

Robot Framework - Quick Guide

Robot Framework - Overview

Robot Framework is an open source test automation framework for acceptance testing and acceptance test-driven development. It follows different test case styles – keyword-driven, behaviour-driven and data-driven for writing test cases. Robot Framework provides support for external libraries, tools which are open source and can be used for automation. The most popular library used is Selenium Library used for web development & UI testing.

Test cases are written using keyword style in a tabular format. You can use any text editor or Robot Integrated Development Environment (RIDE) for writing test cases.

Robot framework works fine on all the Operating Systems available. The framework is built on Python and runs on Jython (JVM) and IronPython (.NET).

Robot Framework Features

In this section, we will look at the different features offered by Robot.

Tabular format for test cases

Robot framework comes with a simple tabular format where the test cases are written using keywords. It is easy for a new developer to understand and write test cases.

Keywords

Robot framework comes with built-in keywords available with robot framework, keywords available from the libraries like Selenium Library (open browser, close browser, maximize browser, etc.). We can also create user-defined keywords, which are a combination of other user-defined keywords or built-in or library keywords. We can also pass arguments to those keywords, which make the user-defined keywords like functions that can be reused.

Variables

Robot framework supports variables – scalar, list and dict. Variables in robot framework are easy to use and are of great help while writing complex test cases.

Libraries

Robot framework has support for a lot of external libraries like SeleniumLibrary, Database Library, FTP Library and http library. SeleniumLibrary is mostly used as it helps to interact with the browsers and helps with web application and UI testing. Robot framework also has its own built-in libraries for strings, date, numbers etc.

Resources

Robot framework also allows the import of robot files with keywords externally to be used with test cases. Resources are very easy to use and are of great help when we need to use some keywords already written for other test projects.

Data driven test cases

Robot framework supports keyword driven style test cases and data driven style. Data driven works with high-level keyword used as a template to the test suite and the test cases are used to share data with the high-level keyword defined in the template. It makes the work very easy for testing UI with different inputs.

Test Case Tagging

Robot framework allows to tag test-cases so that we can either run the tags test-cases or skip the tagged testcases. Tagging helps when we want to run only a group of test cases or skip them.

Reports and Logs

Robot framework provides all the details of test suite, test case execution in the form of report and logs. All the execution details of the test case are available in the log file. The details like whether the test case has failed or passed, time taken for execution, steps followed to run the test case are provided.

RIDE

This editor available with Robot framework helps in writing and running test cases. The editor is very easy to install and use. RIDE makes life easy for writing test cases by providing framework specific code completion, syntax highlighting, etc. Creation of project, test suite, test case, keywords, variables, importing library, executing, tagging the test case is easily done in the editor. Robot framework also provides plugins for eclipse, sublime, Textmate, Pycharm that has support for robot test cases.

Robot Framework Advantages

Robot framework is open source, so anyone who wants to try out can easily do so.

- It is very easy to install and helps in creating and executing test cases. Any new comer can easily understand and does not need any high level knowledge of testing to get started with robot framework.
- It supports keyword-driven, behaviour-driven and data-driven style of writing test cases.
- It is a good support for external libraries. Most used is Selenium Library, which is easy to install and use in robot framework.

Robot Framework Limitations

Robot lacks support for if-else, nested loops, which are required when the code gets complex.

Conclusion

Robot Framework is an open source test automation framework for acceptance testing and acceptance test-driven development. The test cases in Robot Framework are based on keywords written in tabular format, which makes it clear and readable, and conveys the right information about the intention of the test case. For example, to open browser, the keyword used is “**Open Browser**”.

Robot Framework - Environment Setup

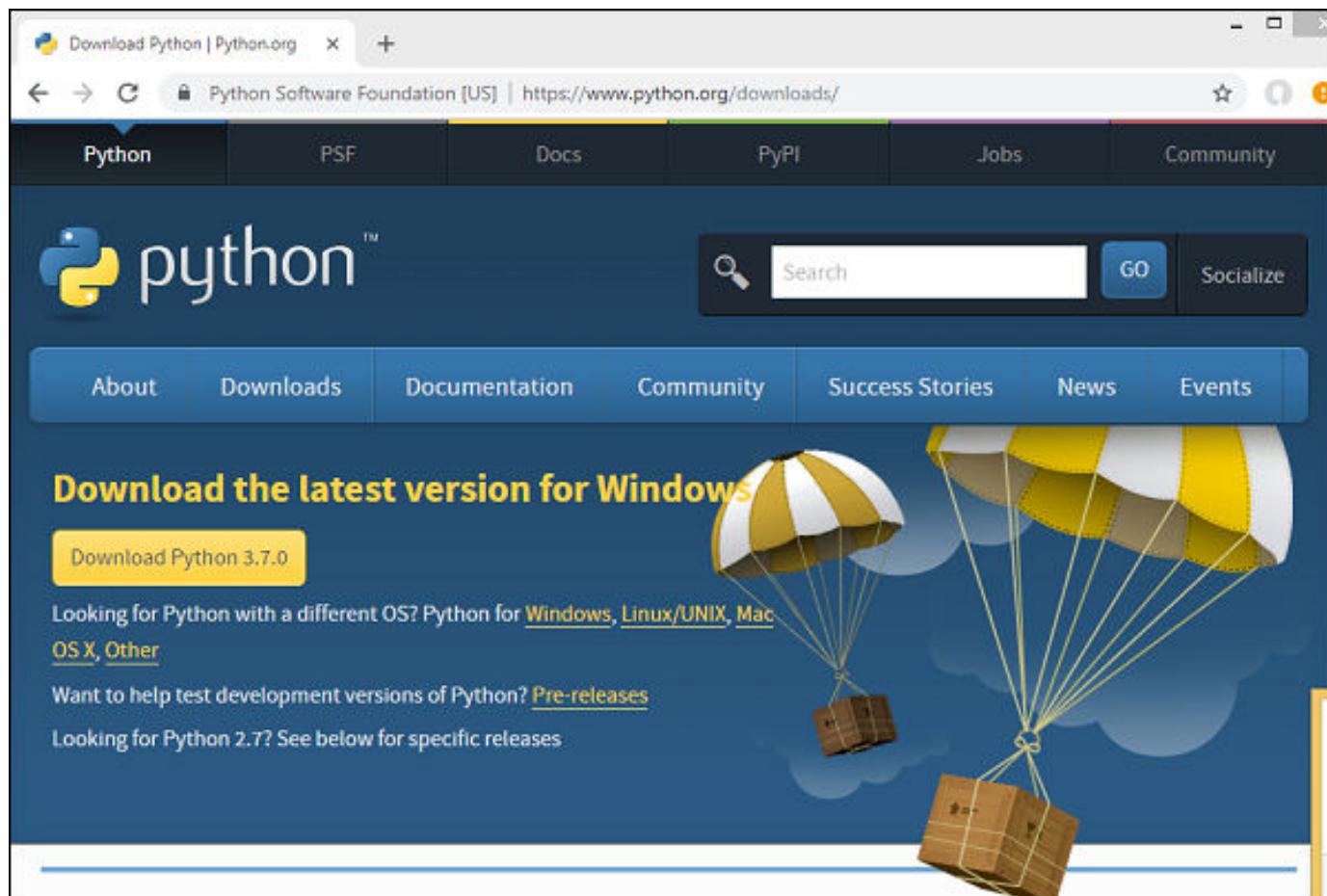
Robot framework is built using python. In this chapter, we will learn how to set up Robot Framework. To work with Robot Framework, we need to install the following –

- Python
- pip
- Robot Framework
- wxPython for Ride IDE
- Robot Framework Ride

Install Python

To install python, go to python official site – <https://www.python.org/downloads/> and download the latest version or the prior version of python as per your operating system (Windows, Linux/Unix, Mac, and OS X) you are going to use.

Here is the screenshot of the python download site –



The latest version available as per release dates are as follows –

Release version	Release date	Click for more	
Python 3.5.6	2018-03-02	Download	Release Notes
Python 3.4.9	2018-03-02	Download	Release Notes
Python 3.7.0	2018-06-27	Download	Release Notes
Python 3.6.6	2018-06-27	Download	Release Notes
Python 2.7.15	2018-05-01	Download	Release Notes
Python 3.6.5	2018-03-28	Download	Release Notes
Python 3.4.8	2018-02-05	Download	Release Notes
Python 3.6.4	2018-01-05	Download	Release Notes

Before you download python, it is recommended you check your system if python is already present by running the following command in the command line –

Windows Installation

```
python --version
```



```
C:\>python --version
'python' is not recognized as an internal or external command,
operable program or batch file.

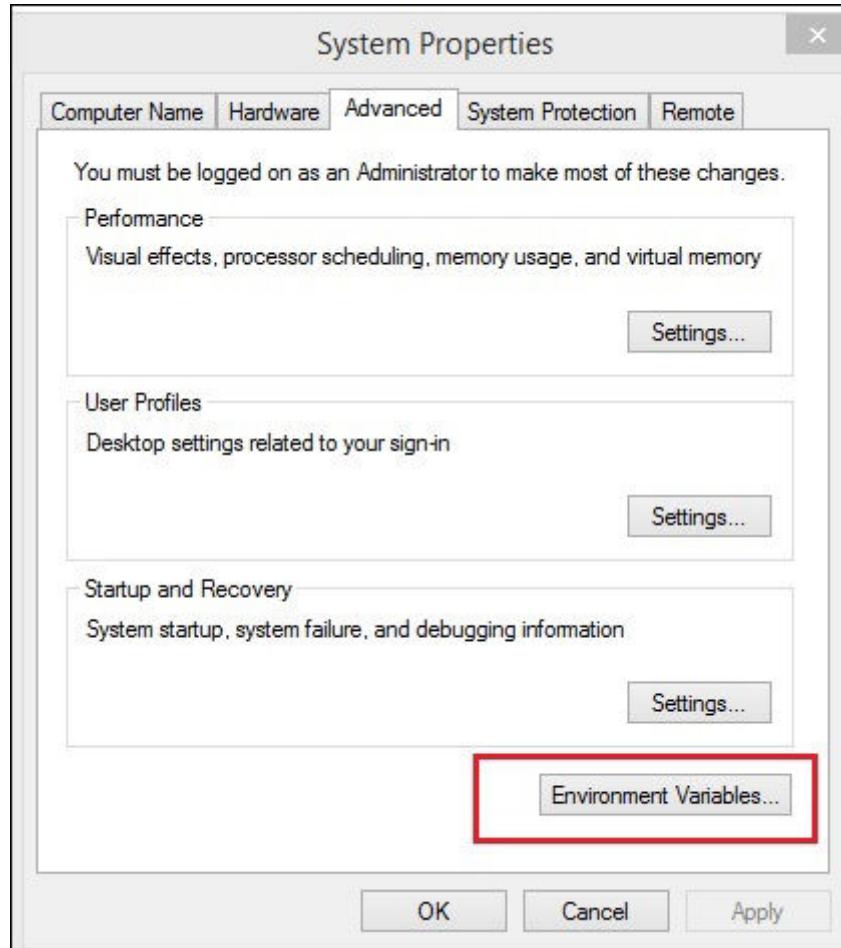
C:\>_
```

If we get the version of python as output then, we have python installed in our system. Otherwise, you will get a display as shown above.

