Enhanced test automation for Web and Desktop apps

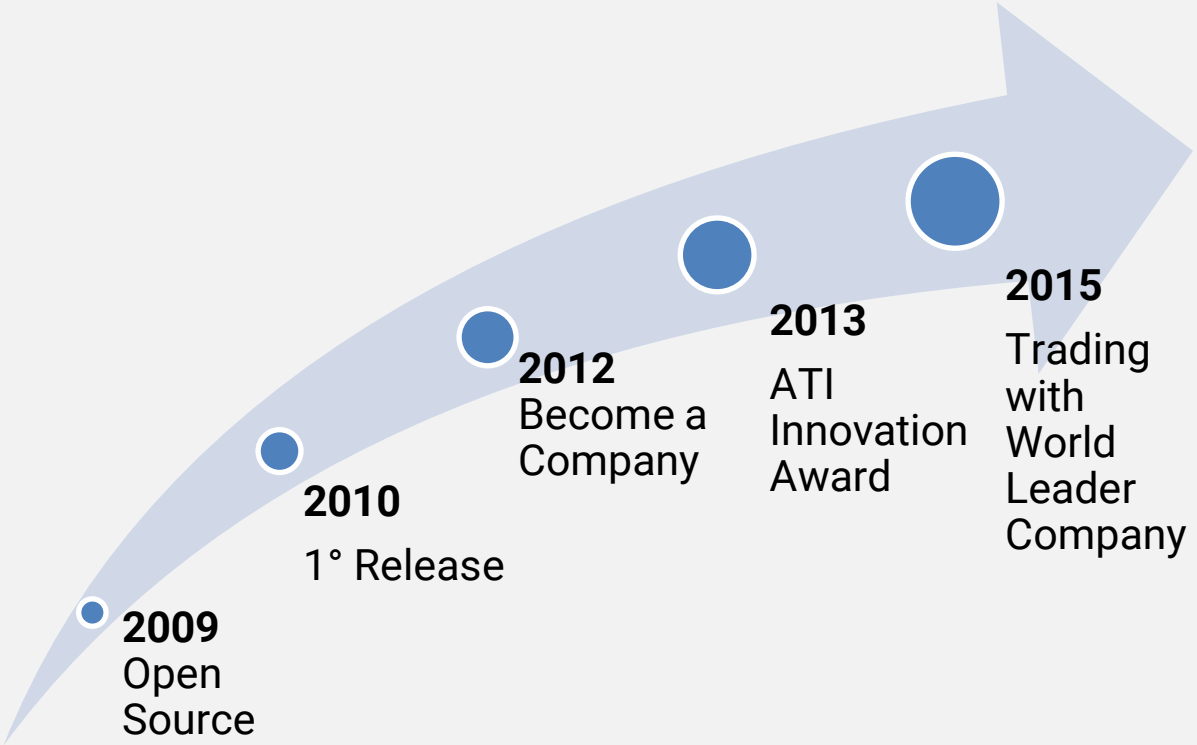
Software Testing Forum 2019

Alfonso Nocella

MAVERYX®
Test it simple!

Introduction

Maveryx is the **Italian testing automation** company that provides one of the most advanced testing automation solution worldwide.



Today

- 14 Released Versions
- 25K Downloads
- 40K Contacts

MAVERYX®
Test it simple!

Challenges (1)

- **Creating and maintaining test artefacts** (maps, objects repositories, recordings, etc…): huge cost in terms of ***time*** and ***effort***;
- **Instrumenting the AUT (Application Under Test) code**
- AUTs in **different technologies need different tools**

Challenges (2)

Time

The Application Under Test is required to start developing tests

Start late, finish later!
50%

Quality

Automation does not replace completely manual testing

Useless activities instead of increasing functional coverage

Cost

- Acquisition €2K-16K
- Know-how €10K-25K
- Maintenance *cost* unpredictable

Costs reduction is hard to achieve

Does automation necessarily be complicated and expensive?

The Idea

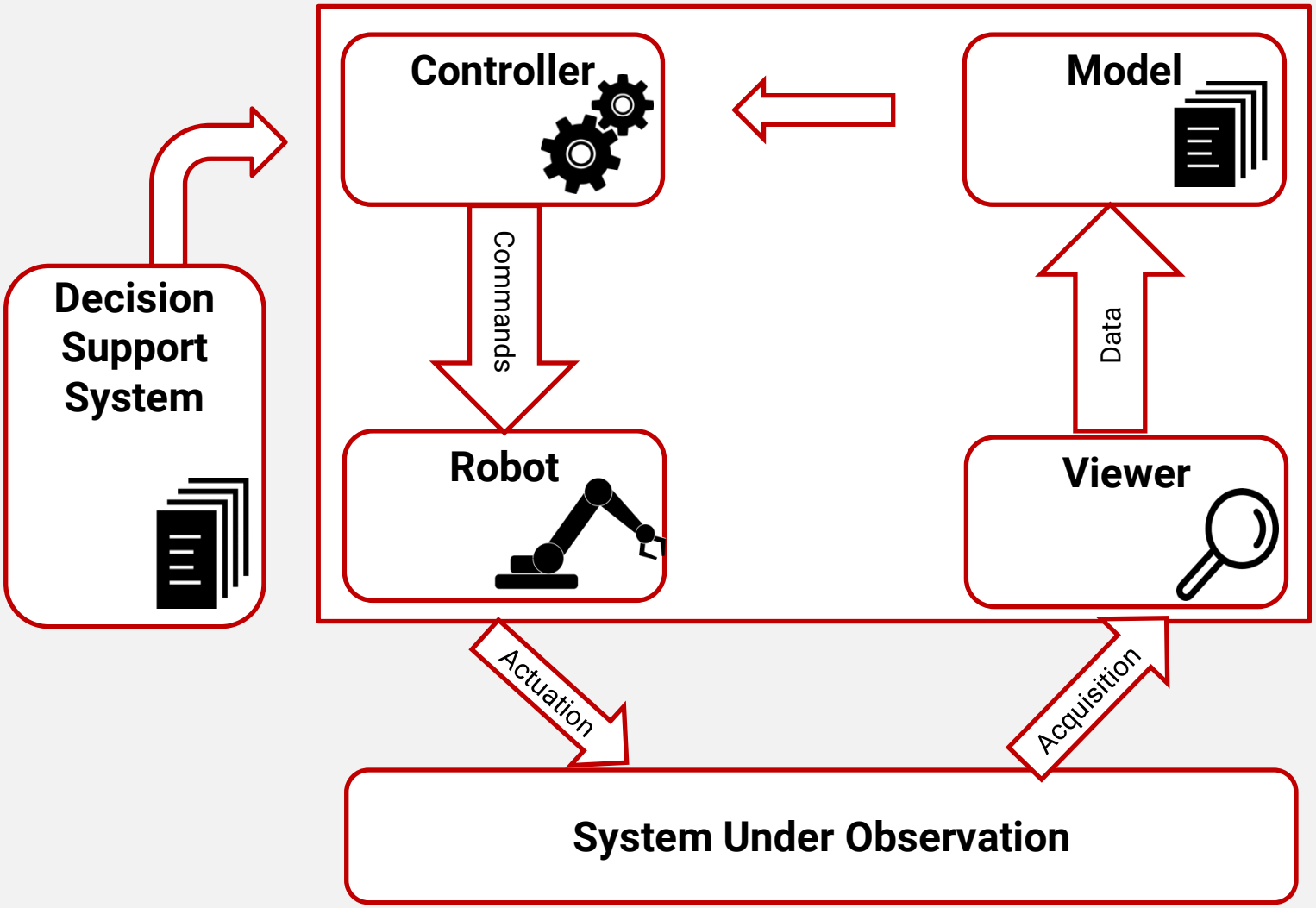
A technology able to operate as a person

- *no artefact*
- *no code instrumentation*
- *no programming skills.*

A technology able to test AUTs from design

- *no matter the development technology*
- *no matter the execution environment*

The Technology



The Framework

The Maveryx Test Automation **Framework** innovates automated testing for:

- ❑ *Functional Testing*
- ❑ *Regression Testing*
- ❑ *Keyword-Driven Testing*
- ❑ *Data-Driven Testing*
- ❑ *Continuous Integration*

Inspection «On-The-Fly»

- No GUI Maps or Object Repositories
- No pre-recording or UI element capture
- No AUT code instrumentation
- No programming skills needed
- Recognition of objects in the User Interface by Images

So innovative ?!

- ❑ **Inspection & recognition at runtime**
- ❑ **One script to test them all**
- ❑ **Testers will be testers**
- ❑ **Multi-Platform**

The Framework is an **expert tool** operating as a **senior human tester**

A Case Study

MAVERYX®
Test it simple!

Challanges won

Time

Test automation runs in *parallel* with software development

Short time-to-market: early start, earlier finish!

