Eclipse & Windows

Quick Start Guide

for Keywords Driven Testing

Release 1.0

© 2017 Maveryx srl.
All rights reserved.

© **Maveryx** srl

All rights reserved. This document is the property of Maveryx srl. It must not be copied partly or in full without obtaining the prior written consent of Maveryx srl. Permission to copy and implement the material contained herein is granted subject to the conditions that any copy or re–publication must bear this legend in full. That any derivative work must bear a notice that it is Maveryx srl copyright document jointly published by the copyright holders and that none of the copyright holders shall have any responsibility or liability whatsoever to any other party arising from the use or publication of the material contained herein.

At the time of going to press, this paper is as thorough and correct as possible: however, information herein contained may have been updated without prior notice after this date. Maveryx srl reserves the right to change the functions of its products at any time without prior notice.

Trademarks

- Eclipse is a trademark of Eclipse Foundation Inc.
- Java and all Java–based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.
- Windows is a registered trademark of Microsoft Corporation in the United States and other countries.
- UNIX is a registered trademark of The Open Group in the United States and other countries.

All other company, product, or service names contained on this site may be trademarks or service marks of others and are the property of their respective owners.

Installing Maveryx

System Requirements

Maveryx requires your computing environment meets some minimum requirements.

Java Version

Maveryx requires a fully compliant J2SE Java Runtime Environment 1.6 or later[1].

Operating System

Maveryx is a 100% Java application and should run correctly on any system that has a compliant Java implementation.

Maveryx has been tested and works with:

- Windows XP, Windows Vista, Windows 7, Windows 8 and Windows 10
- Unix (Linux)
- Mac OS X

Eclipse Version

To install the Maveryx plug-in for Eclipse you must have the following:

- Eclipse Ganymede or later (version 3.4)[2]

Getting Maveryx

The easiest way to begin using Maveryx is to first download the latest production release and install it.

The latest stable version of Maveryx is available at

- http://www.maveryx.com/

Installing Maveryx

Installation on Windows

Windows users can download an installer for the relevant version of Maveryx. To install Maveryx run the Setup program following the on-screen instructions. The setup installs both the Maveryx Framework and the Eclipse plug-in.

NOTE: If you are upgrading your installation of Maveryx from a previous version, you will need to uninstall the old Maveryx version before installing the new version.

---

[1] http://java.sun.com/javase/downloads/index.jsp

[2] http://eclipse.org/downloads

Configuration of the Eclipse Plug-in

Maveryx offers a custom plugin for the Eclipse IDE. This plugin provides a powerful, integrated environment in which to develop Maveryx tests. It extends the capabilities of Eclipse to let you quickly set up new Maveryx projects, build a test suite, debug your tests, and much more.

To configure the Maveryx Eclipse plug-in you must specify the location of your Maveryx directory:

1. Start Eclipse, then select **Window > Preferences**
2. Filter for or navigate to the **Maveryx** page
3. Enter the Maveryx installation path into the **Maveryx Location** field
4. Click **OK**

