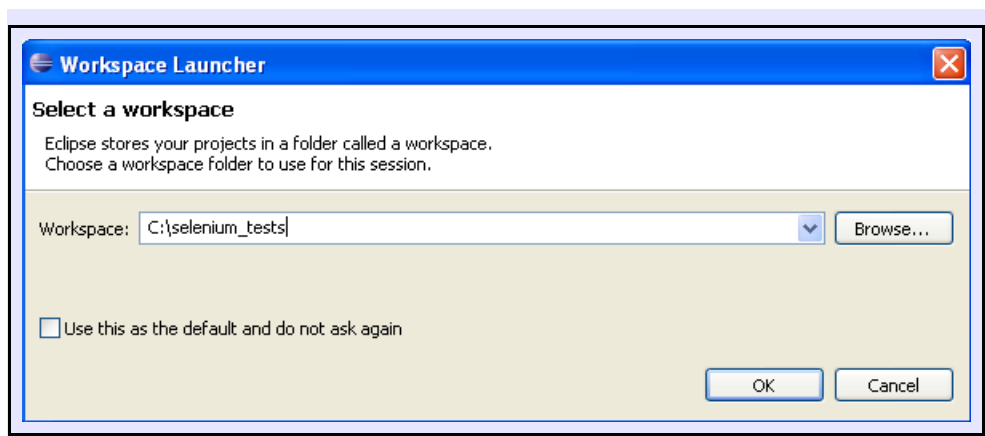


Figure 5.8: Choose a location where Eclipse will save your source code

And then you will see the Eclipse "Welcome" screen, which I always find a bit confusing.

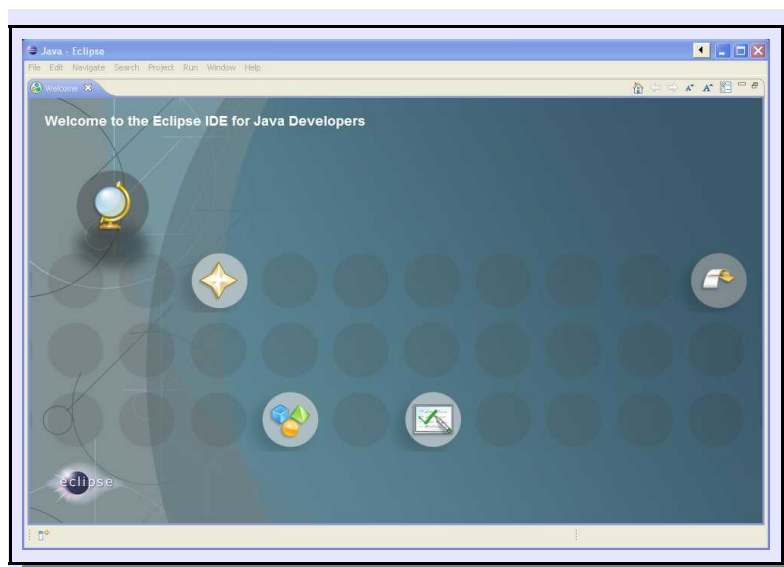


Figure 5.9: Eclipse Welcome Screen

You want to click on the curved arrow to the right which displays the text “Workbench” “Goto the Workbench” when you hover the mouse over it.

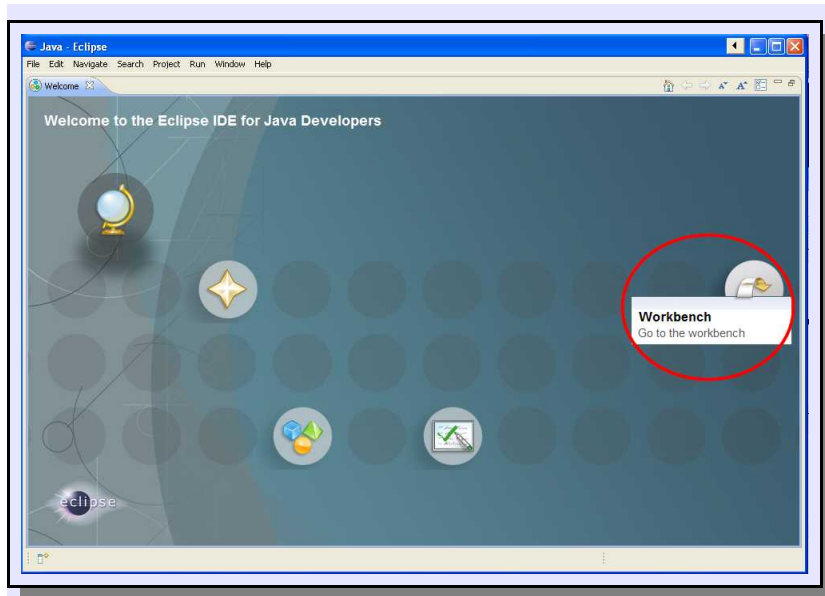


Figure 5.10: Goto Workbench Location on Welcome Screen

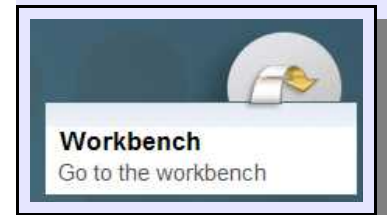


Figure 5.11: Goto Workbench

And then you will see the Eclipse workbench screen that will become very familiar and you will grow to know and love.

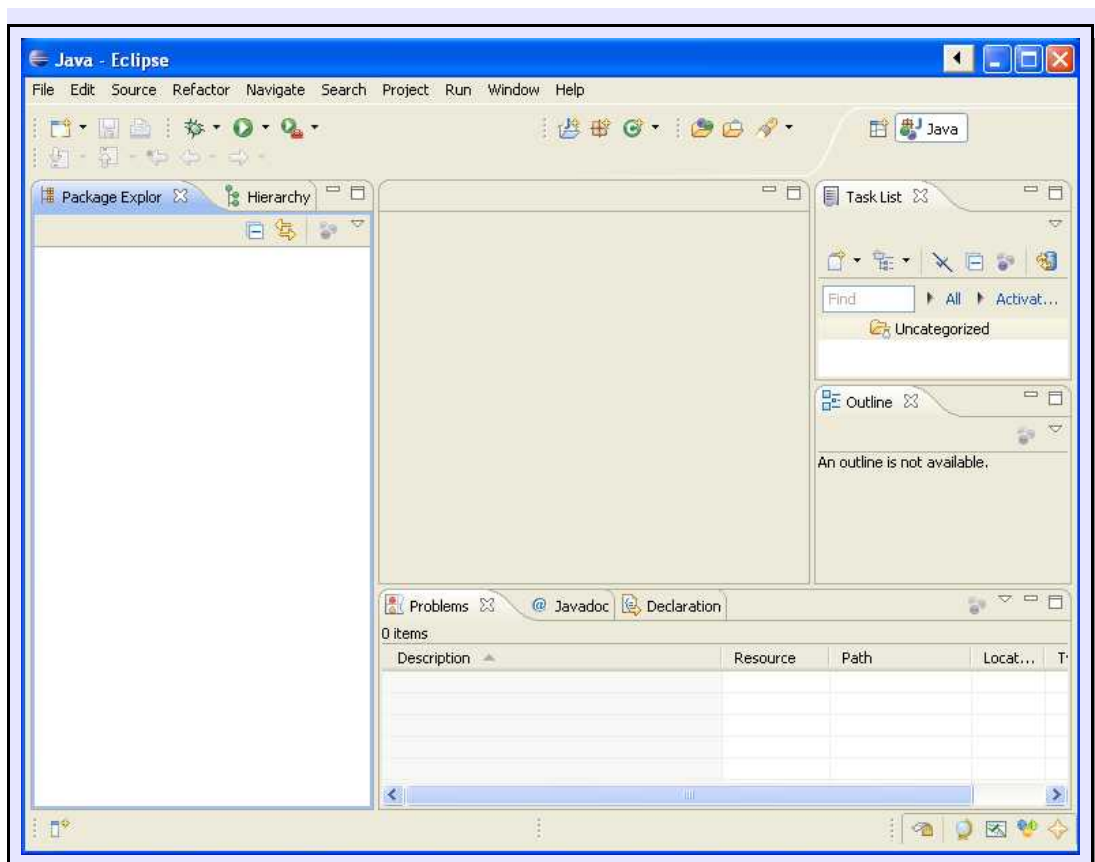


Figure 5.12: The Eclipse Workbench Screen

Third - create a new Java project

We are going to create tests in Java so to do that we need a Java project to store them in. Start by using the file menu in Eclipse to create a new Java project.

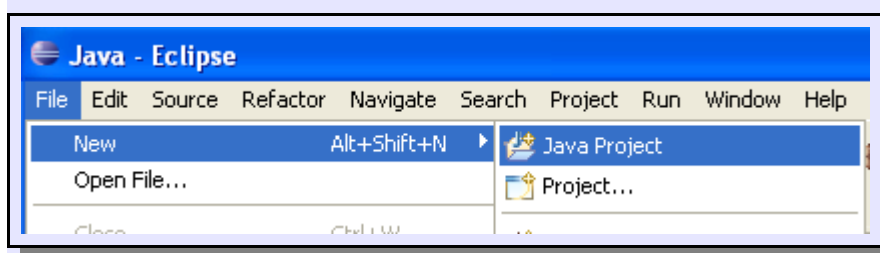


Figure 5.13: Create a New Java Project from the File Menu

Type in the name of your new project - I called mine "InitialSeleniumTests", just leave everything else on the dialog with the default entries and click 'finish'.

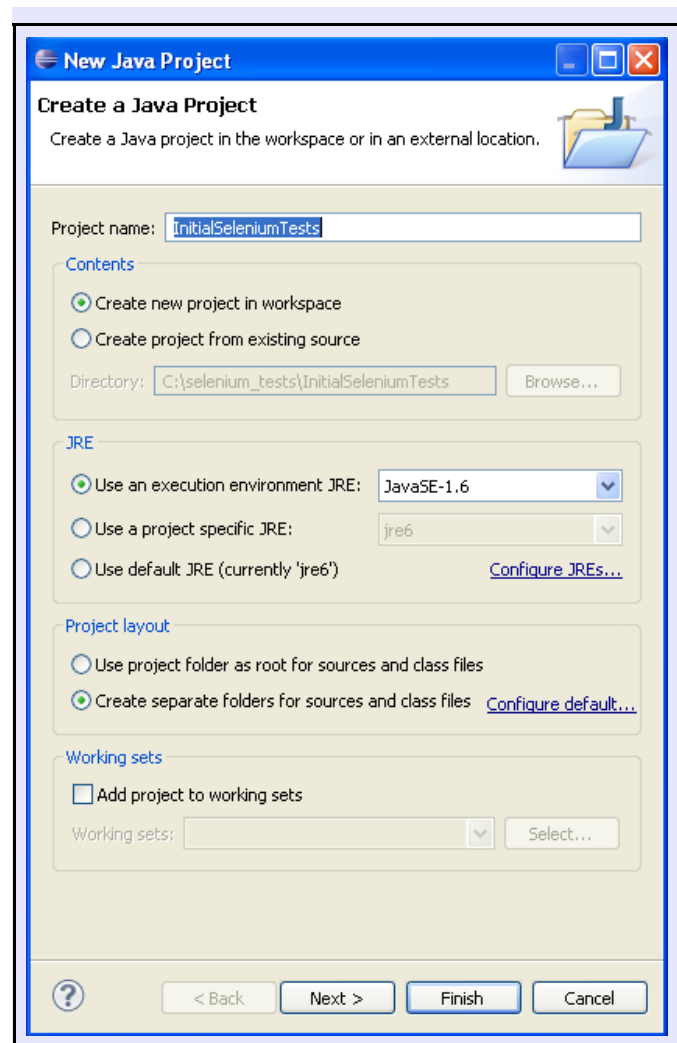


Figure 5.14: Project Named Initial Selenium Tests

This will create a folder in your workspace directory called "InitialSeleniumTests" which has the basic folders that you need when working with Java in Eclipse.

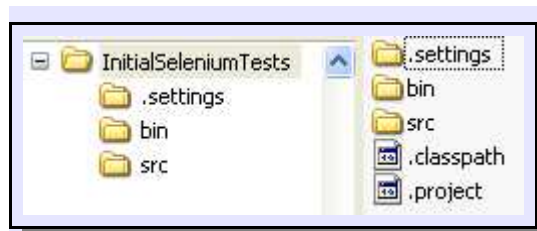


Figure 5.15: Automatically Created Files

In Eclipse, in the Package Explorer tab on the left, you will see a tree with a root node named “InitialSeleniumTests”, this is your project.

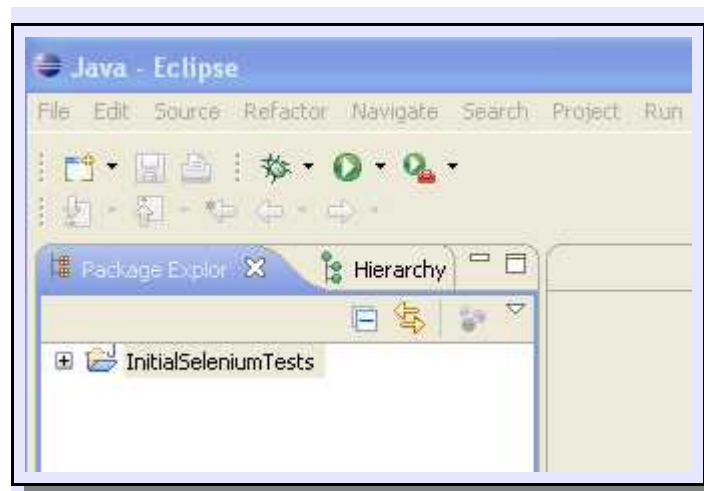


Figure 5.16: Default package view after creation

Now you are ready to start creating Selenium tests in Eclipse. We still have a few changes to make in the Eclipse configuration, but we will make those changes in the context of writing tests so that you will see and understand why we make them.

Chapter 6: Create a JUnit test Using the JUnit export from Selenium-IDE

Introduction

In this section we are going to:

- export an IDE script as a Java JUnit test
- get the exported test running in Eclipse

Export an IDE script as a JUnit test

We have already done this in the preceding section. So in the IDE load the test that you saved “google_for_selenium_rc.htm” or follow the instructions in the section “Capture Play Back – Recording a Script with the IDE” to create a test script.

Load an existing script into the IDE

Start up the Firefox browser. And click on “Selenium IDE” from the Tools menu to display the Selenium IDE.

In the IDE select “File\Open...”

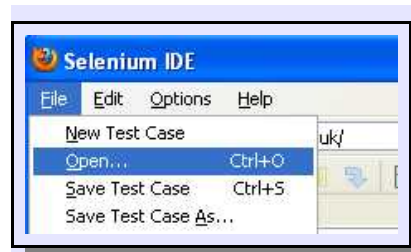


Figure 6.1: select “File\Open...”

Then navigate to the test case and Open it in the IDE. You should see the test loaded into the IDE in the table view.

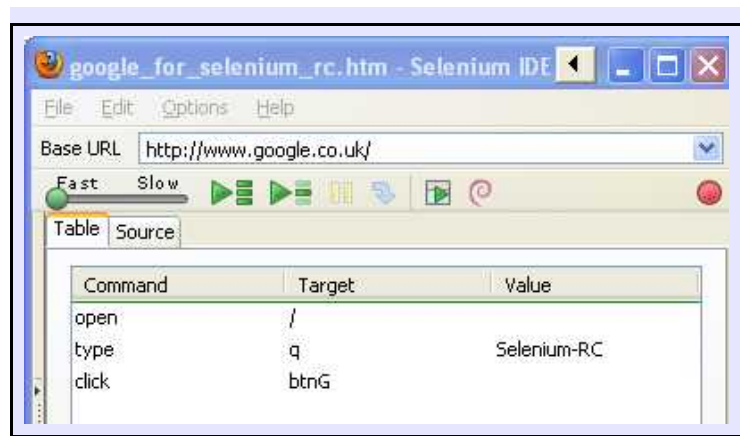


Figure 6.2: Test in table view

Change the format of the script to Java

We need to convert this test into a Java test and we do this by changing the format of the test from the options menu.