Quality

Accuracy improved by:

- test objects recognition
- accommodating changes
- error recovering

Tests *resilient* to frequent changes

Cost

- Std. technologies
- Easy to learn & use
- No maintenance cost

***Reduced* all the automation costs**

Testers

- **No code**, or learn and use complex **XPath Locators**, or **Matchers**, etc…
- **No** effort to capture and maintain any **Object Repository**, **GUI Map**, etc…
- Java & C# scripting, or **Keywords & Blocks for non programmers**

Managers

- ✓ Your team can save a lot of **time** that can be spent **to increase test coverage**
- ✓ You can cut effort & costs of test creation & maintenance, **releasing earlier**
- ✓ **Everyone** in your team can play a **significant role**

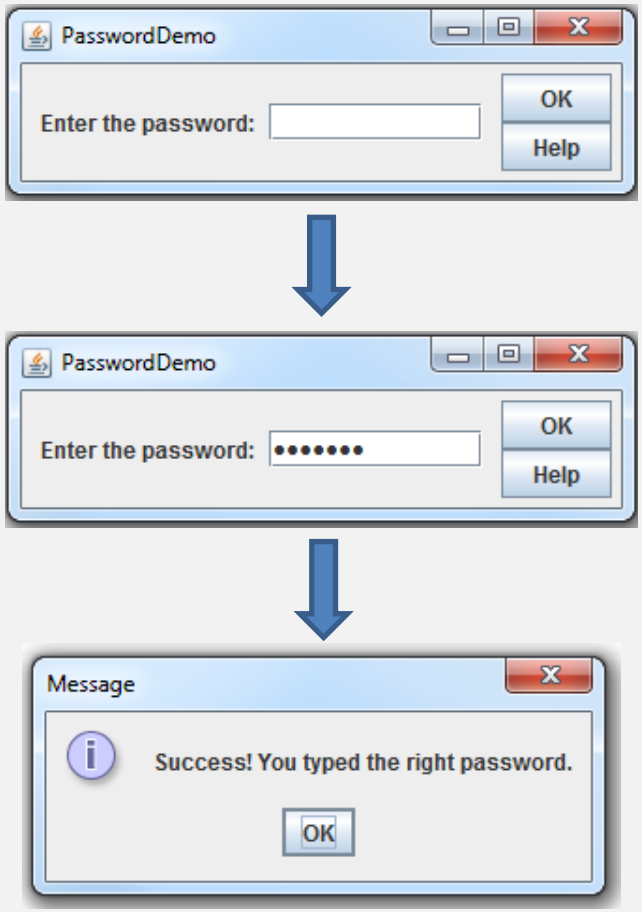
MAVERYX® Test it simple!

Test Scripting

- Taking advantage of the coding skills;
- Using constructs, statements, etc…
- Using the OOP features (e.g. inheritance, polymorphism, recursion, etc…);
- Using design patterns;

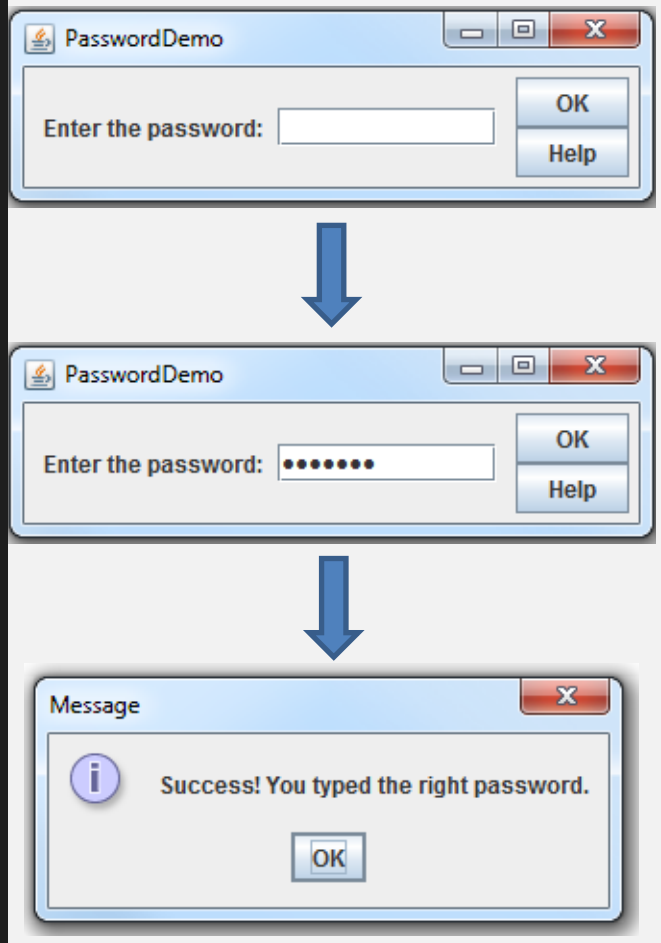
Scripting: Java Example

```java
@Test
public void test001() throws Exception {

    GuiPasswordText t = new GuiPasswordText("Enter the password:");
    assertTrue(t.isEditable()); //check whether the text field is editable
    t.setText("bugaboo");       //enter the password

    GuiButton ok = new GuiButton("OK");
    assertTrue(ok.isEnabled());  //check whether the push button is enabled

    //click the 'OK' button in the main frame to confirm the entered password
    ok.click();

    GuiDialog dialog = new GuiDialog("Message"); //the info message dialog
    GuiLabel message = new GuiLabel("Success!", dialog);

    //check whether the message dialog contains the expected user message
    String expectedMessage = "Success! You typed the right password.";
    assertEquals(expectedMessage, message.getActualId());

    //close the message dialog
    dialog.close();

    //Alternatively,  close the message dialog by clicking the OK button
    //ok.setContainer(dialog);
    //ok.click();
}
```



Scripting: C# Example

```
[Test]
public void Test001()
{
    var t = new GuiPasswordText("Enter the password");
    //check wheter the text field is editable
    Assert.True(t.IsEditable());
    //enter the password
    t.SetText("bugaboo");

    var ok = new GuiButton("OK");
    //check wheter the push button is enabled
    Assert.True(ok.IsEnabled());
    //click the OK button in the main frame
    ok.Click();

    var dialog = new GuiDialog("Message");
    var message = new GuiLabel("Success!", dialog);

    var expectedMessage = "Success! You typed the right password.";
    //check wheter the message dialog contains the excepted message
    Assert.True(message.GetActualId().Equals(expectedMessage));

    //close the dialog
    dialog.Close();
}
```



Scriptless Testing

- No programming skills are needed;
- Easy to learn and use;
- Promotes an improved functional coverage;
- Favours the participation of all the stakeholders;

Scriptless: Excel Example

Test Case :

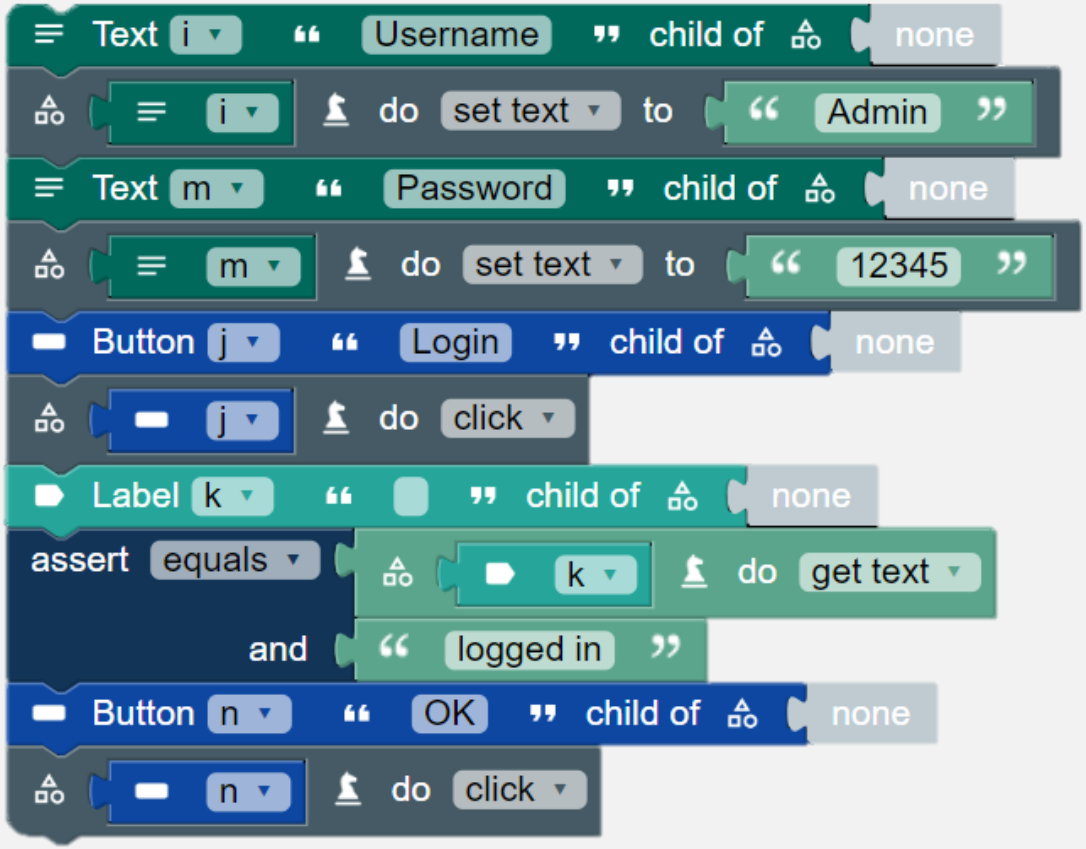
1. Start the Application
2. Enter valid username
3. Enter valid Password
4. Click “Login” button
5. Check the results: “logged in”
6. Click “OK" button
7. Close the Application