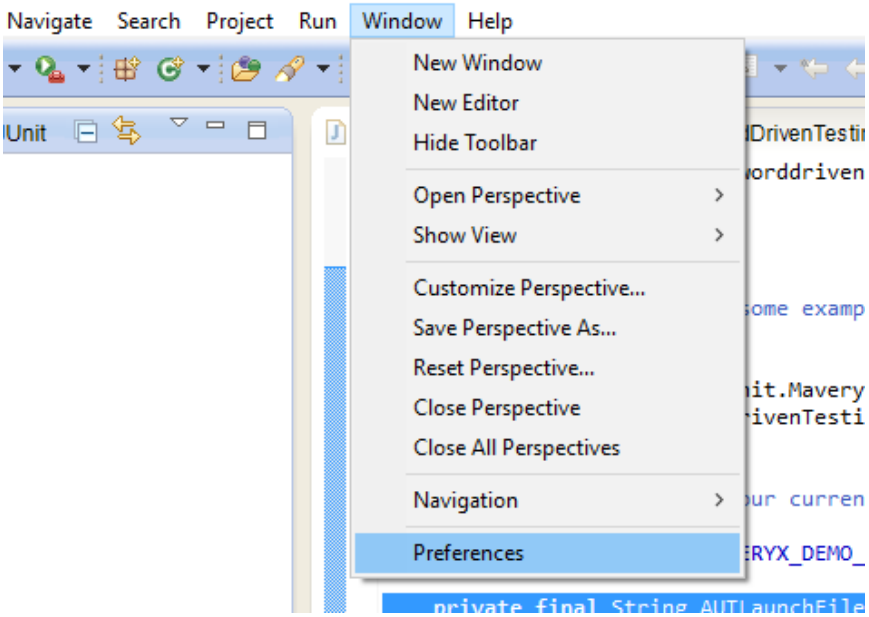


Figure 1 – Window → Preferences

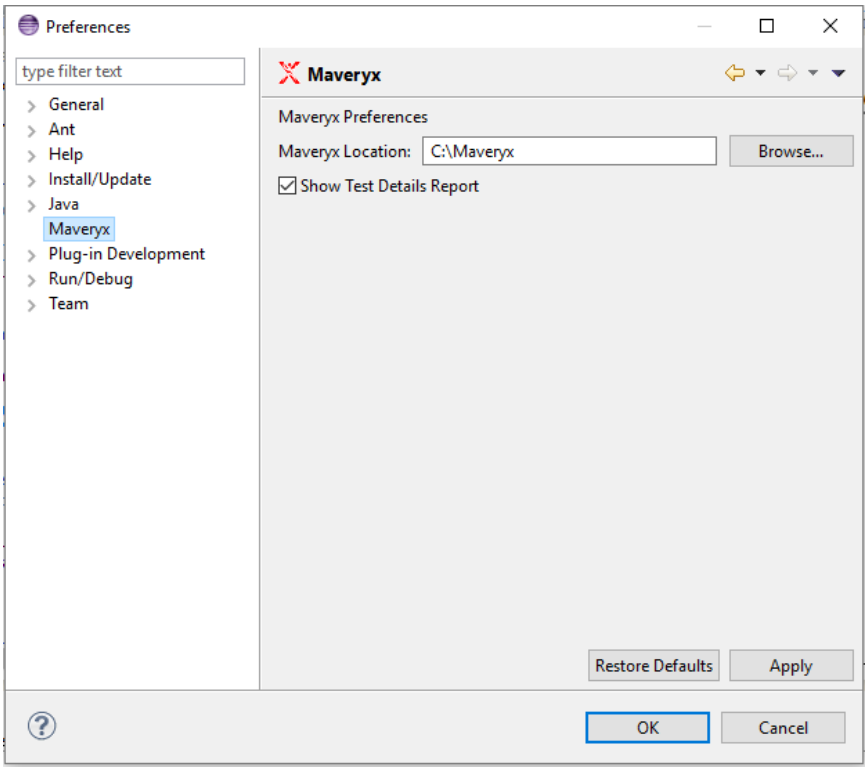


Figure 2 – Maveryx → Browse

Your Eclipse IDE is now set up to develop Maveryx tests,

Building Your First Maveryx Keywords driven test

This section teaches you how to build your first Maveryx test.

You'll learn how to:

1. create a Maveryx test project with the Eclipse plugin
2. create a Maveryx project
3. configure the AUT launch file
4. run the tests and view the results

The application used in the following section is contained into the example.jar file in MAVERYX_HOME/demo/ and is described at

https://docs.oracle.com/javase/tutorial/uiswing/components/passwordfield.html

Creating a new Maveryx project

A Maveryx project contains source code and related files for building a test suite. It has an associated Java builder that can incrementally compile Java source files as they are changed.

To create a new Maveryx test project in workspace:

5. Launch the Eclipse IDE
6. In the Maveryx Eclipse Perspective:
    a. select **File** → **New** → **Maveryx Test Project** (**Figure** 33) or
    b. click the [ ] button on the toolbar
7. In the ***Maveryx Test Project*** window (**Errore. L'origine riferimento non è stata trovata.4**)
    a. enter a name for the project (e.g. MyFirstMaveryxProject)
    b. in the ***JRE*** section make sure that Java/JRE 6 or higher is selected
8. Click **Finish** to create the test project