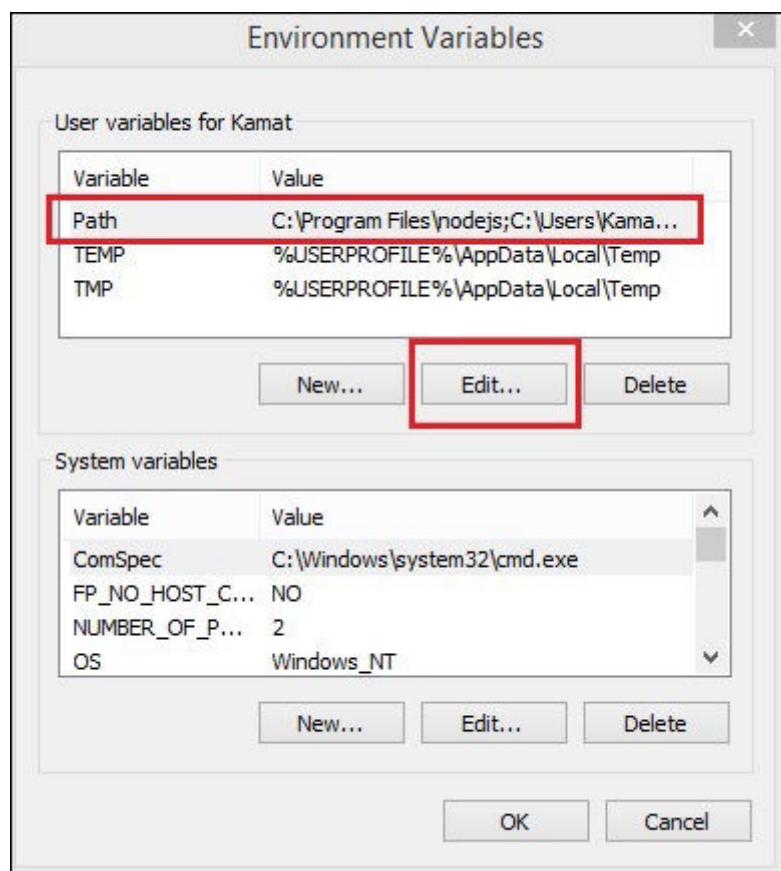
Here, we will download python version 2.7 as it is compatible to the windows 8 we are using right now. Once downloaded, install python on your system by double-clicking on .exe python download. Follow the installation steps to install Python on your system. Once installed, to make python available globally, we need to add the path to environment variables in windows as follows –

Setting path for Windows

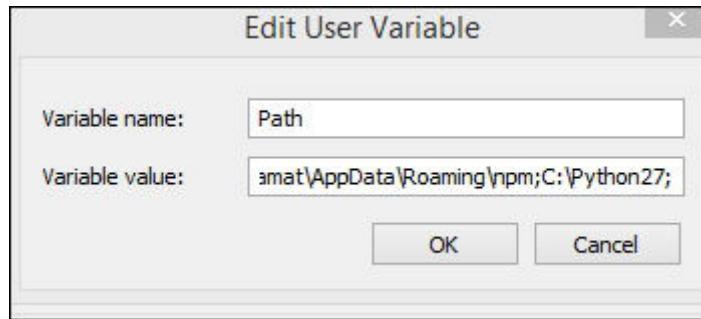
Right-click on My Computer icon and select properties. Click on Advanced System setting and the following screen will be displayed.



Click on *Environment Variables* button highlighted above and it will show you the screen as follows –



Select the *Variable Path* and click the *Edit* button.



Get the path where python is installed and add the same to Variable value at the end as shown above.

Once this is done, you can check if python is installed from any path or directory as shown below –

```
C:\>python --version
Python 2.7.15
C:\>
```

Robot Framework - Unix and Linux Installation

Let us now see a few simple steps to install Python on Unix/Linux machine. Open a Web browser and go to <https://www.python.org/downloads/> .

- Follow the link to download zipped source code available for Unix/Linux.
- Download and extract files.
- Editing the Modules/Setup file if you want to customize some options.
- run `./configure` script
- `make`
- `make install`

This installs Python at standard location `/usr/local/bin` and its libraries at `/usr/local/lib/pythonXX` where XX is the version of Python.

Setting Path at Unix/Linux

To add the Python directory to the path for a particular session in Unix –

In the csh shell

type setenv PATH "\$PATH:/usr/local/bin/python" and press Enter.

In the bash shell (Linux)

type export ATH="\$PATH:/usr/local/bin/python" and press Enter.

In the sh or ksh shell

type PATH="\$PATH:/usr/local/bin/python" and press Enter.

Note – /usr/local/bin/python is the path of the Python directory

Install PIP

Now, we will check for the next step, which is pip installation for python. PIP is a package manager to install modules for python.

PIP gets installed along with python and you can check the same in command line as follows –

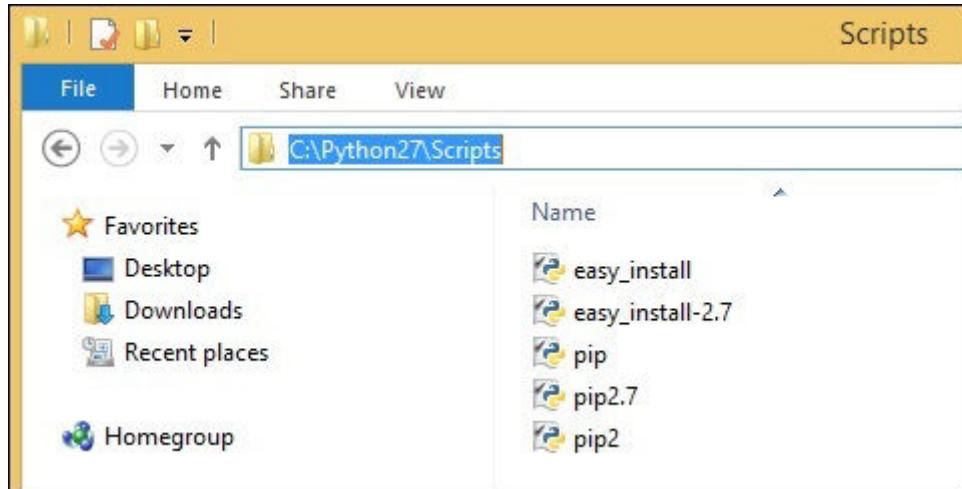
Command

```
pip --version
```

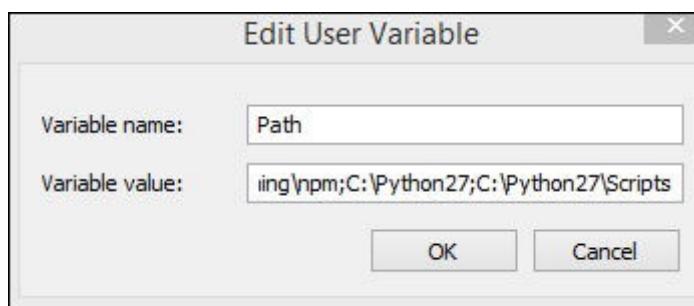


The screenshot shows a Windows Command Prompt window titled "Command Prompt". The window has a yellow header bar. In the black terminal area, the command "C:\>pip --version" is typed, followed by its output: "'pip' is not recognized as an internal or external command, operable program or batch file." Below this, the prompt "C:\>_" is visible. The window has standard window controls (minimize, maximize, close) in the top right corner.

Here we are still not getting the version for pip. We need to add the pip path to Environment variables so that we can use it globally. PIP will be installed in Scripts folder of python as shown below –



Go back to environment variables and add the path of pip to the variables list. Add C:\Python27\SCripts to environment variables as follows –



Now open your command line and check the version of pip installed –

A screenshot of a Command Prompt window titled 'Command Prompt'. The window shows the command 'C:\>pip --version' being run, and the output 'pip 9.0.3 from c:\python27\lib\site-packages (python 2.7)'. The window has a yellow border and a red close button.

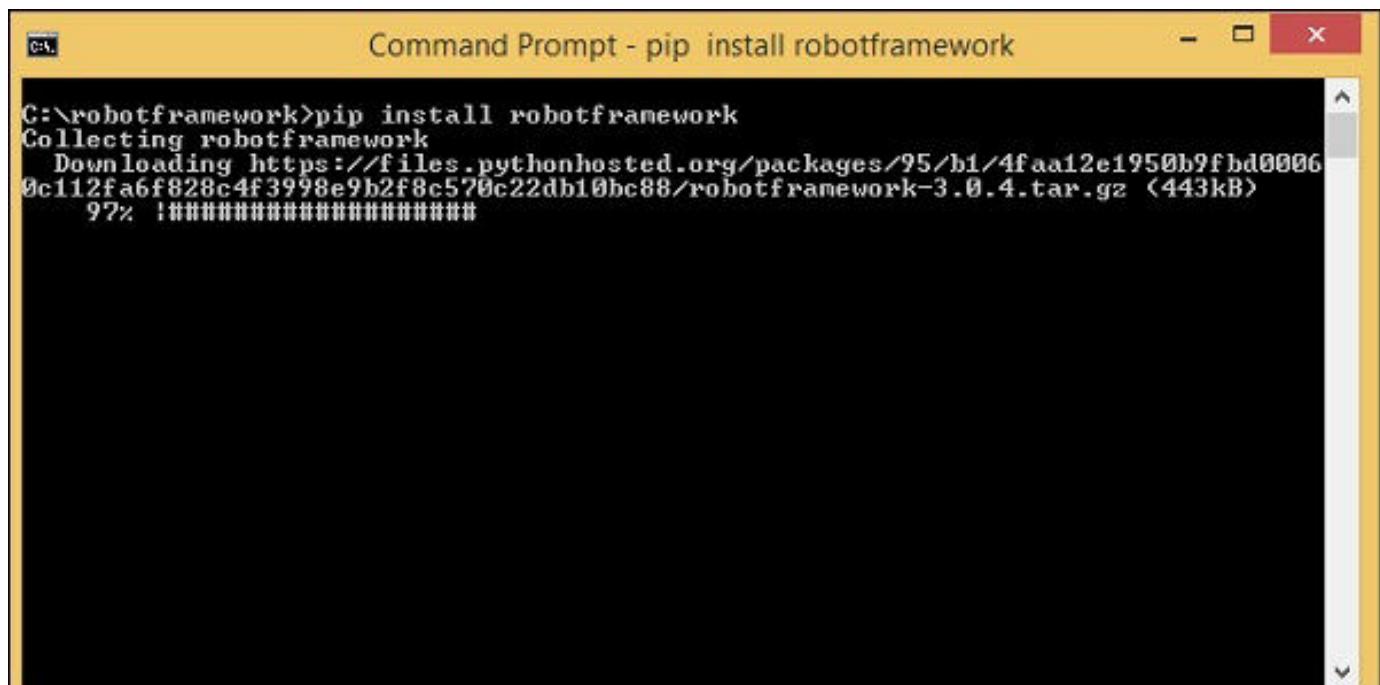
So now, we have python and pip installed.

Install Robot Framework

We will now use pip – python package manager to install the robot framework and the command for it is as follows –

Command

```
pip install robotframework
```



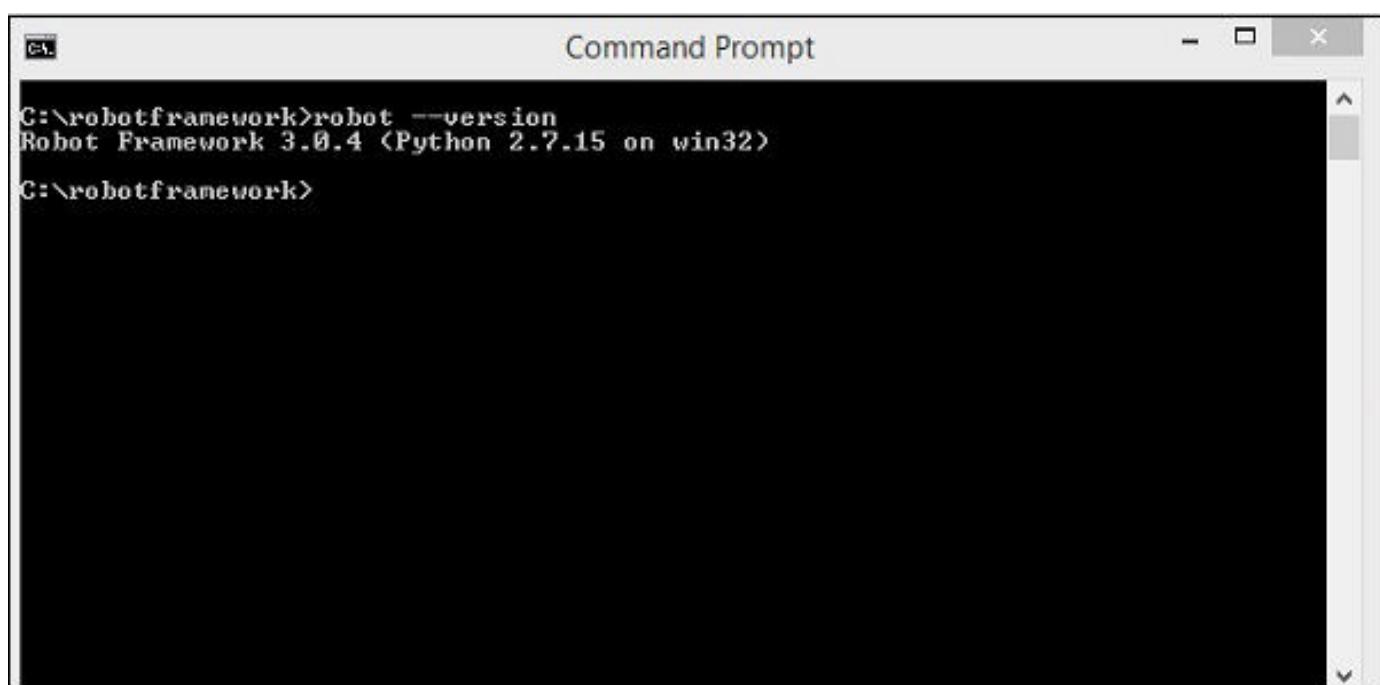
```
C:\>pip install robotframework
Collecting robotframework
  Downloading https://files.pythonhosted.org/packages/95/b1/4faa12e1950b9fb00060c112fa6f828c4f3998e9b2f8c570c22db10bc88/robotframework-3.0.4.tar.gz (443kB)
    97% !#####

```

Once the installation is done, you can check the version of robot framework installed as shown below –

Command

```
robot --version
```



```
C:\>robot --version
Robot Framework 3.0.4 (Python 2.7.15 on win32)
C:\>
```

So, we can see Robot Framework 3.0.4 is installed.