	A	B	C	D	E	F	G
1	Test Case ID :	TC_01	Author(s) :	Maveryx	Test Case		
2	Description :	Click on login button w	Requirement(s) :	REQ_1			
3	OBJECT	NAME	CONTAINER	CONTAINER NAME	ACTION	DATA	DATA
4					START	Login	
5	TEXT	Username	DIALOG	Login	SET_TEXT	dgraham	
6	PASSWORD_TEXT	Password	DIALOG	Login	SET_TEXT	dgraham01	
7	BUTTON	Login	DIALOG	Login	CLICK		
8	LABEL		DIALOG	Login	HAS_TEXT	Hi dgraham! You have successfully logged in	
9	BUTTON	OK	DIALOG	Login	CLICK		
10					CLOSE	Login	

Scriptless: Blockly Example

Test Case :

1. Start the Application
2. Enter valid username
3. Enter valid Password
4. Click “Login” button
5. Check the label: “logged in”
6. Click “OK" button
7. Close the Application



Data-Driven Testing

- **Data-Driven Testing** allows writing the test cases as scripts those read their data from external files or db

- One script to drive the tests and changing the data you can create any number of test cases

Add-ons

Extension plugin mechanism & interfaces

- ❑to support custom controls
- ❑to add new keyword (action) libraries

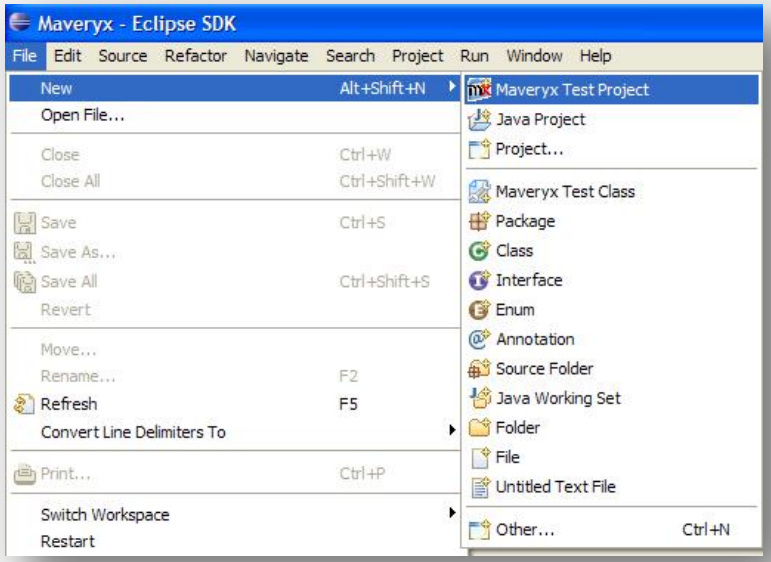
API extension

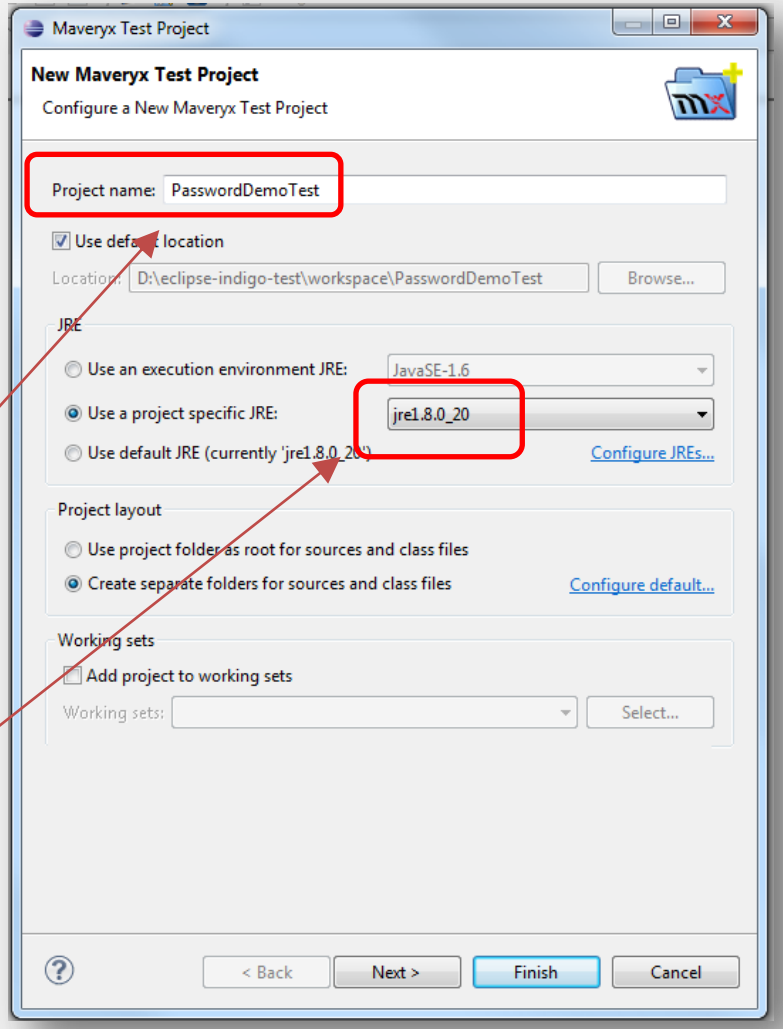
- ❑to create new Test API

Create and run a keyword-driven test

1. **Create a new Maveryx Test Project**
2. Write the test case
3. Run the test

Create New Test Project





1. Select **File** → **New** → **Maveryx Test Project**

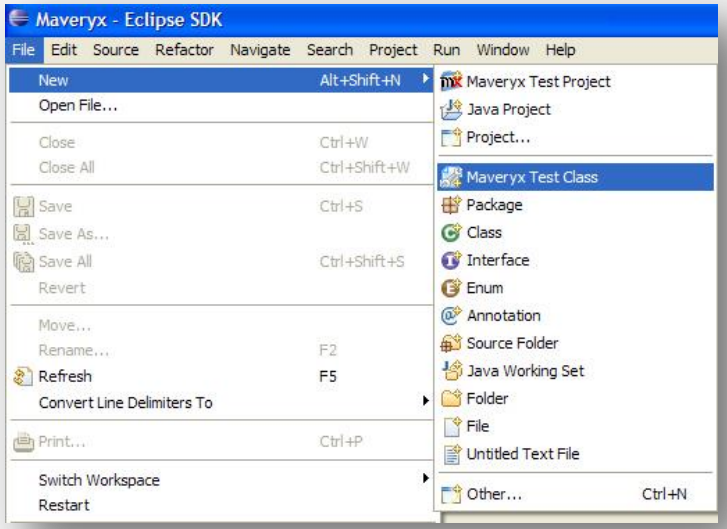
In the **Maveryx Test Project** window

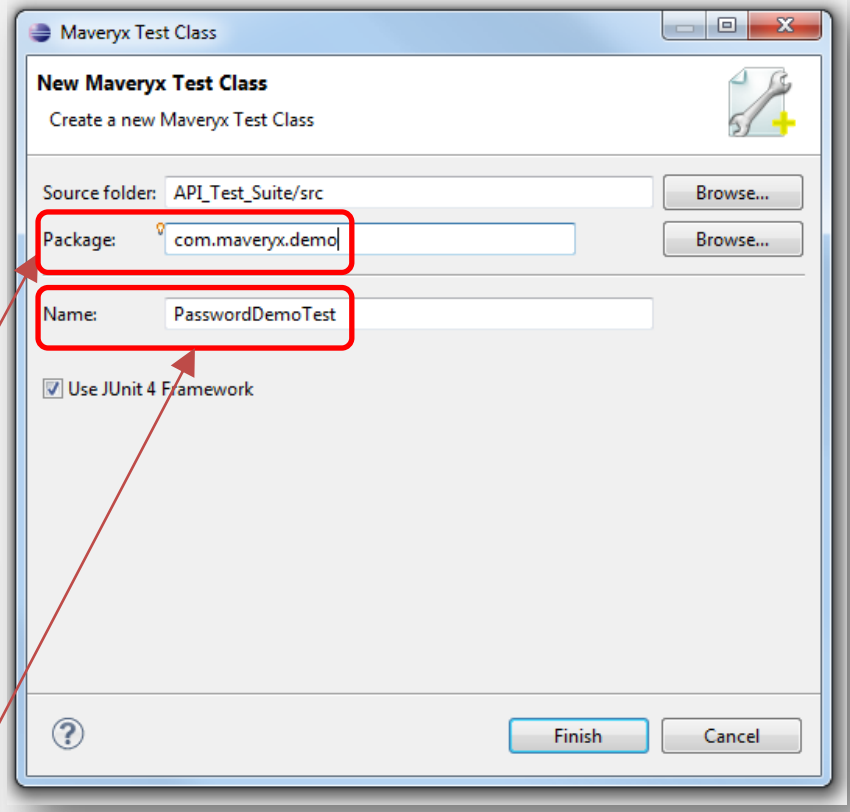
   1. enter the Project name (e.g. "*PasswordDemoTest*")
   2. in the **JRE** section make sure that Java/JRE 8 or higher is selected

2. Click **Finish**

Test it simple!