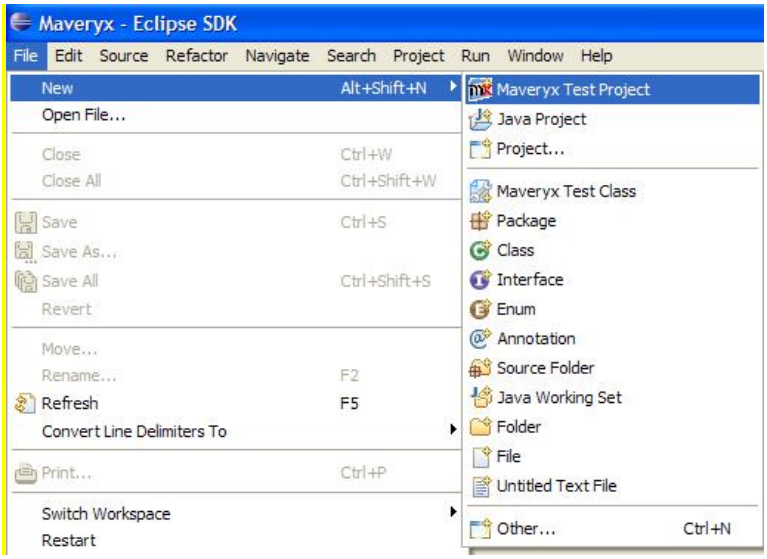


Figure 3 – File → New → Maveryx Test Project

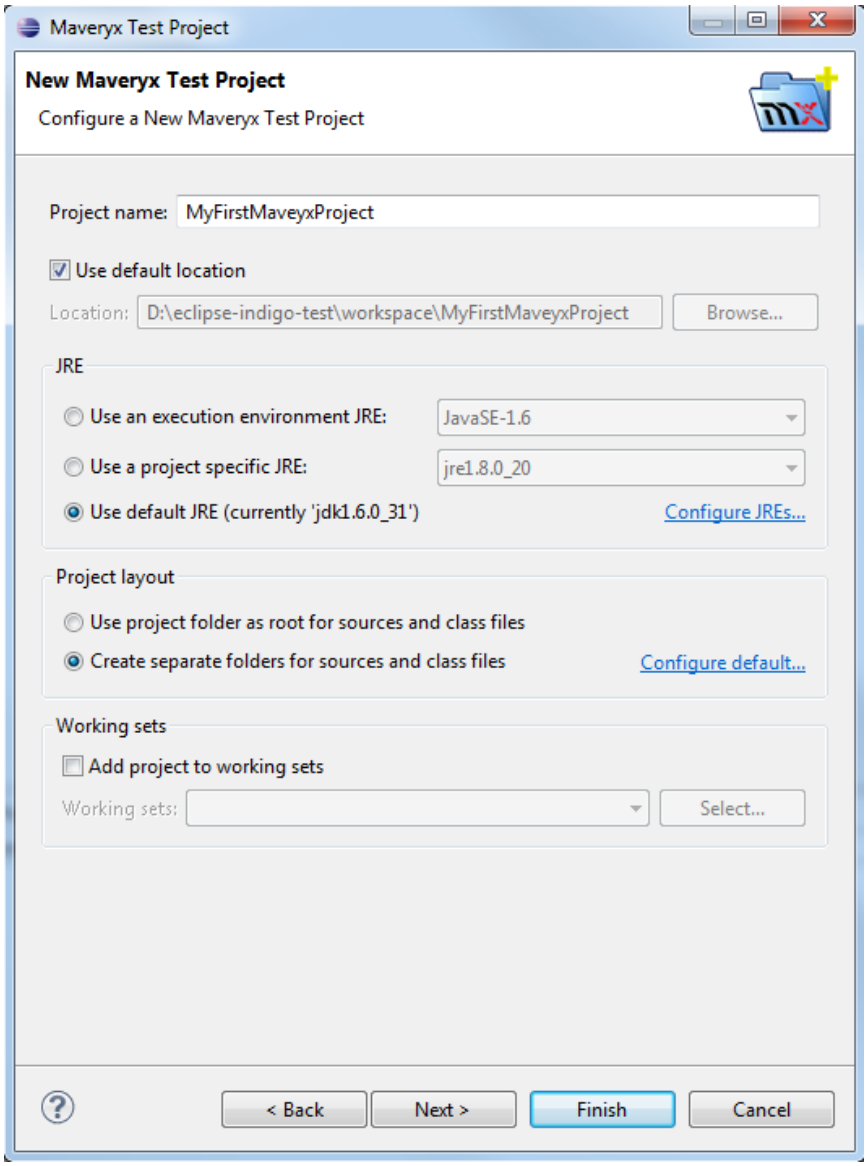


Figure 4 – New Maveryx Test Project

Keyword–driven testing

Keyword-driven tests, also known as table-driven or action-word tests are automated tests that are developed in data tables (independent of the automated test tool used) with a vocabulary of reusable Keywords that correspond to automated testing actions such as "StartApplicaton", "Login" … Each keyword specifies the action to be executed on the application under test and the parameters associated with the action.

Maveryx supports keyword-driven tests written in Excel spreadsheets, comma-separated (CSV) files and XML files. It provides built-in keywords for all supported GUI objects. Maveryx also supports variables and an extension plugin mechanism to add new keywords, variable types, and keyword file formats.

