# Integration with Other Tools

In this chapter, we will cover:

- Configuring Eclipse and Maven for Selenium WebDriver test development
- Configuring IntelliJ IDEA and Maven for Selenium WebDriver test development
- Using Ant for Selenium WebDriver test execution
- Configuring Jenkins for continuous integration
- Using Jenkins and Maven for Selenium WebDriver test execution in continuous integration
- Using Jenkins and Ant for Selenium WebDriver test execution in continuous integration
- Configuring Microsoft Visual Studio for Selenium WebDriver test development
- Automating non-web UI in Selenium WebDriver with AutoIt
- Automating non-web UI in Selenium WebDriver with Sikuli

## Introduction

Selenium WebDriver has been widely used in combination with various tools due to its neat and clean object-oriented API. We can integrate Selenium WebDriver with other tools easily for developing testing frameworks.

The initial sections of this chapter explore Selenium WebDriver's integration with development and build tools such as Eclipse, IntelliJ IDEA, Maven, Ant, Microsoft Visual Studio, and Jenkins CI server. These tools provide an easy way to develop test automation frameworks and extend the capabilities of Selenium WebDriver API. The following recipes will explain how to set up and configure these tools with Selenium.

Lastly, we will explore how to automate non-web GUI using tools such as AutoIt and Sikuli with Selenium WebDriver. Both the tools are famous in the free and open source software world for automating user tasks and provide their own approaches for automating the GUI.

## Configuring Eclipse and Maven for Selenium WebDriver test development

Selenium WebDriver is a simple API that comes to your help for browser automation. However, when using it for testing and building a test framework, there is much more needed. You will need to integrate Selenium WebDriver API with different libraries, tools, and so on, for test development. You will need an Integrated Development Environment (IDE) to build your test project and inject other dependencies to build the framework.

Eclipse is a widely used IDE in the Java world. Eclipse provides a feature-rich environment for Selenium WebDriver test development.

Along with Eclipse, Apache Maven provides support for managing the entire lifecycle of a test project. Maven is used to define project structure, dependencies, build, and test management.

You can use Eclipse and Maven for building your Selenium WebDriver test framework from a single window. Another important benefit of using Maven is that you can get all the Selenium library files and their dependencies by configuring the `pom.xml` file. Maven automatically downloads the necessary files from the repository while building the project.

This recipe will explain how to configure Eclipse and Maven for the Selenium WebDriver test development. Most of the code in this book has been developed in Eclipse and Maven.

### Getting ready

1. Download and set up Maven from `http://maven.apache.org/download.html`.

   Follow the instructions on the Maven download page (see the *Installation Instructions* section on the page).

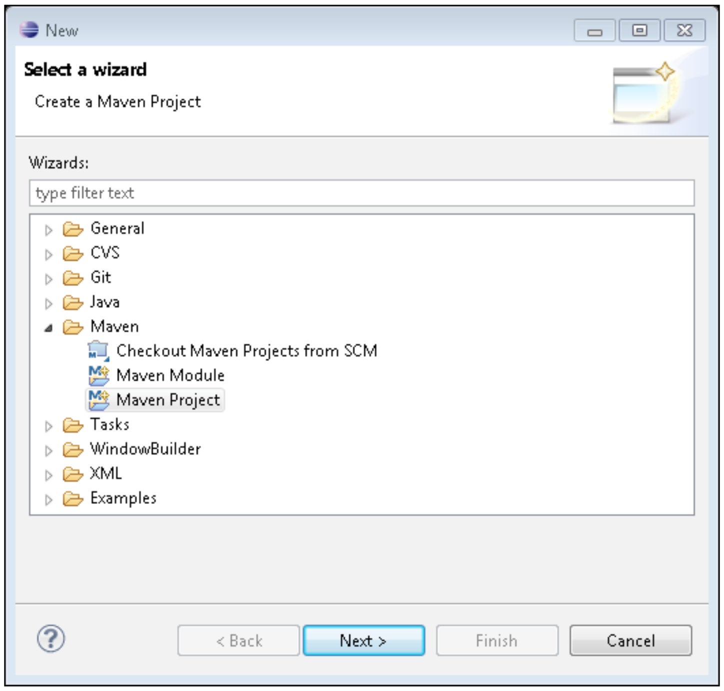
2. Download and set up Eclipse from `http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/junor`.

   For this recipe, Eclipse (Juno) IDE for Java Developers is used to set up Selenium WebDriver Project. This comes with the Maven plugin bundled with other packages.

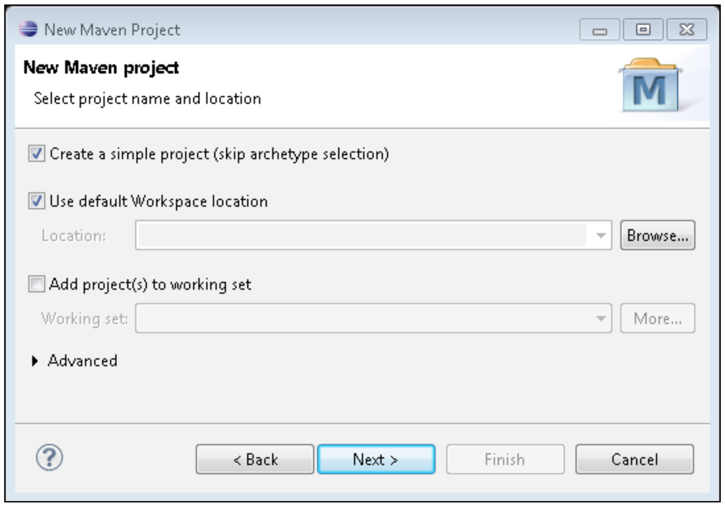
## How to do it...

Let's configure Eclipse with Maven for developing Selenium WebDriver tests using the following steps:

1. Launch the Eclipse IDE.
2. Create a new project by selecting **File |New | Other** from Eclipse Main Menu.
3. On the **New** dialog, select **Maven |Maven Project** as shown in the following screenshot:



4. Next, the **New Maven Project** dialog will be displayed. Select the **Create a simple project (skip archetype selection)** checkbox and set everything to default and click on the **Next** button as shown in the following screenshot:

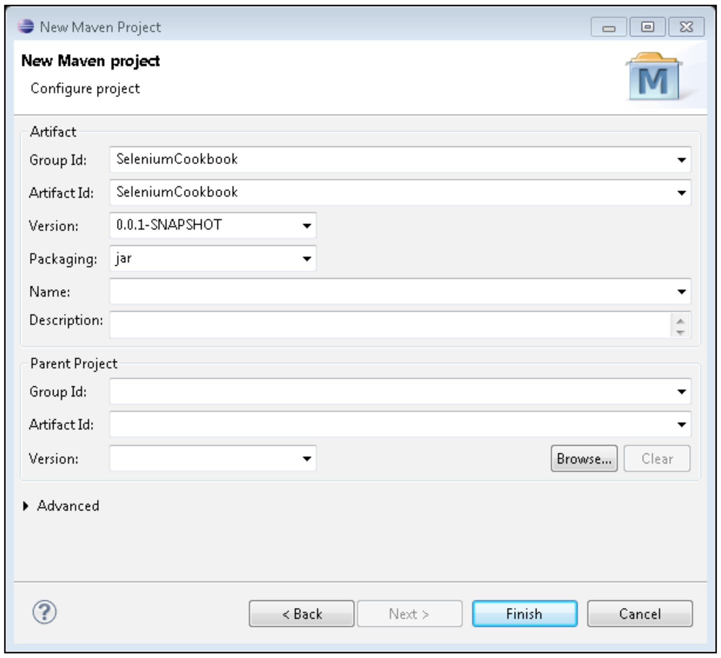

New Maven Project

**New Maven project**

Select project name and location

- [✓] Create a simple project (skip archetype selection)
- [✓] Use default Workspace location
- Location:
- [ ] Add project(s) to working set
- Working set:

5. On the **New Maven Project** dialog box, enter `SeleniumCookbook` in **Group Id:** and **Artifact Id:** textboxes. You can also add a name and description. Set everything to default and click on the **Finish** button as shown in the following screenshot:


New Maven Project

**New Maven project**

Configure project

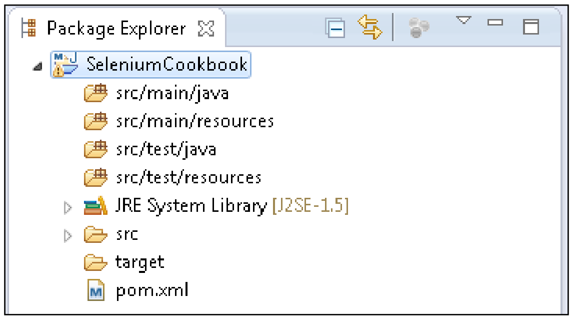
Artifact

| | |
|---|---|
| Group Id: | SeleniumCookbook |
| Artifact Id: | SeleniumCookbook |
| Version: | 0.0.1-SNAPSHOT |
| Packaging: | jar |
| Name: | |
| Description: | |

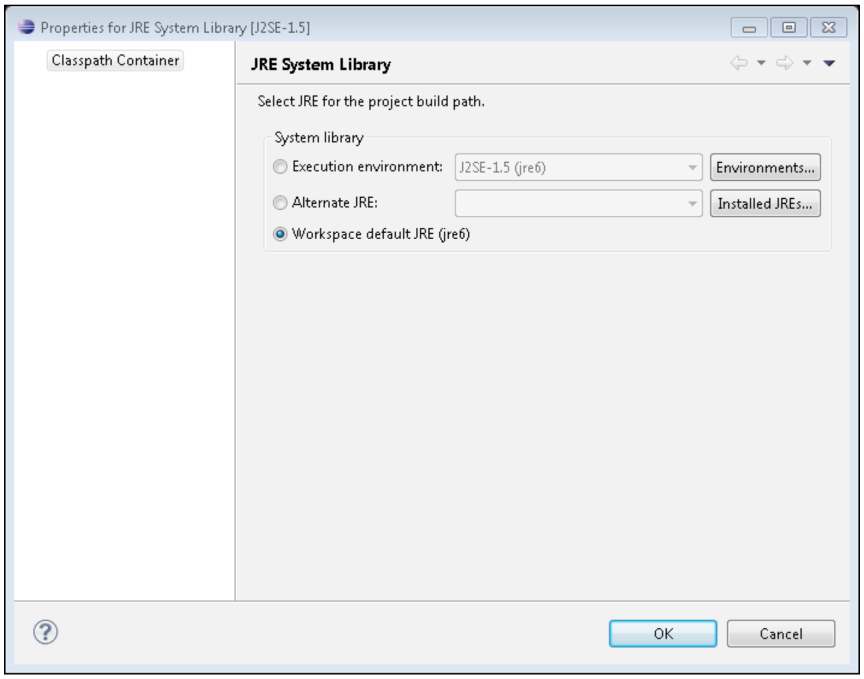
Parent Project

| | |
|---|---|
| Group Id: | |
| Artifact Id: | |
| Version: | |

6. Eclipse will create the **SeleniumCookbook** project with a structure (in **Package Explorer**) similar to the one shown in the following screenshot:



7. Right-click on **JRE System Library [J2SE-1.5]** and select the **Properties** option from the menu.
8. On the **Properties for JRE System Library [J2SE-1.5]** dialog box, make sure **Workspace default JRE (jre6)** is selected. If this option is not selected by default, select this option. The JRE version might change based on the Java version installed on your machine. Click on the **OK** button to save the change as shown in the following screenshot:



Select JRE for the project build path.

System library

Execution environment: J2SE-1.5 (jre6)

Alternate JRE:

[✓ Workspace default JRE (jre6)]

9. Select **pom.xml** from **Package Explorer**. This will open the `pom.xml` file in the editor area with the **Overview** tab open. Select the **pom.xml** tab instead.
10. Add the WebDriver and JUnit dependencies highlighted in the following code snippet, to `pom.xml` in the `<project>` node:

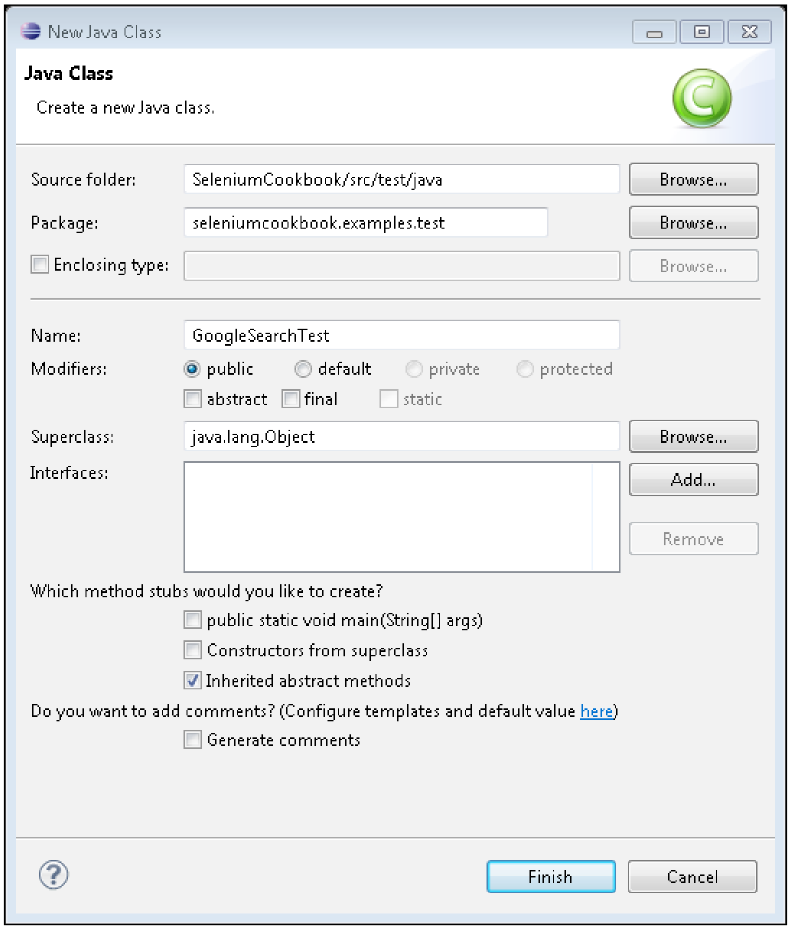
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>SeleniumCookbook</groupId>
    <artifactId>SeleniumCookbook</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>2.25.0</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

11. You can get the latest dependency information for Selenium WebDriver and JUnit from `http://seleniumhq.org/download/maven.html` and `http://maven.apache.org/plugins/maven-surefire-plugin/examples/junit.html` respectively.