Create New Test Script





1. Select **File** → **New** → **Maveryx Test Class**

In the **Maveryx Test Class** window

   1. enter a name for the Package (e.g. "*com.maveryx.demo*")
   2. enter a Name for the test class / script (e.g. "*PasswordDemoTest*")

2. Click **Finish**

Test it simple!

Test Script "stub"

```java
package com.maveryx.demo.java.junit;

import org.junit.After;

@RunWith(com.maveryx.test.junit.MaveryxTestRunner.class)
public class prova {

    /**
      * Change this path to your current application's XML launch file.
    */
    private static final String pathName = "C:\\Maveryx\\demo\\AUTconfiguration.xml";

    /**
     * Default constructor.
     * @throws Exception
    */
    public prova() throws Exception {
        super();
    }

    /**
    * Start the Application-Under-Test.
    * @throws Exception
    */
    @Before
    public void setUp() throws Exception {
        Bootstrap.startApplication(pathName); //start the application under test
    }

    /**
    * Close the Application-Under-Test.
    * @throws Exception
    */
    @After
    public void tearDown() throws Exception {

        Bootstrap.stop(); //close the application under test

    }

    /**
      *  Test 1
      * @throws Exception
    */
    @Test
    public void test001() throws Exception {

        //Write here your test case

    }

}
```

Set the full path (*pathName*) to the **AUT *launch*** file.

e.g. **private final** String *pathName* = "C:/Maveryx/demo/AUT/PasswordDemo.xml";

The static method ***startApplication(pathName)*** in class *Bootstrap* launches the AUT

The static method ***stop()*** in class *Bootstrap* closes the AUT.

Java AUT Launch File

To execute a Java Application-Under-Test it is necessary to create the related AUT launch file.

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

    <WORKING_DIR>./src/resources/AUT/java</WORKING_DIR> <!-- change this path to your working directory -->

    <APPLICATION_NAME>ButtonDemo</APPLICATION_NAME>

    <AUT_ARGUMENTS></AUT_ARGUMENTS>

    <VM_ARGUMENTS></VM_ARGUMENTS>

    <DESCRIPTION>
        Push-Button testing
    </DESCRIPTION>

    <JRE_PATH>${java.home}</JRE_PATH> <!-- change this path to your JRE home -->

    <MAIN_CLASS>com.sun.demo.ButtonDemo</MAIN_CLASS>

    <!-- on UNIX-like and MAC OS X systems change the path separator ';' to ':' -->
    <CLASSPATH>
        <LIB>
            <PATH>examples.jar</PATH> <!-- change this path to your Maveryx installation directory /demo -->
        </LIB>
        <!-- do not change the data below! (except for path separator on UNIX-like and MAC OS X systems) -->
    </CLASSPATH>
</AUT_DATA>
```

MFC & .Net AUT Launch File

To execute a MFC or .NET Application-Under-Test it is necessary to create the related **AUT launch file**.

```
<?xml version="1.0" encoding="UTF-8"?>
<AUT_DATA>
    <EXECUTABLE_PATH>.\src\resources\AUT\windows\Notepad Enhanced.exe</EXECUTABLE_PATH>
    <APPLICATION_NAME>Notepad Enhanced</APPLICATION_NAME>
    <TOOLKIT>WIN</TOOLKIT>
    <TIMEOUT>1000</TIMEOUT>
    <DELTA_CHECK>1000</DELTA_CHECK>
    <AUT_ARGUMENTS></AUT_ARGUMENTS>
</AUT_DATA>
```

Set the absolute or relative path to your AUT executable file

Web AUT Launch File

To execute a Web Application-Under-Test it is necessary to create the related **AUT launch file**.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<AUT_DATA>
    <EXECUTABLE_PATH>C:/Program Files (x86)/Google/Chrome/Application/chrome.exe</EXECUTABLE_PATH>
    <APPLICATION_NAME>CHROME</APPLICATION_NAME>
    <TOOLKIT>WEB</TOOLKIT>
    <AUT_ARGUMENTS>file:///./src/resources/AUT/web/index.html</AUT_ARGUMENTS>
</AUT_DATA>
```