In the MAVERYX_HOME/demo directory there are some examples using the keyword–driven approach.

Creating keyword-driven tests using XML

Maveryx supports XML sources with the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<testcase name="" description="" author="" requirements="">
        <action name="CLICK">
                <object name="Disable middle button" type="BUTTON"/>
                <container name="" type="FRAME"/>
        </action>
        <action name="CLICK">
                <object name="Enable middle button" type="BUTTON"/>
                <container name="ButtonDemo" type="FRAME"/>
        </action>
        <action name="HAS_TOOLTIP">
                <object name="Disable middle button" type="BUTTON"/>
                <container name="ButtonDemo" type=""/>
                <parameters>
                        <param value="Click this button to disable the middle button."/>
                </parameters>
        </action>
        <action name="CLOSE">
                <object name="" type="FRAME"/>
```

testcase: the root element. *testcase* shall consist at least of the following properties:

- *name*: a string label/ID for the test case (OPTIONAL)
- *description*: a brief description of the test case (OPTIONAL)
- *author*: the name of the test case author (OPTIONAL)
- *requirements*: the related requirement(s) or use case(s) (OPTIONAL)

This element shall contain at least of one *action* element.

action: contains the action/keyword to be executed. *action* shall consist of the following properties:

- *name*: the identifier of the action/keyword to be executed

Such element may contain the following elements.

object: the GUI Object to test. *object* shall consist of the following properties:

- *name*: the identifier of the GUI Object to test (OPTIONAL)
- *type*: the type of the GUI Object to test (OPTIONAL)

container: the container/owner of the GUI Object to test (*object*). *container* shall consist of the following properties:

- *name*: the identifier of the container object (OPTIONAL)
- *type*: the type of the container object (OPTIONAL)

parameters: the parameters associated with the action/keyword. *parameters* shall one or more *param* elements (depending on the number of parameters associated with the action).

param: a parameter associated with the action/keyword. *param shall* consist of the following properties:

- *value*: the parameter value

value can be a variable:

- **${file path}** {***testdata* name**} {***data* id**} : to use the data-driven mechanism provided by Maveryx [**Errore. L'origine riferimento non è stata trovata.**] - example ${C:/Maveryx/demo/data/Test_Data.xls} {username} {user1})
- **\\${value}** : to use a custom data-driven mechanism

Creating keyword-driven tests using CSV

Maveryx supports CSV sources with the following structure:

```
Object;Name;Container;Container Name;Action;Data;Data;Data
BUTTON;Disable middle button;FRAME;;CLICK;;;
BUTTON;Enable middle button;FRAME;Button demo;CLICK;;;
BUTTON;Disable middle button;;ButtonDemo;HAS_TOOLTIP;Click this button to disable the
middle button.;
FRAME;;;;CLOSE;;;
```

The first line is the header.

Object: the type of the GUI Object to test (OPTIONAL)

Name: the name of the GUI Object to test (OPTIONAL)

Container: the type of the container/owner object (OPTIONAL)

Container name: the name of the container/owner object (OPTIONAL)

Action: the action/keyword to be executed

Data$_1$ … Data$_N$: the parameters (e.g. input, expected output …) associated with the action/keyword (OPTIONAL)

Data$_i$ can be a variable:

- **${file path} {*testdata* name} {*data* id}** : to use the data-driven mechanism provided by Maveryx (e.g. ${ C:/Maveryx/demo/data/Test_Data.xls} {username} {user1})
- **\\${value}** : to use a custom data-driven mechanism

You can add more Data element, depending on your keyword.

Creating keyword-driven tests using Excel

Maveryx supports MS Excel sources with the following structure (**Figure 5**).

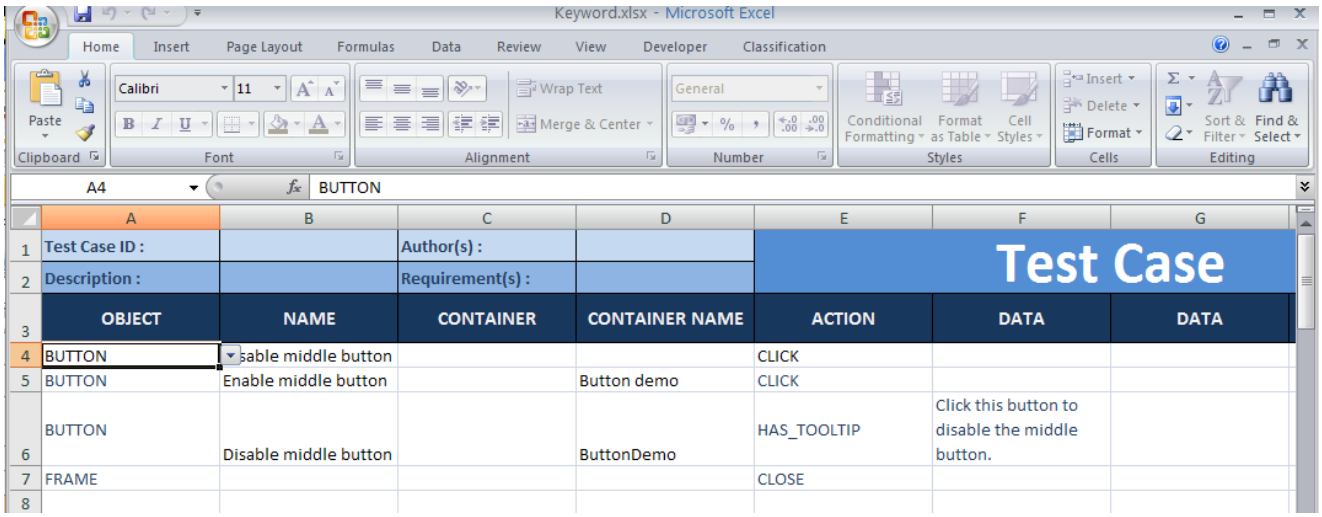


Figure 5 – Keyword-driven test using Excel

The *Test Case* header contains the following information:

- *Test Case ID* : a string label/ID for the test case (OPTIONAL)

- *Description*: a brief description of the test case (OPTIONAL)
- *Author(s)*: the name of the test case author (OPTIONAL)
- *Requirement(s)*: the related requirement(s) or use case(s) (OPTIONAL)

The *Test Case* body contains the following elements:

Object: the type of the GUI Object to test (OPTIONAL). You can choose the value from a list (combo box) of all supported GUI Objects[3].

Name: the name of the GUI Object to test (OPTIONAL)

Container: the types of the container/owner object (OPTIONAL). You can choose the value from a list (combo box) of all supported GUI Objects.

Container name: the name of the container/owner object (OPTIONAL)

Action: the action/keyword to be executed. You can choose the action/keyword from a list (combo box) of all supported ones.

Data$_1$ … Data$_N$: the parameters (e.g. input, expected output …) associated with the action/keyword (OPTIONAL)

Data$_i$ can be a variable:

- **${file path} {*testdata* name} {*data* id}** : to use the data-driven mechanism provided by Maveryx (e.g. ${ C:/Maveryx/demo/data/Test_Data.xls} {username} {user1})
- **\\${value}** : to use a custom data-driven mechanism

You can add more Data element, depending on your keyword.

Setting up the launch file

This section describes the XML format used for launch files.

To execute the application under test you have to create an XML launch file as the one below[4] (available in MAVERYX_HOME/demo/).

```
<?xml version="1.0" encoding="UTF-8"?>
<AUT_DATA>

    <WORKING_DIR>C:\Maveryx\demo</WORKING_DIR> <!-- change this path to the application directory -->

    <APPLICATION_NAME>PasswordDemo</APPLICATION_NAME>

    <AUT_ARGUMENTS></AUT_ARGUMENTS>

    <VM_ARGUMENTS></VM_ARGUMENTS>

    <DESCRIPTION>PasswordDemo test</DESCRIPTION>

    <JRE_PATH>C:\Program Files\Java\jre6\</JRE_PATH> <!-- change this path to your JRE home -->

    <MAIN_CLASS>com.sun.demo.PasswordDemo</MAIN_CLASS>
```

---

[3] You can add new GUI Objects and Actions/Keywords to the list (combo box) by adding values starting from row = 100. The default password to unprotect sheet is '1234'.

[4] We recommend you start from the provided example.

```
        <CLASSPATH>
                <LIB>
                  <PATH>C:\Maveryx\demo\example.jar;</PATH> <!-- change this path to the application
                  executable jar file -->
                </LIB>
        </CLASSPATH>
</ AUT_DATA>
```

AUT_DATA: the root element.

WORKING_DIR: the directory from which the application under test is launched.

APPLICATION_NAME: the name of the application to test.

AUT_ARGUMENTS: the arguments to pass to the application's *main* method.

VM_ARGUMENTS: the Java VM arguments to use to run the application under test. Refer to java help for more details.

DESCRIPTION: a textual description of the application to test. [optional]

JRE_PATH: the full file system path to the HOME directory containing the Java executable used to run the application under test.

MAIN_CLASS: the name of the main class to be invoked.

CLASSPATH: this element contains the classpath. CLASSPATH shall consist of one *or more LIB* elements.

LIB: specifies a JAR file needed to execute the application under test. An element LIB shall consist of *PATH*.