If you open the options menu from the IDE then you will see that the current format of the recorded test script is “HTML”. This is the default recording and playback option for the IDE using an Action Word scripting language called “Selenese”.

So select “Java (JUnit) – Selenium RC” from the “Format” menu.

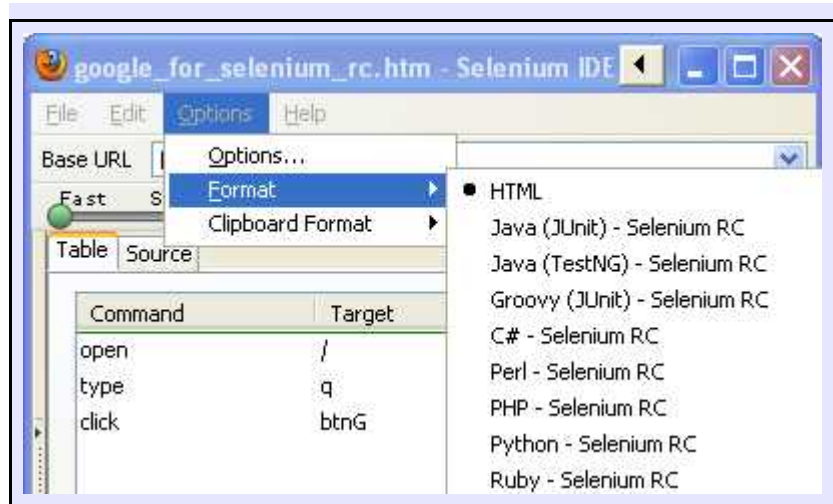


Figure 6.3: Test script formatting options

You should now see the “Source” tab selected and some Java source code displayed.

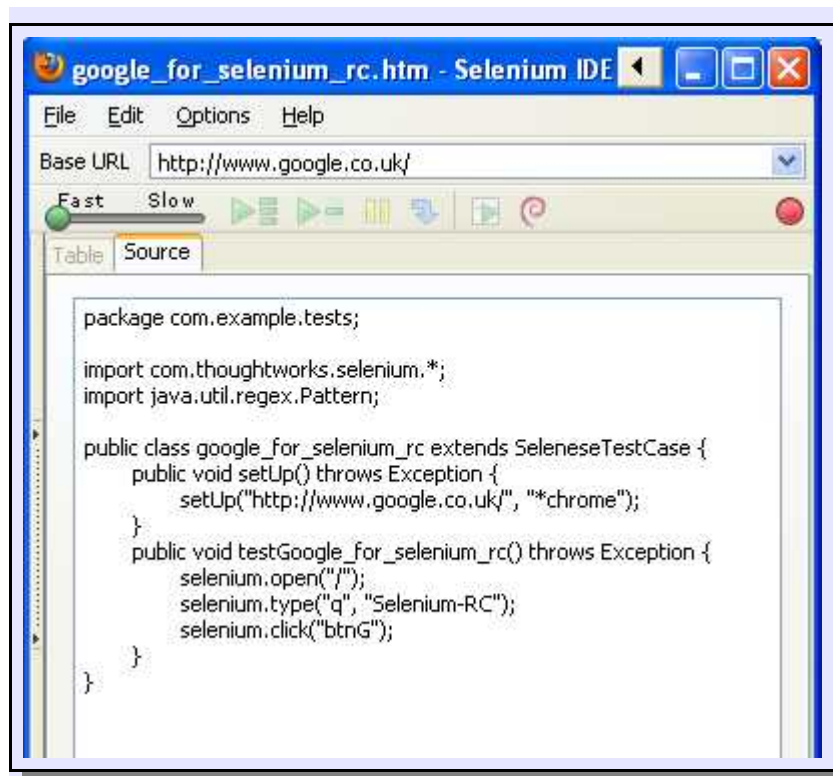


Figure 6.4: Test script displayed as a JUnit test

You don't need to understand this test code yet. We will explain it as we go through this section. But rest assure it will make sense at the end of this chapter.

So the first thing we are going to do is create a new class to store our tests in.

NOTE:

Depending on the order that you did actions in the Selenium-IDE you might find that the base url in the sourcecode view is not www.google.co.uk, but is instead

```
setUp("http://change-this-to-the-site-you-are-testing/", "*firefox");
```

If this happens, all you have to do is edit the code after you have pasted it into the Eclipse IDE so that it has the base url that you are working with. e.g.

```
setUp("http://www.google.co.uk/", "*firefox");
```

Create a New Class in Eclipse

In Java all of our 'tests' are going to be 'methods' of a Java Class.

Java is an Object Oriented programming language so all of the code is built from classes. And the only way to have the program do something is to execute a method of one of the classes.

You don't need to understand this yet and we will revisit this in a later section to explain the basics of classes, methods, objects, packages and other terminology you will encounter.

So, start Eclipse (If you haven't already got it running), and use the Menu system: File \ New \ Class

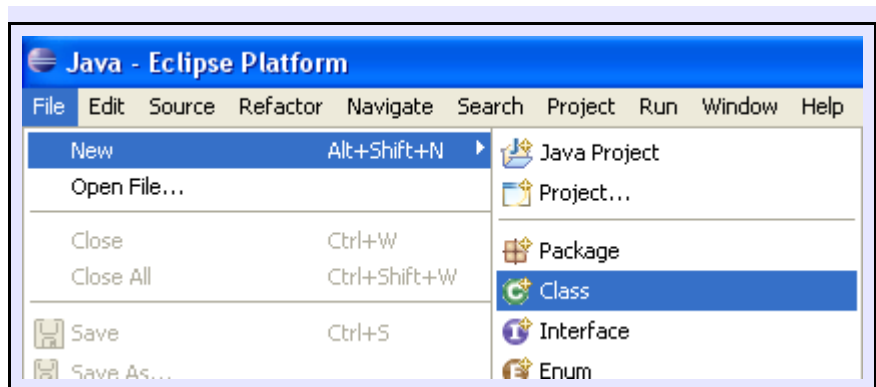


Figure 6.5: Start creating a New Class

The Eclipse "New Java Class" wizard will appear and we can use this to create our basic class - don't worry if you don't understand all the items in the dialog, just follow the instructions, you will understand everything there eventually as you become more experienced with Java. Right now, we just want to create a simple class called "MyFirstSeleniumTests".

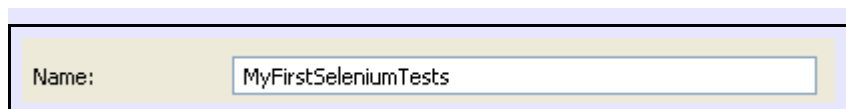


Figure 6.6: Name the class

So in the name field type "MyFirstSeleniumTests" - if you choose a different name then note that you should start the class name with an uppercase letter.

After typing in the name, did you notice that Eclipse showed you a warning message saying "The use of the default package is discouraged". Packages are how we organise our Java code so that we can reuse it and manage it effectively on a large project.

Eclipse provides plenty of warnings to help you code. This warning means that we should really type in a package name - and we will.



Figure 6.7: Warning to choose a package

Sun provide some guidance on package names (see links below), I will use "com.eviltester.seleniumtutorials" and type that into the "Package" field.

- <http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html>
- http://java.sun.com/docs/books/jls/third_edition/html/packages.html#7.7

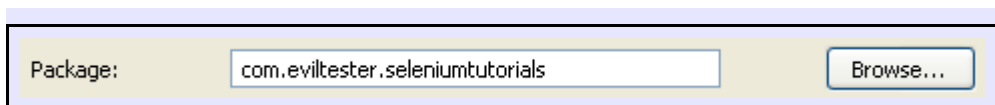


Figure 6.8: Enter a package name

The rest of the dialog can be left with the default values provided by the wizard.

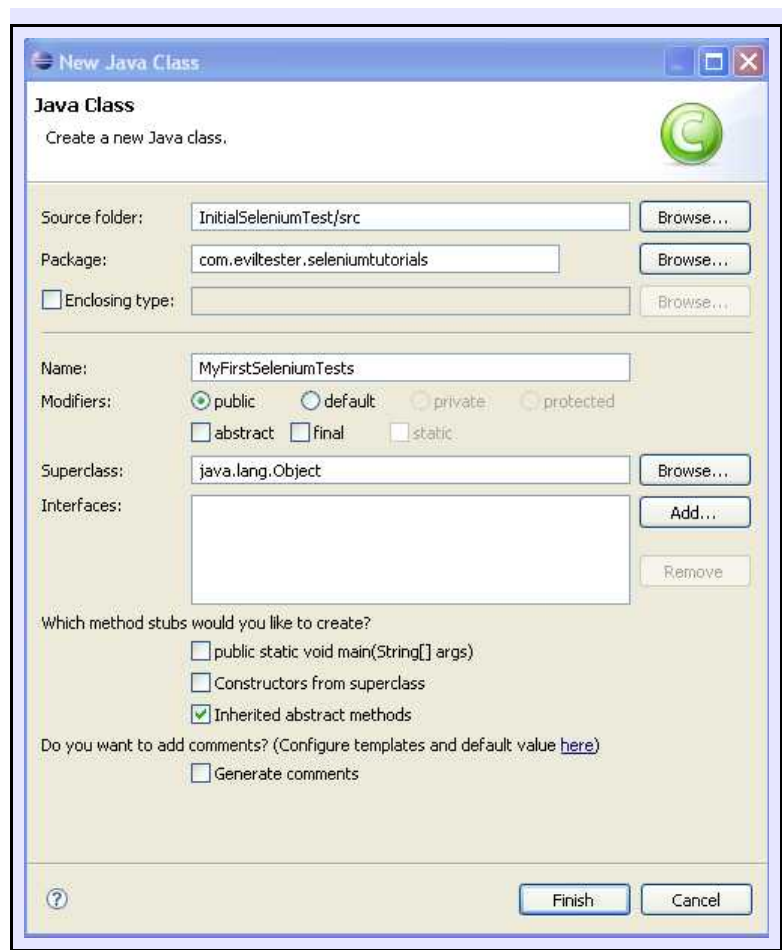


Figure 6.9: New Java Class wizard

And then press finish.

Eclipse will have created an empty class and a package for you.

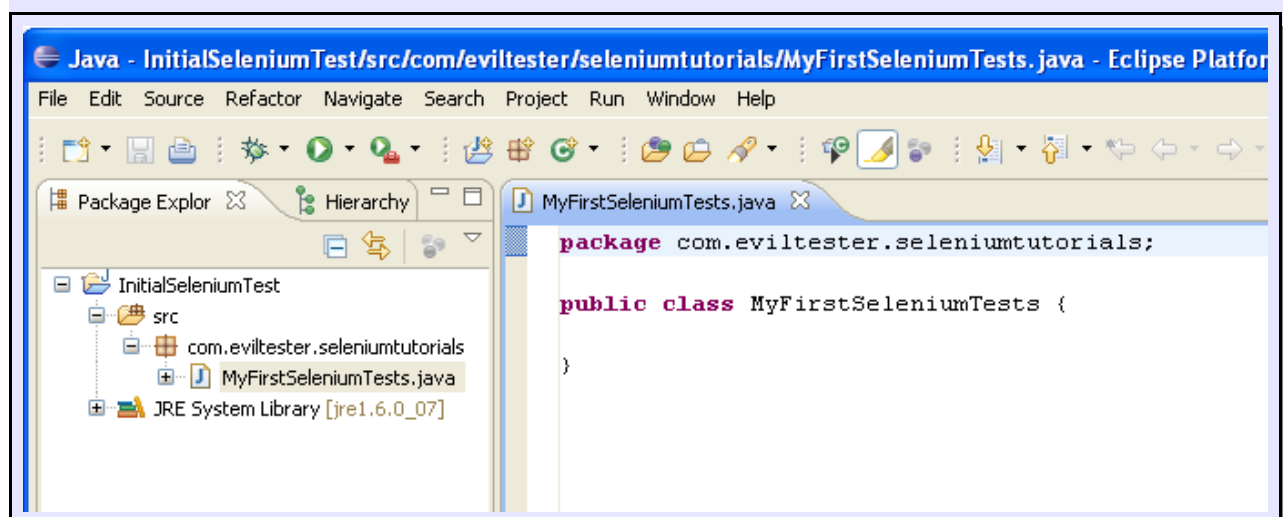


Figure 6.10: Created Class in Eclipse

Now we just have to add the code from the IDE.

Copy & Paste the code from Selenium IDE into Eclipse

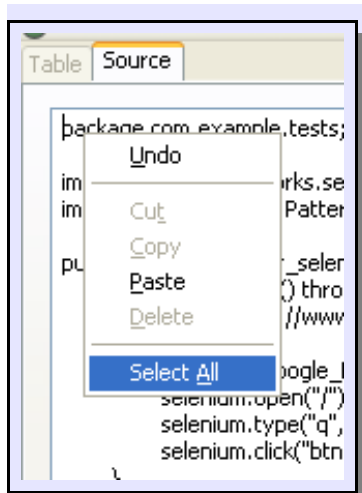


Figure 6.11: Select All

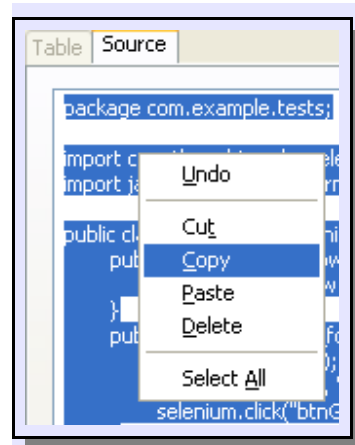


Figure 6.12: And Copy from Context Menu

Copy the code from the IDE into the clipboard.

Either by using the right click context menu to do a “select all” followed by a “copy”, or by clicking on the source code and using the keyboard and pressing Ctrl+a followed by Ctrl+c.

We can't just paste the Selenium IDE code into Eclipse because we need some of the code that the New Class Wizard created for us.

So paste the Selenium IDE code at the end of the MyFirstSeleniumTests.java code so you end up with a code listing like the one below.

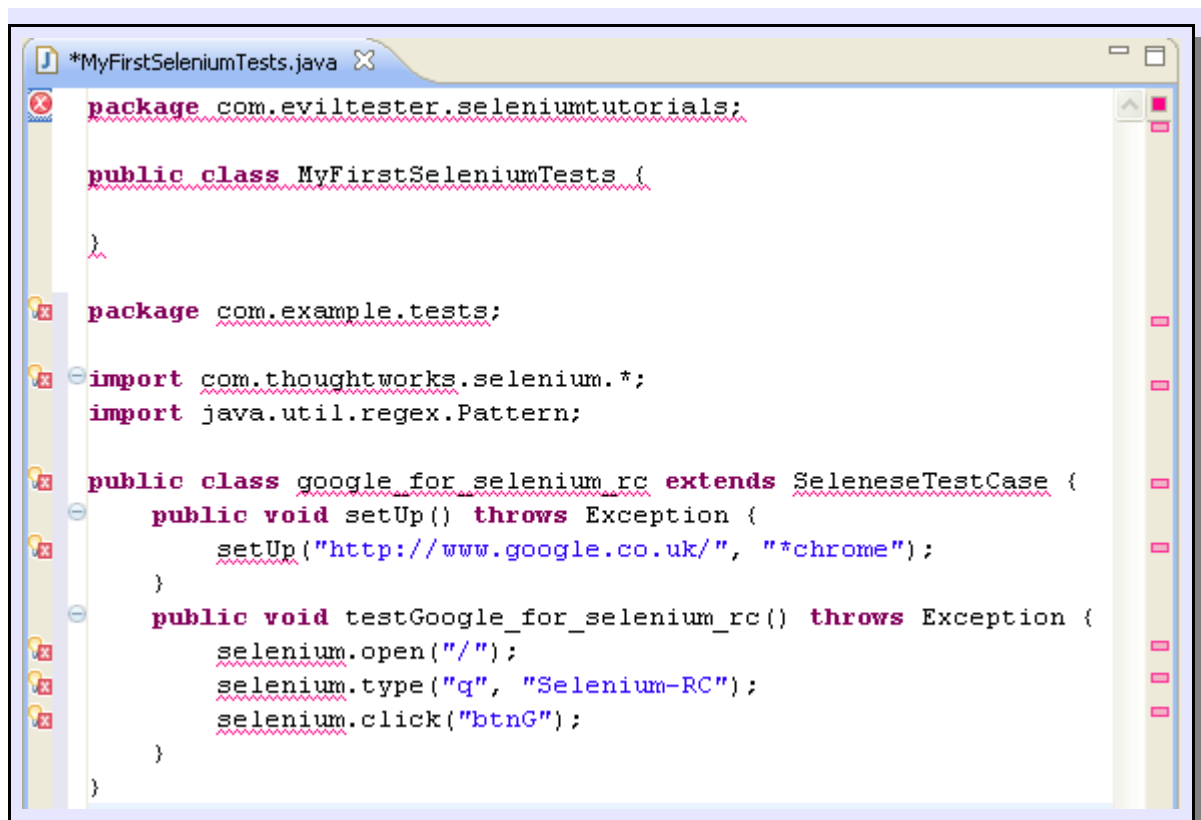


Figure 6.13: Copy and Pasted Class with lots of warning errors

As you can see there are a lot of Eclipse error symbols on the left hand side of the screen. This is because we have written invalid Java with two package declarations and two class declarations. So we will tidy that up:

- remove the line “package com.example.tests;”
 - We can only have one package declaration in each class, and it is at the top of the file.
- copy MyFirstSeleniumTests and overwrite google_for_selenium_rc
- then remove the declaration of MyFirstSeleniumTests that the wizard generated ie. “public class MyFirstSeleniumTests { }”
 - We only want one class declaration per file.