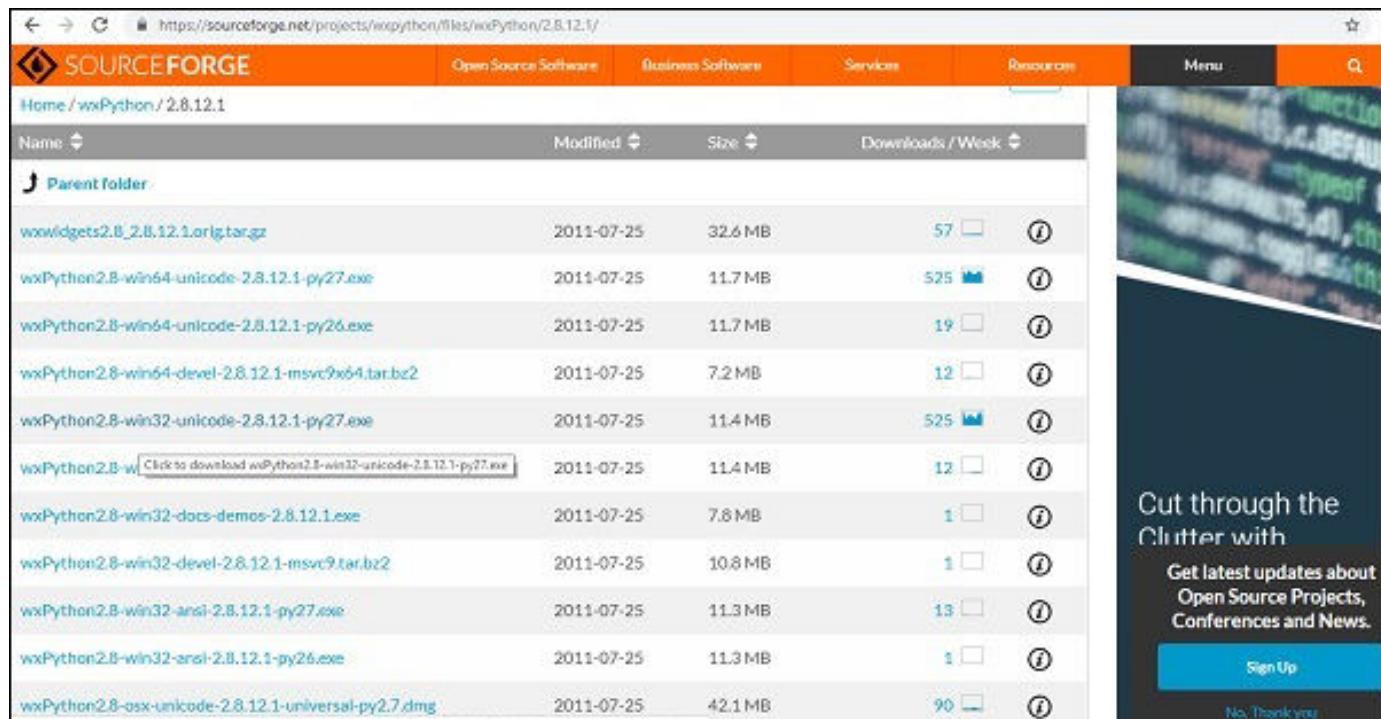
Install wxPython

We need wxPython for Robot Framework Ride, which is an IDE for Robot Framework.

For windows to get the required download for wxPython, go to the following URL –

<https://sourceforge.net/projects/wxpython/files/wxPython/2.8.12.1/>

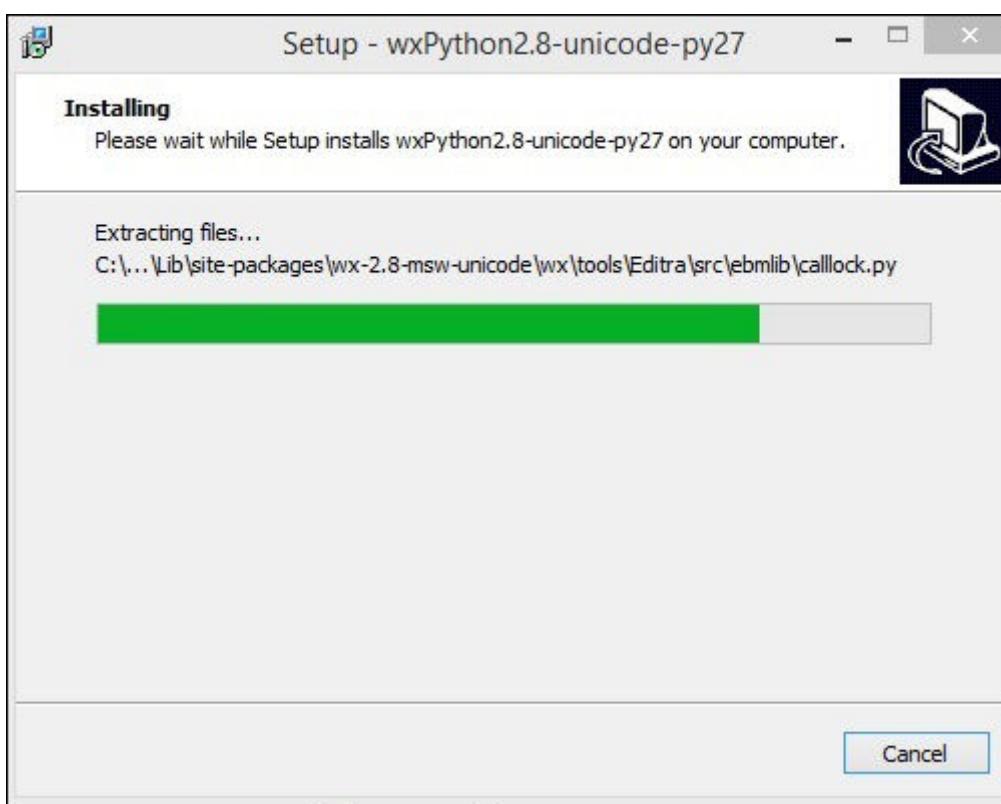
And, download 32 or 64-bit wxpython for windows as per your Windows Operating system.



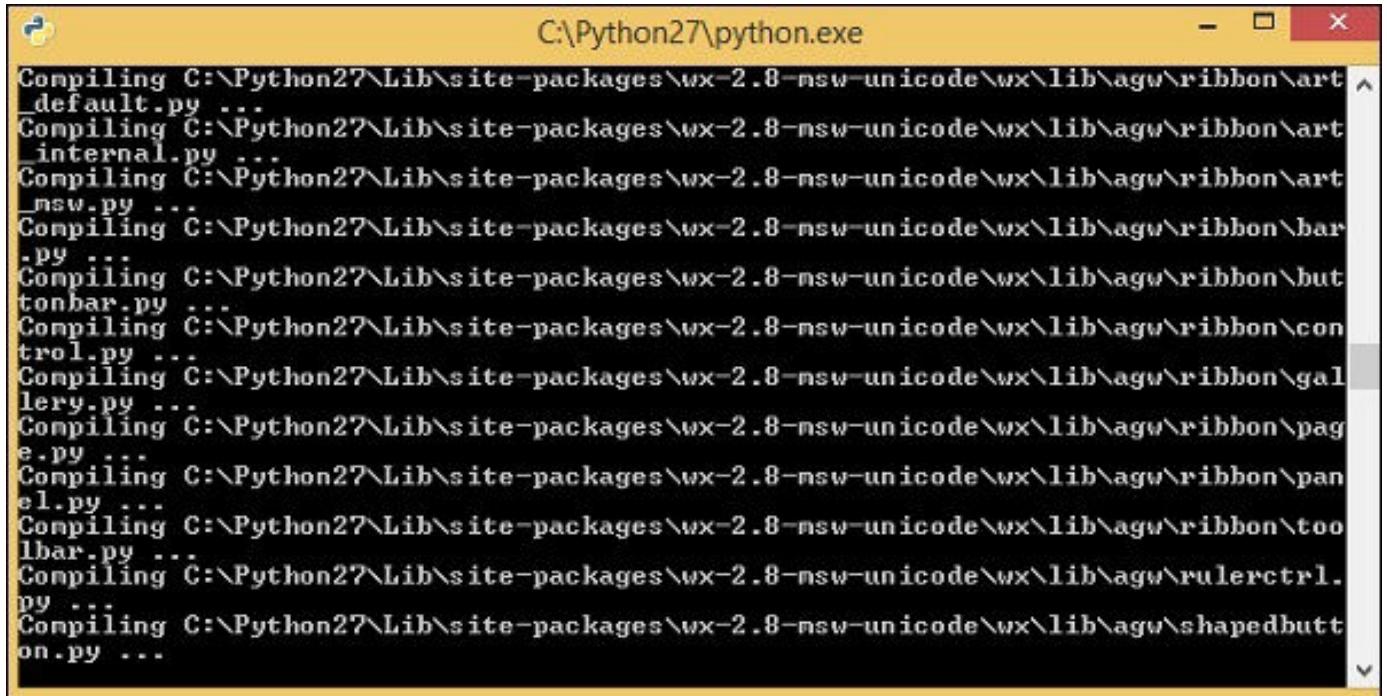
The screenshot shows the SourceForge download page for wxPython 2.8.12.1. The page has a header with the SourceForge logo and navigation links for Open Source Software, Business Software, Services, and Resources. A sidebar on the right features a banner with the text "Cut through the Clutter with" and "Get latest updates about Open Source Projects, Conferences and News.", along with "Sign Up" and "No, Thank you" buttons. The main content area displays a table of download links:

Name	Modified	Size	Downloads / Week	Action
wxWidgets2.8_2.8.12.1.orig.tar.gz	2011-07-25	32.6 MB	57	[Info]
wxPython2.8-win64-unicode-2.8.12.1-py27.exe	2011-07-25	11.7 MB	525	[Info]
wxPython2.8-win64-unicode-2.8.12.1-py26.exe	2011-07-25	11.7 MB	19	[Info]
wxPython2.8-win64-devel-2.8.12.1-msvc9x64.tar.bz2	2011-07-25	7.2 MB	12	[Info]
wxPython2.8-win32-unicode-2.8.12.1-py27.exe	2011-07-25	11.4 MB	525	[Info]
wxPython2.8-win32-unicode-2.8.12.1-py27.exe	2011-07-25	11.4 MB	12	[Info]
wxPython2.8-win32-docs-demos-2.8.12.1.exe	2011-07-25	7.8 MB	1	[Info]
wxPython2.8-win32-devel-2.8.12.1-msvc9.tar.bz2	2011-07-25	10.8 MB	1	[Info]
wxPython2.8-win32-ansi-2.8.12.1-py27.exe	2011-07-25	11.3 MB	13	[Info]
wxPython2.8-win32-ansi-2.8.12.1-py26.exe	2011-07-25	11.3 MB	1	[Info]
wxPython2.8-osx-unicode-2.8.12.1-universal-py2.7.dmg	2011-07-25	42.1 MB	90	[Info]

Download the 32-bit wxPython and install the same.