[ TestNG is another widely used unit-testing framework in Java World. If you want to add TestNG support to the project instead of JUnit, you can get its Maven entry at `http://testng.org/doc/maven.html`. ]

12. Select **src/test/java** in **Package Explorer** and right-click for the menu. Select **New | Class**.

13. Enter `seleniumcookbook.examples.test` in the **Package:** textbox and `GoogleSearchTest` in the **Name:** textbox and click on the **Finish** button as shown in the following screenshot:



New Java Class

**Java Class**

Create a new Java class.

Source folder: SeleniumCookbook/src/test/java

Package: seleniumcookbook.examples.test

[ ] Enclosing type:

Name: GoogleSearchTest

Modifiers: [● public] ○ default ○ private ○ protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[ ] public static void main(String[] args)

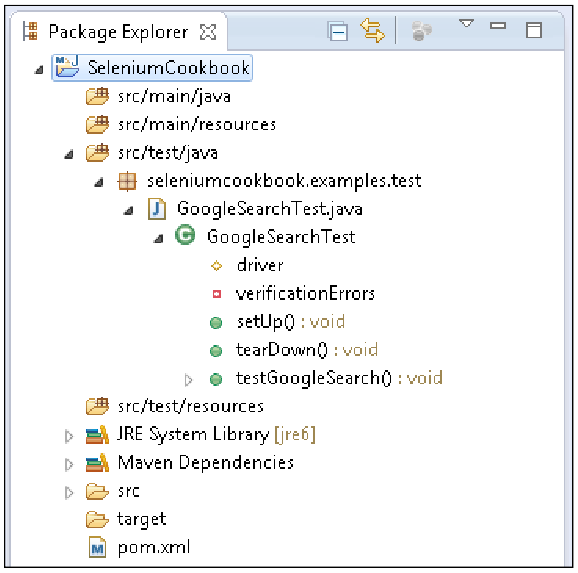
[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

14. This will create the **GoogleSearchTest** class as shown in the following screenshot:



Package Explorer

```
SeleniumCookbook
    src/main/java
    src/main/resources
    src/test/java
        seleniumcookbook.examples.test
            GoogleSearchTest.java
                GoogleSearchTest
                    driver
                    verificationErrors
                    setUp() : void
                    tearDown() : void
                    testGoogleSearch() : void
    src/test/resources
    JRE System Library [jre6]
    Maven Dependencies
    src
    target
    pom.xml
```

15. Add the following code in the `GoogleSearchTest` class:

```
package seleniumcookbook.examples.test;

import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.By;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.junit.*;
import static org.junit.Assert.*;

public class GoogleSearchTest {

    protected WebDriver driver;
    private StringBuffer verificationErrors =
        new StringBuffer();

    @Before
    public void setUp(){
        driver = new FirefoxDriver();
        driver.get("http://www.google.com");
    }

    @Test
    public void testGoogleSearch() {
        try {

            // Find the text input element by its name
            WebElement element =
                driver.findElement(By.name("q"));

            // Enter something to search for
            element.sendKeys
                ("Selenium testing tools cookbook");

            // Now submit the form. WebDriver will find
            //the form for us from the element
            element.submit();

            // Google's search is rendered dynamically
            //with JavaScript.
            // Wait for the page to load, timeout after
            //10 seconds
            (new WebDriverWait(driver, 10)).until
            (new ExpectedCondition<Boolean>() {
                public Boolean apply(WebDriver d) {
                    return d.getTitle().toLowerCase().
                        startsWith("selenium testing
                        tools cookbook");
                }
            });

            // Should see: selenium testing tools
            //cookbook - Google Search
            assertEquals("selenium testing tools cookbook
                - Google Search", driver.getTitle());
        } catch (Error e) {
            //Capture and append Exceptions/Errors
            verificationErrors.append(e.toString());
        }
    }

    @After
    public void tearDown() throws Exception {
        //Close the browser
```

```
            driver.quit();

            String verificationErrorString =
                verificationErrors.toString();
            if (!"".equals(verificationErrorString)) {
                fail(verificationErrorString);
            }
        }
    }
```

16. To run the tests in the Maven lifecycle, select the **SeleniumCookbook** project in **Package Explorer**. Right-click on the project name and select **Run As |Maven test**. Maven will execute all the tests from the project.

## How it works...

Eclipse provides the ability to create Selenium WebDriver test projects easily with its Maven plugin, taking away the pain of project configurations, directory structure, dependency management, and so on. It also provides a powerful code editor for writing the test code.

When you set up a project using Maven in Eclipse, it creates the `pom.xml` file, which defines the configuration of the project and its structure. This file also contains the dependencies needed for building, testing, and running the code. For example, the following shows dependency information about Selenium WebDriver that we added in `pom.xml`:

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>2.25.0</version>
</dependency>
```

Most of the open source projects publish this information on their website. In this case, you can check `http://seleniumhq.org/download/maven.html`; you can also get this information from Maven Central at `http://search.maven.org/#browse`. Maven will automatically download libraries and support files mentioned for all the dependencies and add to the project without you needing to find, download, and install these files to the project. This saves a lot of our time and effort while managing the dependency-related tasks.

Maven also generates a standard directory structure for your code for easier management and maintenance. In the previous example, it created the `src/test/java` folder for test code and the `src/test/resources` folder to maintain resources needed for testing, such as test data files, utilities, and so on.

Maven provides lifecycle steps such as building the test code and running the test. If you are working with the Java development team, then you might find the application code and test code together in Maven. Here, Maven supports building the application code, then firing the tests and releasing the software to production.

## There's more...

Maven can also be used to execute the test from the command line. To run tests from the command line, navigate to the `SeleniumCookbook` project folder through the command line and type the following command:

```
mvn clean test
```

This command will traverse through all the subdirectories and run the clean command to delete/remove earlier build files. It will then build the project and run the tests. You will see the results at the end of execution on command line.

## See also

- The *Configuring IntelliJ IDEA and Maven for Selenium WebDriver test development* recipe
- The *Using Jenkins and Maven for Selenium WebDriver test execution in continuous integration* recipe

## Configuring IntelliJ IDEA and Maven for Selenium WebDriver test development

IntelliJ IDEA is another very popular IDE available for Java development from JetBrains. Unlike Eclipse, IntelliJ IDEA is a commercial tool. However, JetBrains offers a community edition with limited features for free use.

This recipe will describe the steps for configuring IntelliJ IDEA and Maven for Selenium WebDriver test development.

## Getting ready

1. Download and install IntelliJ IDEA from `http://www.jetbrains.com/idea/download/index.html`.

   JetBrains offers IntelliJ IDEA Ultimate and Community editions. The Ultimate Edition is a commercial version of the product whereas the Community Edition is offered free, and only supports Java, Groovy, and Scala Projects. In this recipe, the Community Edition is used to set up the Selenium WebDriver project.

2. Download and install Maven from `http://maven.apache.org/download.html`.

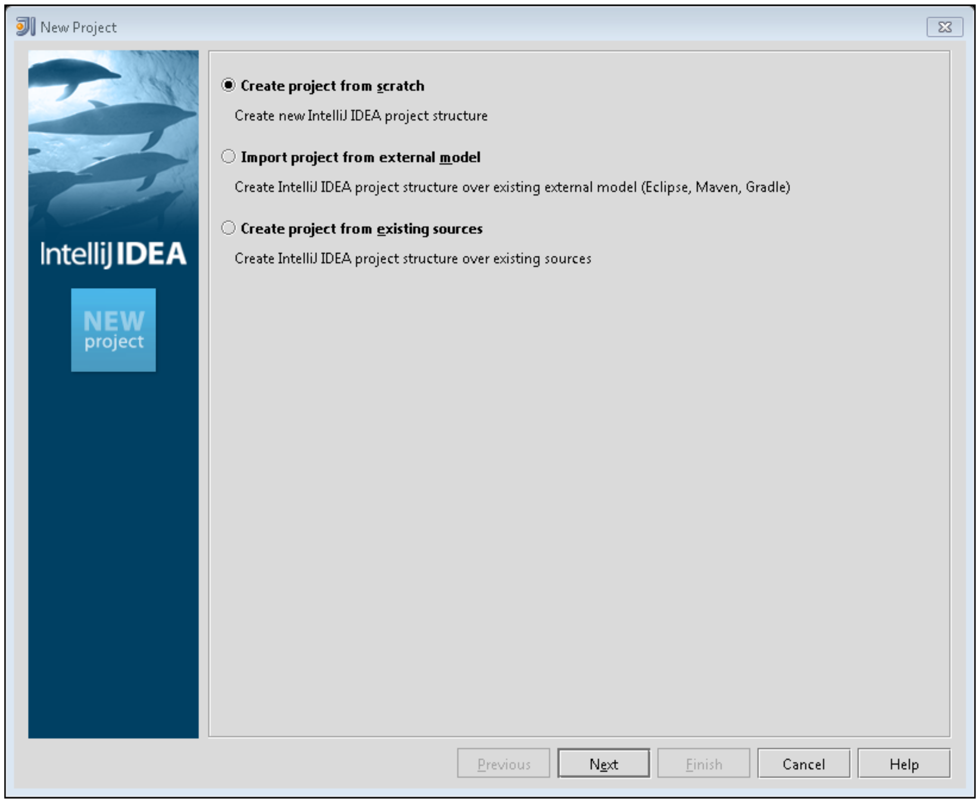
   Follow the instructions on the Maven download page (see the *Installation Instructions* section on the page).

[ If you are using IntelliJ IDEA for the first time, please configure JSDK by referring to the steps at `http://www.jetbrains.com/idea/webhelp/configuring-global-project-and-module-sdks.html`. ]

## How to do it...

Let's configure IntelliJ IDEA with Maven for developing Selenium WebDriver tests, using the following steps:

1. Launch the IntelliJ IDEA.
2. Create a new project by selecting **File | New Project** from the IntelliJ main menu. The **New Project** dialog box will be displayed as shown in the following screenshot:

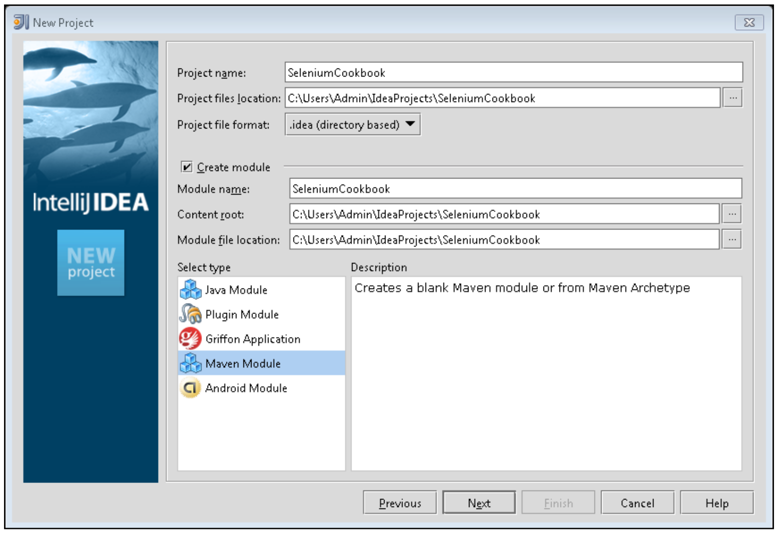


● **Create project from scratch**
Create new IntelliJ IDEA project structure

○ **Import project from external model**
Create IntelliJ IDEA project structure over existing external model (Eclipse, Maven, Gradle)

○ **Create project from existing sources**
Create IntelliJ IDEA project structure over existing sources

3. On the **New Project** dialog, select the **Create project from scratch** radio button and click on the **Next** button as shown in the following screenshot:



New Project

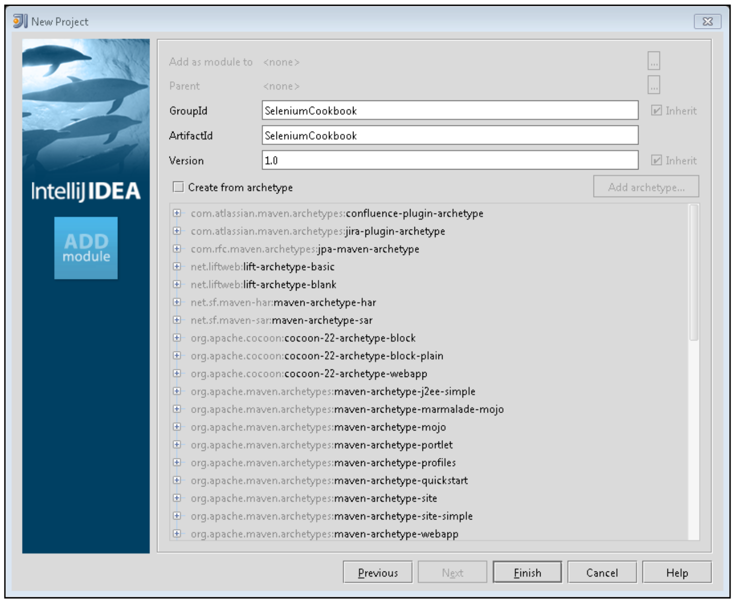
Project name: SeleniumCookbook
Project files location: C:\Users\Admin\IdeaProjects\SeleniumCookbook
Project file format: .idea (directory based)

☑ Create module
Module name: SeleniumCookbook
Content root: C:\Users\Admin\IdeaProjects\SeleniumCookbook
Module file location: C:\Users\Admin\IdeaProjects\SeleniumCookbook

Select type: Java Module, Plugin Module, Griffon Application, Maven Module (selected), Android Module

Description: Creates a blank Maven module or from Maven Archetype

4. On the **New Project** dialog, enter `SeleniumCookbook` in the **Project name:** textbox. Select the **Maven Module** option from the **Select type** list.
5. Leave the remaining options with their default value and click on the **Next** button as shown in the following screenshot:



New Project

Add as module to <none>
Parent <none>
GroupId: SeleniumCookbook ☑ Inherit
ArtifactId: SeleniumCookbook
Version: 1.0 ☑ Inherit

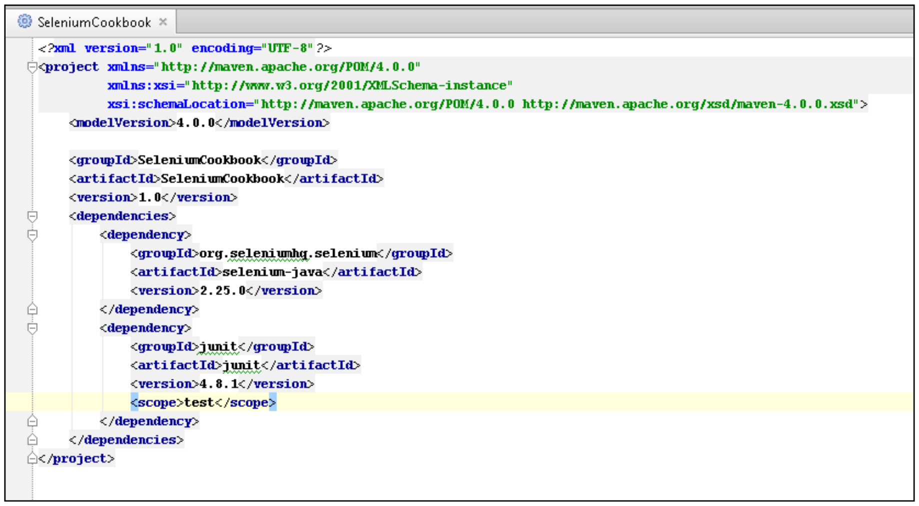
☐ Create from archetype