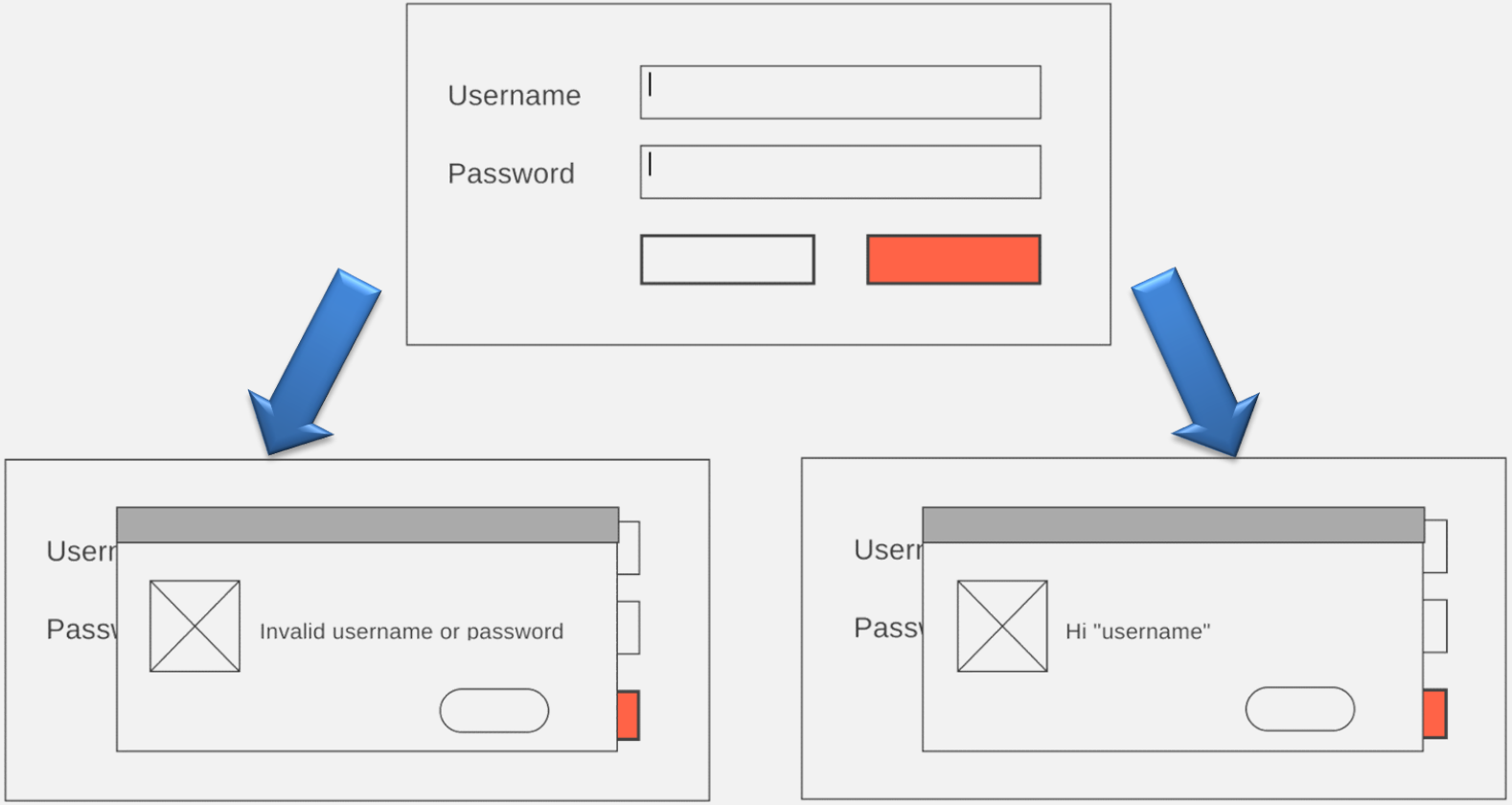
Set the URL of the AUT

Set the path of the browser **you** want to use for your tests

Create and run a keyword-driven test

1. Create a new Maveryx Test Project
2. **Write the test case**
3. Run the test

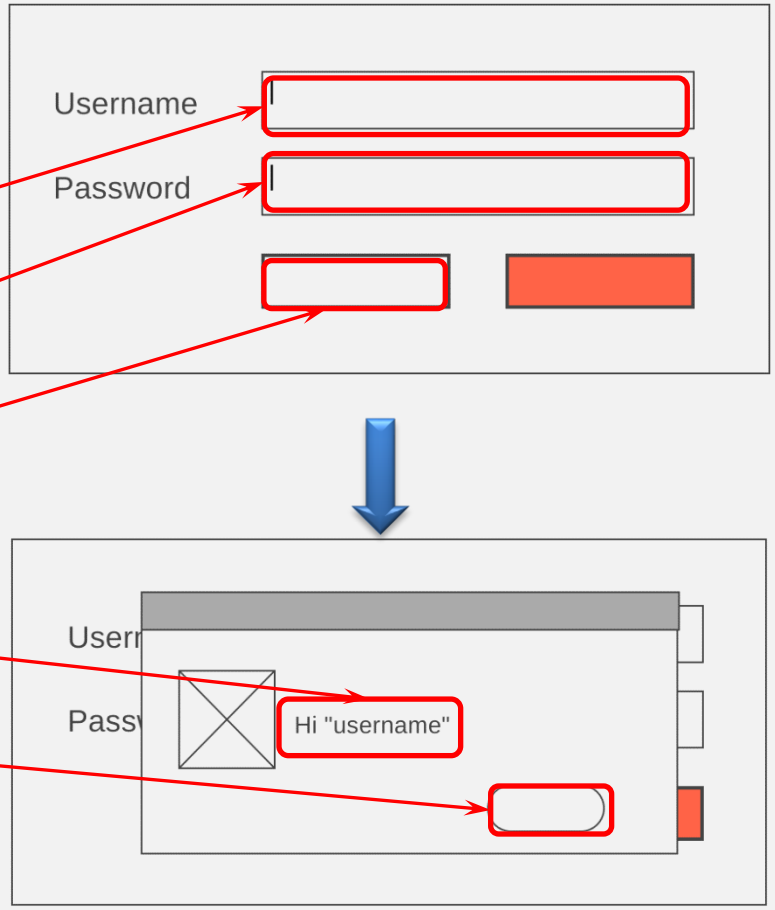
The Sample AUT



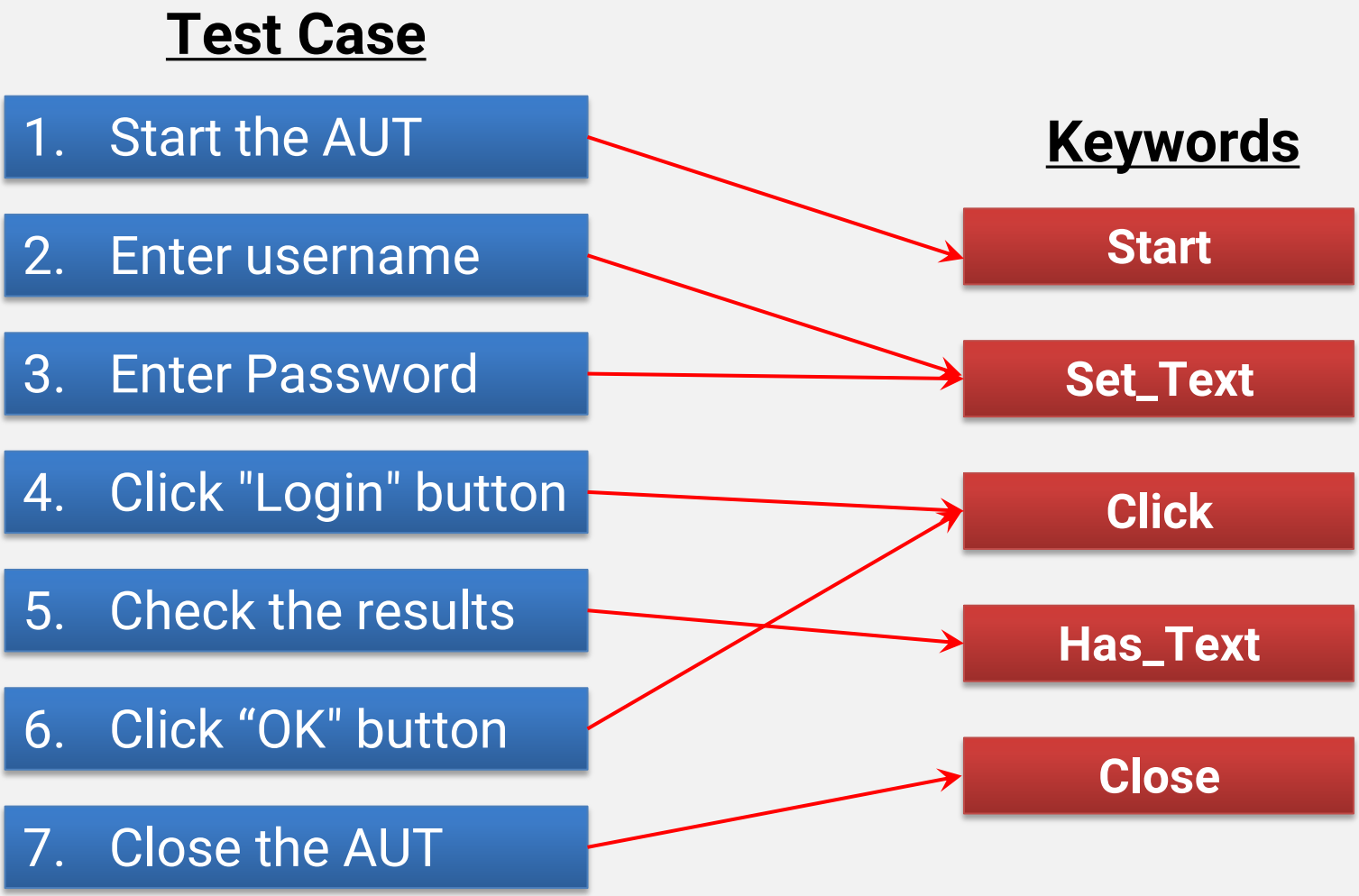
Test Case #001

Test Case : TC_01

1. Start the Application
2. Enter valid username
3. Enter valid Password
4. Click “Login” button
5. Check the results: Hi "username”
6. Click “OK" button
7. Close the Application



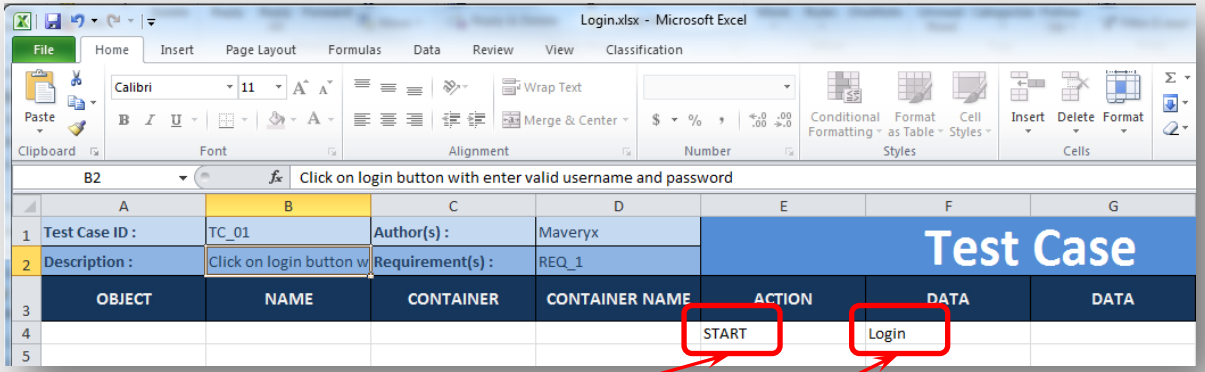
Identify Keywords



Design Test step 1

Test Case : TC_01

1. **Start the Application**
2. Enter valid username
3. Enter valid Password
4. Click “Login” button
5. Check the results: “Hi alfonso”
6. Click “OK" button
7. Close the Application

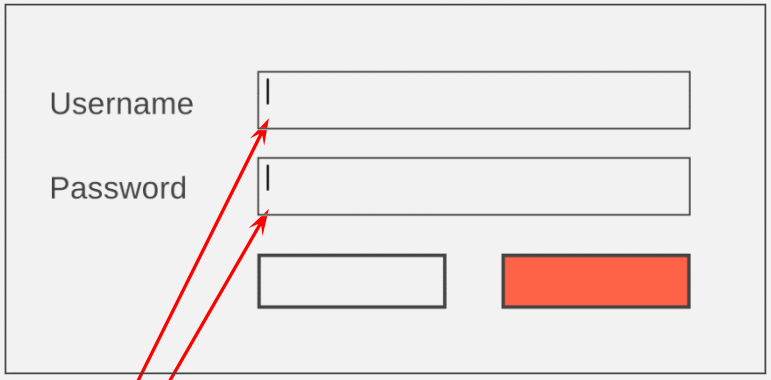


Keyword	Data / Input	Description
START	*AUT lauch file path*	Launch the AUT

Design Test step 2 & 3

Test Case : TC_01

1. Start the Application
2. **Enter valid username**
3. **Enter valid Password**
4. Click “Login” button
5. Check the results: “Hi alfonso”
6. Click “OK" button
7. Close the Application



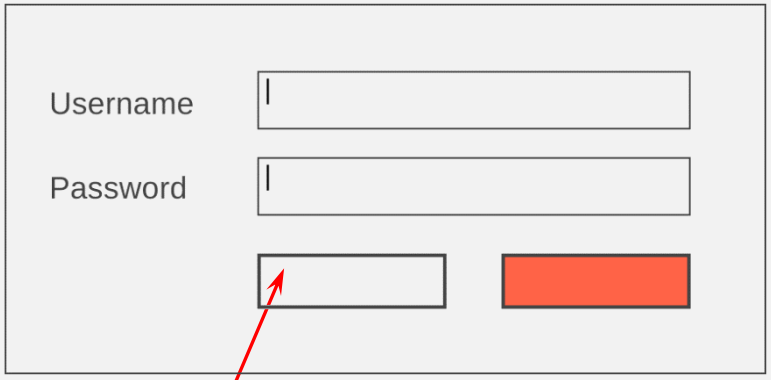
1	Test Case ID :		Author(s) :		Test Case		
2	Description :		Requirement(s) :				
3	OBJECT	NAME	CONTAINER	CONTAINER NAME	ACTION	DATA	DATA
4					START	Login	
5	TEXT	Username			SET_TEXT	alfonso	
6	PASSWORD_TEXT	Password			SET_TEXT	alfonsopwd	
7							