This should give you a listing like the following, which although valid Java is not yet correct because we need to make a few more changes to make it work in Eclipse:

```

package com.eviltester.seleniumtutorials;

import com.thoughtworks.selenium.*;
import java.util.regex.Pattern;

public class MyFirstSeleniumTests extends SeleneseTestCase {
    public void setUp() throws Exception {
        setUp("http://www.google.co.uk/", "*firefox");
    }
    public void testGoogle_for_selenium_rc() throws Exception {
        selenium.open("/");
        selenium.type("q", "Selenium-RC");
        selenium.click("btnG");
    }
}

```

Resolve Import Errors

The first thing we need to resolve is the error on the first import statement.

```
import com.thoughtworks.selenium.*;
```

Import statements are how we tell Java about the methods and classes we are using from external packages.

If we hover over the red cross at the side of this line then Eclipse will tell us what the problem is.



Figure 6.14: Eclipse Explains the Error

In this case it is "The import com.thoughtworks.selenium.* cannot be resolved" which means that Eclipse does not know where to find this package because we haven't told Eclipse where the selenium library is yet.

We can fix this error automatically by clicking on the red cross to the left of the source code line.

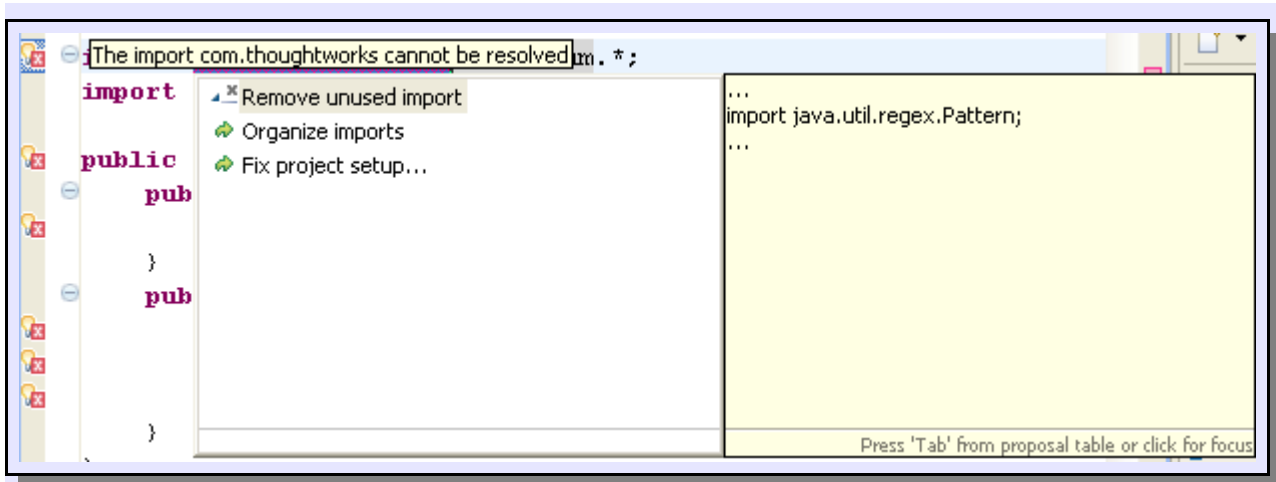


Figure 6.15: Try to Automatically fix the error

Double click on the "Fix project setup..." option and Eclipse will tell us that it doesn't know how to do that automatically.

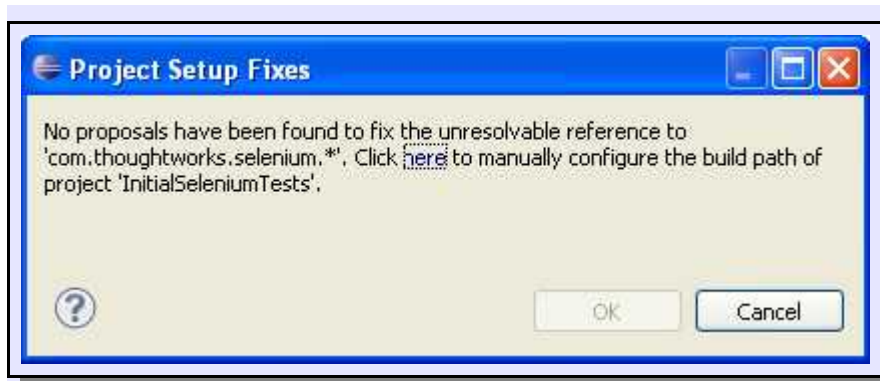


Figure 6.16: Manually fix the error

So we accept Eclipse's offer to allow us to manually configure the build path for the project. By clicking on the "here" in the dialog text "click here to manually configure the build path of project 'InitialSeleniumTests'."

The 'build path' is how java knows what external libraries are used in this project. Java searches through those libraries looking at the different packages to find the classes and methods that we want to re-use from those libraries.

When we click 'here', we will see a subset of the properties of the project we are working on. We want to add a new library so click on the "Libraries" tab.

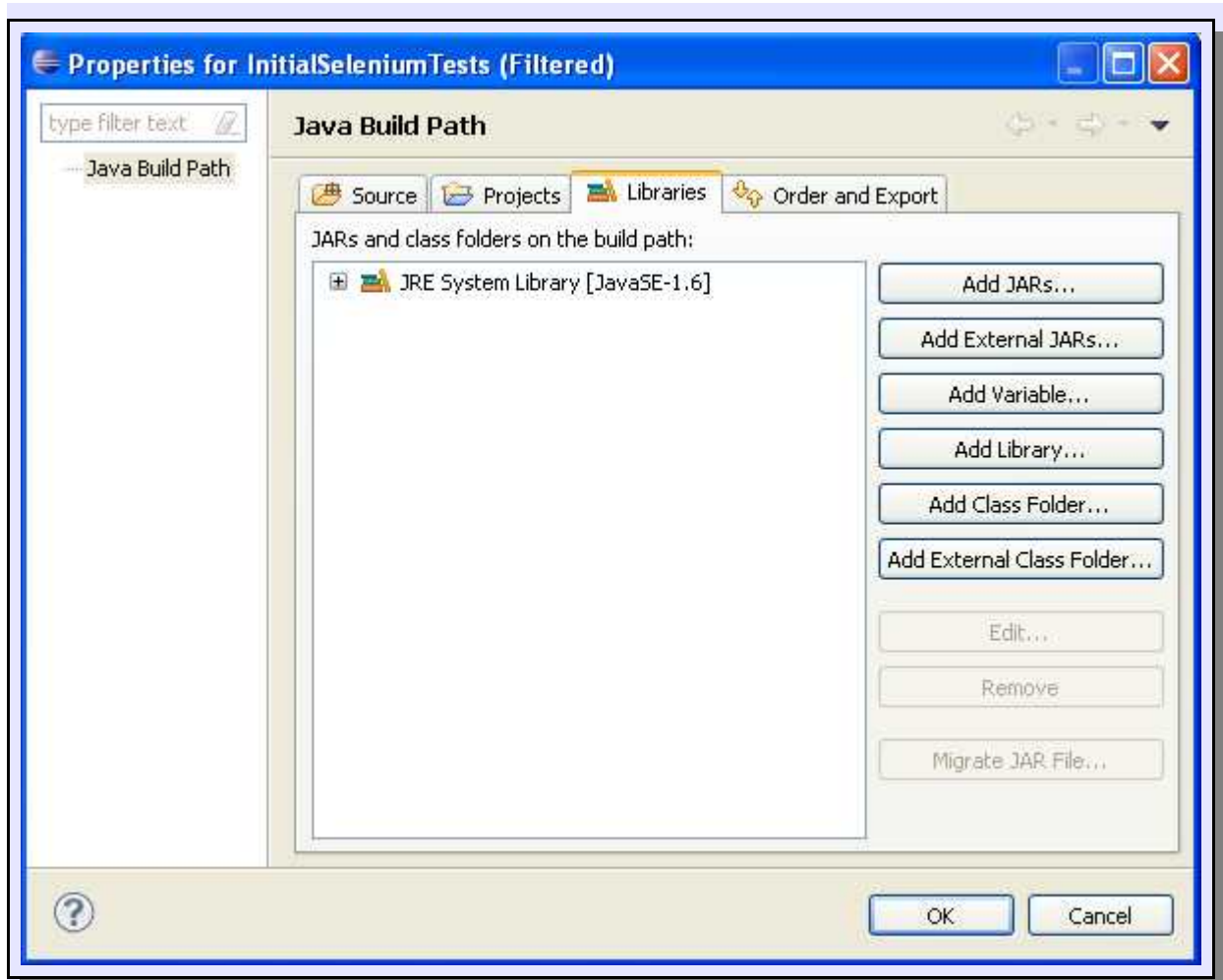


Figure 6.17: Basic properties for the default project

We want to amend the Java Build Path by pressing the [Add External JARs...] button. Navigate to your Selenium directory, then into the 'java client directory'

`C:\selenium\selenium-remote-control-1.0.3\selenium-java-client-driver-1.0.1`

And select:

- selenium-java-client-driver.jar, and

Upon clicking [open] from the file dialog, That file should be displayed in the Libraries list now.

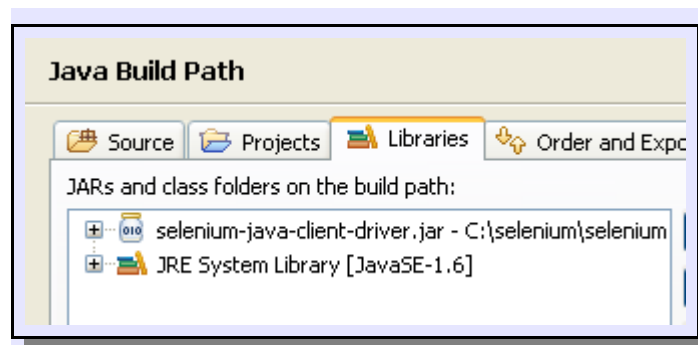


Figure 6.18: Client driver added to libraries

Click [OK] to exit the dialog.

The display of the test should now look better, with only one error symbol displayed by Eclipse.

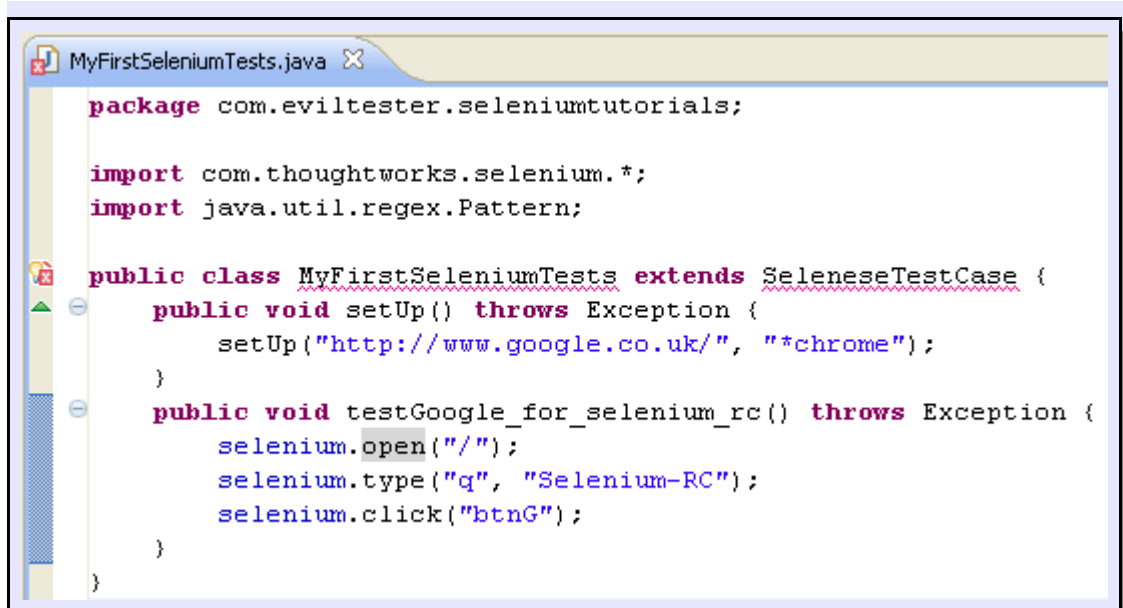


Figure 6.19: Only one error left

Add JUnit to the build path

As before, hover over the symbol to see what errors Eclipse reports.

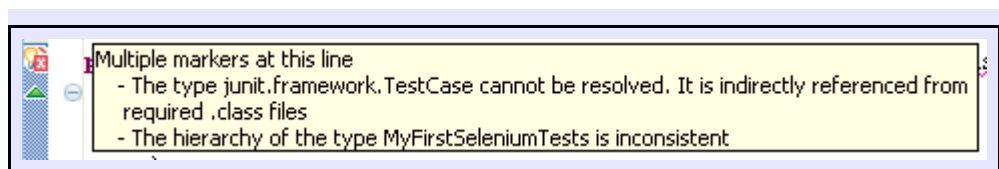


Figure 6.20: JUnit TestCase not defined

In the first error Eclipse reports that we haven't added JUnit to the build path.

Click on the error symbol and choose "Configure build path..." from the list of options.

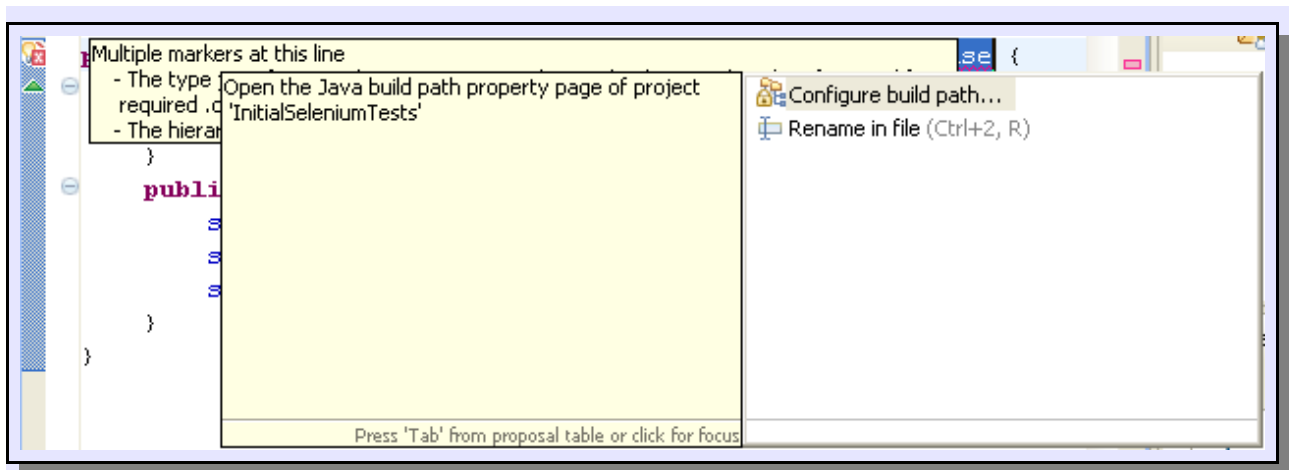


Figure 6.21: Configure build path for JUnit

This time we are going to choose the [Add Library...] button because Eclipse is distributed with JUnit. From the list of libraries presented, choose JUnit and click [Next >].