```
com.atlassian.maven.archetypes:confluence-plugin-archetype
com.atlassian.maven.archetypes:jira-plugin-archetype
com.rfc.maven.archetypes:jpa-maven-archetype
net.liftweb:lift-archetype-basic
net.liftweb:lift-archetype-blank
net.sf.maven-har:maven-archetype-har
net.sf.maven-sar:maven-archetype-sar
org.apache.cocoon:cocoon-22-archetype-block
org.apache.cocoon:cocoon-22-archetype-block-plain
org.apache.cocoon:cocoon-22-archetype-webapp
org.apache.maven.archetypes:maven-archetype-j2ee-simple
org.apache.maven.archetypes:maven-archetype-marmalade-mojo
org.apache.maven.archetypes:maven-archetype-mojo
org.apache.maven.archetypes:maven-archetype-portlet
org.apache.maven.archetypes:maven-archetype-profiles
org.apache.maven.archetypes:maven-archetype-quickstart
org.apache.maven.archetypes:maven-archetype-site
org.apache.maven.archetypes:maven-archetype-site-simple
org.apache.maven.archetypes:maven-archetype-webapp
```

6. Leave the remaining options to their default value and click on the **Finish** button.
7. A new project will be created with the `pom.xml` file open in the editor area.
8. Now add the following Selenium WebDriver and JUnit dependency in `pom.xml`.

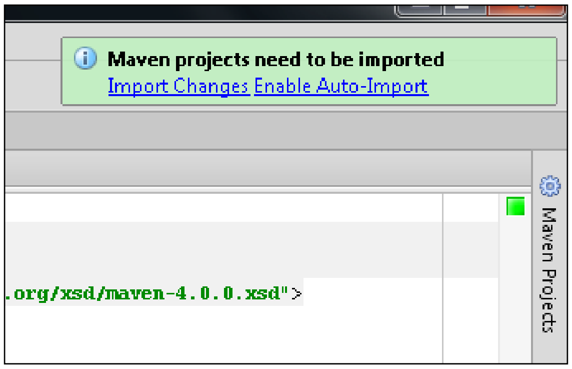
```
<dependencies>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>2.25.0</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.8.1</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

9. Your POM should look as shown in the following screenshot:



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>SeleniumCookbook</groupId>
    <artifactId>SeleniumCookbook</artifactId>
    <version>1.0</version>
    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>2.25.0</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

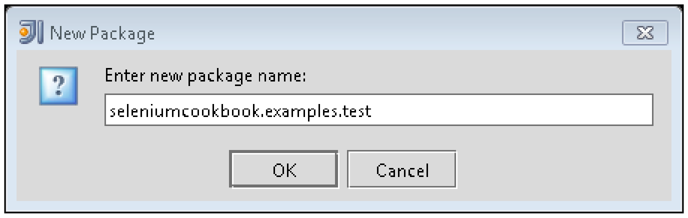
10. You may optionally get a **Maven projects need to be imported** message pop up displayed at the top-right corner as shown in the following screenshot:



Maven projects need to be imported
Import Changes Enable Auto-Import

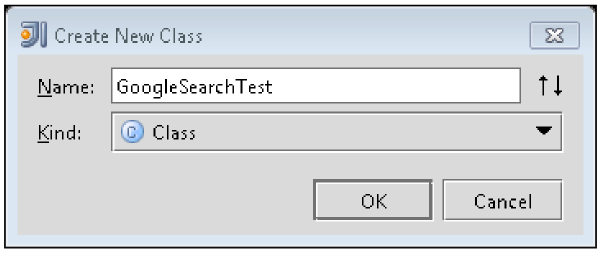
```
.org/xsd/maven-4.0.0.xsd">
```

11. Click on **Enable Auto-Import** on the **Maven projects need to be imported** message pop up.
12. In **Project Explorer**, select and right-click on the **test/java** folder. On the pop up menu, select **New | Package**. This will display a **New Package** dialog box as shown in the following screenshot:



Enter new package name: seleniumcookbook.examples.test

13. Add a new Java class to this newly created package by right-clicking on the **seleniumcookbook.example.test** package, and on the pop up menu select **New |Java Class**. This will display the **Create New Class** dialog as shown in the following screenshot:

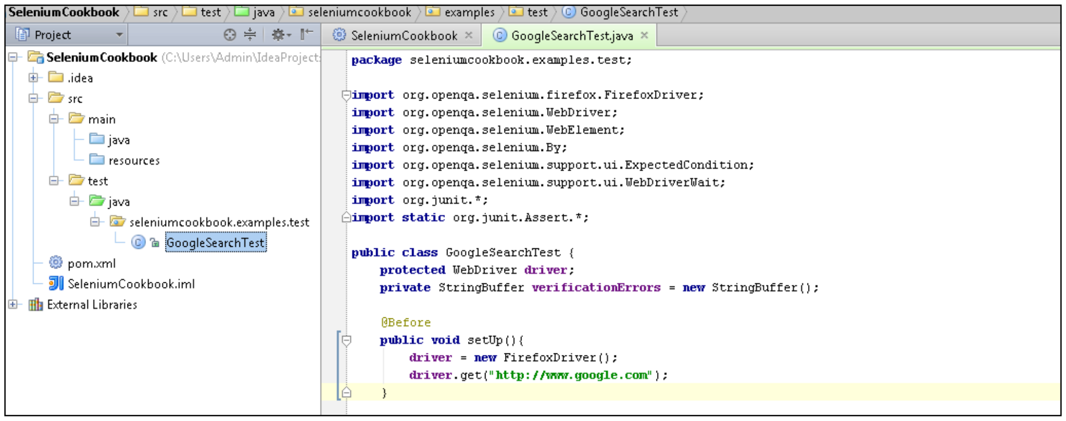


Name: GoogleSearchTest
Kind: Class

14. Enter `GoogleSearchTest` in the **Name:** textbox and click on the **OK** button.
15. The editor will create the `GoogleSearchTest` class in a new tab.
16. Copy the following code to the `GoogleSearchTest` class:

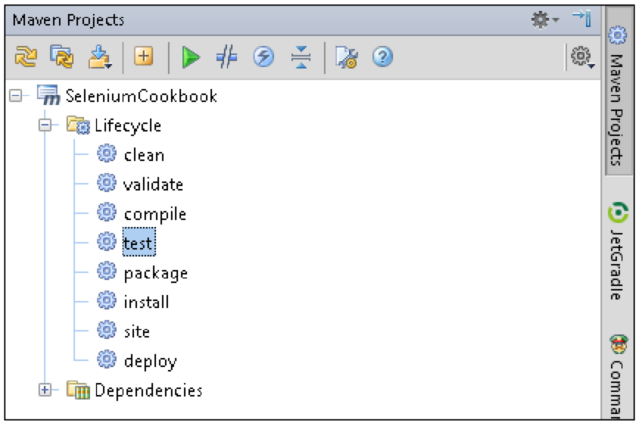
```
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.By;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.junit.*;
import static org.junit.Assert.*;

public class GoogleSearchTest {
    protected WebDriver driver;
    private StringBuffer verificationErrors =
        new StringBuffer();

    @Before
    public void setUp(){
        driver = new FirefoxDriver();
        driver.get("http://www.google.com");
    }

    @Test
    public void testGoogleSearch() {
        try {
            // Find the text input element by its name
            WebElement element =
                driver.findElement(By.name("q"));

            // Enter something to search for
            element.sendKeys("Selenium testing tools
                cookbook");

            // Now submit the form. WebDriver will find
           //the form for us from the element
            element.submit();

            // Google's search is rendered dynamically
            //with JavaScript.
            // Wait for the page to load, timeout after
            //10 seconds
            (new WebDriverWait(driver, 10)).until(new
                ExpectedCondition<Boolean>() {
```

```
                public Boolean apply(WebDriver d) {
                    return d.getTitle().toLowerCase().
                        startsWith("selenium testing
                        tools cookbook");
                }
            });

            // Should see: selenium testing tools
            //cookbook - Google Search
            assertEquals("selenium testing tools cookbook
                - Google Search", driver.getTitle());
        } catch (Error e) {
            //Capture and append Exceptions/Errors
            verificationErrors.append(e.toString());
        }
    }

    @After
    public void tearDown() throws Exception {
        //Close the browser
        driver.quit();

        String verificationErrorString =
            verificationErrors.toString();
        if (!"".equals(verificationErrorString)) {
            fail(verificationErrorString);
        }
    }
}
```


SeleniumCookbook > src > test > java > seleniumcookbook > examples > test > GoogleSearchTest
Project
SeleniumCookbook
GoogleSearchTest.java
SeleniumCookbook (C:\Users\Admin\IdeaProject
.idea
src
main
java
resources
test
java
seleniumcookbook.examples.test
GoogleSearchTest
pom.xml
SeleniumCookbook.iml
External Libraries

```
package seleniumcookbook.examples.test;

import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.By;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.junit.*;
import static org.junit.Assert.*;

public class GoogleSearchTest {
    protected WebDriver driver;
    private StringBuffer verificationErrors = new StringBuffer();

    @Before
    public void setUp(){
        driver = new FirefoxDriver();
        driver.get("http://www.google.com");
    }
```

17. Click on the **Maven Projects** option in the right sidebar. This will open the **Maven Projects** explorer with the **SeleniumCookbook** project. Select and expand the project. Select **Lifecycle** and then the **test** option to execute the previous test with Maven as shown in the following screenshot:



## How it works...

After Eclipse, IntelliJ IDEA is the widely used IDE available for developing software with Java. It provides a highly productive environment for test development with its advanced user interface, code editor, and IntelliSense features.