Keyword	Data / Input	Description
SET_TEXT	*Text*	Set the text into a text field

Design Test step 4

Test Case : TC_01

1. Start the Application
2. Enter valid username
3. Enter valid Password
4. **Click “Login” button**
5. Check the results: “Hi alfonso”
6. Click “OK" button
7. Close the Application



1	Test Case ID :		Author(s) :		Test	
2	Description :		Requirement(s) :			
3	OBJECT	NAME	CONTAINER	CONTAINER NAME	ACTION	DATA
4					START	Login
5	TEXT	Username			SET_TEXT	alfonso
6	PASSWORD_TEXT	Password			SET_TEXT	alfonsopwd
7	BUTTON	Login			CLICK	
8						

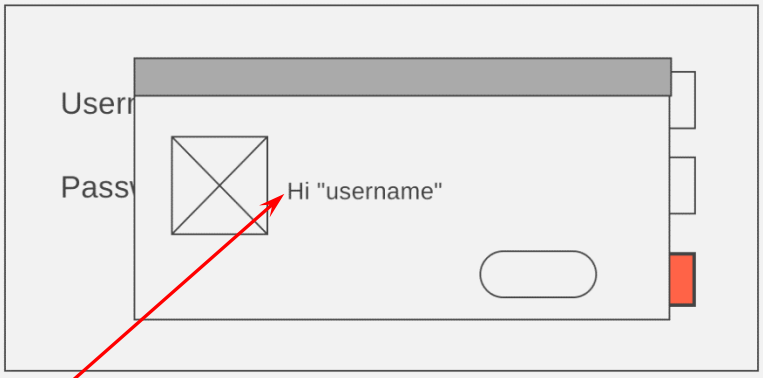
Keyword	Data / Input	Description
CLICK		Click the button

Test it simple!

Design Test step 5

Test Case : TC_01

1. Start the Application
2. Enter valid username
3. Enter valid Password
4. Click “Login” button
5. **Check the results: “Hi alfonso”**
6. Click “OK" button
7. Close the Application



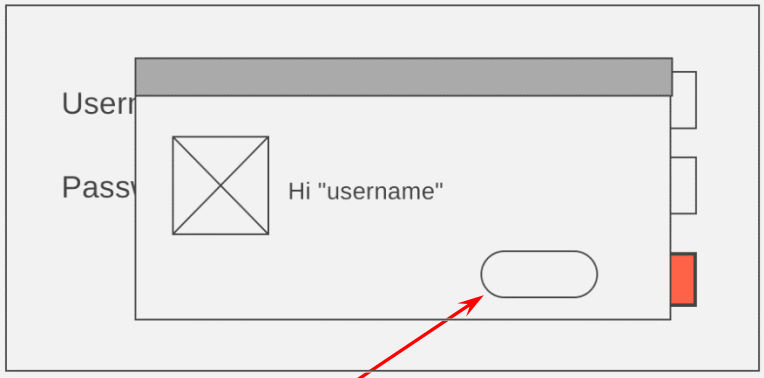
					Test Case		
1	Test Case ID :		Author(s) :				
2	Description :		Requirement(s) :				
3	OBJECT	NAME	CONTAINER	CONTAINER NAME	ACTION	DATA	DATA
4					START	Login	
5	TEXT	Username			SET_TEXT	alfonso	
6	PASSWORD_TEXT	Password			SET_TEXT	alfonsopwd	
7	BUTTON	Login			CLICK		
8	LABEL				HAS_TEXT	Hi alfonso	
9							

Keyword	Data / Input	Description
HAS_TEXT	*Text*	Check whether the given LABEL has the actual *Text*

Design Test step 6

Test Case : TC_01

1. Start the Application
2. Enter valid username
3. Enter valid Password
4. Click “Login” button
5. Check the results: “logged in”
6. **Click “OK" button**
7. Close the Application



1	Test Case ID :		Author(s) :		Test Case		
2	Description :		Requirement(s) :				
3	OBJECT	NAME	CONTAINER	CONTAINER NAME	ACTION	DATA	DATA
4					START	Login	
5	TEXT	Username			SET_TEXT	alfonso	
6	PASSWORD_TEXT	Password			SET_TEXT	alfonsopwd	
7	BUTTON	Login			CLICK		
8	LABEL				HAS_TEXT	Hi alfonso	
9	BUTTON	OK			CLICK		
10							

Keyword	Data / Input	Description
CLICK		Click the button

Test it simple!

Design Test step 7

Test Case : TC_01

1. Start the Application
2. Enter valid username
3. Enter valid Password
4. Click “Login” button
5. Check the results: “logged in”
6. Click “OK" button
7. **Close the Application**

1	Test Case ID :		Author(s) :		Test Case		
2	Description :		Requirement(s) :				
3	OBJECT	NAME	CONTAINER	CONTAINER NAME	ACTION	DATA	DATA
4					START	Login	
5	TEXT	Username			SET_TEXT	alfonso	
6	PASSWORD_TEXT	Password			SET_TEXT	alfonsopwd	
7	BUTTON	Login			CLICK		
8	LABEL				HAS_TEXT	Hi alfonso	
9	BUTTON	OK			CLICK		
10					CLOSE	Login	
11							

Keyword	Data / Input	Description
CLOSE		Close the AUT

Create and run a keyword-driven test

1. Create a new Maveryx Test Project
2. Write the test case
3. **Run the test**

Create the Driver Script

ScriptlessLoginTest.java

```java
package org.maveryx.demo;

import org.junit.After;

@RunWith(org.maveryx.test.junit.MaveryxTestRunner.class)
public class ScriptlessLoginTest {

    public ScriptlessLoginTest() throws Exception {
        super();
    }

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    /**
     * Click on login button entering valid username and password
     * Expected: Successfull login
     */
    @Test
    public void tc_01() throws Exception {
        new KeywordDrivenTestManager().run("data/Login_01.xls");
    }

    /**
     * Click on login button entering invalid username and password
     * Expected: Login failed
     */
    @Test
    public void tc_02() throws Exception {
        new KeywordDrivenTestManager().run("data/Login_02.xls");
    }
}
```

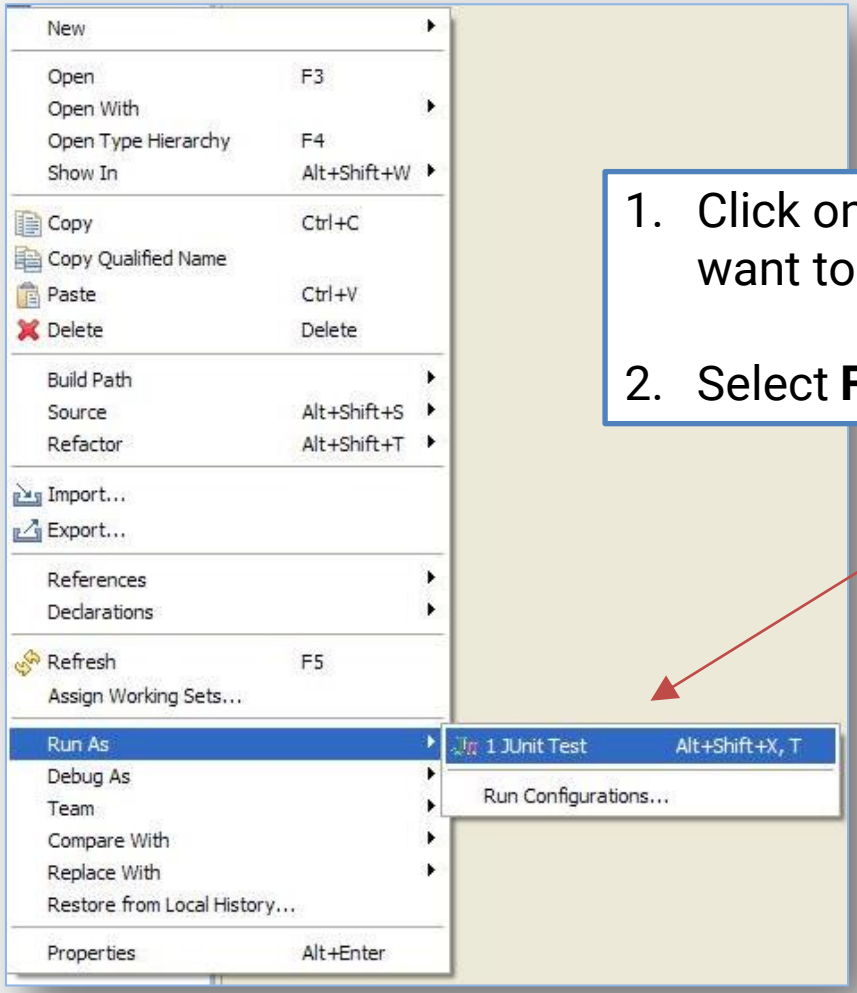
a. Create a new object ***KeywordDrivenTestManager***

```java
new KeywordDrivenTestManager()
```

b. Call the method **run()** specifying the path of the Excel file

```java
run("data/Login_01.xls");
```