Once the installation is done, it opens the command line and auto runs some commands as shown below –



```
C:\Python27\python.exe
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\ribbon\art
_default.py ...
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\ribbon\art
_internal.py ...
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\ribbon\art
_msu.py ...
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\ribbon\bar
.py ...
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\ribbon\but
tonbar.py ...
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\ribbon\con
trol.py ...
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\ribbon\gal
lery.py ...
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\ribbon\pag
e.py ...
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\ribbon\pan
el.py ...
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\ribbon\tit
lebar.py ...
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\rulerctrl.
py ...
Compiling C:\Python27\Lib\site-packages\wx-2.8-msw-unicode\wx\lib\agw\shapedbutt
on.py ...
```

wxPython is now installed. This module is required for the RIDE Ide to be used for Robot Framework which is the next step.

On Linux, you should be able to install wxPython with your package manager. For example, on Debian based systems such as Ubuntu running sudo apt-get install pythonwxgtk2.8 ought to be enough.

On OS X

, you should use wxPython binaries found from the wxPython download page. wxPython2.8 only has 32 bit build available, so Python must be run in 32-bit mode also. This can be done globally by running

```
> defaults write com.apple.versioner.python Prefer-32-Bit -bool yes
```

or, just for the RIDE execution –

```
> VERSIONER_PYTHON_PREFER_32_BIT=yes ride.py
```

Install Ride

Ride is Robot Framework IDE. We can use pip to install it as shown below.

Command

```
pip install robotframework-ride
```

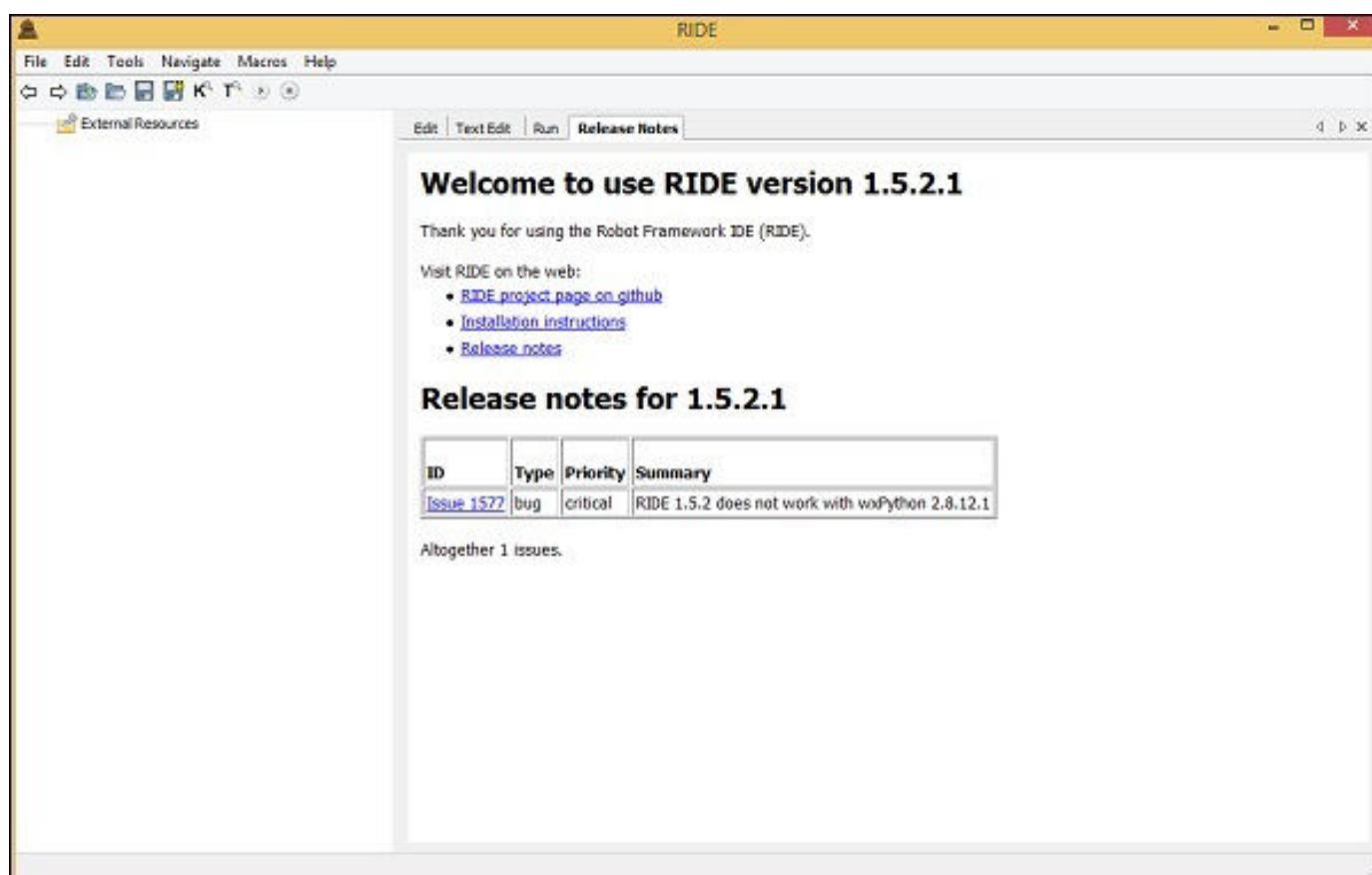
```
C:\>pip install robotframework-ride
Collecting robotframework-ride
  Downloading https://files.pythonhosted.org/packages/3c/14/a5f97f5cf5e981f01e8c0b4c405b0dfc9bc86500cabb044d2c462f73004a/robotframework-ride-1.5.2.1.tar.gz (576 kB)
    100% [=====] 583kB 246kB/s
```

Once the installation is done, open the command prompt and type the following command to open the Ride-IDE.

Command

```
ride.py
```

The above command opens the IDE as follows –



So we are done with the installation of Robot Framework and can get started working with it.

Conclusion

We now know how to install python, pip, robot framework and also get RIDE installed to work with test cases in robot framework.

Robot Framework - Introduction to Ride

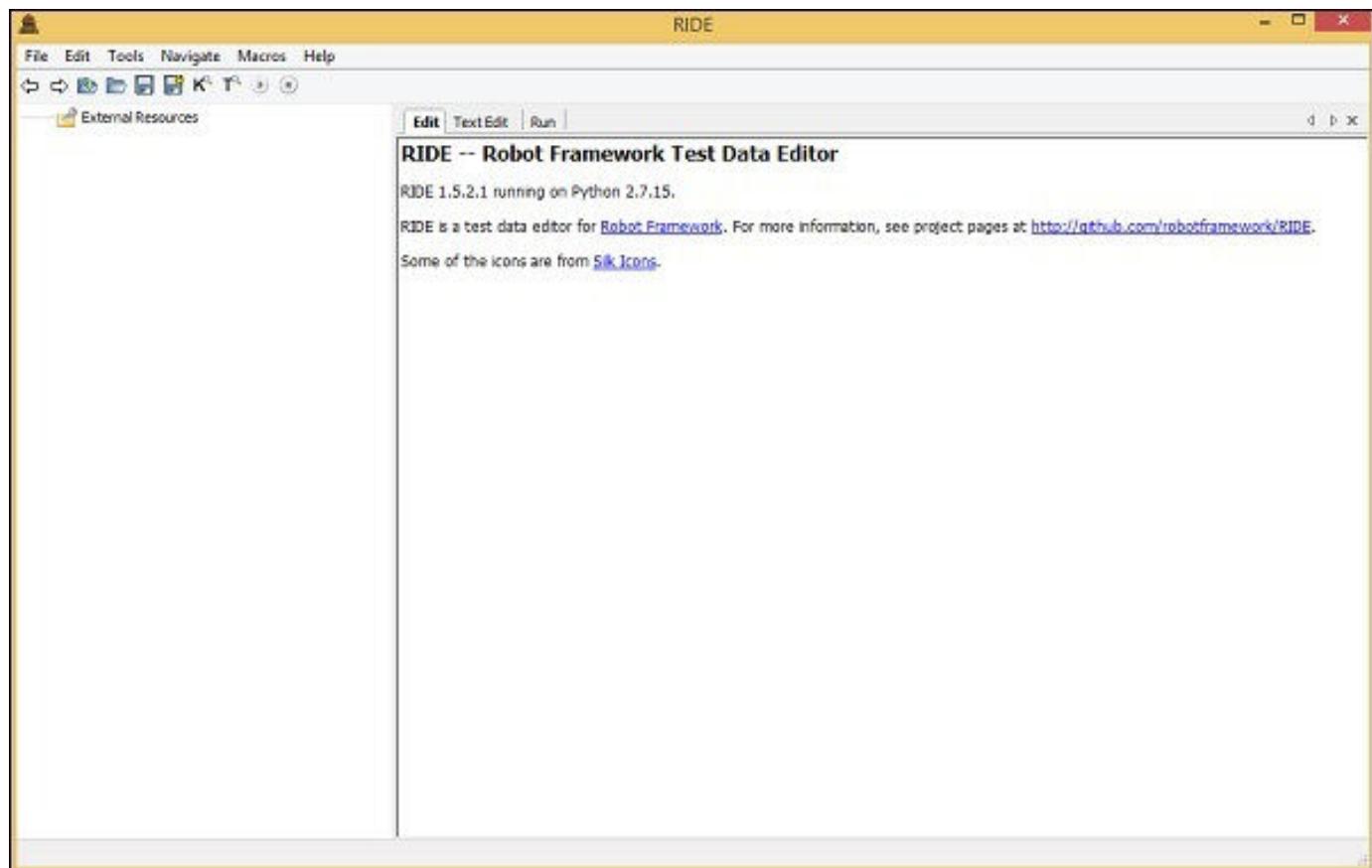
Ride is a testing editor for Robot Framework. Further, we will write test cases in Ride. To start Ride, we need to run the command shown below.