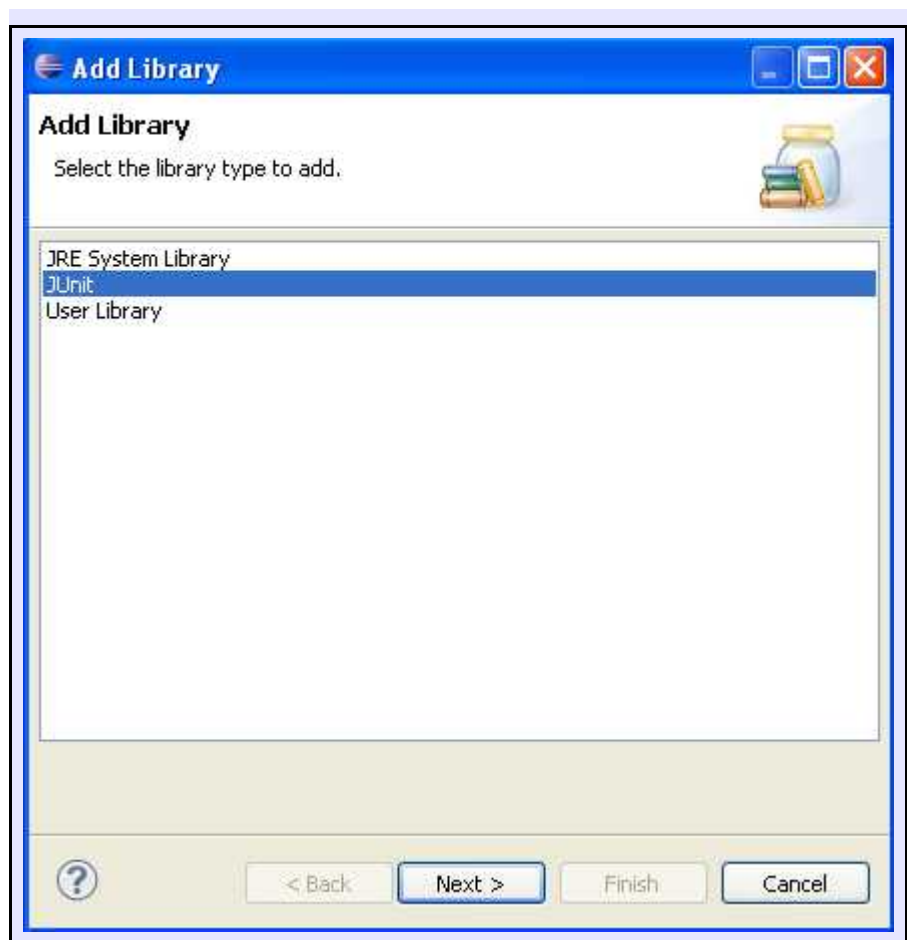


Figure 6.22: JUnit distributed as a standard library with Eclipse

From the list of JUnit library versions, choose “JUnit 4” and click [Finish].

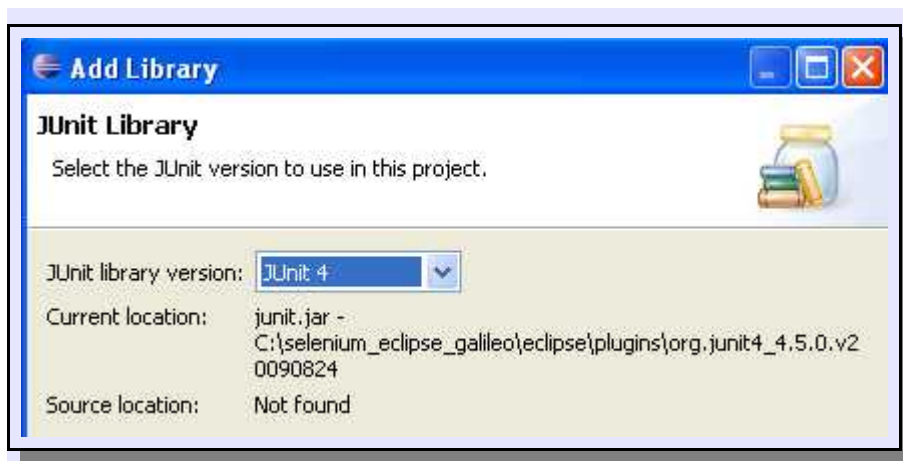


Figure Choose JUnit 4 from the drop down

Click [OK] on the properties dialog and you should see your test with a single warning symbol next to the “import java.util.regex.Pattern;” line.

Remove the unused import

If we hover over this warning then we will see Eclipse report that the imported classes are never used.

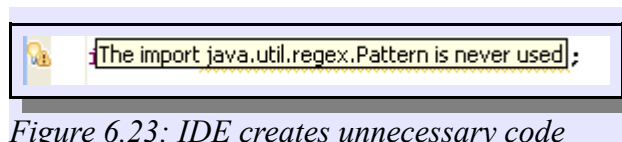


Figure 6.23: IDE creates unnecessary code

So we can safely use the option provided when we click on the warning symbol to remove the unused import.



Figure 6.24: Remove the line throwing a warning

And having done all this, our code should be clean and ready to run:

```
package com.eviltester.seleniumtutorials;
import com.thoughtworks.selenium.*;

public class MyFirstSeleniumTests extends SeleneseTestCase {
```

```

    public void setUp() throws Exception {
        setUp("http://www.google.co.uk/", "*firefox");
    }
    public void testNew() throws Exception {
        selenium.open("/");
        selenium.type("q", "Selenium-RC");
        selenium.click("btnG");
    }
}

```

Run the JUnit test

We run this by right clicking on the “MyFirstSeleniumTests.java” in the Package Explorer.

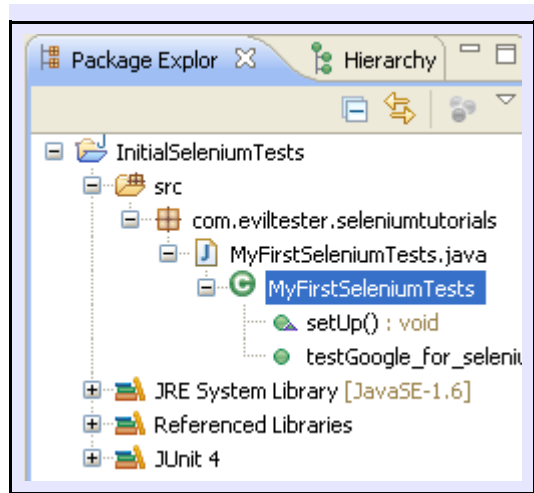


Figure 6.25: The Test in the Package Explorer

And selecting “Run as > JUnit test”.

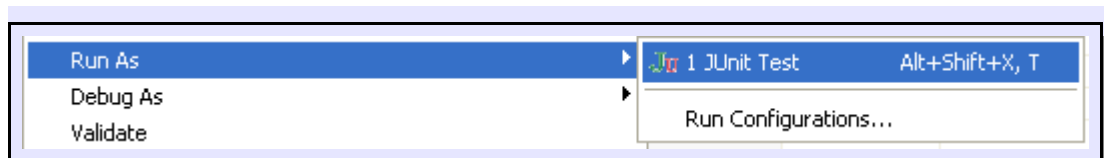


Figure 6.26: Run the test as a JUnit test

Try running it, remember first of all to start the Selenium-RC server as explained in earlier sections.

If you don't then Selenium client will display an error in the Eclipse console, asking if you have started the server.

Allow it through the Firewall

If you encounter any Firewall prompts when you run it then you should allow Eclipse or java access through your Firewall.

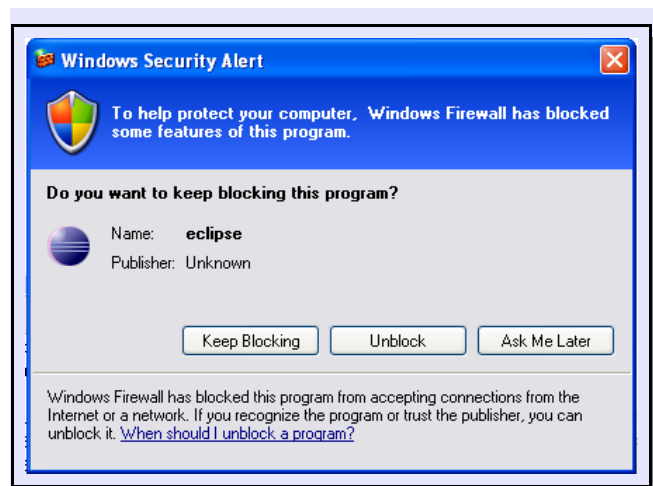


Figure 6.27: Allow Eclipse through the firewall

Seeing the test running

You should see the Selenium window appear. And underneath it the Google window.

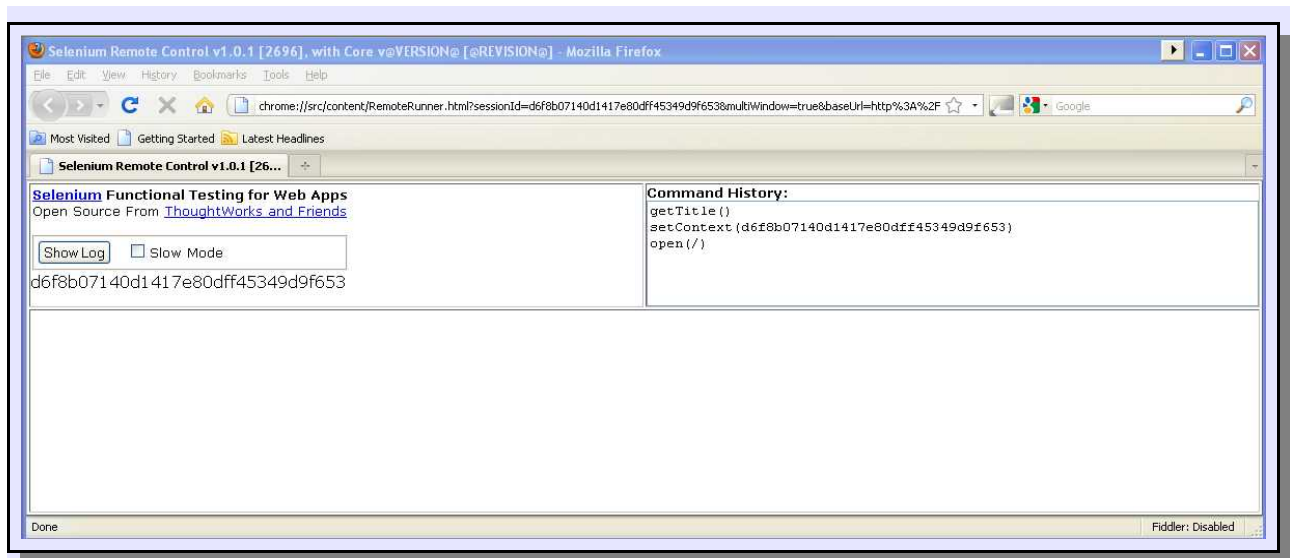


Figure 6.28: The Selenium window shows the commands and any errors that occur.

NOTE:

If you didn't see this then you may have firewall problems (you could check by disabling the firewall while you try to run the test and see if that helps - if it does then you may need to do one or more of the following: open port 4444, add Eclipse, java and the browser to your exclusions list).

In the Google window you can see the actual test execution happening. Of course since it is a small test it may have run too quickly for you to see the details. So we will cover that in the next major section after you check the results.

You should also have seen a bunch of messages appear in your console where you are running

Selenium Server.

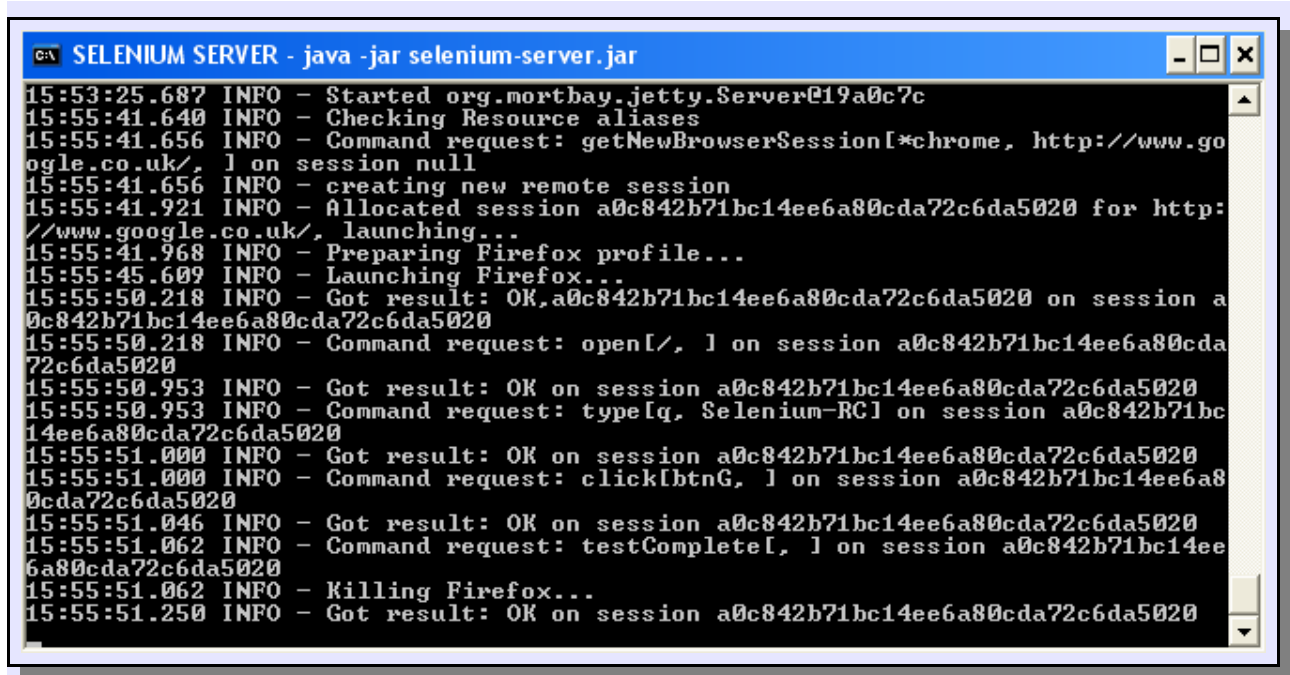


Figure 6.29: Messages from Selenium Server

From the console window you can gain a little insight into how Selenium works. Because we are using Firefox, Selenium creates a new Firefox profile for executing the test. This means that if we have any plugins etc in Firefox then they won't be available to the Selenium test. You can also see each statement in our test "open", "type", "click" sent through to the server as a 'Command Request' and Selenium-RC execute it within a session and return an OK result to the client.

The console can prove invaluable when investigating Selenium errors so read the output during your testing to help you continue to learn.

Check the results

After the run you should notice that the JUnit tab is now selected on the right.

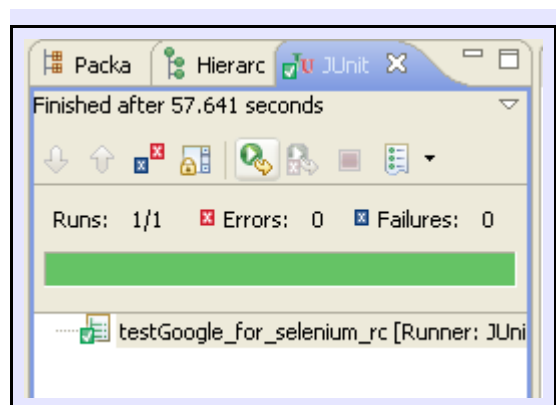


Figure 6.30: JUnit tab after a successful run

This reports that we ran 1 test, with no errors. And the green bar means that all our tests passed. If

the test had failed then the bar would be red and we would see the failing tests with a red cross next to them.