Run a Test Script



1. Click on the test class or package you want to run

2. Select **File** → **Run As** → **JUnit Test**

Alternatively

By command line ***KeywordDrivenTestingCLI*** with the following arguments:

For example:

```
KeywordDrivenTestingCLI –o "C:\Report" "C:\Test\test001.xls"
```

The test script *C:\Test\test001.xls* is executed and the test report is stored in *C:\Report*

Or:

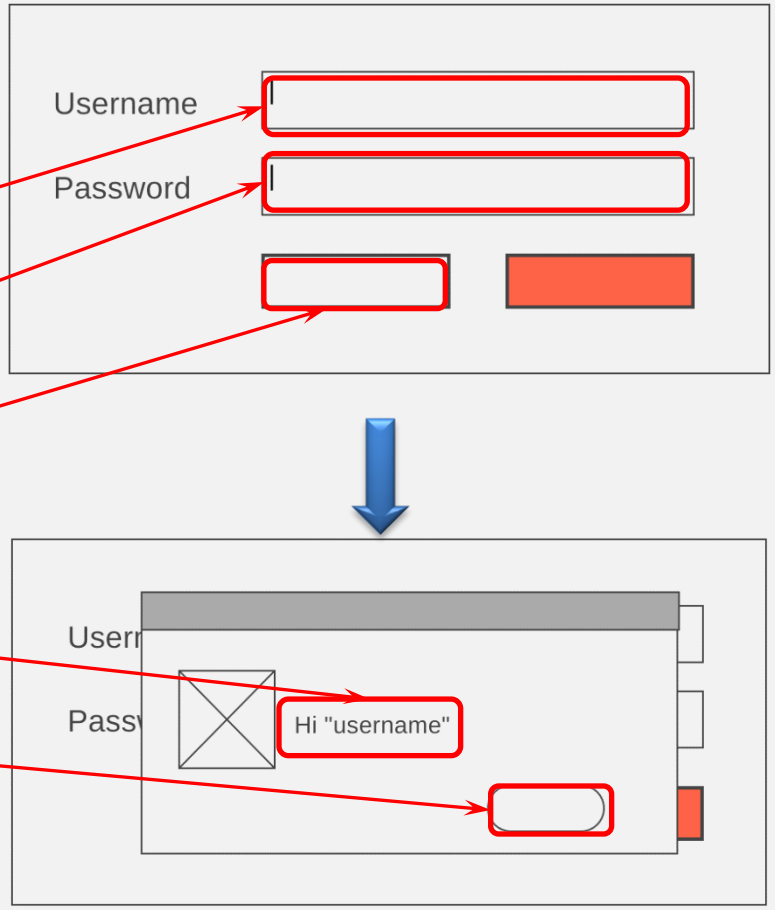
```
KeywordDrivenTestingCLI –o "C:\Report" –e
"C:\Test\test001.xls " "C:\Test"
```

All test scripts in *C:\Test* (including subfolders) are executed except *C:\Test\test001.xls* and the test report is stored in *C:\Report*

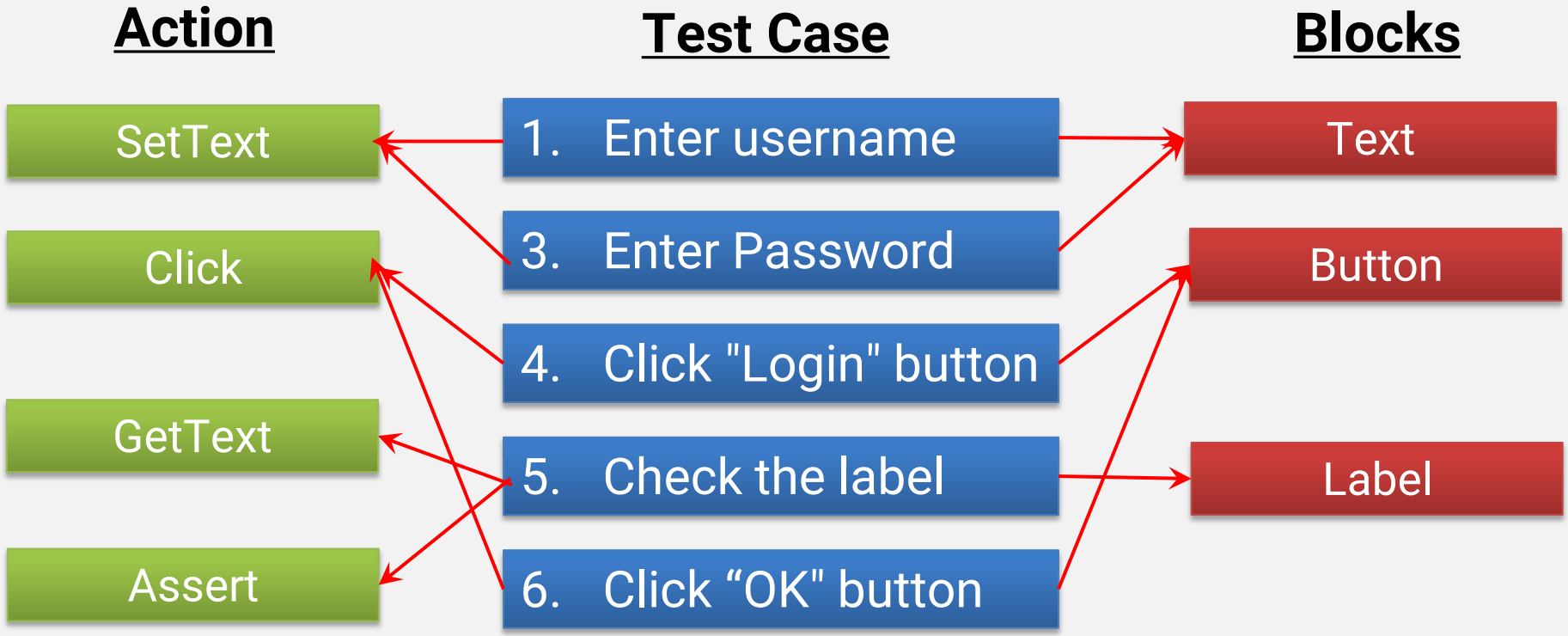
Test Case #001

Test Case : TC_01

1. Start the Application
2. Enter valid username
3. Enter valid Password
4. Click “Login” button
5. Check the results: Hi "username”
6. Click “OK" button
7. Close the Application