Command

```
ride.py
```



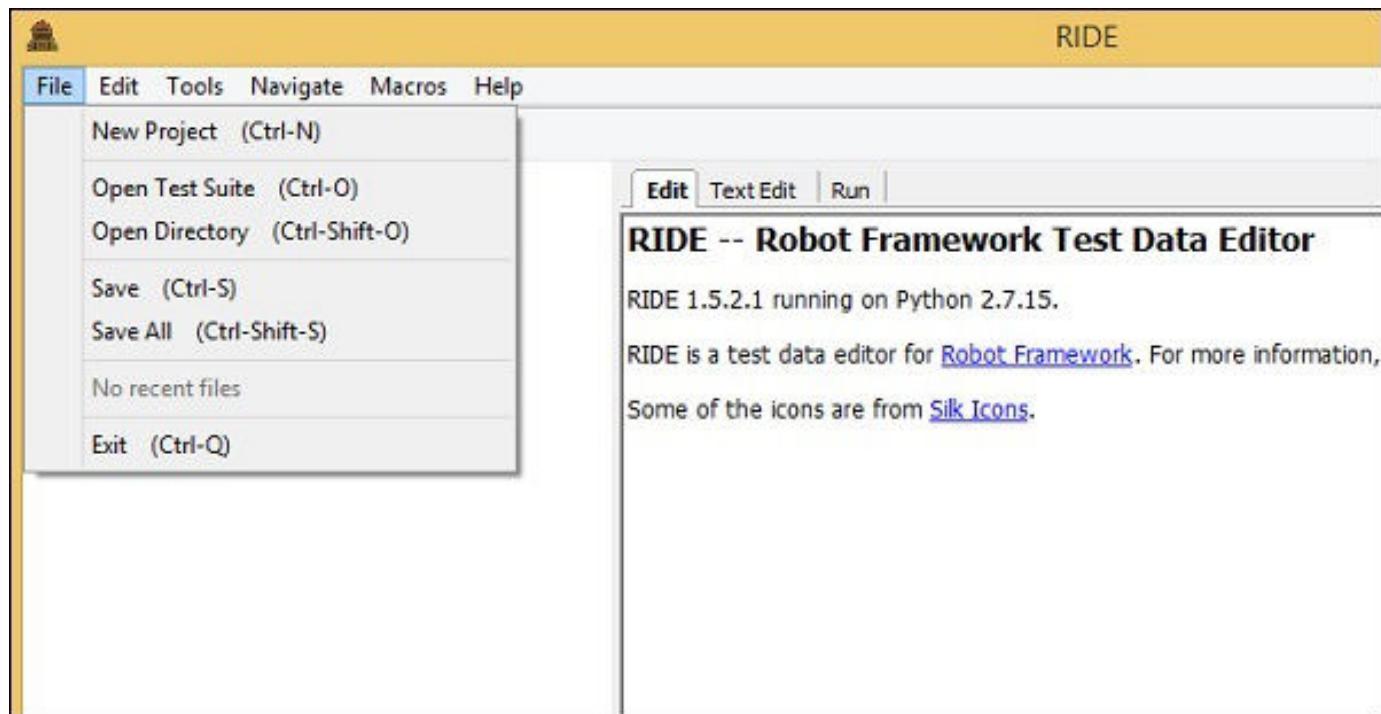
The above command will open the IDE as shown in the following screenshot –



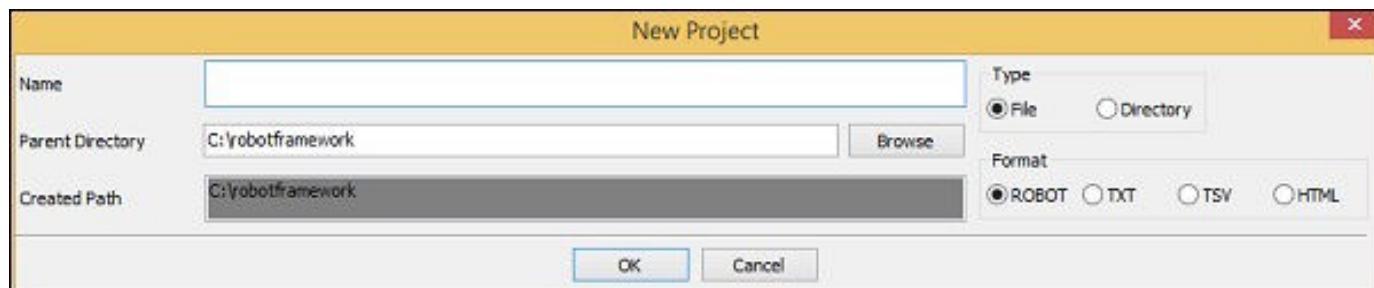
In this chapter, we will walk through the editor to see what options and features are available in the IDE. The options and features will help us in testing our project.

Create New Project

Go to File and click on New Project as shown below –

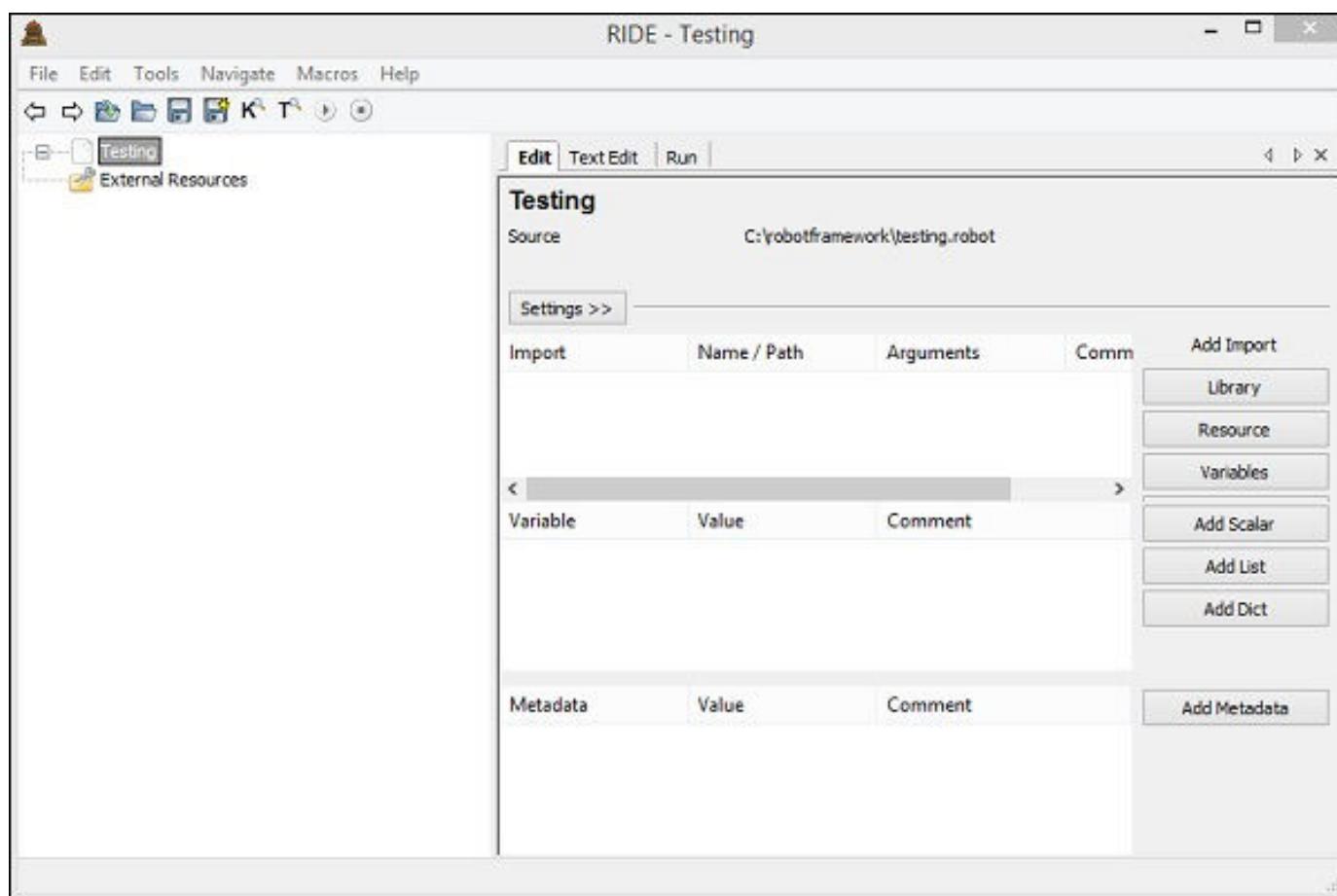


The following screen will appear when you click New Project.



Enter the name of the project. Created Path is the path where the project will get saved. You can change the location if required. The project can be saved as File or directory. You can also save the project in format like ROBOT, TXT, TSV or HTML. In this tutorial, we are going to use the format ROBOT and how to write and execute test-cases.

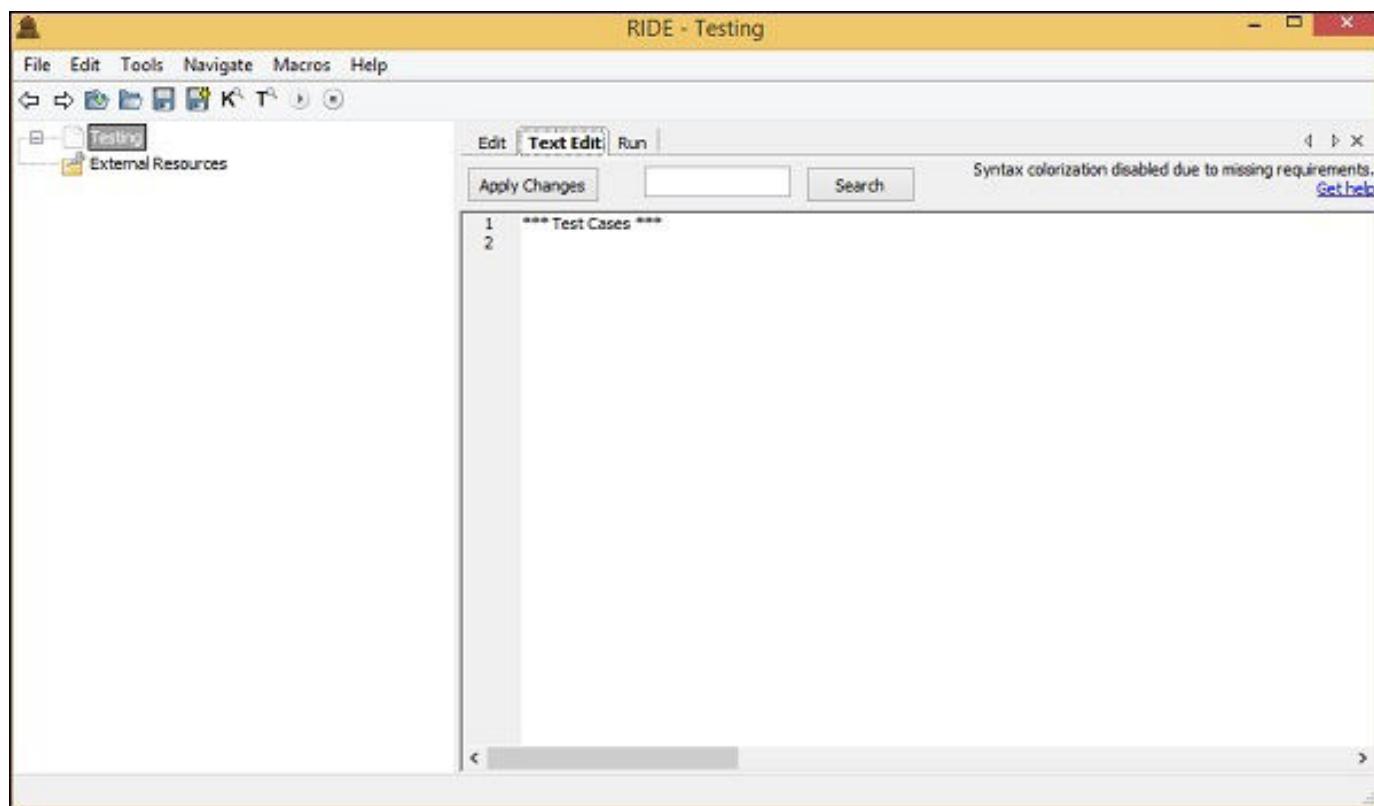
Now, we will add a project as a file the way it is shown below. The project is named Testing and the following screen appears after the project is created.