It went too quickly!

OK, we will now use the debugger to run the test slowly and see it in action.

Set a break point by right clicking on the left bar in Eclipse beside your source code on the line with the `setUp` command.

```
setUp("http://www.google.co.uk/", "*firefox");
```

Choose “Toggle Breakpoint” from the displayed popup menu.

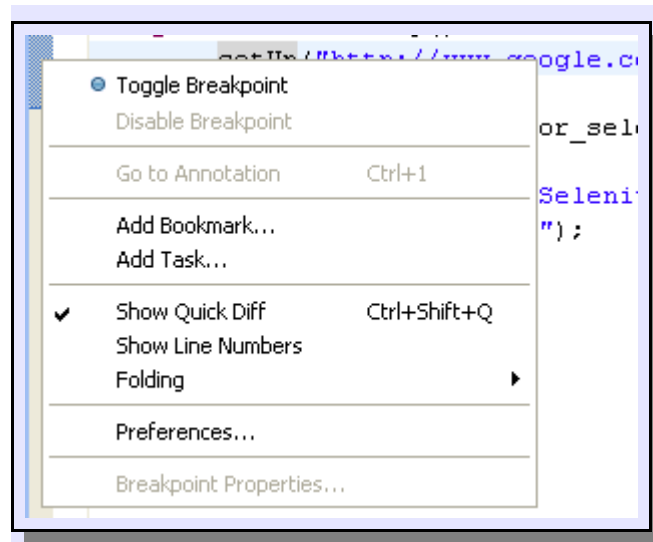


Figure 6.31: Toggle Breakpoint in the context menu

If you now see a round blue circle in the margin next to that line then it means you have set a breakpoint.

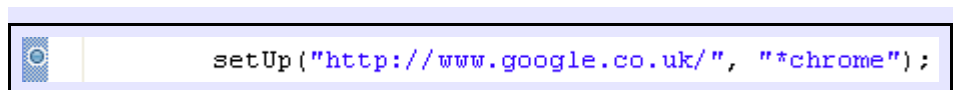


Figure 6.32: A line in Eclipse with a breakpoint set

The breakpoint means that when we run this test in debug mode. The execution of the test will pause when we hit that line and we can manually control the execution of the test.

Add another breakpoint to the line of code that reads:

```
selenium.open("/");
```

So you should see two breakpoints set in your code.

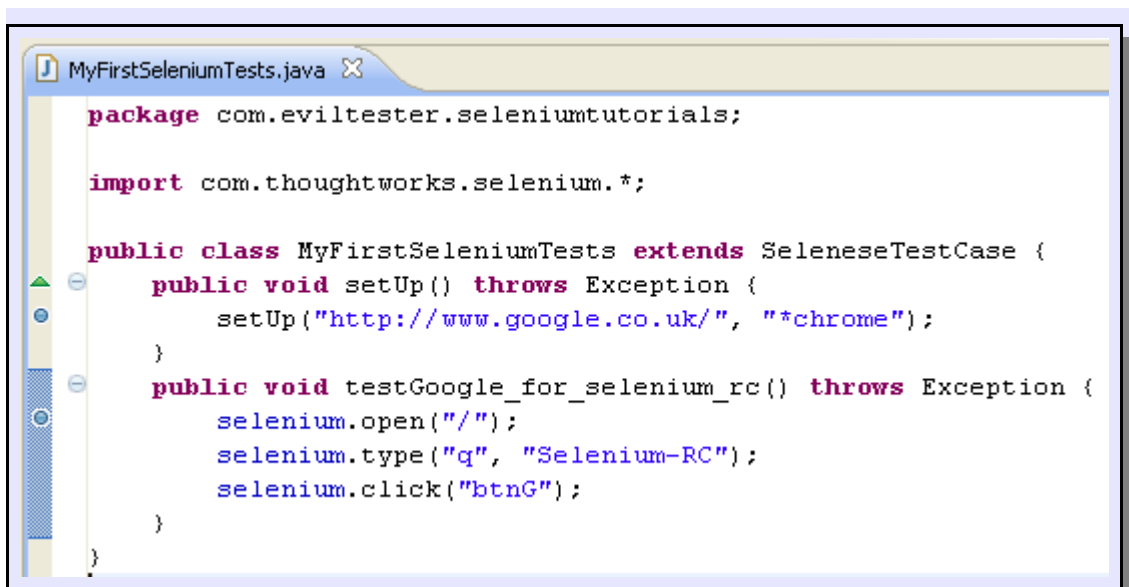


Figure 6.33: Two breakpoints visible in left info bar

Run the test in debug mode

Now we want to run the test in debug mode.

Switch back to the Package Explorer view by clicking on the Package Explorer tab.

And instead of right clicking on the test class and choosing "Run As\ JUnit Test" we want to choose "Debug As \ JUnit Test". Or select the test class name in the Package Explorer and press Alt+Shift+d, followed by 't'.

The first thing you may notice is a "Confirm Perspective Switch" dialog asking if you want to launch the Debug view. Click the "[x] remember my decision" option and choose [Yes].

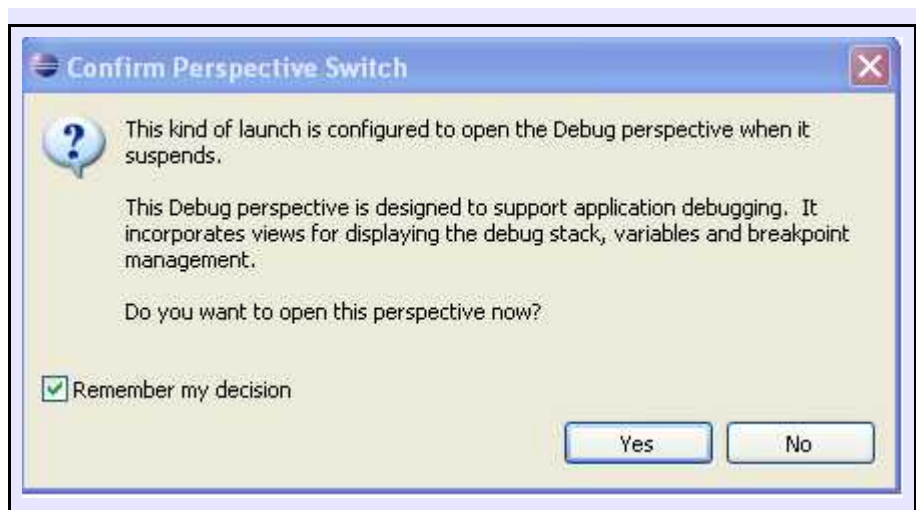


Figure 6.34: Confirm the switch of perspective

You should then find yourself in a new window layout of Eclipse. The Debug perspective.

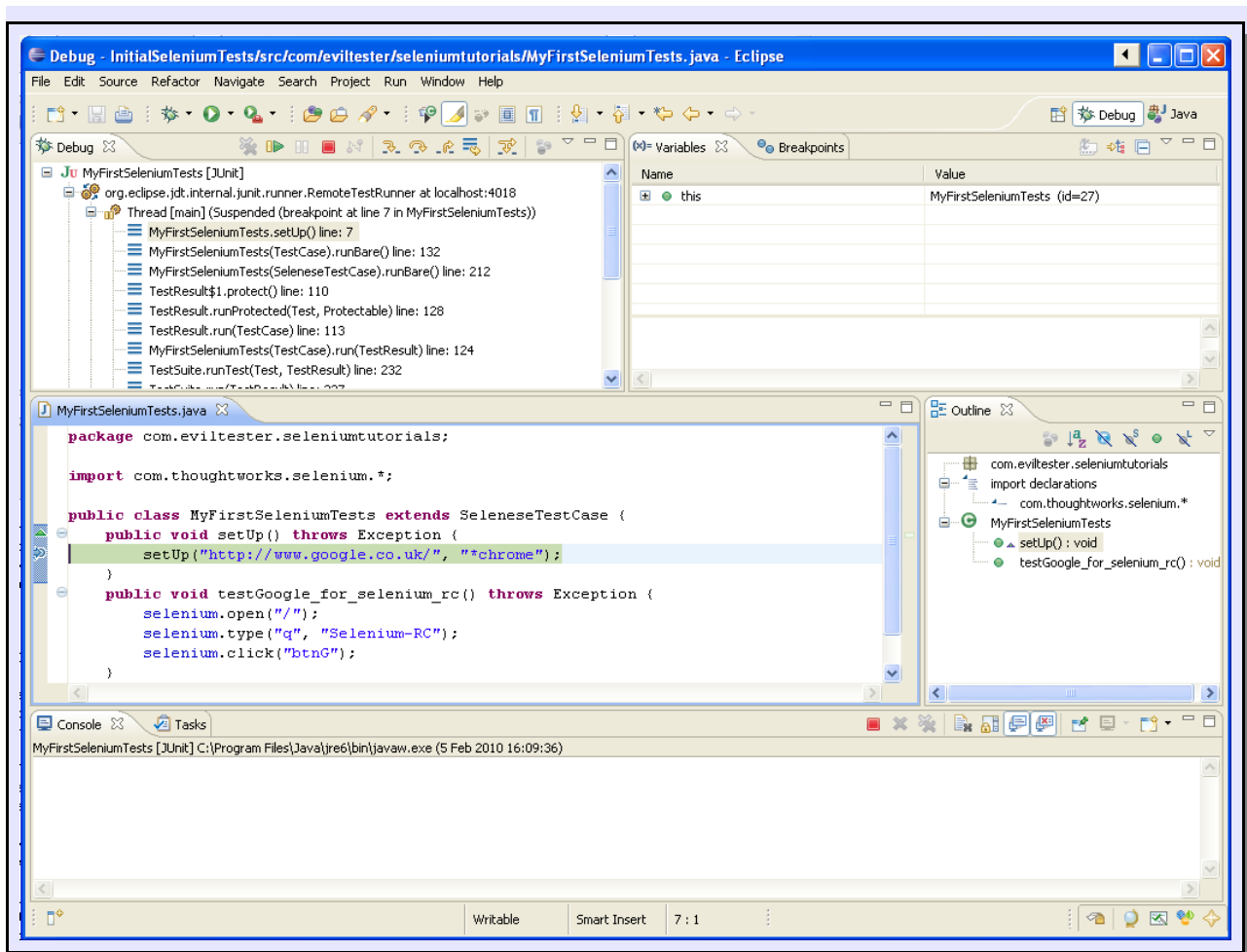


Figure 6.35: The Debug Perspective

We will use the toolbar to navigate through the code.



Figure 6.36: The debug toolbar

Each of the buttons has a shortcut key equivalent. Hover the mouse over the icon to learn what function each one has. We will only use two in this tutorial:

-  Step Over (F6)
-  Resume (F8)

Your test should have stopped executing on the `setUp("http://www.google.co.uk/", "firefox");` line. And this should be highlighted green.

This line has not executed yet.

This code was provided by the selenium-server jar in the SeleneseTestCase class. We will look into the class in more detail later, so for now all you really need to know is that this will setup our Selenium session and start the browser for us. We will step over this line by pressing F6.

“Step Over” means that it will execute in the debugger but we will not debug through all the lines

of code that were written to make 'setup' happen.

When we step over the code, either by pressing the F6 key, or clicking on the icon, the "Selenium Remote Control" window should have been displayed. Eclipse now highlights the closing brace in the setUp method in green because this is the next line to be executed.

If you notice the url in the execution browser you can see that it hasn't visited google yet and still has a selenium-server command listed:

- <http://localhost:4444/selenium-server/core/Blank.html?start=true>

Press "Step Over (F6)" again and you should see the Google homepage appear in your "Selenium Remote Control" window.

Press "Step Over (F6)" again and you will see a Class File Editor showing TestCase.class. This is a JUnit class and we haven't associated any source code. This is not an error. We can still debug through the test. This is why we set the second breakpoint.

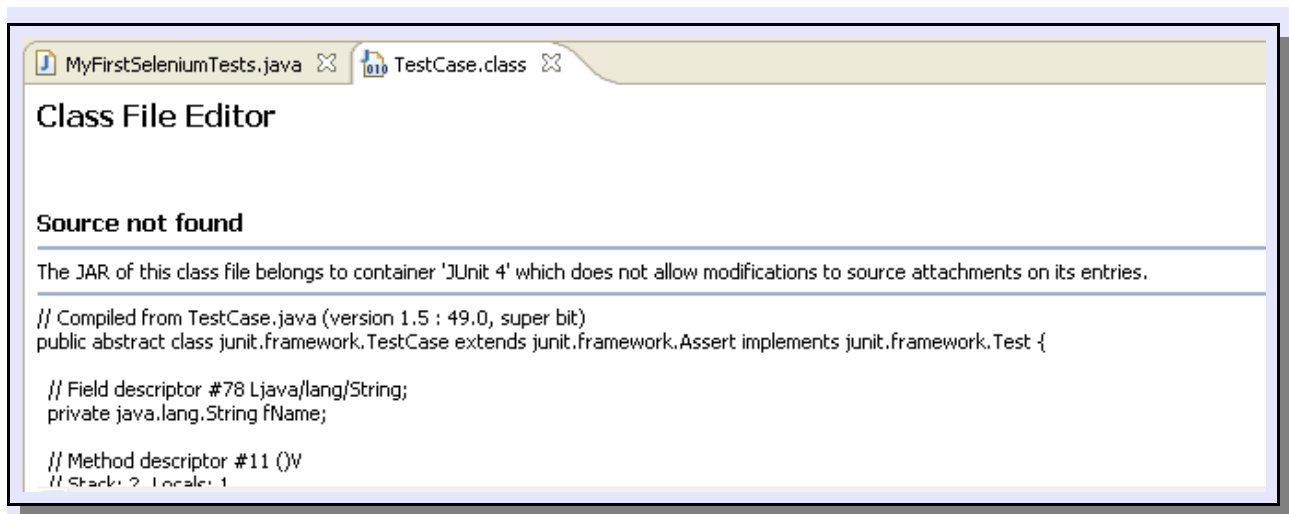


Figure 6.37: JUnit TestCase class code not found

So click the resume button or press F8. And this will execute the code to get to the next line that we are interested in seeing execute under the debugger.

Now you should see the selenium.open("/"); line selected in green.

Press "Step Over (F6)" and you should see Selenium execute that command and open Google in the window below.