It also provides integration with Maven, Ant, and various other tools for project and build management and SCM tools for version control.

## See also

- The *Configuring Eclipse and Maven for Selenium WebDriver test development* recipe
- The *Using Jenkins and Maven for Selenium WebDriver test execution in continuous integration* recipe

## Using Ant for Selenium WebDriver test execution

Apache Ant is a popular build management tool available for Java developers. It is similar to Apache Maven, but does not support project management and dependency management features like Maven. It's a pure build tool.

Ant is another choice for running Selenium WebDriver tests from the command line or through **continuous integration** (**CI**) tools such as Jenkins.

In this recipe, we will add Ant support to the `SeleniumCookbook` project created in the *Configuring Eclipse and Maven for Selenium WebDriver test development* recipe.

## Getting ready

You can download and install WinAnt on Windows. WinAnt comes with an installer that will configure Ant through the installer. The WinAnt installer is available at `http://code.google.com/p/winant/`.

You can also download and configure Ant from `http://ant.apache.org/bindownload.cgi` for other OS platforms. This recipe uses WinAnt on the Windows OS.

## How to do it...

Let's set up the `SeleniumCookbook` project for Ant with following steps:

1. Create a `lib` folder and copy the JAR files for the dependencies used for this project, that is, Selenium WebDriver and JUnit.
2. Create the `build.xml` file in the project folder with the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="test" default="exec" basedir=".">

    <property name="src" value="./src" />
    <property name="lib" value="./lib" />
    <property name="bin" value="./bin" />
    <property name="report" value="./report" />
    <path id="test.classpath">
        <pathelement location="${bin}" />
        <fileset dir="${lib}">
            <include name="**/*.jar" />
        </fileset>
    </path>

    <target name="init">
        <delete dir="${bin}" />
        <mkdir dir="${bin}" />
    </target>

    <target name="compile" depends="init">
        <javac source="1.6" srcdir="${src}" fork="true"
            destdir="${bin}" >
            <classpath>
                        <pathelement path="${bin}">
                        </pathelement>
```

```
                <fileset dir="${lib}">
                    <include name="**/*.jar" />
                </fileset>
            </classpath>
        </javac>
    </target>

    <target name="exec" depends="compile">
        <delete dir="${report}" />
        <mkdir dir="${report}" />
            <mkdir dir="${report}/xml" />
        <junit printsummary="yes" haltonfailure="no">
            <classpath>
                <pathelement location="${bin}" />
                <fileset dir="${lib}">
                    <include name="**/*.jar" />
                </fileset>
            </classpath>