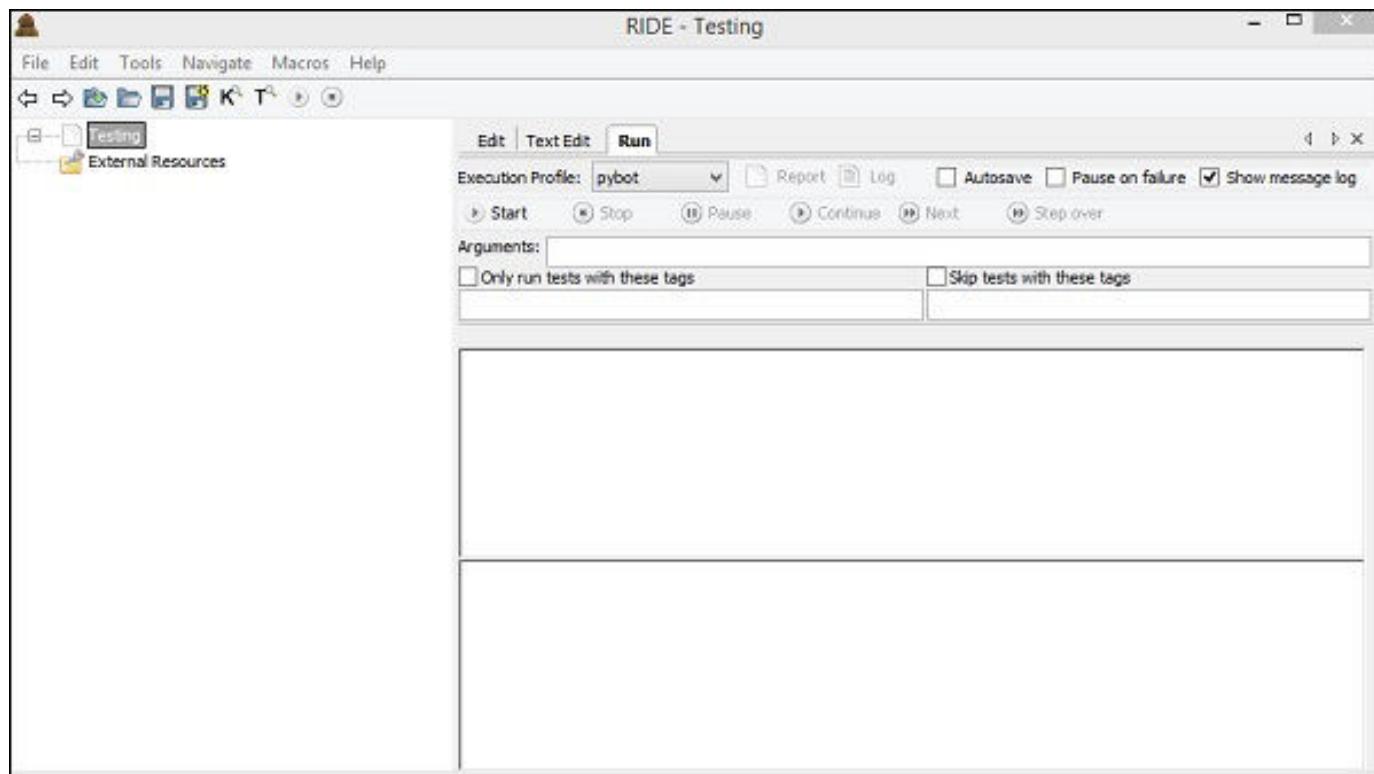
The name of the project is shown on the left side and on the right side we can see three tabs Edit, TextEdit and Run.

Edit has a lot of options on the UI as shown above. In this section, we can add data required to run our test cases. We can import Library, Resource, Variables, Add scalar, Add list, Add dict and Add Metadata.

The details added in the Edit section will be seen in the next tab, Text Edit. You can write the code here in text edit section.



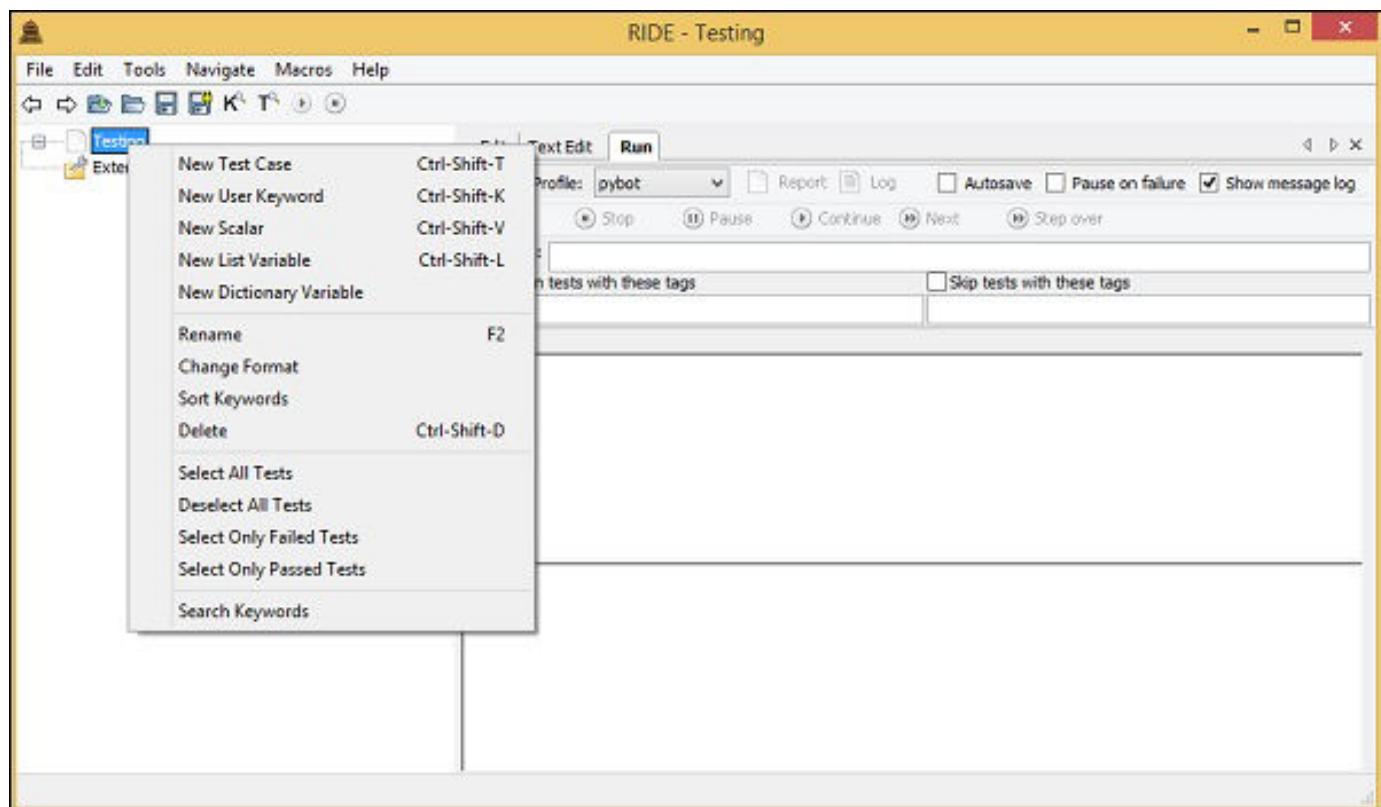
If there is any change added in Textedit, it will be seen in the Edit section. Therefore, both the tabs Edit and TextEdit are dependent on each other and the changes done will be seen on both. Once the test cases are ready, we can use the third tab Run to execute them.



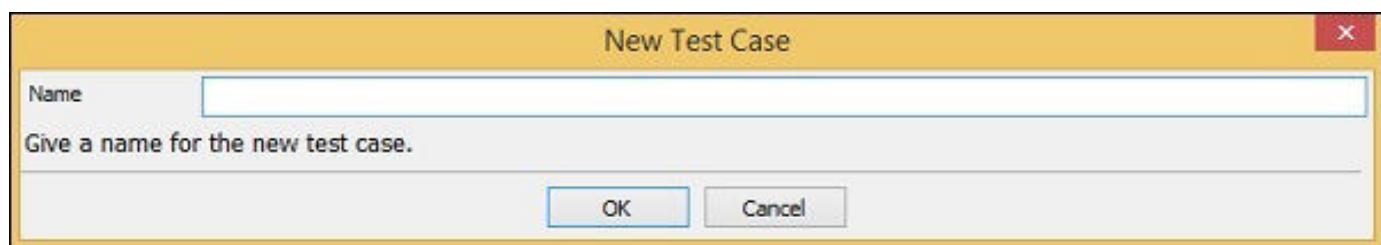
The Run UI is as shown above. It allows to run the test case and comes with options like start, stop, pause continue, next test case, step over, etc. You can also create Report, Log for the test cases you are executing.

To create a test case, we have to do the following –

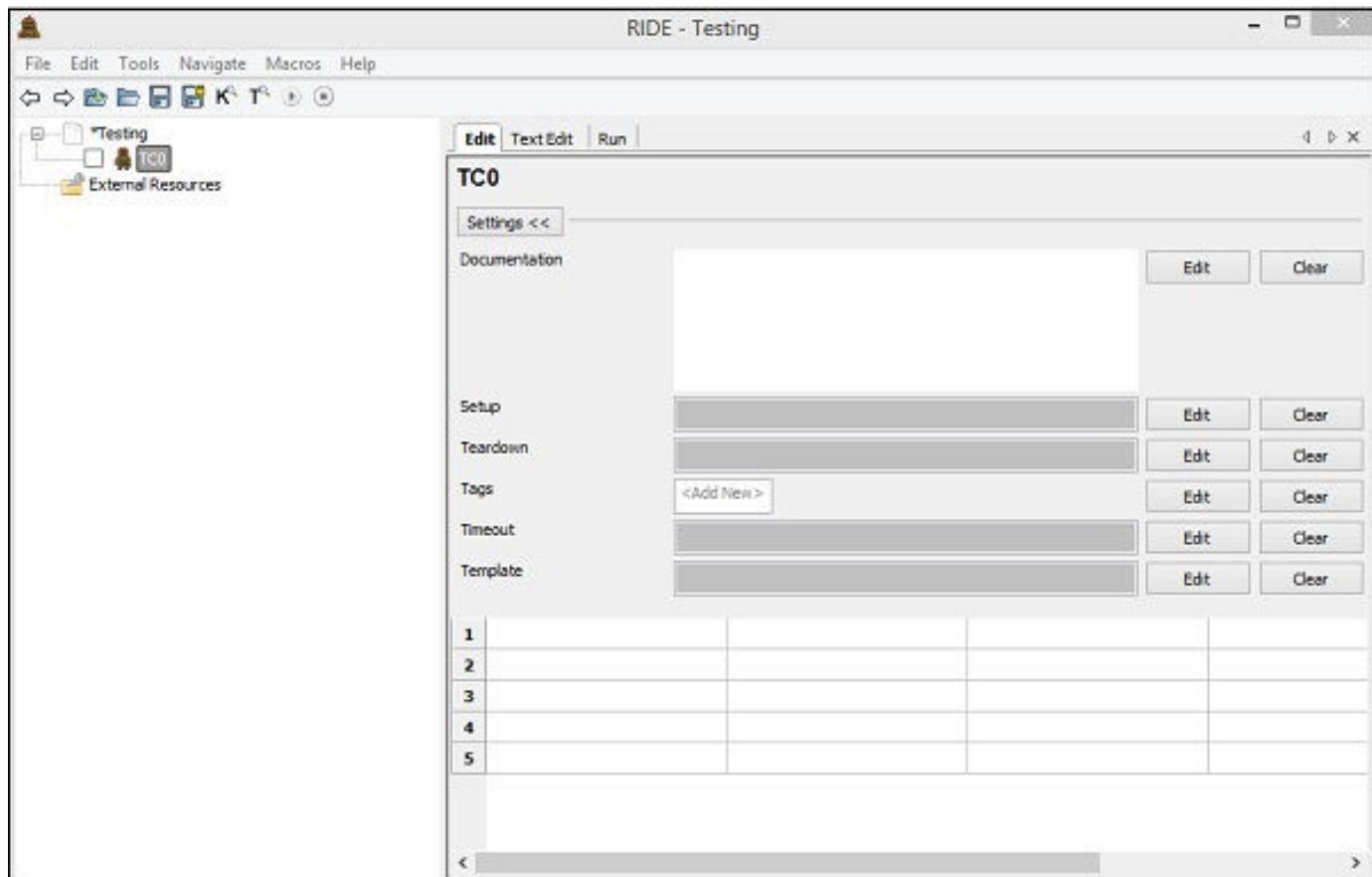
Right-click on the project created and click on new test case as shown below –



Upon clicking New Test Case, a screen appears as shown below –



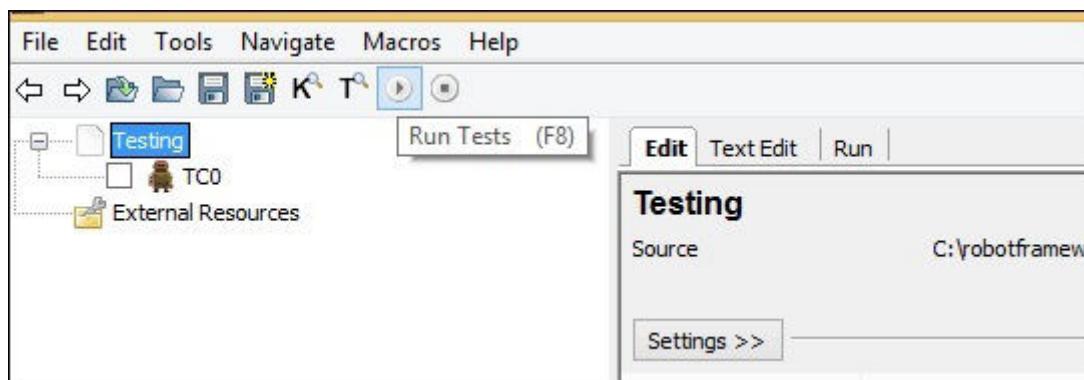
Enter the name of the test case and click OK. We have saved the test case as TC0. The following screen appears once the test case is saved.