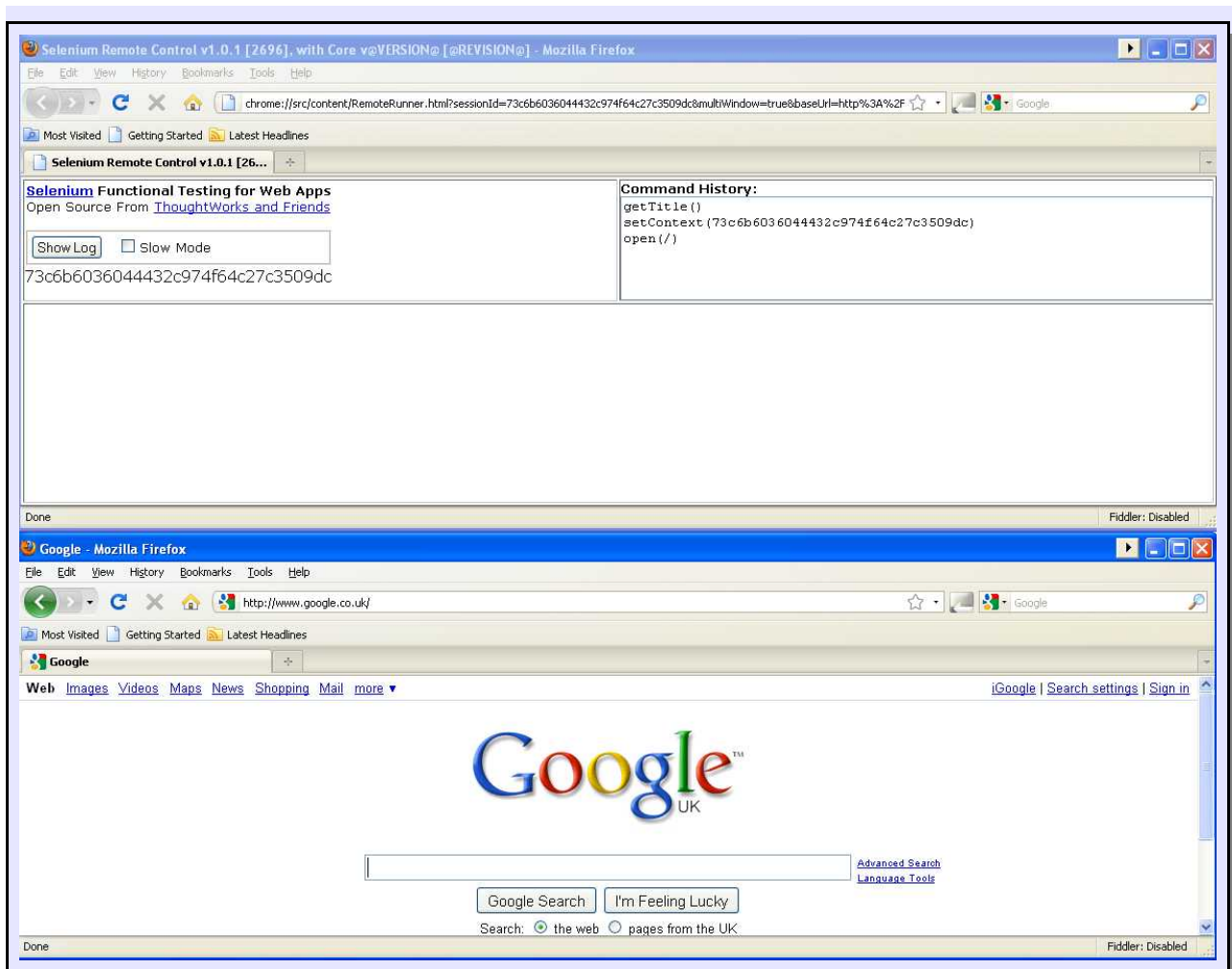


Figure 6.38: The test running

Keep pressing "Step Over (F6)" to watch the test execute and when happy that you have seen enough press "Resume (F8)" to skip over the rest of the test code to finish the test.

Click the [Java] perspective button in the top right to go back to your more familiar Package Explorer view.



Figure 6.39: Java Perspective Button

Create and Import some more IDE Converted tests

Just to make sure that we know how to import tests and you can see how easy it will be in the future now that we have done the hard work of setting up the build path.

We will now record and import another test into this class.

Start up Firefox and the Selenium IDE.

Make sure record is on.



Figure 6.40: Record is on

Make sure Options\Format is set to HTML.

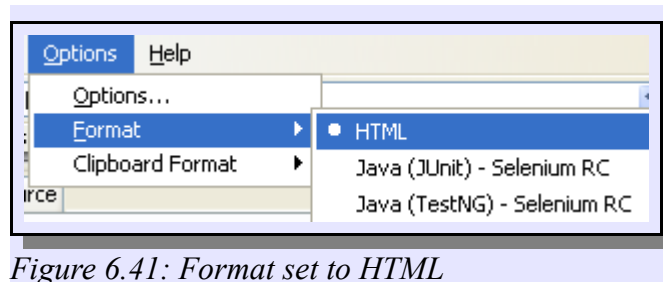


Figure 6.41: Format set to HTML

And that you have the Table view showing.

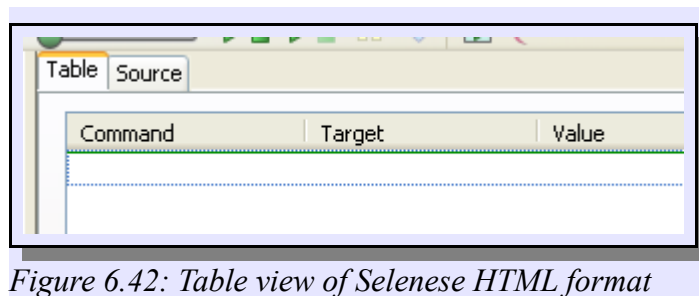


Figure 6.42: Table view of Selenese HTML format

This time record actions where you:

- type in "evil tester" to the search box
- click on "Images"
- and then perform a Google Search by pressing the "Search Images" button.

You should have a test like the following:

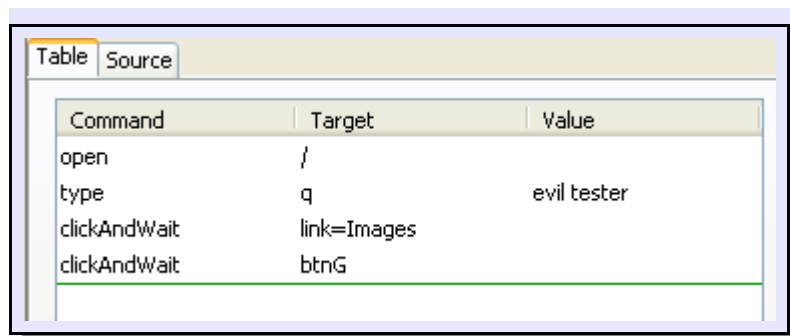


Figure 6.42: Recorded Script in Selenese Table view

Run the test again to make sure you have created something repeatable by clicking the Play button.



Figure 6.43: Play button on left

And if that worked, then we can change the format to “Java (JUnit) – Selenium RC” and import it into Eclipse.

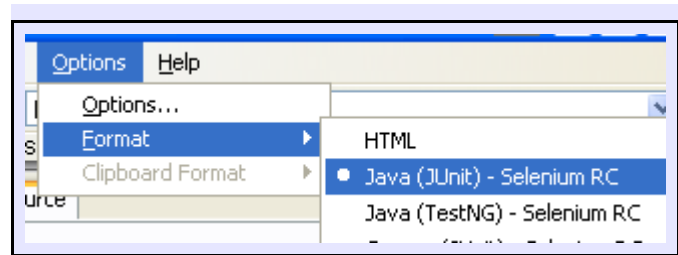


Figure 6.44: Select Java (JUnit) format

But this time, all we want to copy into Eclipse is the 'test' code i.e.

```
public void testUntitled() throws Exception {
    selenium.open("/");
    selenium.type("q", "evil tester");
    selenium.click("link=Images");
    selenium.waitForPageToLoad("30000");
    selenium.click("btnG");
    selenium.waitForPageToLoad("30000");
}
```

Paste that code into Eclipse beneath the existing test and before the closing brace “}” of the class.

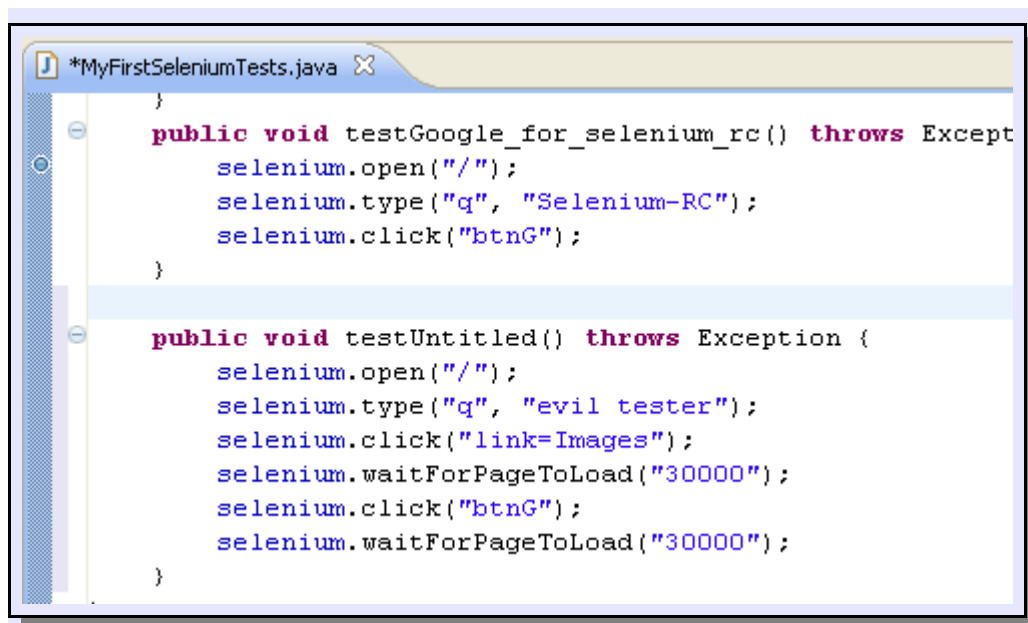


Figure 6.45: Just the method code pasted into Eclipse

Then rename the test. i.e. Change “testUntitled” to “testGoogleForEvilTesterImages”

```
public void testGoogleForEvilTesterImages() throws Exception
    selenium.open("/");
```

Figure 6.46: The renamed test in Eclipse

Click on the class and choose “run \ JUnit test”, then Eclipse will run all the tests in that class.

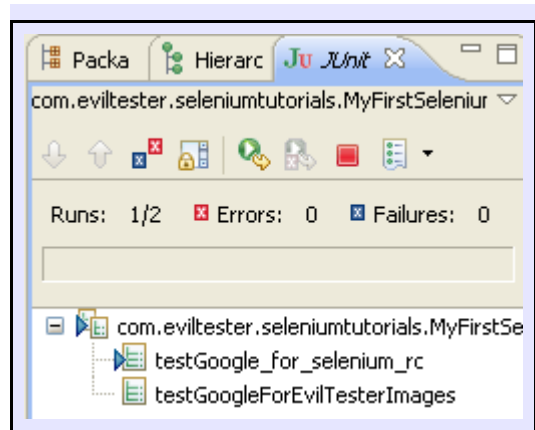


Figure 6.47: JUnit running both tests

You should now have enough information to record your own tests and add them into either the existing class, or create new classes to house them.

Hopefully you can see that we can easily use the IDE to generate the basics for simple tests. In a later chapter we will see how to identify the locators for the .type and .click methods, and how you can learn to write tests like this by hand.

Learning to write these tests by hand is important because you will learn how the application hangs together and to write robust tests.

For quickly generating basic tests, copying code in from the IDE can be very useful.

In practice I use the IDE to help capture certain actions that I want to do, but may be having difficulty figuring out the exact command sequence, or frames that controls are embedded within.

Exercises

Instructions: Create a new class called MyFirstSetOfExercises. This is where you will create some more tests from the IDE and paste them in. You should try and create all tests in the IDE and then simply change the format to JUnit and paste them into Eclipse.

Exercise:

1. search for an image, assert title of page,
2. add an assert that a specific image is present
3. change the value of the search to see the test fail
4. use verify instead of asserts

Chapter 7: The Annotated Generated Test

Creating the test from the IDE was simple, but there is still quite a lot to learn about the automatically generated test, as it hides a lot of complexity from us that we need to understand to get the best from Selenium.

As we look at the generated test we will learn a lot about Java, Object Oriented programming and how Selenium works. We will put that to good use in the next chapter where we create tests without using the `SeleneseTestCase` class and in this way we will take complete control of our Selenium test cases.

Use Attach Source to see the Selenium Driver Source Code

The first thing we will do is configure Eclipse to let us see the source code for the Selenium Driver.

So in Eclipse, highlight the text “`SeleneseTestCase`” (you can do this by double clicking on the text).

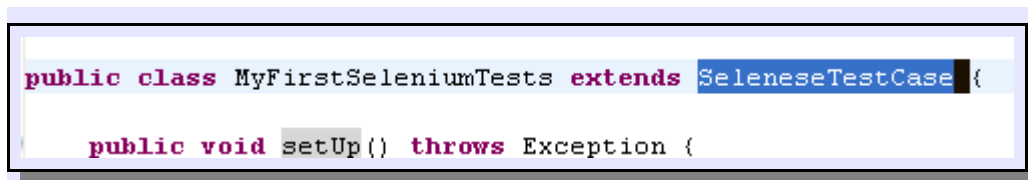


Figure 7.1: Highlighted `SeleneseTestCase`

Then either:

- right click and choose “Open Declaration” from the context menu or
- press F3 – the shortcut key to Open Declaration

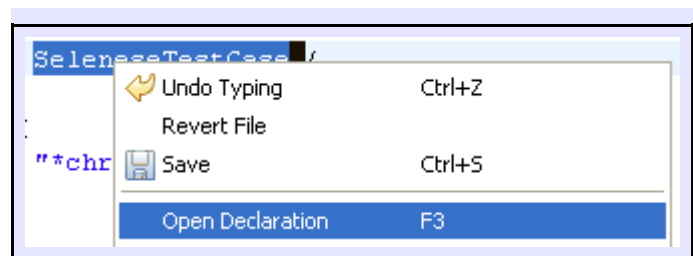


Figure 7.2: Open Declaration from Context Menu

“Open Declaration” is a very useful function in Eclipse to take you to the implementation source code for the source item you selected. In this case we want to see the source code for the Class declaration of `SeleneseTestCase`.

You should see a Class File Editor window, with the message “Source not found”. This is because we haven't told Eclipse where the source code for the Selenium Java Driver resides on the disk.

We can easily fix this by pressing the [Attach Source...] button and navigating to the `selenium-java-client-driver-sources.jar`.

Chapter 14: Chapter - Basic HTML Theory

This chapter will provide you with a basic understanding of HTML. Just enough to cover the basic requirements for writing Selenium tests.

If you visit a web page in your browser, and then view source on that page. You will see that an HTML page has a very simple structure.