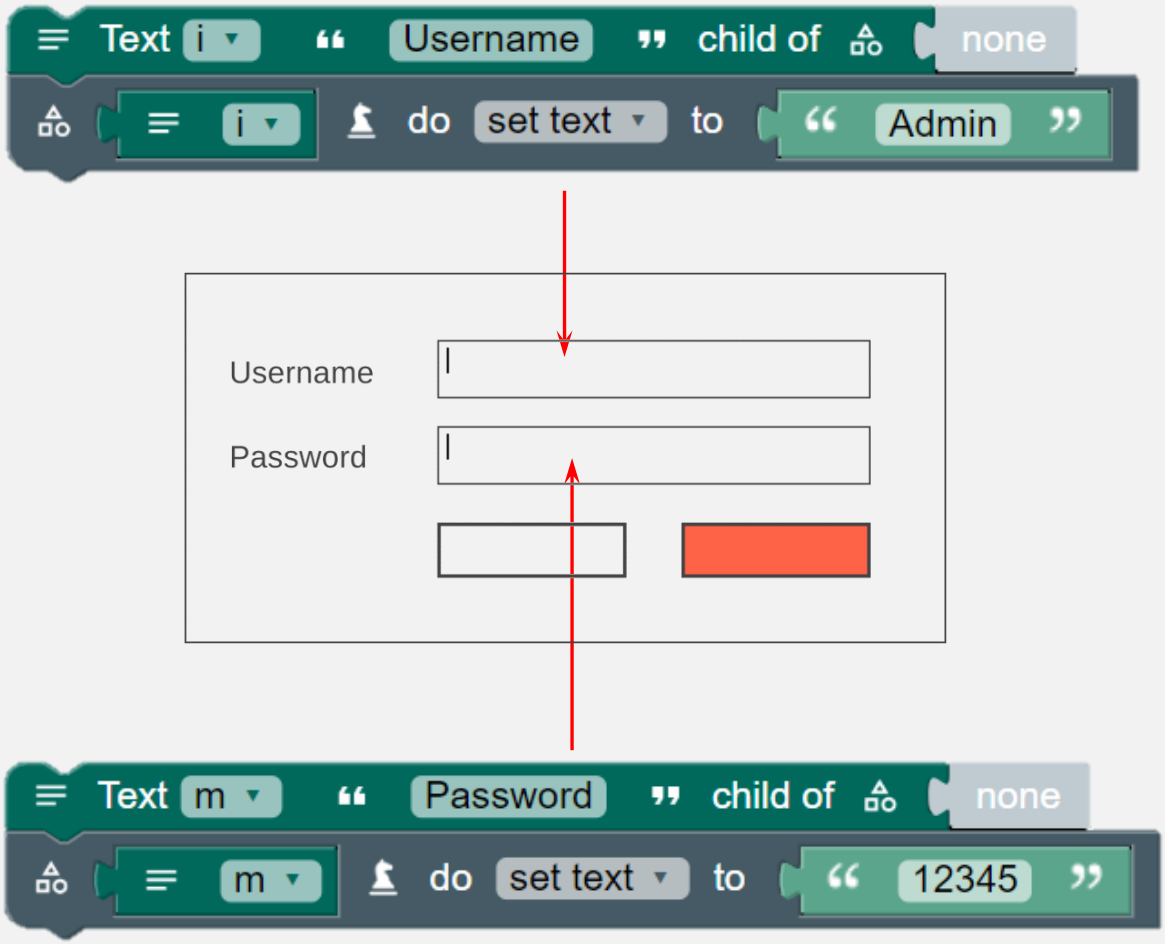
Identify Blocks



Design Test step 2 & 3

Test Case : TC_01

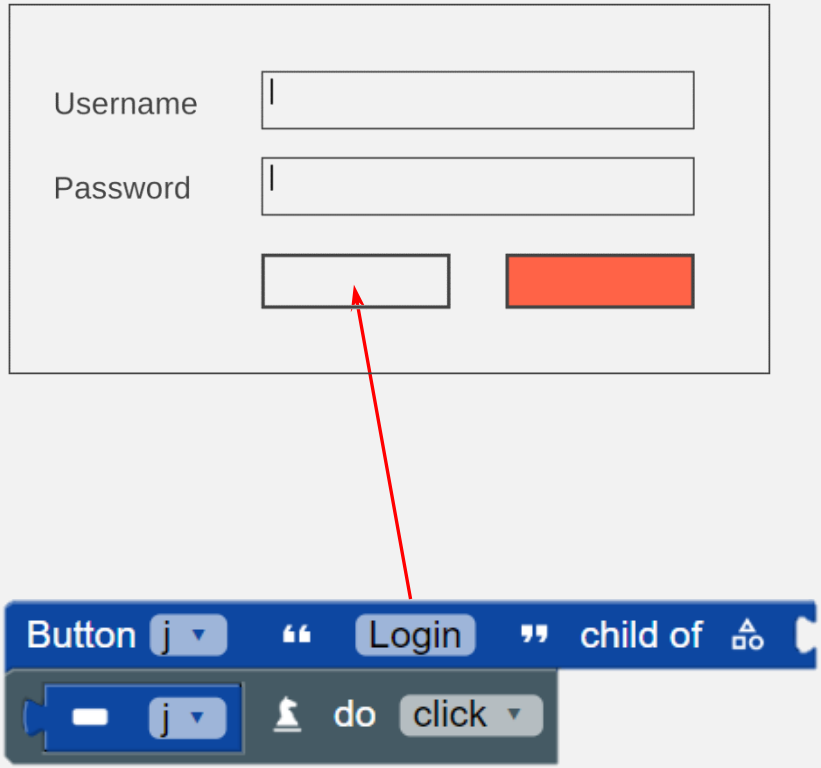
1. **Enter valid username**
2. **Enter valid Password**
3. Click “Login” button
4. Check label: “Hi Admin”
5. Click “OK" button



Design Test step 4

Test Case : TC_01

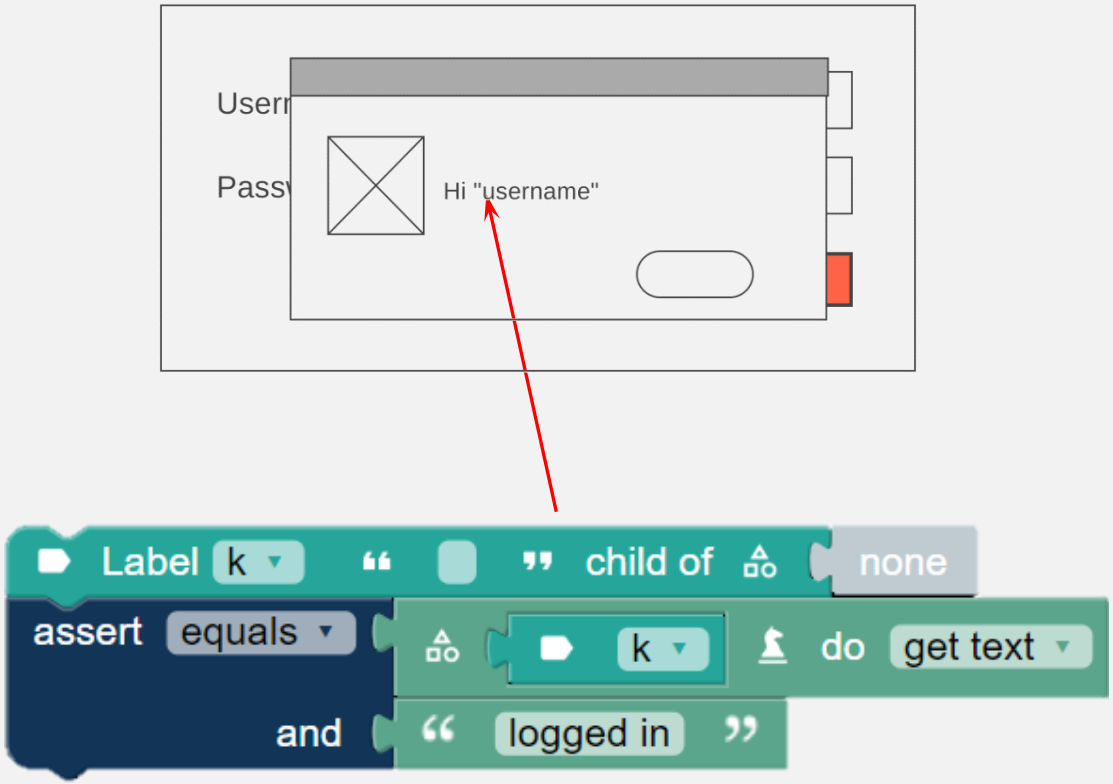
1. Enter valid username
2. Enter valid Password
3. **Click “Login” button**
4. Check label: “Hi Admin”
5. Click “OK" button



Design Test step 5

Test Case : TC_01

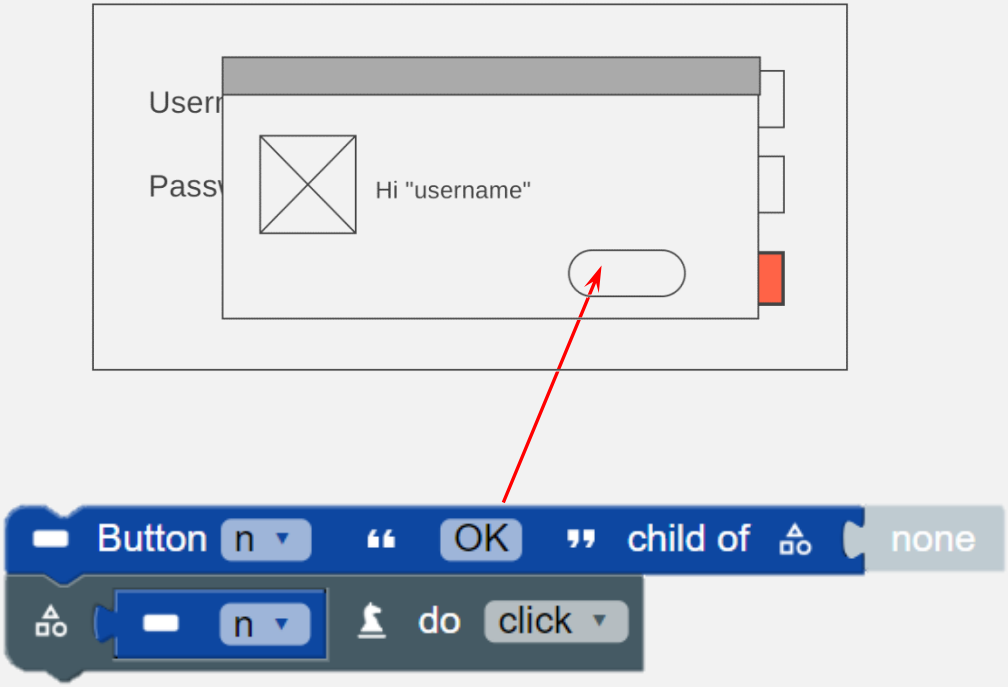
1. Enter valid username
2. Enter valid Password
3. Click “Login” button
4. **Check label: “Hi Admin”**
5. Click “OK" button



Design Test step 6

Test Case : TC_01

1. Enter valid username
2. Enter valid Password
3. Click “Login” button
4. Check label: “Hi Admin”
5. **Click “OK" button**



Customer Case

Boeing Defence Australia

The team develops Mission Computing Software for an airborne command and control platform.

- This project was the major mid-life upgrade and the team wanted to introduce automated testing.
- The challenge was to find a tool that it could integrate with a large existing codebase quickly and efficiently, that would provide a robust framework for the testing of additional new features.

BDA

"Maveryx's unique innovative technology made the tool selection task less difficult. Their technology, as opposed to other established GUI automated test tools, promotes efficiency for a start from scratch test automation solution on a legacy application by not having to spend a long time developing an object repository."

Paul D. Ellis, Principal Software Engineer at BOEING

DBA Environment

- Nightly GUI automated test runs:
  - Unit testing;
  - Build Verification Testing;

- Weekly collaborative automated test runs.

The code-base is instrumented for memory error detection and code coverage.

Maveryx is used to drive the execution paths of the instrumented executables and libraries.

THANK YOU!

MAVERYX®
Test it simple!