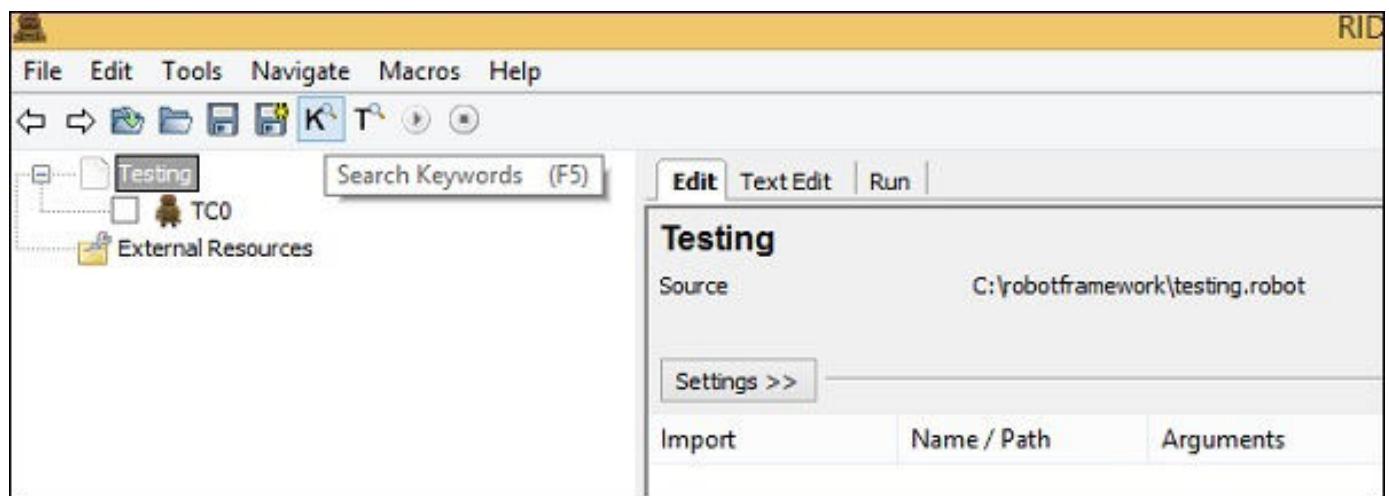
The test case has options like Documentation, setup, teardown, tags, timeout and Template. They have an edit button across it; upon clicking the button a screen appears wherein, you can enter the details for each option. We will discuss the various parameters of these details in our subsequent chapters.

The test cases can be written in tabular format as shown below. Robot framework test cases are keyword based and we can write the test-cases using built-in keywords or keywords imported from the library. We can also create user-defined keywords, variables, etc. in robot framework.

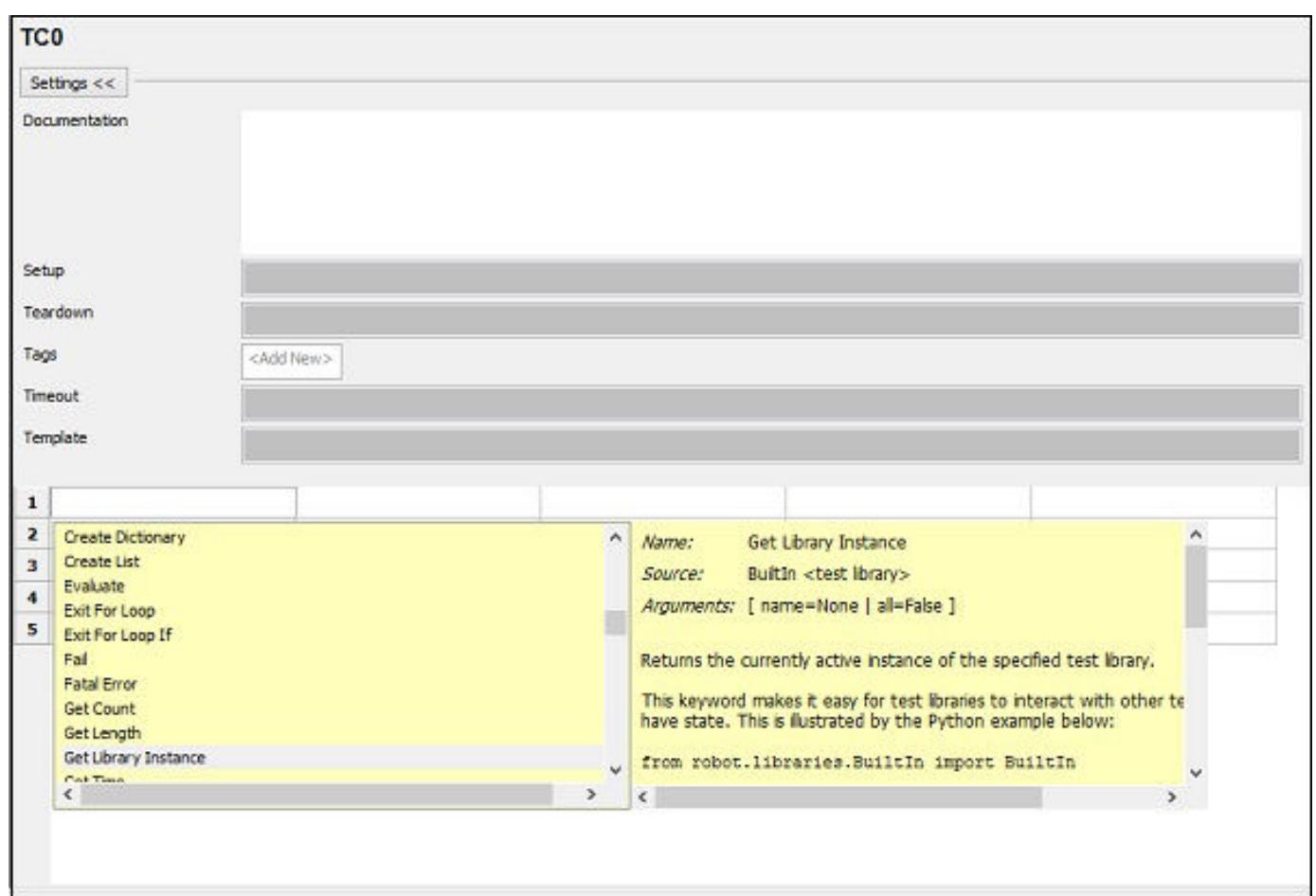
There are shortcuts available in the navigation bar to run/stop test case as shown below –



The search keyword option can be used as shown in the screenshot below –



To get the list of keywords available with robot framework, simple press **ctrl+space** in the tabular format as shown below and it will display all the keywords available –



In case, you cannot remember the keyword, this will help you get the details. We have the details available across each keyword. The details also show how to use the related keyword. In our next chapter, we will learn how to create our first test case in ride.

Conclusion

In this chapter, we have seen the features available with RIDE. We also learnt how to create test cases and execute them.

Robot Framework - First Test Case Using Ride

We will explore RIDE and work on our first test case.

Open Ride from command prompt or you can create a shortcut of ride on your desktop.

From command line

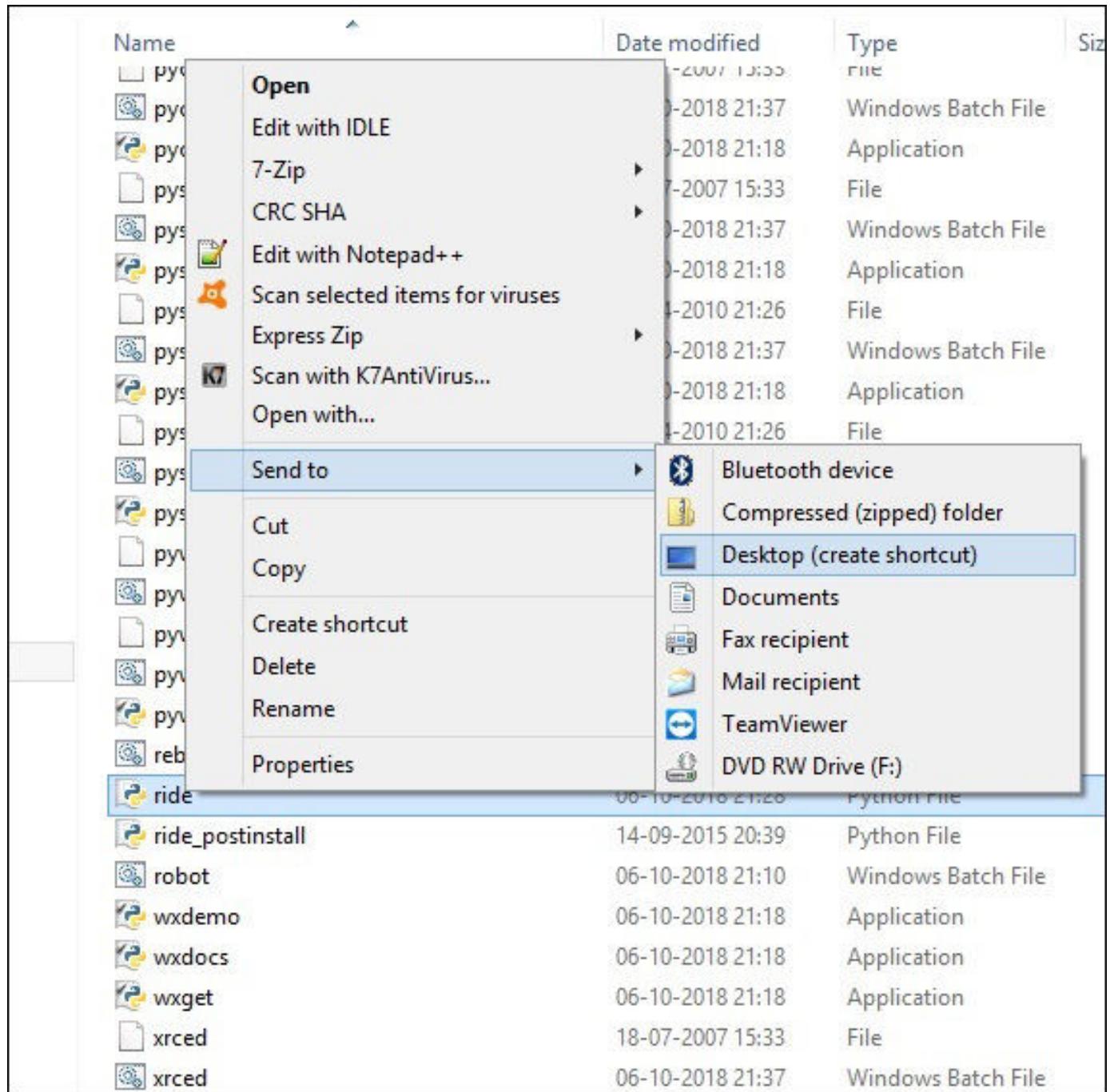
```
ride.py
```

From Desktop

Go to the path where ride is installed; for windows, it is **C:\Python27\Scripts**.