            <test name="seleniumcookbook.
                examples.test.GoogleSearchTest"
                haltonfailure="no" todir="${report}/xml"
                outfile="TEST-result">
                <formatter type="xml" />
            </test>
        </junit>
        <junitreport todir="${report}">
            <fileset dir="${report}/xml">
                <include name="TEST*.xml" />
            </fileset>
            <report format="frames"
                todir="${report}/html" />
        </junitreport>
    </target>
</project>
```

3. Navigate to the project directory through the command line and type the following command:

   **`ant`**

   This will trigger the build process. You will see the test running. At the end, Ant will create a `report` folder in the project folder. Navigate to the `html` subfolder in the `report` folder and open the `index.html` file to view the results.

### How it works...

Ant needs a `build.xml` file with all configurations and steps that are needed to build the project. We can add steps for report generation, sending e-mail notification, and so on to `build.xml`. Ant provides a very dynamic framework for defining steps in the build process.

Ant also needs the necessary library/JAR files to be copied in the `lib` folder, which are needed for building the project.

Ant scans for the entire tests in the project and executes these tests in a way similar to Maven.

### See also

- The *Using Jenkins and Ant for Selenium WebDriver test execution in continuous integration* recipe

## Configuring Jenkins for continuous integration

Jenkins is a popular continuous integration server in the Java development community. It is derived from the Hudson CI server. It supports SCM tools including CVS, Subversion, Git, Mercurial, Perforce, and ClearCase, and can execute Apache Ant and Apache Maven based projects as well as arbitrary shell scripts and Windows batch commands.

Jenkins can be deployed to set up an automated testing environment where you can run Selenium WebDriver tests unattended based on a defined schedule, or every time changes are submitted in SCM.

In this recipe, we will set up Jenkins Server for running Maven and Ant projects. Later recipes describe how Ant and Maven is used to run Selenium WebDriver tests with Jenkins.

### Getting ready

Download and install the Jenkins CI server from `http://jenkins-ci.org/`. For this recipe, the Jenkins Windows installer is used to set up Jenkins on a Windows 7 machine.

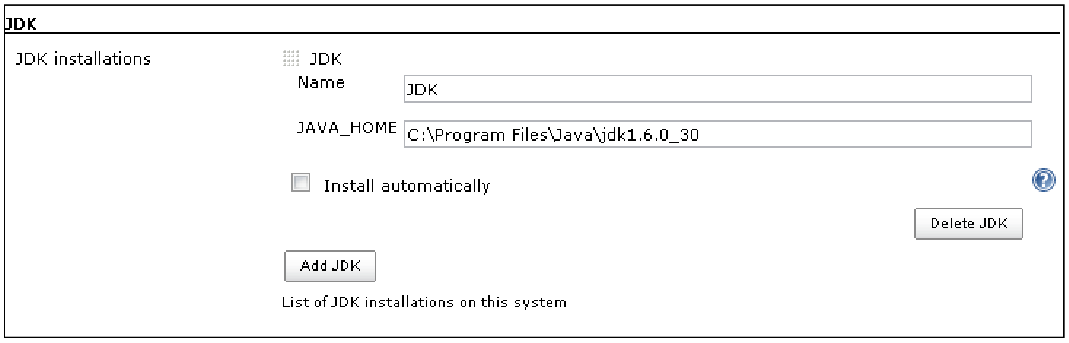
## How to do it...

Before using Jenkins, we need to set up the following options in the Jenkins configuration:

1. Navigate to the **Jenkins Dashboard** (`http://localhost:8080` by default) in the browser window.
2. On **Jenkins Dashboard**, click on the **Manage Jenkins** link.
3. On the **Manage Jenkins** page, click on the **Configure System** link.

### Adding JDK

1. On the **Configure System** page, locate the **JDK** section.
2. Click on the **Add JDK** button in the **JDK** section.
3. Specify `JDK6` in the **Name** field and unselect the **Install automatically** checkbox.
4. In the **JAVA_HOME** textbox, enter the path of the JDK folder from your system. In the following screenshot, `C:\Program Files\Java\jdk1.6.0_30` has been specified:



JDK

JDK installations

JDK

Name: JDK

JAVA_HOME: C:\Program Files\Java\jdk1.6.0_30

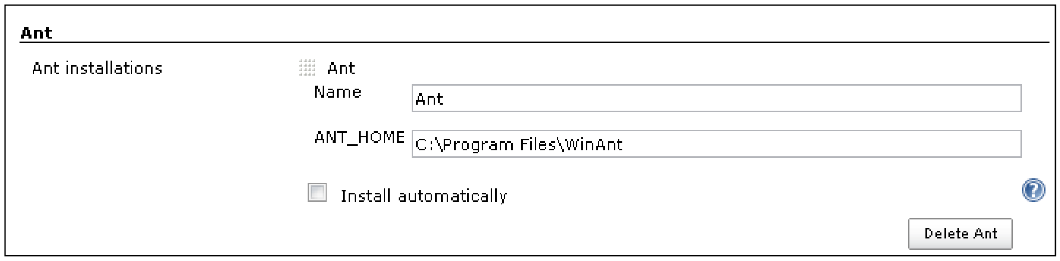
Install automatically [ ]

List of JDK installations on this system

### Adding Ant

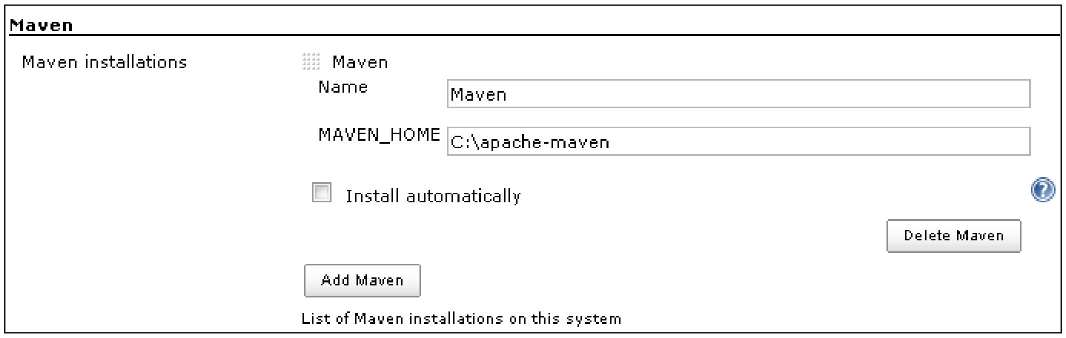
1. On the **Configure System** page, locate the **Ant** section.
2. Click on the **Add Ant** button in the **Ant** section.
3. Specify `Ant` in the **Name** field and unselect the **Install automatically** checkbox.

4. In the **ANT_HOME** textbox, enter the path of the Ant folder from your system. In the following screenshot, `C:\Program Files\WinAnt` has been specified for the WinAnt version:



## Adding Maven

1. On the **Configure System** page, locate the **Maven** section.
2. Click on the **Add Maven** button in **Maven** section.
3. Specify `Maven` in the **Name** field and unselect the **Install automatically** checkbox.
4. In the **MAVEN_HOME** textbox, enter the path of the Maven folder from your system. In the following screenshot, **MAVEN_HOME** contains **C:\apache-maven**:



5. Click on the **Save** button to save the configuration.

## There's more...

Jenkins also runs a Selenium standalone server which can be used as a remote web driver. Using Jenkins master/slave architecture, we can build a distributed build and test environment for large-scale test automation projects.

## See also

- The *Using Jenkins and Maven for Selenium WebDriver test execution in continuous integration* recipe
- The *Using Jenkins and Ant for Selenium WebDriver test execution in continuous integration* recipe

## Using Jenkins and Maven for Selenium WebDriver test execution in continuous integration

Jenkins supports Maven for building and testing a project in continuous integration. In this recipe, we will set up Jenkins to run tests from a Maven project.

### Getting ready

Running tests with Jenkins and Maven needs both the tools installed on the machine. In this recipe, the `SeleniumCookbook` project is used from the earlier *Configuring Eclipse and Maven for Selenium WebDriver test development* recipe.

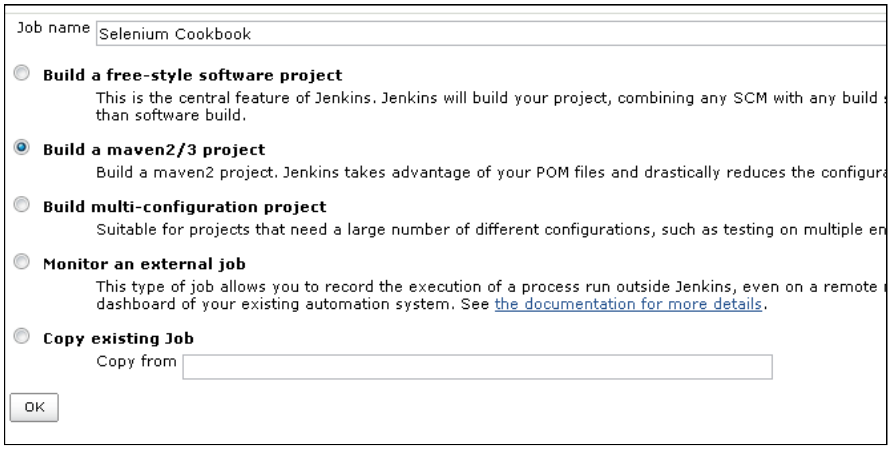
This recipe refers to Subversion as **Source Code Management** (**SCM**) tool for the `SeleniumCookbook` project.

You can use various SCM tools along with Jenkins. If Jenkins does not support a SCM tool that you are using, please check the Jenkins plugin directory for specific SCM tool plugins.

### How to do it...

1. Navigate to the **Jenkins Dashboard** (`http://localhost:8080` by default) in the browser window.
2. On **Jenkins Dashboard**, click on the **New Job** link to create a CI job.
3. Enter `Selenium Cookbook` in the **Job name** textbox.

4. Select the **Build a maven2/3 project** radio button as shown in the following screenshot:



Job name: Selenium Cookbook

( ) **Build a free-style software project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build
than software build.

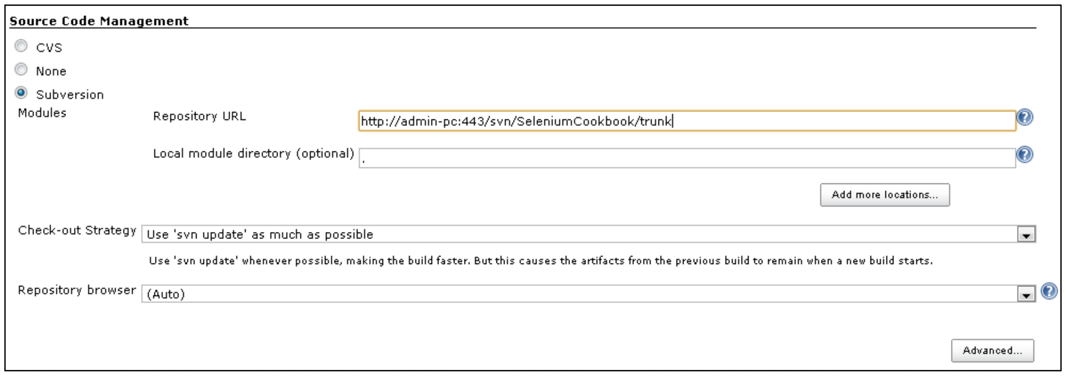
(●) **Build a maven2/3 project**
Build a maven2 project. Jenkins takes advantage of your POM files and drastically reduces the configura

( ) **Build multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple en

( ) **Monitor an external job**
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote
dashboard of your existing automation system. See the documentation for more details.

( ) **Copy existing Job**
Copy from

5. Click on **OK**.
6. A new job will be created with the specified name.
7. On the job configuration page, go to the **Source Code Management** section and select the **Subversion** radio button.
8. Enter the URL of your test code in the **Repository URL** textbox as shown in the following screenshot. Optionally, Jenkins will ask for Subversion login details. Provide user credentials as configured on your SVN server.



**Source Code Management**

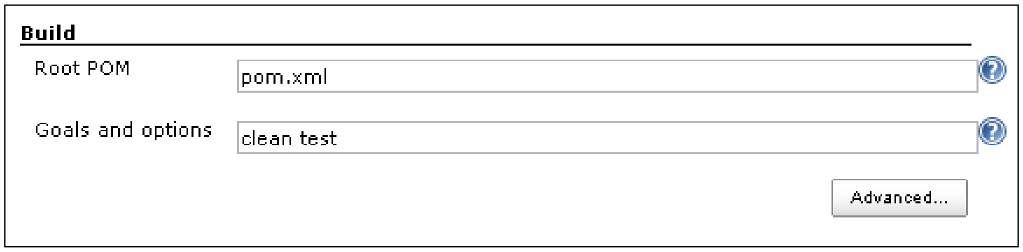
( ) CVS
( ) None
(●) Subversion

Modules
Repository URL: http://admin-pc:443/svn/SeleniumCookbook/trunk
Local module directory (optional): .

Check-out Strategy: Use 'svn update' as much as possible
Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.

Repository browser: (Auto)

9. Go to the **Build** section. In the **Root POM** textbox, enter `pom.xml` and in the **Goals and options** textbox, enter `clean test` as shown in the following screenshot:

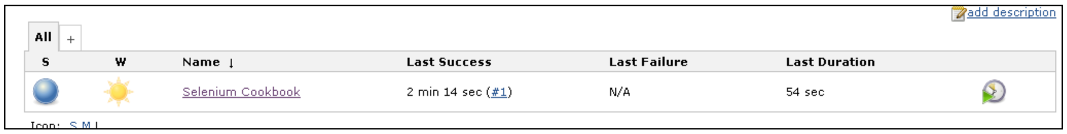


**Build**

| | |
|---|---|
| Root POM | pom.xml |
| Goals and options | clean test |

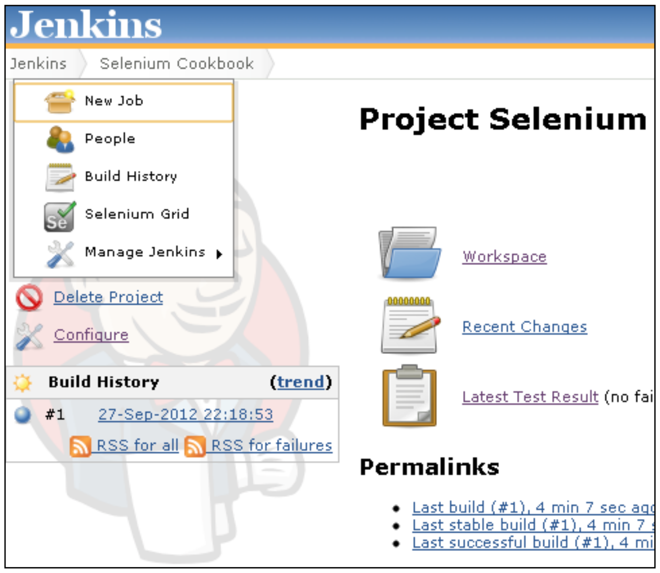
10. On the **Selenium Cookbook** project page, click on the **Build Now** link. Go back to the **Jenkins Dashboard**.

11. Maven builds the project and executes tests from the project. Once the build process is completed, click on the **Selenium Cookbook** project from the list as shown in the following screenshot:



| S | W | Name ↓ | Last Success | Last Failure | Last Duration |
|---|---|---|---|---|---|
| ● | ☀ | Selenium Cookbook | 2 min 14 sec (#1) | N/A | 54 sec |

The **Selenium Cookbook** project page displays the build history and links to the results as shown in the following screenshot:



**Project Selenium**

**Build History** (trend)

#1 27-Sep-2012 22:18:53

RSS for all RSS for failures

Workspace

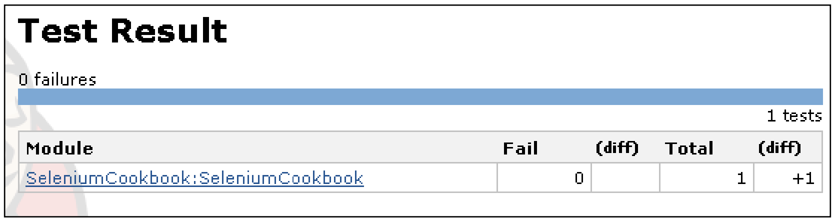
Recent Changes

Latest Test Result (no fai

**Permalinks**

- Last build (#1), 4 min 7 sec ago
- Last stable build (#1), 4 min 7
- Last successful build (#1), 4 mi

12. Click on the **Latest Test Result** link to view the test results as shown in the following screenshot:



**Test Result**

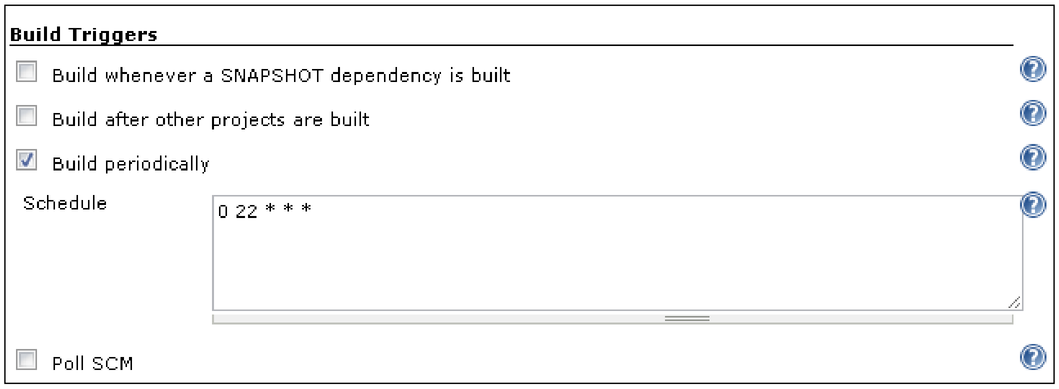
0 failures

1 tests

| Module | Fail | (diff) | Total | (diff) |
|---|---|---|---|---|
| SeleniumCookbook:SeleniumCookbook | 0 |  | 1 | +1 |

## Scheduling build for automatic execution

1. Go to the **Selenium Cookbook** project configuration in Jenkins.
2. In the **Build Triggers** section, select the **Build periodically** checkbox.
3. Enter `0 22 * * *` in the **Schedule** textbox as shown in the following screenshot. This will trigger the build process every day at 10 p.m. and the test will run unattended.



**Build Triggers**

- [ ] Build whenever a SNAPSHOT dependency is built
- [ ] Build after other projects are built
- [x] Build periodically

Schedule: 0 22 * * *

- [ ] Poll SCM

4. Click on the **Save** button to save the configurations.

## How it works...

Using the **Build a Maven2/3 Project** option, Jenkins supports building and testing Maven projects.

Jenkins supports various SCM tools such as CVS, and Subversion. To get the source code from SCM, specify the repository location and check out the strategy. Since Maven is used in this example, specify the path of root POM and Maven Goal.

While building the project, Jenkins gets the latest source from SCM to the Jenkins project workspace. It will then call Maven with specified goals. When the build process is complete, Jenkins gets the test results from Maven and displays these results on the project page.

### Scheduling builds

One of the important features of Jenkins is that it automatically triggers the build, based on defined criteria. Jenkins provides multiple ways to trigger the build process under the Build Trigger configuration. Build Trigger can be set at a specific time. In the previous example, it is set to trigger the process every day at 10 p.m. This provides the ability to run tests unattended, nightly, so that you can see the results the next morning.

### Test results

Jenkins provides the ability to display test results by reading the results files generated by unit test frameworks. It also archives these results, which can be used to generate various metrics over time.

## See also

- The *Configuring Jenkins for continuous integration* recipe

# Using Jenkins and Ant for Selenium WebDriver test execution in continuous integration

Ant can also be configured to run tests in continuous integration with Jenkins. In this recipe, we will set up Jenkins to run tests with Ant.

## Getting ready

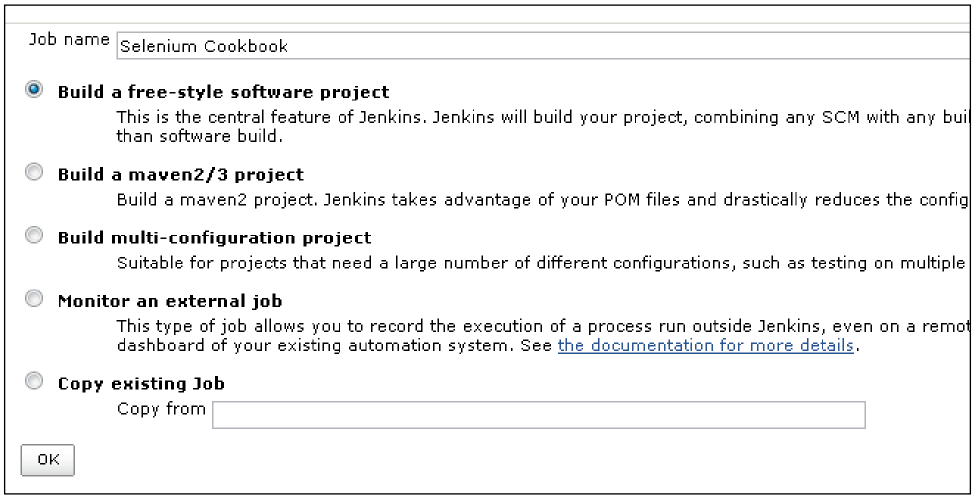
Running tests with Jenkins and Ant needs both the tools installed on the machine. Refer to the *Using Ant for Selenium WebDriver test execution* and *Configuring Jenkins for continuous integration* recipes to install and configure Ant and Jenkins.

## How to do it...

Let's configure Jenkins and Ant for running tests in CI:

1. Navigate to the **Jenkins Dashboard** (`http://localhost:8080` by default) in the browser window.
2. On **Jenkins Dashboard**, click on the **New Job** link to create a CI job.
3. Enter `Selenium Cookbook` in the **Job name:** textbox.

4. Select the **Build a free-style software project** radio button as shown in the following screenshot:



Job name: Selenium Cookbook

(●) **Build a free-style software project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any bui
than software build.

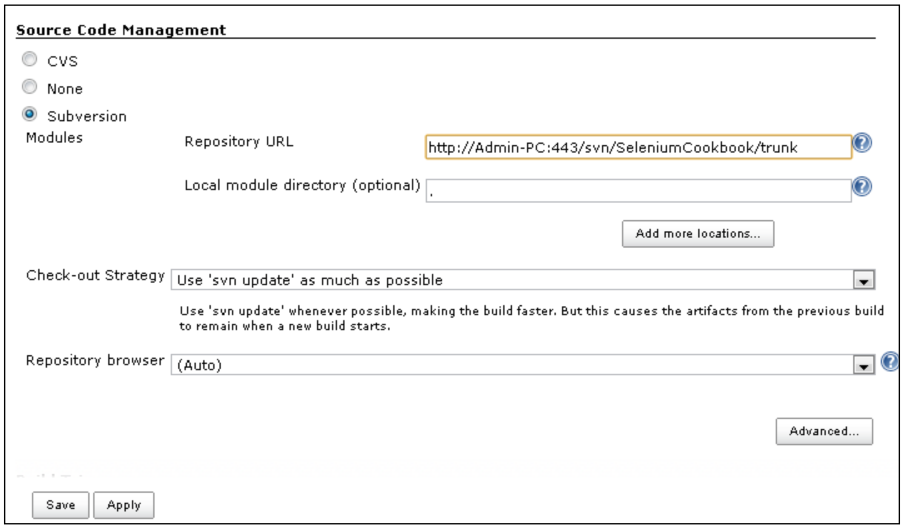
( ) **Build a maven2/3 project**
Build a maven2 project. Jenkins takes advantage of your POM files and drastically reduces the config

( ) **Build multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple

( ) **Monitor an external job**
This type of job allows you to record the execution of a process run outside Jenkins, even on a remot
dashboard of your existing automation system. See the documentation for more details.

( ) **Copy existing Job**
Copy from

5. Click on **OK**.
6. A new job will be created with the **Selenium Cookbook** name.
7. On the job configuration page, go to the **Source Code Management** section and select the **Subversion** radio button.
8. Enter the URL of your test code in the **Repository URL** textbox as shown in the following screenshot. Optionally, Jenkins will ask for Subversion login details. Provide user credentials as configured on your SVN server.



**Source Code Management**

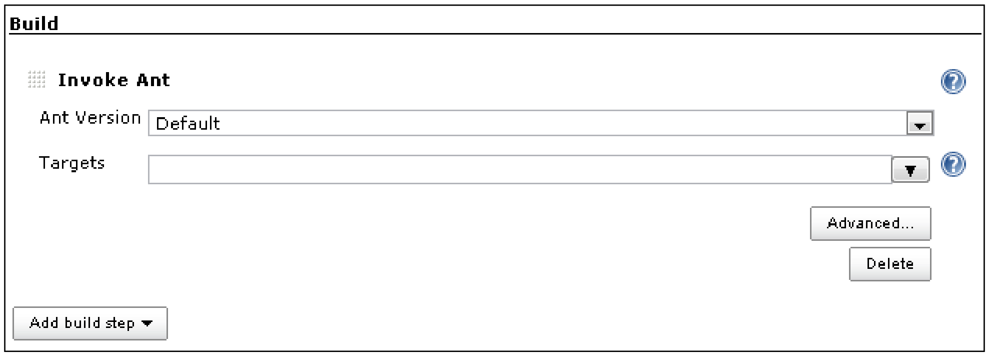
( ) CVS
( ) None
(●) Subversion

Modules
Repository URL: http://Admin-PC:443/svn/SeleniumCookbook/trunk
Local module directory (optional): .

Check-out Strategy: Use 'svn update' as much as possible
Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.

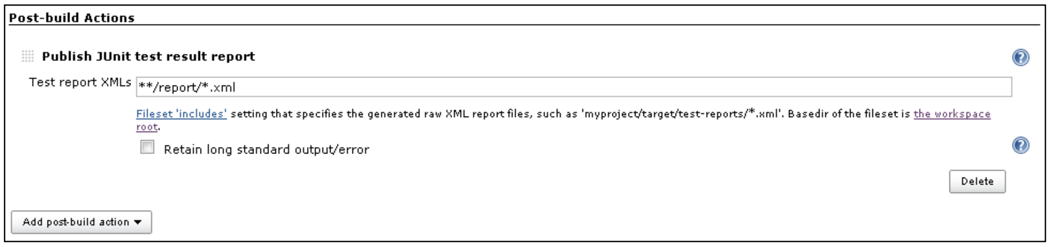
Repository browser: (Auto)

9. Go to the **Build** section. Click on the **Add build step** button once again and select **Invoke Ant** option from the drop-down list.
10. The **Ant version** textbox will display **Default** as shown in the following screenshot:



Ant Version: Default
Targets:

11. Go to the **Post-build Actions** section.
12. Click on the **Add post-build action** button and select **Publish JUnit test result report** from the drop-down list.
13. In the **Test report XMLs** textbox, enter `**/report/*.xml` as shown in the following screenshot:

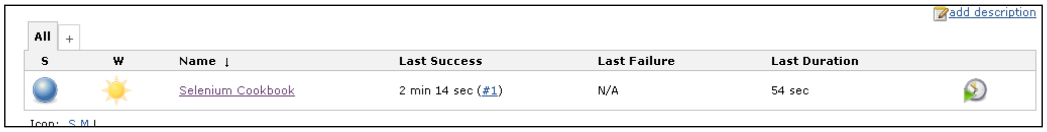


Test report XMLs: **/report/*.xml

Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/*.xml'. Basedir of the fileset is the workspace root.

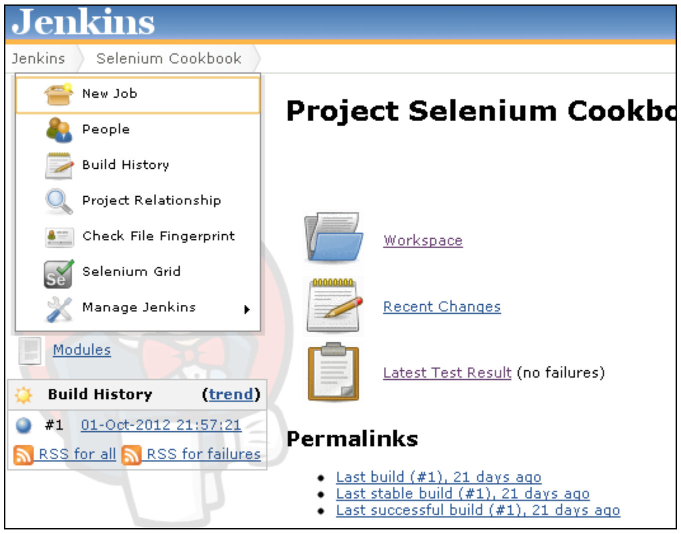
☐ Retain long standard output/error

14. Click on the **Save** button to save the configuration.
15. Go back to the **Jenkins Dashboard** page.
16. Click on the **Schedule a Build** button to trigger the build. Ant will execute the test. Once the build process is completed, click on the **Selenium Cookbook** project from the list as shown in the following screenshot:

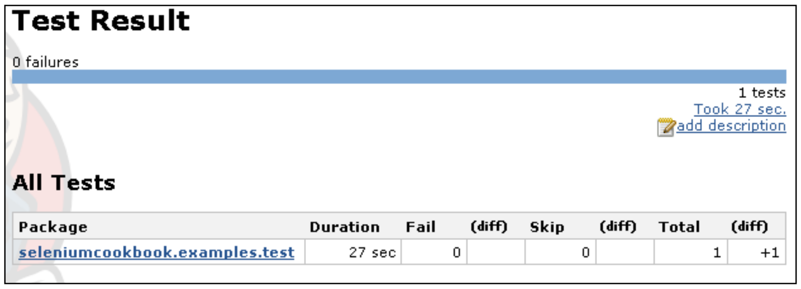


| S | W | Name ↓ | Last Success | Last Failure | Last Duration |
|---|---|---|---|---|---|
| ● | ☀ | Selenium Cookbook | 2 min 14 sec (#1) | N/A | 54 sec |

The project page displays the build history and links to the results as shown in the following screenshot:



17. Click on the **Latest Test Result** link to view the test results as shown in the following screenshot:

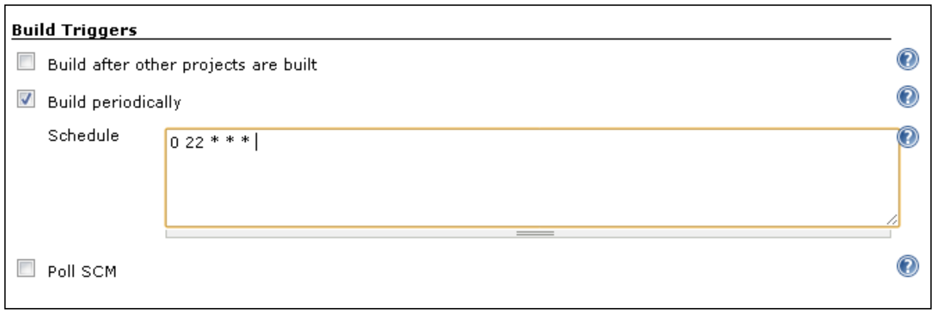


| Package | Duration | Fail | (diff) | Skip | (diff) | Total | (diff) |
|---|---|---|---|---|---|---|---|
| seleniumcookbook.examples.test | 27 sec | 0 | | 0 | | 1 | +1 |

## Scheduling build for automatic execution

1. Go to the Selenium Cookbook project configuration in Jenkins.
2. In the **Build Triggers** section, select the **Build periodically** checkbox.

3. Enter `0 22 * * *` in the **Schedule** textbox as shown in the following screenshot. This will trigger the build process every day at 10 p.m.



**Build Triggers**

- [ ] Build after other projects are built
- [✓] Build periodically

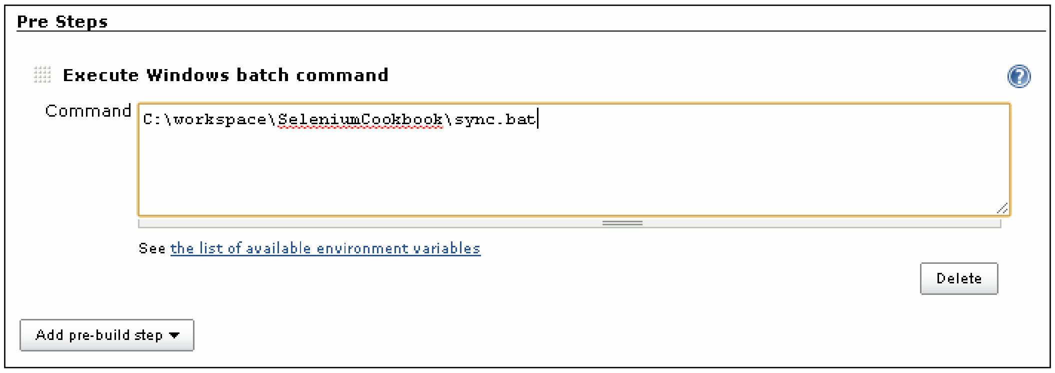
Schedule: `0 22 * * *`

- [ ] Poll SCM

4. Click on the **Save** button to save the configurations.

## There's more...

The preceding example uses SCM to get the test source code to the Jenkins workspace. However, if you are not using SCM, you can copy the code from the source folders to the Jenkins workspace, so that it gets the latest version of test source code. This can be done by writing a batch or a shell script that will clean the Jenkins project workspace and copy the latest test code to the workspace. For running this batch file, use the **None** option in the **Source Code Management** section and specify the batch file path in **Pre Steps | Execute Windows batch command** or the **Execute shell** option as shown in the following screenshot:



**Pre Steps**

**Execute Windows batch command**

Command:
```
C:\workspace\SeleniumCookbook\sync.bat
```

See the list of available environment variables

## See also

- The *Using Ant for Selenium WebDriver test execution* recipe
- The *Configuring Jenkins for continuous integration* recipe

## Configuring Microsoft Visual Studio for Selenium WebDriver test development

Selenium WebDriver provides .NET bindings for developing Selenium tests with the .NET platform. For using the Selenium WebDriver API, you need to reference Selenium WebDriver libraries to the project. Microsoft Visual Studio being the major IDE used in the .NET world, setting up the Selenium WebDriver support has become easier with NuGet Package Manager (`http://nuget.org/`).

This recipe explains setting up Selenium WebDriver in Microsoft Visual Studio 2012 using NuGet.

### Getting ready

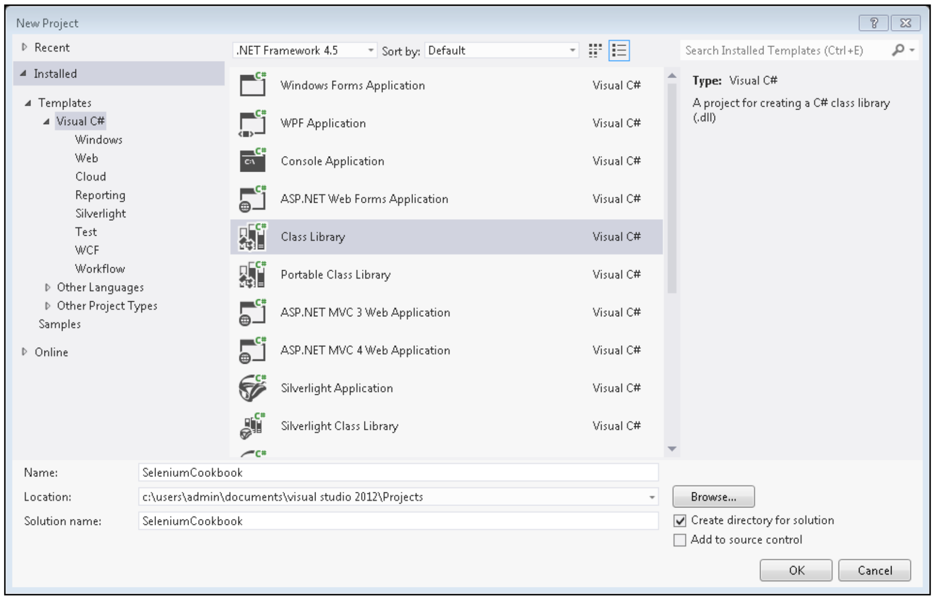
NuGet comes bundled with Microsoft Visual Studio 2012. However, for Microsoft Visual Studio 2010, download and install NuGet from `nuget.codeplex.com`.

### How to do it...

Let's configure Microsoft Visual Studio 2012 for developing Selenium WebDriver tests, using the following steps:

1. Launch the Microsoft Visual Studio.
2. Create a new project by selecting **File |New | Project** from the main menu.

3. On the **New Project** dialog box, select **Visual C# | Class Library Project**. Name the project as **SeleniumCookbook** and click on the **OK** button as shown in the following screenshot:


New Project
Recent
Installed
Templates
Visual C#
Windows
Web
Cloud
Reporting
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Samples
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
ASP.NET Web Forms Application Visual C#
Class Library Visual C#
Portable Class Library Visual C#
ASP.NET MVC 3 Web Application Visual C#
ASP.NET MVC 4 Web Application Visual C#
Silverlight Application Visual C#
Silverlight Class Library Visual C#
Browse...
OK
Cancel

Type: Visual C#

A project for creating a C# class library (.dll)

Name: SeleniumCookbook

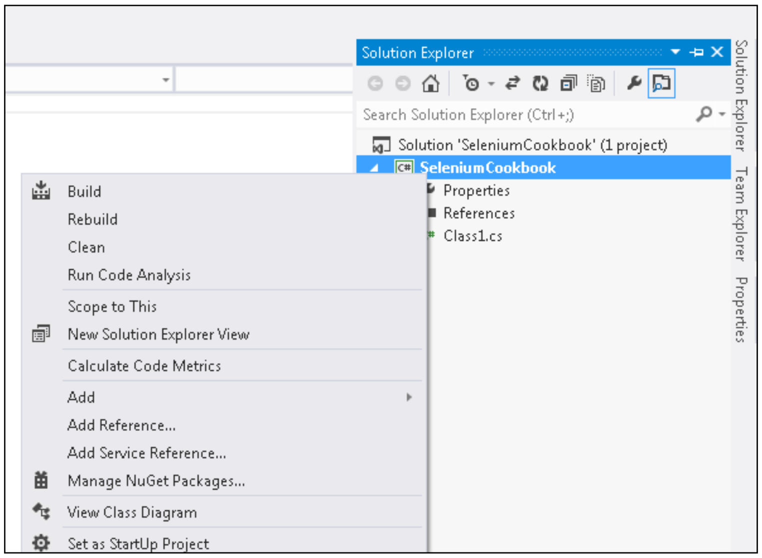
Location: c:\users\admin\documents\visual studio 2012\Projects

Solution name: SeleniumCookbook

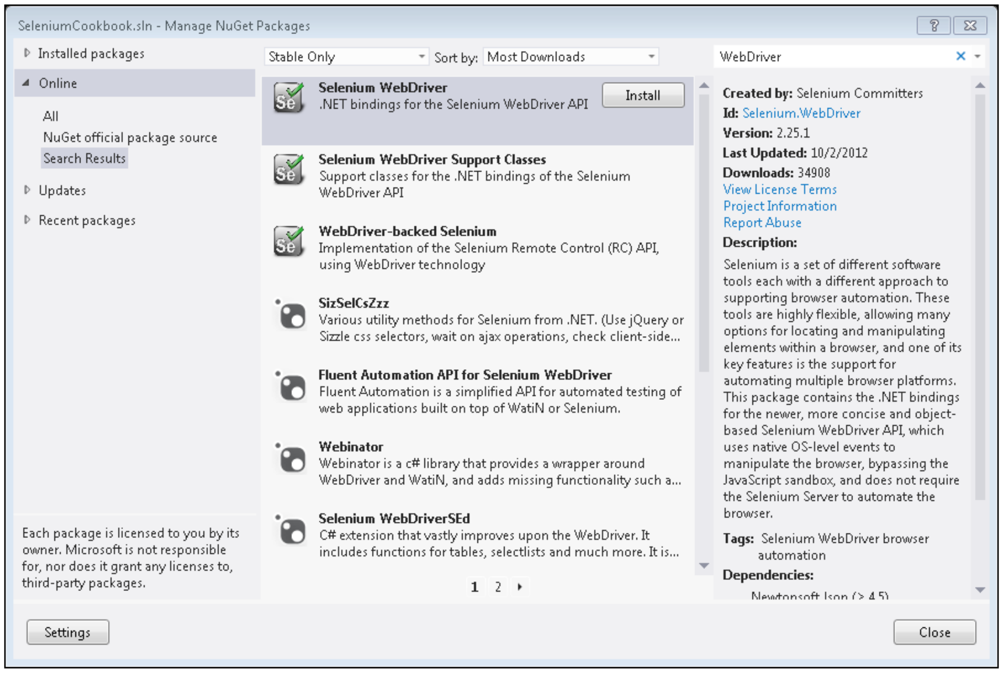
☑ Create directory for solution

☐ Add to source control

4. Next, add WebDriver and NUnit using NuGet. Right-click on the **SeleniumCookbook** solution in **Solution Explorer** and select **Manage NuGet Packages...** as shown in the following screenshot:



5. On the **SeleniumCookbook.sln - Manage NuGet Packages** dialog box, select **Online** and search for the `WebDriver` package. The search will result in the following suggestions:



**Selenium WebDriver**
.NET bindings for the Selenium WebDriver API

**Selenium WebDriver Support Classes**
Support classes for the .NET bindings of the Selenium WebDriver API

**WebDriver-backed Selenium**
Implementation of the Selenium Remote Control (RC) API, using WebDriver technology

**SizSelCsZzz**
Various utility methods for Selenium from .NET. (Use jQuery or Sizzle css selectors, wait on ajax operations, check client-side...

**Fluent Automation API for Selenium WebDriver**
Fluent Automation is a simplified API for automated testing of web applications built on top of WatiN or Selenium.

**Webinator**
Webinator is a c# library that provides a wrapper around WebDriver and WatiN, and adds missing functionality such a...

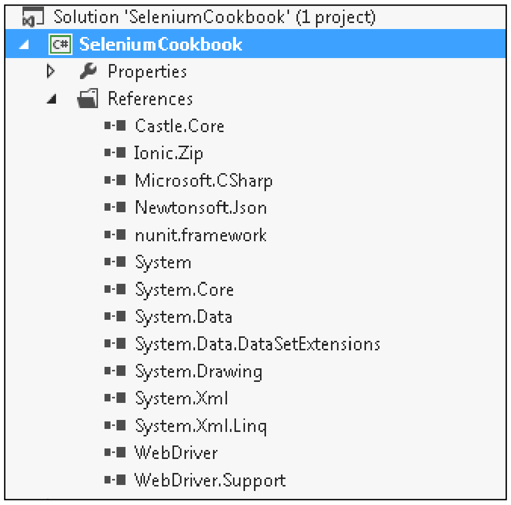
**Selenium WebDriverSEd**
C# extension that vastly improves upon the WebDriver. It includes functions for tables, selectlists and much more. It is...

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

**Created by:** Selenium Committers
**Id:** Selenium.WebDriver
**Version:** 2.25.1
**Last Updated:** 10/2/2012
**Downloads:** 34908
View License Terms
Project Information
Report Abuse
**Description:**
Selenium is a set of different software tools each with a different approach to supporting browser automation. These tools are highly flexible, allowing many options for locating and manipulating elements within a browser, and one of its key features is the support for automating multiple browser platforms. This package contains the .NET bindings for the newer, more concise and object-based Selenium WebDriver API, which uses native OS-level events to manipulate the browser, bypassing the JavaScript sandbox, and does not require the Selenium Server to automate the browser.
**Tags:** Selenium WebDriver browser automation
**Dependencies:**
Newtonsoft.Json (≥ 4.5)

6. Select **Selenium WebDriver** from the list and click on the **Install** button. Repeat this step for **Selenium WebDriver Support Classes**.
7. Next, search for the **NUnit** package on the **SeleniumCookbook.sln - Manage NuGet Packages** dialog box.
8. Select **NUnit** from the search result and click on the **Install** button.
9. Close the **SeleniumCookbook.sln - Manage NuGet Packages** dialog box.

10. Expand the **References** tree for **SeleniumCookbook** solution in **Solution Explorer**. References for WebDriver and NUnit is added to the project as shown in the following screenshot:



```
Solution 'SeleniumCookbook' (1 project)
  SeleniumCookbook
    Properties
    References
      Castle.Core
      Ionic.Zip
      Microsoft.CSharp
      Newtonsoft.Json
      nunit.framework
      System
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Drawing
      System.Xml
      System.Xml.Linq
      WebDriver
      WebDriver.Support
```

11. The `SeleniumCookbook` project is ready for test development. Go on adding new tests as needed.

## How it works...

NuGet Package Manager adds the external dependencies to Microsoft Visual Studio projects. It lists all available packages and automatically downloads and configures packages to the project. It also installs dependencies for the selected packages automatically. This saves a lot of effort in configuring the projects initially.

## Automating non-web UI in Selenium WebDriver with AutoIt

Selenium WebDriver is a pure browser automation API and works with HTML/web elements. It does not support native UI built with C++, .NET, Java, or any other desktop technologies. It also does not support Flex, Flash, or Silverlight native controls out of the box.

While testing applications that interact with native UI, it becomes difficult to automate the functionality involved. For example, the web application provides a file upload feature that invokes native OS UI for selecting a file.

We can use tools such as AutoIt to handle native UI. AutoIt is a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting. By using AutoIt, we can simulate a combination of keystrokes, mouse movement, and window/control manipulation in order to automate. It is a very small, self-contained utility. It runs on all versions of Windows operating system. AutoIt scripts can be compiled as self-contained executables.

AutoIt has certain limitations in terms of OS support as it is not supported on Linux and Mac OSX and it will not work with RemoteWebDriver.

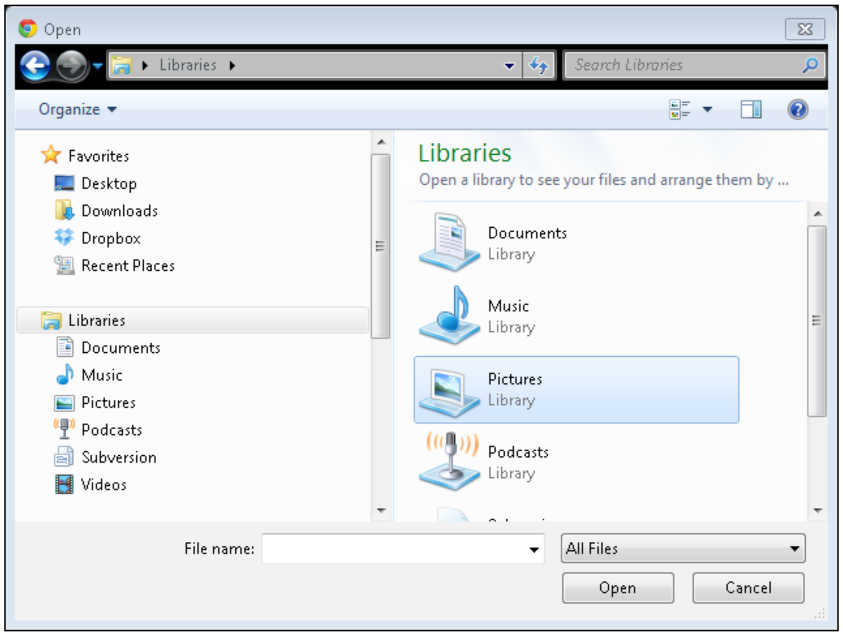
In this recipe, we will explore integration of AutoIt with Selenium WebDriver for testing the file upload functionality on a sample page.

## Getting ready

Download and install AutoIt tools from `http://www.autoitscript.com/site/autoit/downloads/`.

## How to do it...

For implementing the file upload functionality, there are a number of libraries or plugins available, which provide a number of additional features for uploading files. We will use the jQuery File Upload plugin. It offers multiple ways in which users can upload files on the server. Users can either drag-and-drop a file on the browser or can click on the **Add Files** button, which opens the native **Open** dialog box.



We will automate a test where the user can upload a file by clicking on the **Add Files** button. This invokes the **Open** dialog box as shown in previous screenshot. We will create an AutoIt script to automate interactions on this dialog box.

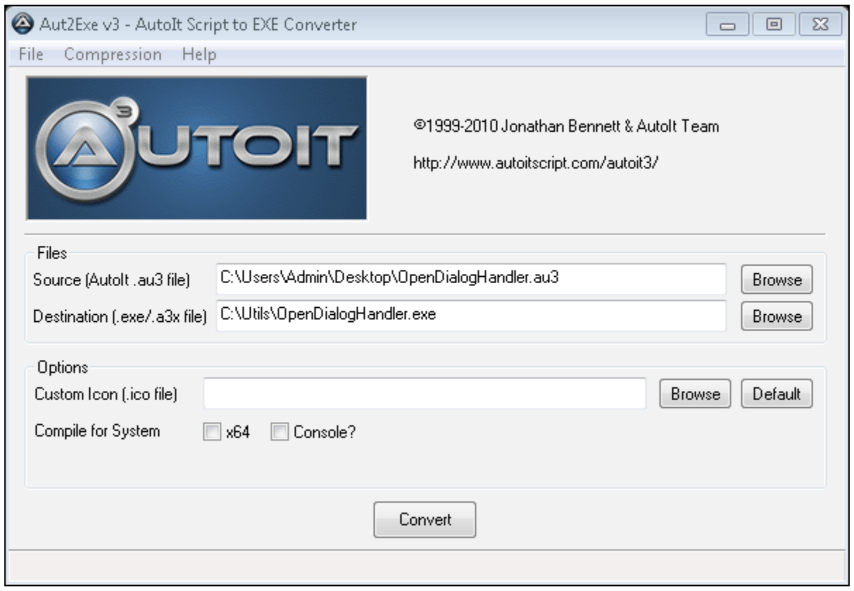
## Creating the AutoIt script

Let's create an AutoIt script which works with the **Open** dialog box.

1. Launch **SciTE Script Editor** from **Start | AutoIt**.
2. Create a new script by selecting **File |New** from the **SciTE Script Editor** main menu.
3. Name and save this file as **OpenDialogHandler.au3** (AutoIt scripts have `.au3` extension) and copy the following code to the script, then save the script:

```
WinWaitActive("Open","","20")
If WinExists("Open") Then
    ControlSetText("Open","","Edit1",$CmdLine[1])
    ControlClick("Open","","&Open")
EndIf
```

4. Launch the `Compile Script to .exe` utility from **Start | AutoIt**. By using the `Compile Script to .exe` utility, we will convert the AutoIt script into an executable file.
5. On the **Aut2Exe v3 - AutoIt Script to EXE Converter** window, enter the path of the AutoIt script in the **Source (AutoIt .au3 file)** textbox and path of executable in the **Destination (.exe/.a3x file)** textbox and click on the **Convert** button as shown in the following screenshot:



Source (AutoIt .au3 file) C:\Users\Admin\Desktop\OpenDialogHandler.au3
Destination (.exe/.a3x file) C:\Utils\OpenDialogHandler.exe

## Using OpenDialogHandler in Selenium WebDriver script

1. Now create a test where we can click on the **Add Files** button and then call the `OpenDialogHandler.exe` and validate that the specified file is uploaded on the page.
2. Create a new test class named `FileUpload` and copy the following code in to it:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.By;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;

import org.junit.*;
import static org.junit.Assert.*;

public class FileUpload {
    protected WebDriver driver;

    @Before
    public void setUp() {
        driver = new ChromeDriver();
        driver.get("http://blueimp.github.com/jQuery-
            File-Upload/");
    }

    @Test
    public void testFileUpload() throws
        InterruptedException {
        try {

            //Click on Add Files button
            driver.findElement(By.className("fileinput-
                button")).click();

            //Call the OpenDialogHandler, specify the
            //path of the file to be uploaded
            Runtime.getRuntime().exec(new String[]
                {"C:\\Utils\\OpenDialogHandler.exe",
                "\"C:\\Users\\Admin\\Desktop\\
                Picture1.png\""});

            //Wait until file is uploaded
            boolean result = (new WebDriverWait
                (driver, 30)).until(new
                ExpectedCondition<Boolean>() {
                public Boolean apply(WebDriver d) {
                    return d.findElement(By.xpath
                    ("//table[@role='presentation']"))
                    .findElements(By.tagName
                     ("tr")).size() > 0;
            }});

            assertTrue(result);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @After
    public void tearDown() {
        driver.close();
    }
}
```