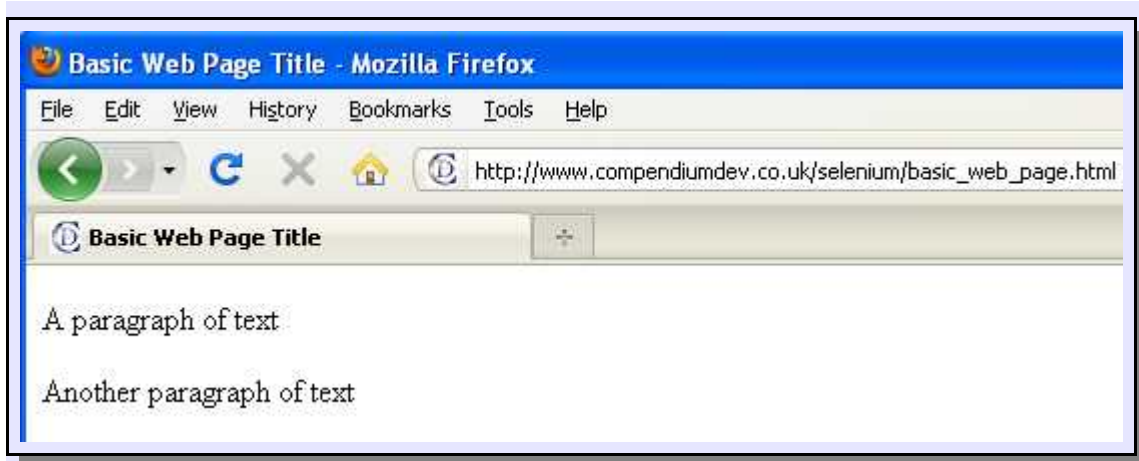


Figure 14.1 : A very simple HTML page

Open http://www.compendiumdev.co.uk/selenium/basic_web_page.html in a browser and view the source code and you will see a very simple set of markup.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Basic Web Page Title</title>
  </head>
  <body>
    <p id="para1">A paragraph of text</p>
    <p id="para2">Another paragraph of text</p>
  </body>
</html>
```

Page Structure

An HTML page has a very simple structure.

- A doctype to help the browser identify what to do with the file
- an <html> block which contains the page code
 - a <head> section
 - a <body> section

Elements & Tags

HTML consists of nested elements. Elements are represented by an opening tag: e.g. <p>, followed by the corresponding closing tag </p>.

In a well formed XHTML document each opening tag has a corresponding closing tag. Some older

doctypes allow you to leave off the closing tag and let the browser try and work out where an element finishes.

Some elements do not have any content, so don't have an open and close tag, they are self contained e.g. `
`.

Because elements are nested, an HTML document can be viewed as a tree. This allows XPath to be used to traverse it.

Attributes

Elements can have attributes e.g. The id in `<p id="para1">`.

Attributes help to specify display styles, code, and in the case of id – a unique identifier for the element in the document.

Expanded

Just to be clear, in the line: `<p id="para1">A paragraph of text</p>`

- The Paragraph **element** is: `<p id="para1">A paragraph of text</p>`
- The Paragraph element **content** (or **text**) is: "A paragraph of text"
- The **tags** are: `<p>`, `</p>`
- The **attribute** is: 'id'
- The attribute **value** is: "para1"

Chapter 17: Learning The Selenium API

Now that you have learned the basics of the unit test framework, Eclipse and simple Java. It is time to start learning the Selenium API.

Selenium has a very simple and unstructured API. Basically a long list of methods on the Selenium object. This makes the API large and unwieldy, but it also makes it very simple for beginners which is one reason why you don't need to know much Java to get started.

The most up to date documentation for the API is found in the Selenium source code, and since it is added as JavaDoc comments it will appear in Eclipse as the comments in the code completion documentation.

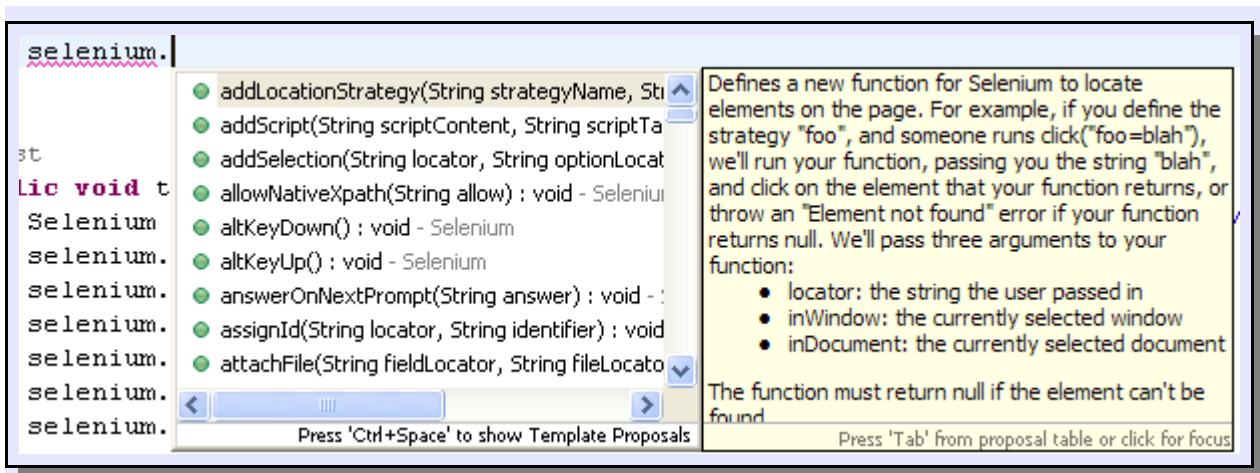


Figure 17.1 : JavaDoc displayed as code completion contextual help

In the following few chapters we will build some familiarity with the API methods by using them to complete common tasks that you will encounter when testing web sites.

Chapter 21: Start Selenium Programmatically

The Selenium distribution's `selenium-server.jar` can also be called programmatically. So that instead of having to type “`java -jar selenium-server.jar`” at the command line, we can have our tests start up and close down a server when they run.

To do this we use the `startSeleniumServer()` command from the `SeleniumServer` Class.

Add Selenium Server to the project

You need to add `selenium-server.jar` to your class path. As before, amend the properties of your project to amend the Java Build Path, and choose to add `selenium-server.jar` as an external jar.

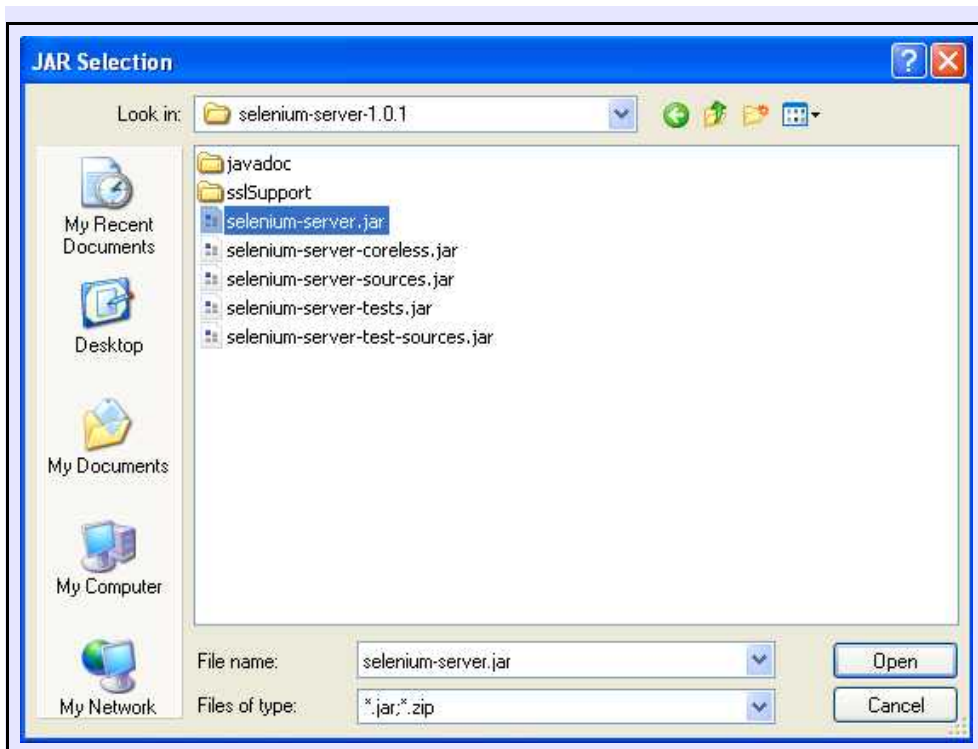


Figure 21.1: Add selenium-server.jar to the project

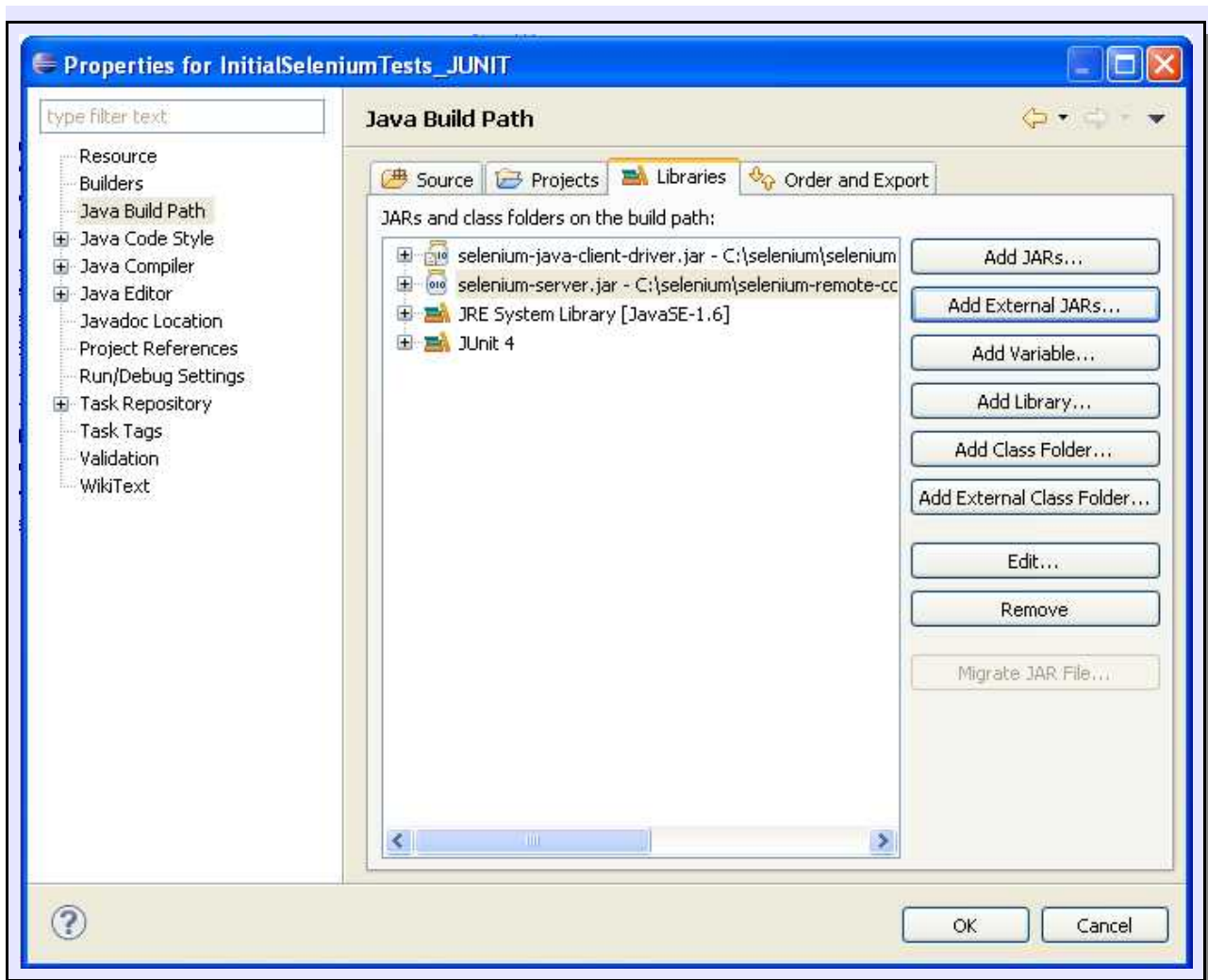


Figure 21.2: Properties Dialog with selenium-server.jar added to build path

Start it in the code

You start Selenium Server by declaring a `SeleniumServer`, creating a new instance and then starting it. We wrap it with a try catch block because `SeleniumServer` will throw an exception if it can't start.

```
SeleniumServer seleniumServer=null;
try {
    seleniumServer = new SeleniumServer();
    seleniumServer.start();
} catch (Exception e) {
    e.printStackTrace();
}
```

Then connect to the server in the normal way. Your server has still started on your local machine using the same port.

```
Selenium selenium=null;
selenium = new DefaultSelenium("localhost", 4444,
    "**firefox", "http://www.google.com");
selenium.start();
```

Then do your normal test commands.

```
selenium.open("http://www.google.com");
selenium.type("xpath=//input[@name='q']", "Selenium-RC");
```

```
selenium.click("xpath=//input[@name='btnG' and @type='submit']");
selenium.waitForPageToLoad("10000");
```

And close selenium in the normal way.

Then close down the server.

```
if(seleniumServer!=null){
    seleniumServer.stop();
}
```

So a full individual test to do this would look like the following:

```
package com.eviltester.seleniumtutorials;

import org.junit.Test;
import org.openqa.selenium.server.SeleniumServer;
import com.thoughtworks.selenium.DefaultSelenium;
import com.thoughtworks.selenium.Selenium;

public class SeleniumServerInATest {

    @Test
    public void doASearchOnGoogle() {

        SeleniumServer seleniumServer=null;

        try {
            seleniumServer = new SeleniumServer();
            seleniumServer.start();
        } catch (Exception e) {
            e.printStackTrace();
        }

        Selenium selenium=null;
        selenium = new DefaultSelenium("localhost", 4444,
            "**firefox", "http://www.google.com");
        selenium.start();

        selenium.open("http://www.google.com");
        selenium.type("xpath=//input[@name='q']", "Selenium-RC");
        selenium.click("xpath=//input[@name='btnG' and @type='submit']");
        selenium.waitForPageToLoad("10000");

        selenium.close();
        selenium.stop();

        if(seleniumServer!=null){
            seleniumServer.stop();

            selenium
        }
    }
}
```

Obviously we wouldn't start and stop the server in each test so we would refactor this to use the @BeforeClass, @AfterClass annotations to start the server.

The Start Up routine explored

Typically the Selenium Server will not start because something else is already using the Port: e.g. Another instance of Selenium.

BindException Handling

We could amend the code to check if the exception raised was a BindException. If this was the case then we could assume that a selenium server was already running and use the existing Selenium Server. e.g.


```

Try {
    seleniumServer = new SeleniumServer();
    seleniumServer.start();
} catch (java.net.BindException bE){
    // could not bind, assume that server is currently running
    // and carry on
    System.out.println("could not bind - carrying on");
}
catch (Exception e) {
    // any other exception stop
    throw new IllegalStateException("Can't start selenium server", e);
}

```

Stop Existing Server

We might choose to stop the running server and start a new one.

You would have to decide if this was a valid thing to do in your environment or not. If during the course of your testing you could not start a server then it might mean that your test setup was not closing down the server properly. And rather than 'working around' this, you might find it more beneficial to work out how to stop the server automatically during your previous testing.

But since I think this is a useful technique to know, I include it here.

This will get a little more complicated so if you are new to Java I don't necessarily expect you to understand all of this.

Our main block doesn't change very much:

```

String stopSeleniumCommand =
    "http://localhost:4444/selenium-server/driver/?cmd=shutDownSeleniumServer";
try {
    seleniumServer = new SeleniumServer();
    seleniumServer.start();
} catch (java.net.BindException bE){
    // could not bind, assume that server is currently running
    // and carry on
    System.out.println("could not bind - carrying on");
    // try and stop it
    if( runHTTPCommand(stopSeleniumCommand)){
        try {
            seleniumServer = new SeleniumServer();
            seleniumServer.start();
        } catch (Exception e) {
            throw new IllegalStateException(
                "Could not stop existing server on blocked port 4444", e);
        }
    }
}
catch (Exception e) {
    // any other exception stop and start
    throw new IllegalStateException("Can't start selenium server", e);
}

```

This time, if we could not bind, we send an HTTP request to the server, and if it is a selenium server on that port, it will stop the server.

If that seemed to work then we try and start the server again.

The runHTTPCommand that does the magic to stop Selenium uses a little more complicated Java.