```
<LIB>
      <PATH>C:\Maveryx\demo\example.jar;</PATH>
</LIB>
```

PATH: the full file system path to a JAR file needed to execute the application under test

If your application classpath consists of *n* JAR files, you can create *n* LIB elements, one for each JAR, or only one LIB element concatenating all paths.

For example, if your classpath consists of *lib1.jar*, *lib2.jar* and *lib3.jar* in *C:\myApp* you can create:

```
<LIB>
      <PATH>C:\myApp\lib1.jar;</PATH>
</LIB>
<LIB>
      <PATH>C:\myApp\lib2.jar;</PATH>
</LIB>
<LIB>
      <PATH>C:\myApp\lib3.jar;</PATH>
</LIB>
```

or

```
<LIB>
      <PATH>C:\myApp\lib1.jar;C:\myApp\lib2.jar;C:\myApp\lib3.jar;</PATH>
</LIB>
```

Running a keyword-driven test

To run a keyword–driven test, you have to create a new Maveryx Test Class.

In the test script you have to first import the relevant libraries:

```
import org.maveryx.keydriven.KeywordDrivenTestManager;
```

Then you have to create a new KeywordDrivenTestManager() object and invoke the run() method:

```
new KeywordDrivenTestManager().run(MAVERYX_DEMO_DIR + "data/Keyword.xlsx");
```

The run method may take as input two parameters:

- the keyword-driven test to run (MANDATORY)
- a list of arguments (OPTIONAL)

In case of an Excel file the list of arguments shall contain the Name/ID of the sheets to be executed. If no argument is passed, all sheets within the file are executed.