## How it works...

AutoIt provides an API for automating native Windows UI control. In this example, we used the `WinWaitActive()` function, which waits for a window. It takes the title of the expected window, text of the window, and timeout as parameters. We supplied the title as `Open` and time out of `20` seconds. The text of the windows is passed as blank, as the title is enough to identify the window.

```
WinWaitActive("Open","","20")
```

Once the open window is activated, the `ControlSetText()` function is called to enter the text in the **File name:** textbox.

```
ControlSetText("Open","","Edit1",$CmdLine[1])
```

This function takes the title of the window, text of the window, control ID, and text that needs to be entered in the textbox. Similar to locators in Selenium, AutoIt identifies controls using control IDs. You can find control IDs using AutoIt V3 Window Info tool installed with AutoIt. You can spy on a window or control using this tool and find out various properties. In this example, the `ControlSetText()` function takes the last parameter as `$CmdLine[1]`. Instead of hardcoding the path of filename that we want to upload, we will pass the filename to AutoIt script using command line arguments; `$CmdLine[1]` will hold this value.

For clicking on the **Open** button, the `ControlClick()` function is called. This function takes the title of the window, text of the windows, and control ID as parameters.

```
ControlClick("Open","","&Open")
```

You can find more about AutoIt API in AutoIt help documentation.

Using the **Aut2Exe v3 - AutoIt Script to EXE Converter** utility, AutoIt script is compiled as executable that is `OpenDialogHandler.exe`.

The Selenium WebDriver test calls this executable file by using the `exec()` method of the `RunTime` class, which allows the Java code to interact with the environment in which the code is running. The complete path of `OpenDialogHandler` as well as the path of the file to be uploaded is passed through the `exec()` method. Please note that `OpenDialogHandler` needs quotes for arguments.

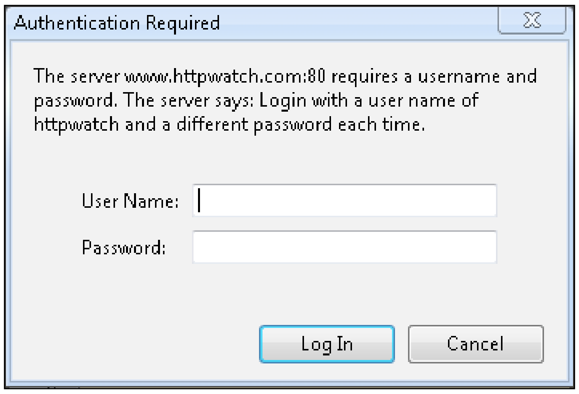
```
Runtime.getRuntime().exec(new String[]
    {"C:\\Utils\\OpenDialogHandler.exe",
    "\"C:\\Users\\Admin\\Desktop\\Picture1.png\""});
```

## There's more...

You can use AutoIt scripts with other Selenium WebDriver bindings such as .NET, Ruby, or Python as long as these languages allow you to call external processes.

AutoIt also comes with a lightweight AutoItX COM library that can be used with languages that support **COM** (**Component Object Model**). Using the COM API will save you from writing the AutoIt script and compiling it in to an executable.

You will come across web applications using HTTP authentication which requires users to authenticate before displaying the application. An HTTP authentication dialog is displayed as shown in the following screenshot:



The server www.httpwatch.com:80 requires a username and password. The server says: Login with a user name of httpwatch and a different password each time.

This dialog box is displayed by using native UI. In addition, the layout and controls on this dialog box may change for different browsers. In the following example, AutoItX API is called directly in the Ruby script for automating the HTTP authentication dialog box displayed in Google Chrome:

```
require 'rubygems'
require 'selenium-webdriver'
require 'test/unit'
require 'win32ole'

class HttpAuthTest < Test::Unit::TestCase
    def setup
        @driver = Selenium::WebDriver.for :chrome
        @driver.get 'http://www.httpwatch.com/
            httpgallery/authentication'
        @verification_errors = []
    end

    def test_http_auth_window
        #Create instance of AutoItX3 control.
        #This will provide access to AutoItX COM API
        au3 = WIN32OLE.new("AutoItX3.Control")

        #Get the Display Image button and
        #click on it to invoke the Http Authentication dialog
        display_image_button = @driver.find_element :id =>
            "displayImage"
        display_image_button.click

        #Wait for couple of seconds for Http Authentication
        #dialog to appear
        sleep(2)

        #Check if Http Authentication dialog exists and
        #enter login details using send method
        result = au3.WinExists("HTTP Authentication -
            Google Chrome")
        if result then
            au3.WinActivate("HTTP Authentication -
                Google Chrome","")
            au3.Send("httpwatch{TAB}")
            au3.Send("jsdhfkhkhfd{Enter}")
        end
        assert_equal 1, result
    end

    def teardown
        @driver.quit
        assert_equal [], @verification_errors
    end
end
```

In the previous example, Ruby's `win32ole` module is used to create an instance of AutoItX COM interface.

```
    au3 = WIN32OLE.new("AutoItX3.Control")
```

By using this interface, the AutoIt methods are called for interacting with non-web UI.

### See also

- The *Automating non-web UI in Selenium WebDriver with Sikuli* recipe

## Automating non-web UI in Selenium WebDriver with Sikuli

Sikuli is another tool that can be used along with Selenium WebDriver for automating non-web UI. It uses visual identification technology to automate and test graphical user interfaces (GUI). The Sikuli script automates anything you see as a user on the screen rather than an API.

Sikuli is supported on Windows, Linux, and Mac OSX operating systems. Similar to Selenium IDE, Sikuli provides an IDE for script development and API that can be used within Java.

Sikuli works well for non-web UI. However, it also has certain limitations as it is not supported by RemoteWebDriver. The Sikuli script might fail if it does not find a captured image due to overlapping windows at runtime.

In this recipe, we will explore integration of Sikuli API with Selenium WebDriver for testing non-web UI.

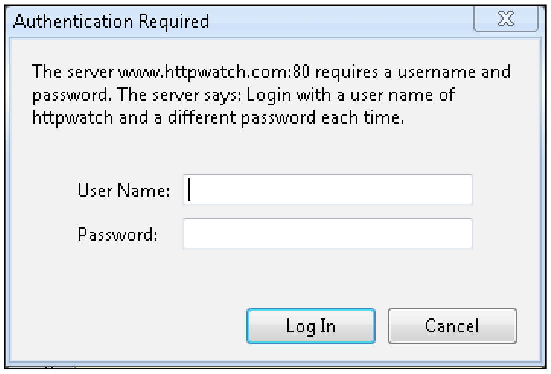
### Getting ready

Download and install Sikuli from `http://sikuli.org/`.

## How to do it...

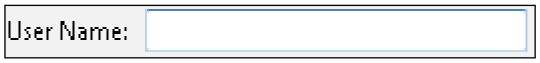
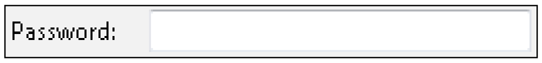
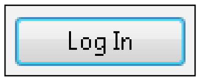
We will use the HTTP authentication example from the previous recipe to automate the steps with Sikuli, as follows:

1. Before we automate the steps on the **Authentication Required** dialog box, we need to capture images of the **User Name:** and **Password:** textboxes and the **Log In** button:


Authentication Required

The server www.httpwatch.com:80 requires a username and password. The server says: Login with a user name of httpwatch and a different password each time.

2. Capture the screenshot of the **Authentication Required** dialog box and extract images as shown next for each control:

| Control | Screenshot |
|---|---|
| **User Name:** textbox |  |
| **Password:** textbox |  |
| **Log In** button |  |

3. Save these extracted images separately in **PNG** (**Portable Network Graphics**) format, in a folder that can be accessed from tests easily.
4. Add the `Sikuli-script.jar` file to the test project. You can get this file from the Sikuli installation folder.
5. Create a new test, name it `HttpAuthTest` and copy the following code to this test:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.By;

import org.sikuli.script.FindFailed;
import org.sikuli.script.Screen;

import org.junit.*;
import static org.junit.Assert.fail;

public class HttpAuthTest {
    private WebDriver driver;
    private StringBuffer verificationErrors = new
        StringBuffer();

    @Before
    public void setUp() {
        driver = new ChromeDriver();
        driver.get("http://www.httpwatch.com/
            httpgallery/authentication/");
    }

    @Test
    public void testHttpAuth() throws
        InterruptedException {
        driver.findElement(By.id
        ("displayImage")).click();

        //Get the system screen.
        Screen s = new Screen();

        try {

            //Sikuli type command will use the image file
            //of the control
            //and text that needs to be entered in to the
            //control
            s.type("C:\\UserName.png", "httpwatch");
            s.type("C:\\Password.png","dhjhfj");

            //Sikuli click command will use the image
            //file of the control
            s.click("C:\\Login.png");

        } catch (FindFailed e) {
            //Sikuli raises FindFailed exception it fails
            //to locate the image on to the screen
            e.printStackTrace();
        }
        }

        @After
        public void tearDown() {
            driver.close();
            String verificationErrorString =
                verificationErrors.toString();
            if (!"".equals(verificationErrorString)) {
                fail(verificationErrorString);
            }
        }
    }
```

## How it works...

As Sikuli uses visual identification technology to identify and interact with windows and controls, it requires images of the controls to perform the action. During the execution, it locates the regions captured in the image on the screen and performs the specified action.

```
s.type("C:\\UserName.png", "httpwatch");
```

In this example, it searches the screen for a region that matches the image captured in `UserName.png` and performs the `type ()` command. Sikuli replays these commands as if a real user is working on the screen.

Sikuli may fail to interact with the application if the region captured in the image is not displayed on the screen or overlapped with other windows.

## There's more...

By using Sikuli, you can automate RIA technologies such as Flash and Silverlight, along with the Selenium WebDriver.

## See also

- The *Automating non-web UI in Selenium WebDriver with AutoIt* recipe