```

private boolean runHTTPCommand(String theCommand) throws IOException{
    URL url = new URL(theCommand);

    URLConnection seleniumConnection = url.openConnection();
    seleniumConnection.connect();

    InputStream inputStream = seleniumConnection.getInputStream();
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    byte[] buffer = new byte[2048];

```

```

        int streamLength;
        while ((streamLength = inputStream.read(buffer)) != -1) {
            outputStream.write(buffer, 0, streamLength);
        }
        inputStream.close();

        // give command some time to finish
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        String stringifiedOutput = outputStream.toString();
        if (stringifiedOutput.startsWith("OK"))
            return true;

        return false;
    }
}

```

This code, requires the addition of the following imports:

```

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.net.URLConnection;

```

Basically it:

- creates a connection to run the command.
- Issues the HTTP command
- retrieves the output result
- waits for the Selenium server to shutdown fully
- If the output result starts with “OK” then it returns true
- otherwise returns false

Try this code by adding it in a test and running it when a Selenium Server is already running from the command line. You should see the server shut down, and then the test run.

Custom Remote Control Configurations

Because we are now starting Selenium-RC from the command line, we have the option to create custom configurations. At its most simple this might just mean choosing a different port to run the server against, but we can control any element of Selenium-RC that is also configurable from the command line.

To do this we use the RemoteControlConfiguration class which is also contained in Selenium-server.jar:

```

import org.openqa.selenium.server.RemoteControlConfiguration;

```

In the following test I create the server on a different port, using the RemoteControlConfiguration object:

```

@Test
public void remoteControlConfigExample() {

    final int SELENIUM_PORT = 8888;
    final int SELENIUM_TIMEOUT = 45;

    RemoteControlConfiguration rcConfig = new RemoteControlConfiguration();
    rcConfig.setTimeoutInSeconds(SELENIUM_TIMEOUT);
    rcConfig.setPort(SELENIUM_PORT);
}

```

```

SeleniumServer seleniumServer=null;
try {
    seleniumServer = new SeleniumServer(rcConfig);
    seleniumServer.start();
} catch (Exception e) {
    e.printStackTrace();
}
Selenium selenium=null;

selenium = new DefaultSelenium("localhost", SELENIUM_PORT,
    "**firefox", "http://www.google.com");
selenium.start();

selenium.open("http://www.google.com");
selenium.type("xpath=//input[@name='q']", "Selenium-RC");
selenium.click("xpath=//input[@name='btnG' and @type='submit']");
selenium.waitForPageToLoad("10000");

selenium.close();
selenium.stop();

if(seleniumServer!=null){
    seleniumServer.stop();
}
}

```

Note:

I also have to use the same port when I define the DefaultSelenium server.

And I have declared SELENIUM_PORT and SELENIUM_TIMEOUT as final because they are constants that will not change in the test.

Chapter 22: Running Tests Outside Eclipse

Thus far we have run all our tests from within Eclipse.

This has allowed us to get used to writing tests and debugging them but has not prepared us for a production environment.

In a production environment, a robust test suite will run as part of a continuous integration build and on multiple browser configurations.

To support this we must have the ability to run our tests outside of the Eclipse environment.

Many tools exist to support this way of working and for simplicity I will demonstrate these principles using Ant, Hudson and Subversion.

What is Ant?

Ant is a build tool written in Java. This means that given a configuration file, written in XML, it will carry out a series of actions to compile the java application, run the tests, report the results, and package up the files we need for deployment.

<http://ant.apache.org/>

Other build systems exist, but ant is simple enough to get to grips with that it makes a suitable tool for beginners.

What is Hudson?

Hudson is a continuous integration server written in Java. This means that you can setup repeating jobs which will build your software and run your tests regularly.

<http://wiki.hudson-ci.org/>

Again, other continuous integration systems exist, but Hudson has a very simple install routine and easy to use front end GUI, again making it suitable for beginners.

What is Subversion?

Subversion is a version control system. We will store our code in the version control system and allow Hudson to check out the code as required to run the most recent version of the tests automatically.

<http://subversion.tigris.org/>

Chapter 23: Using Ant to Run Selenium Tests

Install Ant

To Install Ant we are basically going to follow the instructions listed in the Ant manual

<http://ant.apache.org/manual/index.html>

1. Download the current version of Ant from the binary downloads page <http://ant.apache.org/bindownload.cgi> (I downloaded the zipped version, at the time of writing this was “apache-ant-1.8.0-bin.zip”)
2. Extract the contents of the archive to a folder on your drive. I unarchived the contents to <c:\apache-ant-1.8.0>

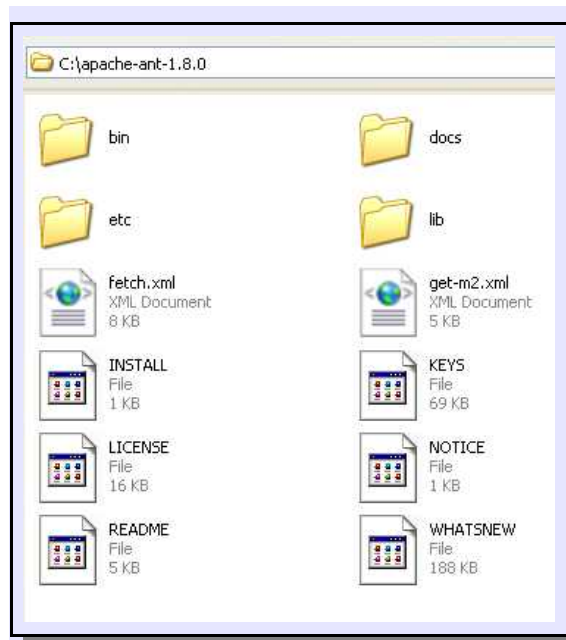


Figure 23.1 : Ant installed to c:\apache-ant-1.8.0

We still have additional configuration to do but first we need to install a JDK version of Java.

Install Java JDK

Because we are now moving into the territory of compiling java code to run outside of Eclipse, we need to install the Java Development Kit version of Java. This is the version that has the java compiler.

1. Visit the Java JDK download page <http://java.sun.com/javase/downloads/index.jsp>
2. Choose to download the most recent JDK

Chapter 36: The Future of Selenium

The Selenium tool suite is constantly being worked on. Having been embraced by Google as one of their automation tools, with many of the key figures in Selenium development working at Google, the future for Selenium looks rosy.

It can often seem daunting choosing between the many open-source software testing tools to know which ones will be supported and maintained over the years. The fact that Google works with it means that there will remain a talented set of programming and testing talent behind the tool.

Selenium is also used in spin off tools like the load testing tool from BrowserMob.com, which uses the selenium API as its scripting language from JavaScript to write the automated tests.

The current direction for Selenium, which should manifest as Selenium 2.0 merges the code bases for Selenium 1.0 and WebDriver.

Selenium takes the automation approach of driving the website through JavaScript with the Selenium RC server using Selenium Core to automate the site. This has some issues with security and cross browser scripting which is why there are so many ways of instantiating a browser session for the supported browsers. e.g. `*iexplore`, `*firefox`, `*firefoxproxy`, `*iexploreproxy`

WebDriver takes a different approach. It calls the browsers directly through their respective automation interfaces. This means that all tests run in the browser and have no security issues as it is the raw browser that takes all the actions.

WebDriver has a different API than Selenium. It is a far more Object Oriented interface and can seem cleaner. WebDriver also supports the use of HTMLUnit which means you can test websites without creating a 'real' browser so tests can run faster and more easily in a Continuous Integration environment.

For Selenium 2.0 the plan is to support both APIs:

- the Selenium API (large set of commands off the Selenium interface)
- the WebDriver API (object oriented browser automation based interface)

This means that you should still be able to write tests using the Selenium API, but have the tool drive the browser directly. This should see an end to some of the JavaScript injection issues and make the tests faster to run.

It also means that the WebDriver API can be used with the Selenium JavaScript injection approach, allowing WebDriver access to browsers which it doesn't yet have a native automation driver for.

All of this basically means that:

- Selenium is still being developed
- The Selenium API you have started learning with this text will still exist and be usable in Selenium 2.0

The time you have spent learning Selenium should be an investment in time and you can look forward to continuing to learn more in the future.

If you want to experiment with 2.0...

Since we have built the SeleniumManager abstraction class to remove the details of selenium for us, we can easily start using Selenium 2.0 by making a few changes to this class.

You can find the official guide to migrating from Selenium 1.x to Selenium 2.0 on the Selenium web site: http://seleniumhq.org/docs/09_webdriver.html#emulating-selenium-rc

This small section is designed to help you take the first steps towards experimenting with Selenium 2.0 using the basic frameworks that we have explored in this book.

So first, download the Selenium 2.0 client drivers from the selenium web site:

- <http://code.google.com/p/selenium/downloads/list>

You want to download the selenium-java-2.x.zip where x is the most recent version of the archive:

- e.g. Selenium-java-2.0a4.zip

Extract this folder to somewhere on your drive.

- I used [c:\selenium2](#)

In your properties in Eclipse for the test project in the Libraries tab, [Add External Class Folder...] and choose the folder you extracted the archive to e.g.

- C:\selenium2\selenium-java-2.0a4

Now you only have a few changes to make to the Selenium Manager to allow you to use Selenium 2.0 instead of Selenium 1.x

- Add a new field in SeleniumManager called useSeleniumTwo

```
public class SeleniumManager {  
  
    SeleniumServer seleniumServer=null;  
    Selenium selenium=null;  
  
    String host;  
    String port;  
    String browser;  
    private boolean useSeleniumTwo=true;  
  
    ...  
}
```

- Manually change this from true to false depending on whether you want to use Selenium 2.x or not.
 - You can easily expand the class to have some helper methods to do this. See the SeleniumManager.java for an example
- Add code to return a WebDriverBackedSelenium instance instead of a Selenium 1.x instance

```
public void start(String baseUrl) throws IOException {  
  
    readProperties(); // get the properties from a file  
    readSystemProperties(); // allow system properties to override properties file  
  
    if(useSeleniumTwo){  
        WebDriver driver = new FirefoxDriver();  
        selenium = new WebDriverBackedSelenium(driver, baseUrl);  
        return;  
    }  
  
    ...  
}
```

- Use Eclipse to fix the include errors

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebDriverBackedSelenium;
import org.openqa.selenium.firefox.FirefoxDriver;
```

- The above includes should be added by Eclipse automatically as you fix the errors in the code, or you can add them manually.

The above is the minimum amount of code required to use the Firefox driver for Selenium 2.0 to have all tests which use the SeleniumManager running under Selenium 2.0

So now if you run any of the tests that you have created which use the SeleniumManager then they should run using Selenium 2.x instead of Selenium 1.x

You will find that not all tests run because there will be some subtle differences required when using Selenium 2.x, but hopefully the vast majority of the tests should run and pass successfully.

Selenium 2.x is still being worked on, but even now you can see that the migration path from Selenium 1.x to Selenium 2.x should not take an awful lot of work.

Note:

You can find more comprehensive code for using Selenium 2.0 in the SeleniumManager.java class.

It is commented out by default so as not to require everyone to amend the properties to reference Selenium 2.0

Chapter 38: In Closing

If you have worked your way through the book to this section then you should now have a good grasp on the basics creating production ready tests in Java using Selenium-RC with the Selenium 1.0 interface.

I know that readers have approached this book with very different levels of experience with programming, automated testing in general, and Selenium.

While I have tried to explain everything as well as I can, I still value feedback from readers to help me improve future versions of this text.

There is still a lot to learn, but you should have enough knowledge that you don't get phased by the prospect of learning more on your own.

This chapter I will suggest some avenues for additional learning, but the key way of learning Selenium is to use it.

So:

- write lots of tests,
- refactor your tests,
- keep learning java,
- keep an eye on new updates to selenium,
- read the Selenium source code

Additional Reading

For Selenium:

- read the official documentation on the Selenium website <http://seleniumhq.org/docs/>
- subscribe to the email feed for the official Selenium Users group <http://groups.google.com/group/selenium-users>
- subscribe to the RSS feed at the Selenium Stack exchange <http://area51.stackexchange.com/proposals/4693/selenium>

For Java:

- Books:
 - Agile Java: Crafting Code with Test-Driven Development by Jeff Langr
 - An excellent learning guide to Java which teaches Java using Test Driven development.
 - Implementation Patterns by Kent Beck
 - An overview of effective Java coding which is a tremendous confidence boost to beginners to Java that keeping your code simple is a perfectly valid coding style used by very experienced developers.
 - Effective Java by Joshua Bloch

- An overview of effective Java practices which speeds up your learning tremendously.
- Growing Object-oriented Software – guided by tests by Steve Freeman and Nat Pryce
 - An excellent overview of abstraction, refactoring and automated unit testing.

Videos:

- Watch the Eclipse and Java tutorials <http://eclipsetutorial.sourceforge.net/>
- Watch Google Tech Talks on Java and Selenium
- Watch the videos from previous Google Test Automation Conference

Web Searches:

- I use Google all the time to search for answers to my queries typically of the form “<thing I don't know> java example”

Subscribe to the following blogs:

- <http://element34.ca>
 - Adam Goucher's Selenium blog
- <http://seleniumhq.wordpress.com/>
 - The Official Selenium Blog
- <http://eviltester.com>
 - The author's testing blog which contains Selenium posts
- <http://www.theautomatedtester.co.uk/>
 - David Burns' testing blog. Author of “Selenium 1.0 Testing Tools: Beginner's Guide”
- <http://seleniumexamples.com/blog/>
 - Examples and tips for selenium
- <http://testingreflections.com/>
 - A Testing blogs aggregator where you will find many posts on testing and Selenium.

Use the mindmap:

- Use the mindmap in “Appendix A - Selenium API MindMap” as a summary of the commands and various categories the commands fit into.

Download and study the source-code for this book:

- Follow the instructions in Chapter “Play along at home” and download the source-code:
 - http://www.compendiumdev.co.uk/selenium/InitialSeleniumTests_JUNIT.zip
- Install a subversion client and checkout the refactored source-code from:
 - <http://svn2.xp-dev.com/svn/seleniumsimplified/trunk/FinalSeleniumTests>

Appendix A - Selenium API MindMap

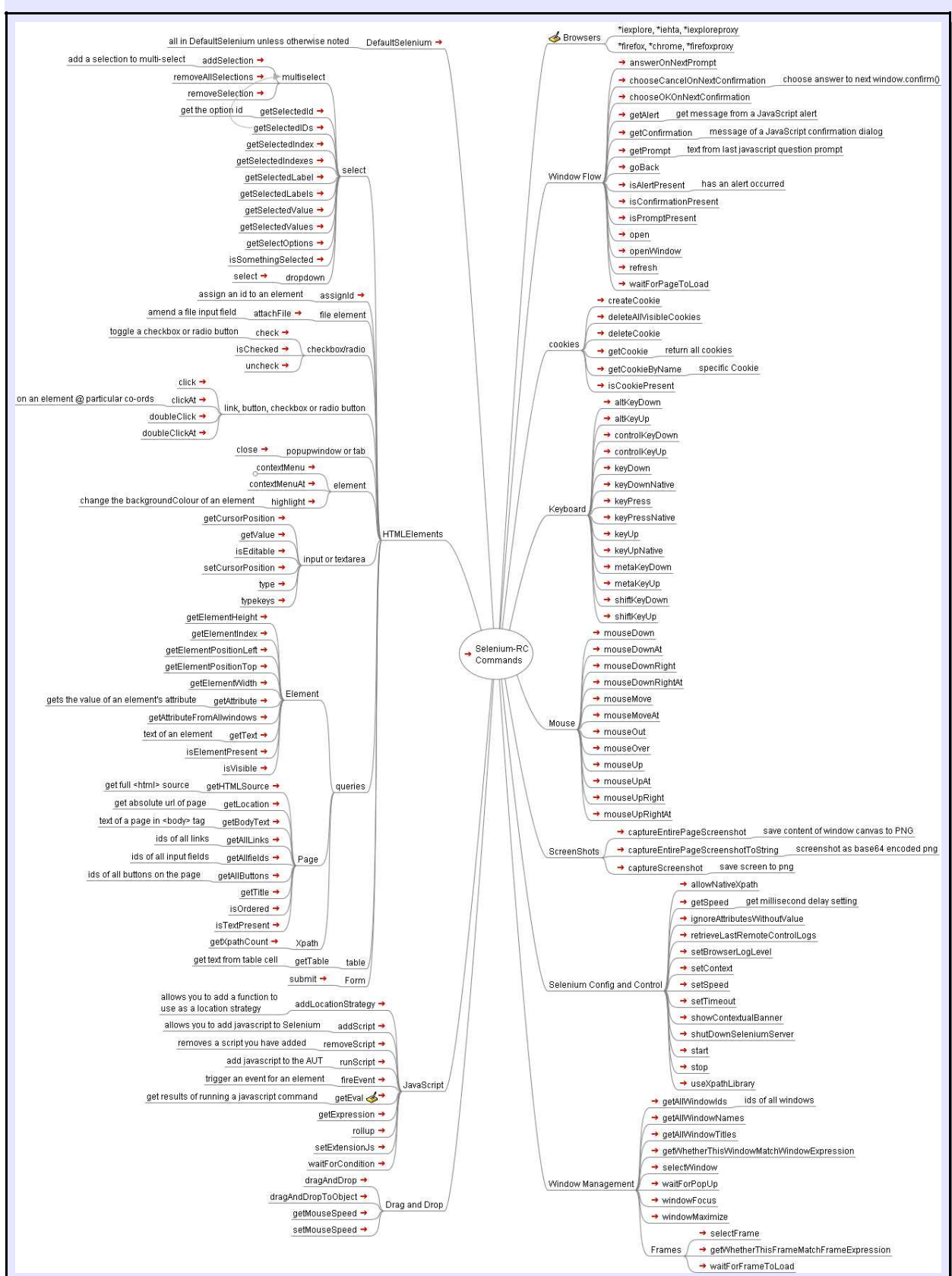


Figure 1 : Categorised Listing of the Selenium Commands