```
new KeywordDrivenTestManager().run(MAVERYX_DEMO_DIR + "data/Keyword.xlsx", new String[]{"Sheet1"});
```

In case of a Comma-Separated file the list of arguments shall contain:

- the values delimiter (default ‘ ; ’)
- the quote delimiter (default ‘ ” ’)

If no argument is passed the default values are used.

To run a test as JUnit Test (**Figure** 67):

1. In the Package Explorer, select the Java compilation unit containing the test you want to launch
2. Press the Run [ ] button in the workbench toolbar or select **Run** → **Run** from the workbench menu. Alternatively, select **Run As** → **JUnit Test** from the Package Explorer pop-up menu, select **Run** → **Run As** → **JUnit Test** in the workbench menu bar, or select **Run As** → **JUnit Test** in the drop-down menu on the Run tool bar button
3. Your test is now launched

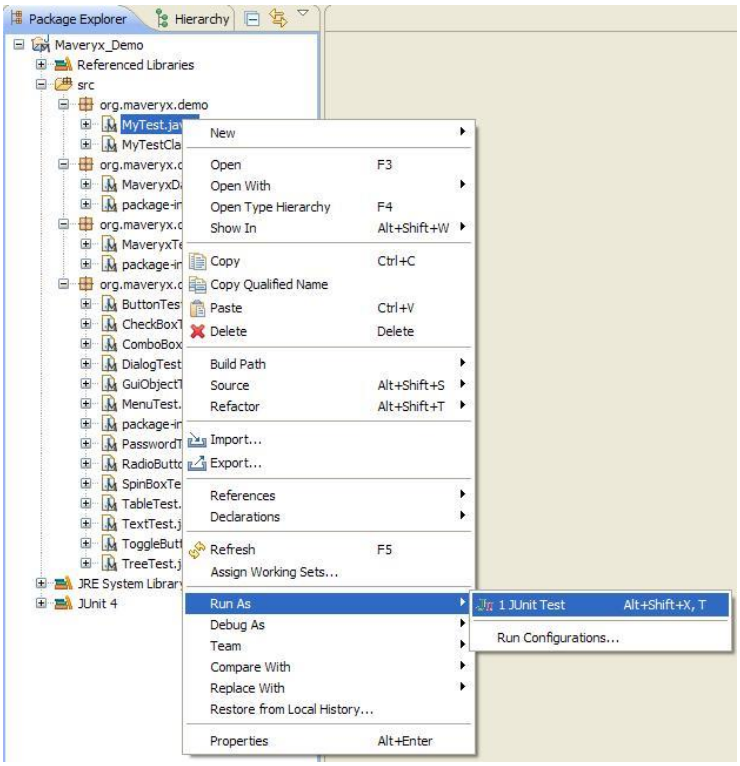


Figure 6 – Run as JUnit Test

To run a test as Java class:

1. In the Package Explorer, select the Java compilation unit containing the test you want to launch
2. Press the Run [ ] button in the workbench toolbar or select **Run** → **Run** from the workbench menu. Alternatively, select **Run As** → **Java Application** from the Package Explorer pop-up menu, select **Run** → **Run As** → **Java Application** in the workbench menu bar, or select **Run As** → **Java Application** in the drop-down menu on the Run tool bar button
3. Your test is now launched

You can also launch your test scripts by selecting the Maveryx project instead of the compilation unit

Viewing Test Results in Eclipse

Maveryx comes with powerful reports and metrics to enable users to quickly and easily interpret the test results.

Before executing the tests, to open the ***Report*** view[5], in the Eclipse toolbar, select **Window** → **Show View** → **Reports** (**Figure 7**)

[5] The *Report View* is automatically loaded when the *Maveryx Perspective* is opened.

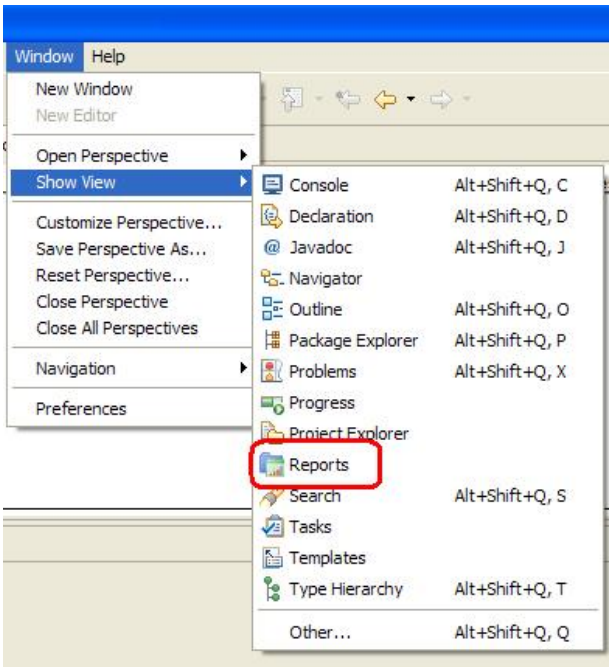


Figure 7 – Open the Report view

When you run your tests, Maveryx displays the status of the tests (**Figure 8**) in the ***Report*** view.

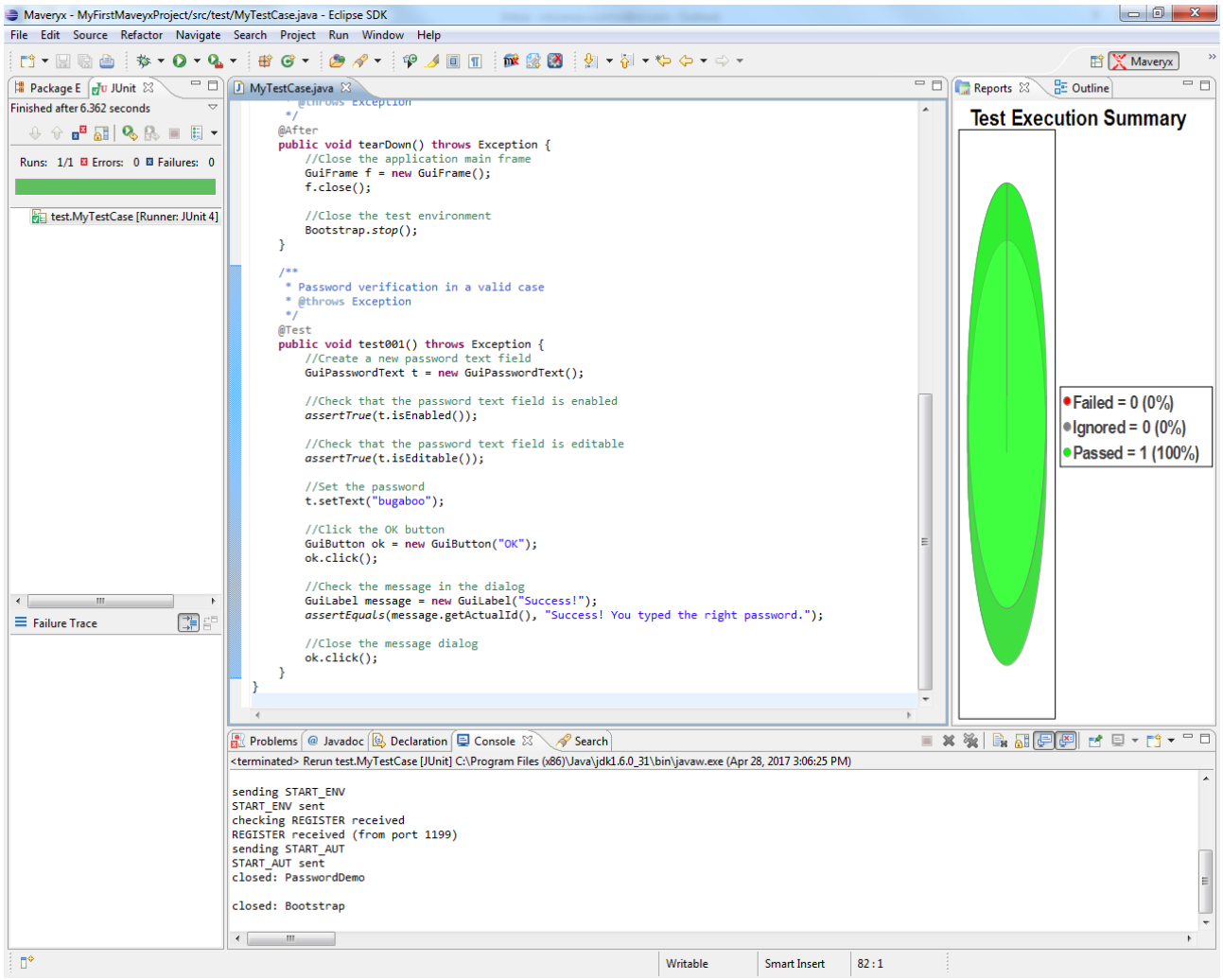
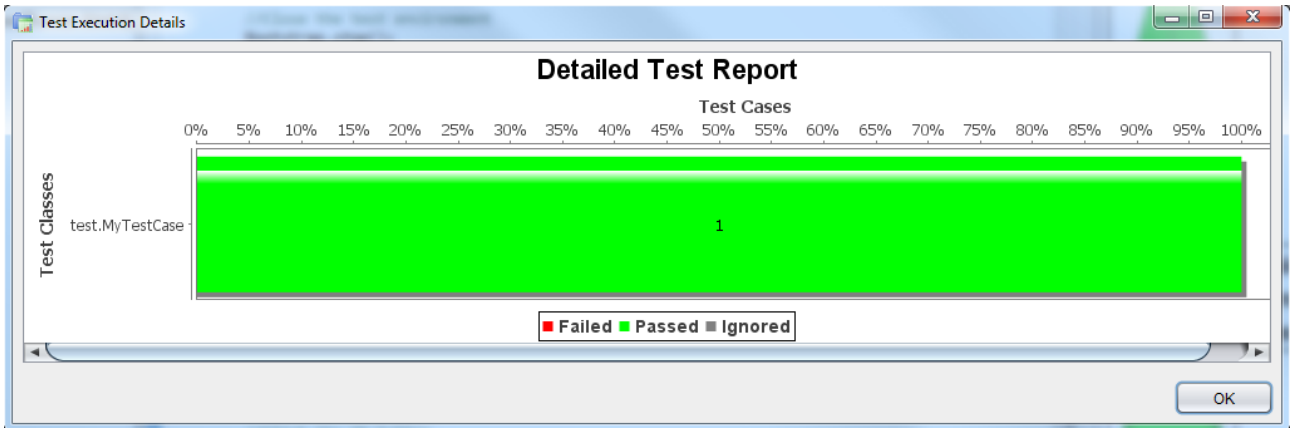


Figure 8 – Test Execution Status

To view more detailed reports and metrics at the end of each test execution:

1. Select **Window** → **Preferences** to open the Preferences panel (**Errore. L'origine riferimento non è stata trovata.1**)
2. Select ***Maveryx*** from the left panel
3. Select ***Show Detailed Test Reports***
4. Click **Apply**, and then **OK.**

After executing all tests the Test Execution Details window (**Errore. L'origine riferimento non è stata trovata.0**) will appear listing the status of all tests.


Test Execution Details
Detailed Test Report
Test Cases
0% 5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75% 80% 85% 90% 95% 100%
Test Classes
test.MyTestCase
1
Failed Passed Ignored
OK