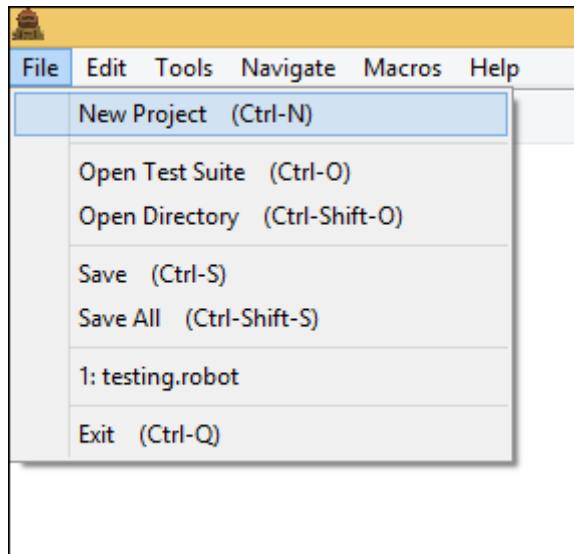
C:\Python27\Scripts			
	Name	Date modified	Type
	pycrust	10-07-2007 15:33	File
	pycrust	06-10-2018 21:37	Windows Batch File
	pycrust	06-10-2018 21:18	Application
	pyshell	18-07-2007 15:33	File
	pyshell	06-10-2018 21:37	Windows Batch File
	pyshell	06-10-2018 21:18	Application
	pyslices	14-04-2010 21:26	File
	pyslices	06-10-2018 21:37	Windows Batch File
	pyslices	06-10-2018 21:18	Application
	pysliceshell	14-04-2010 21:26	File
	pysliceshell	06-10-2018 21:37	Windows Batch File
	pyslicesshell	06-10-2018 21:18	Application
	pywrap	18-07-2007 15:33	File
	pywrap	06-10-2018 21:37	Windows Batch File
	pywxrc	29-02-2008 13:32	File
	pywxrc	06-10-2018 21:37	Windows Batch File
	pywxrc	06-10-2018 21:18	Application
	rebot	06-10-2018 21:10	Windows Batch File
	ride	06-10-2018 21:28	Python File
	ride_postinstall	14-09-2015 20:39	Python File
	robot	06-10-2018 21:10	Windows Batch File
	wxdemo	06-10-2018 21:18	Application

Right-click on ride.py and click **Send To -> Desktop** (create shortcut).

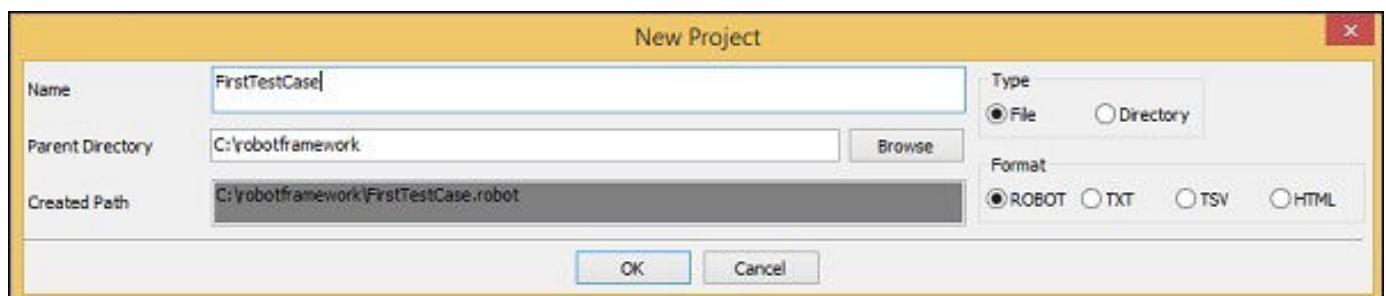


You will now see an icon of ride on your desktop. You can click on it to open the ride editor.

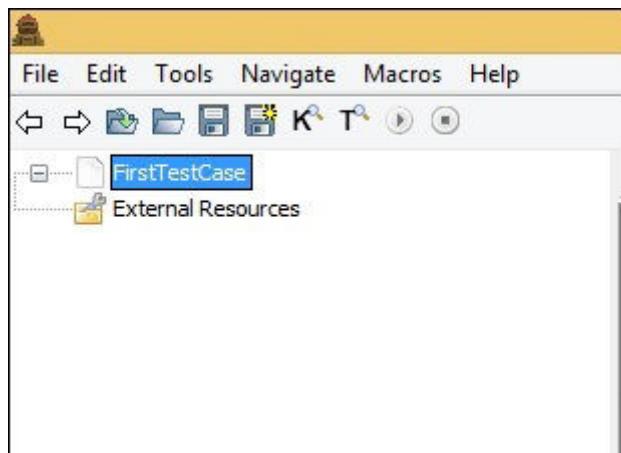
Let us start with our first test case in ride. Open the editor and click on File -> New Project.



Click on *New Project* and enter the name of the project.

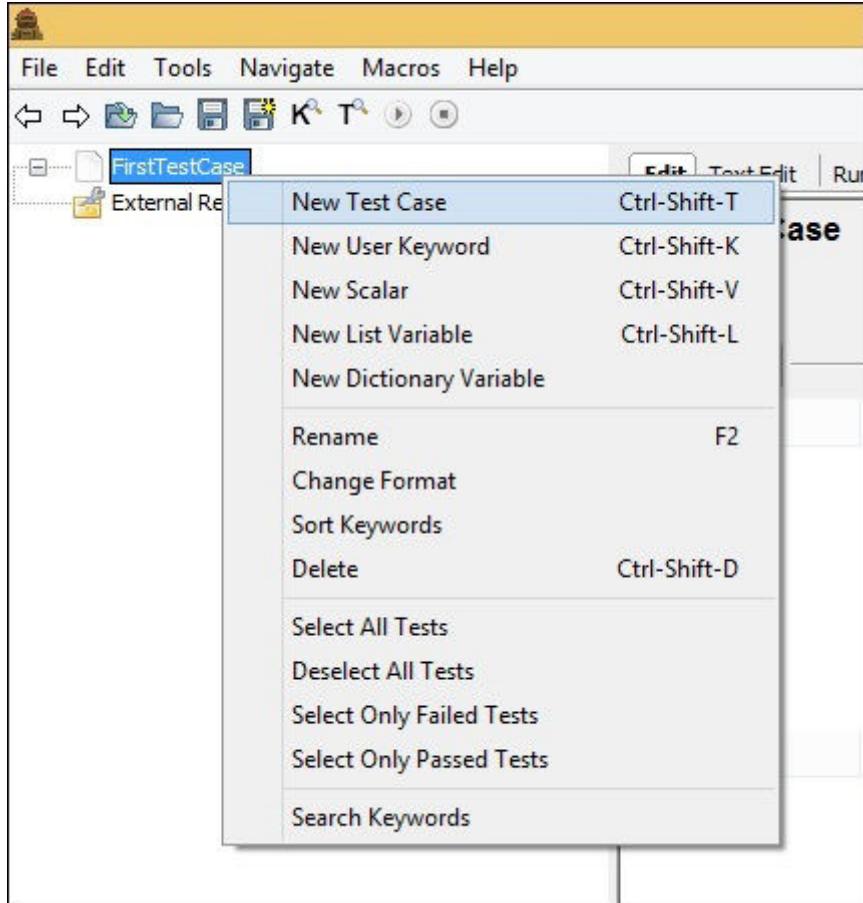


Parent Directory is the path where the project will be saved. You can change the path if required. I have created a folder called robotframework and will save all the files in that folder.

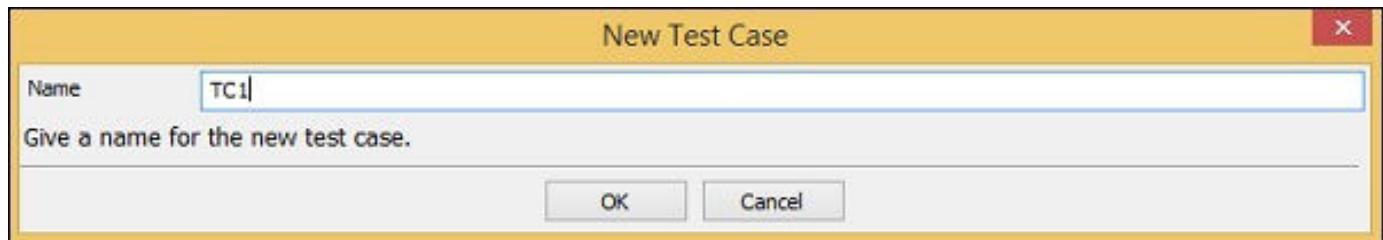


Project *FirstTestCase* is created.

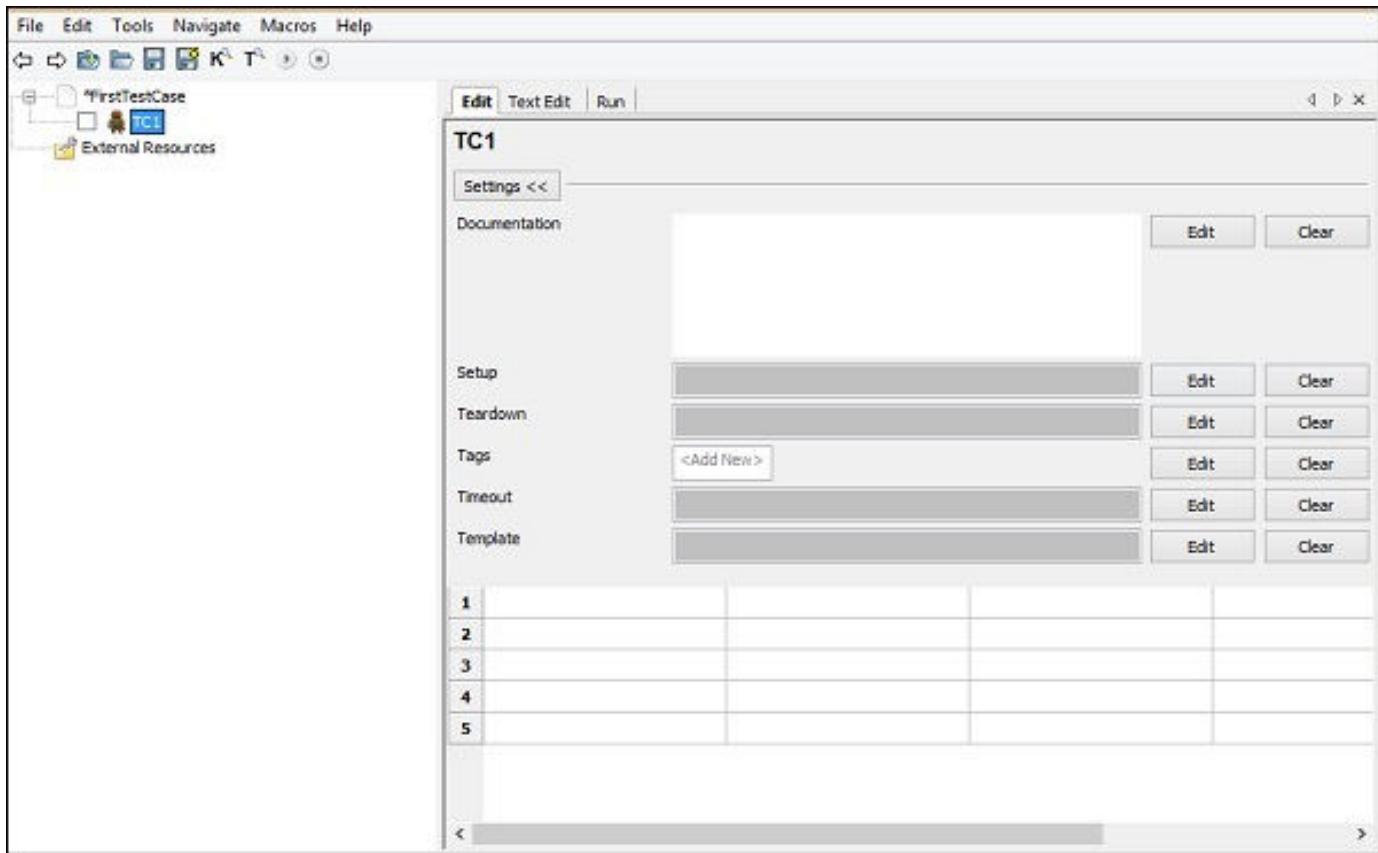
To create test case, right-click on the project.



Click *New Test Case*.



Enter the name of the test case and click *OK*.



There are 3 tabs shown for the test case created – *Edit, Text Edit and Run*.

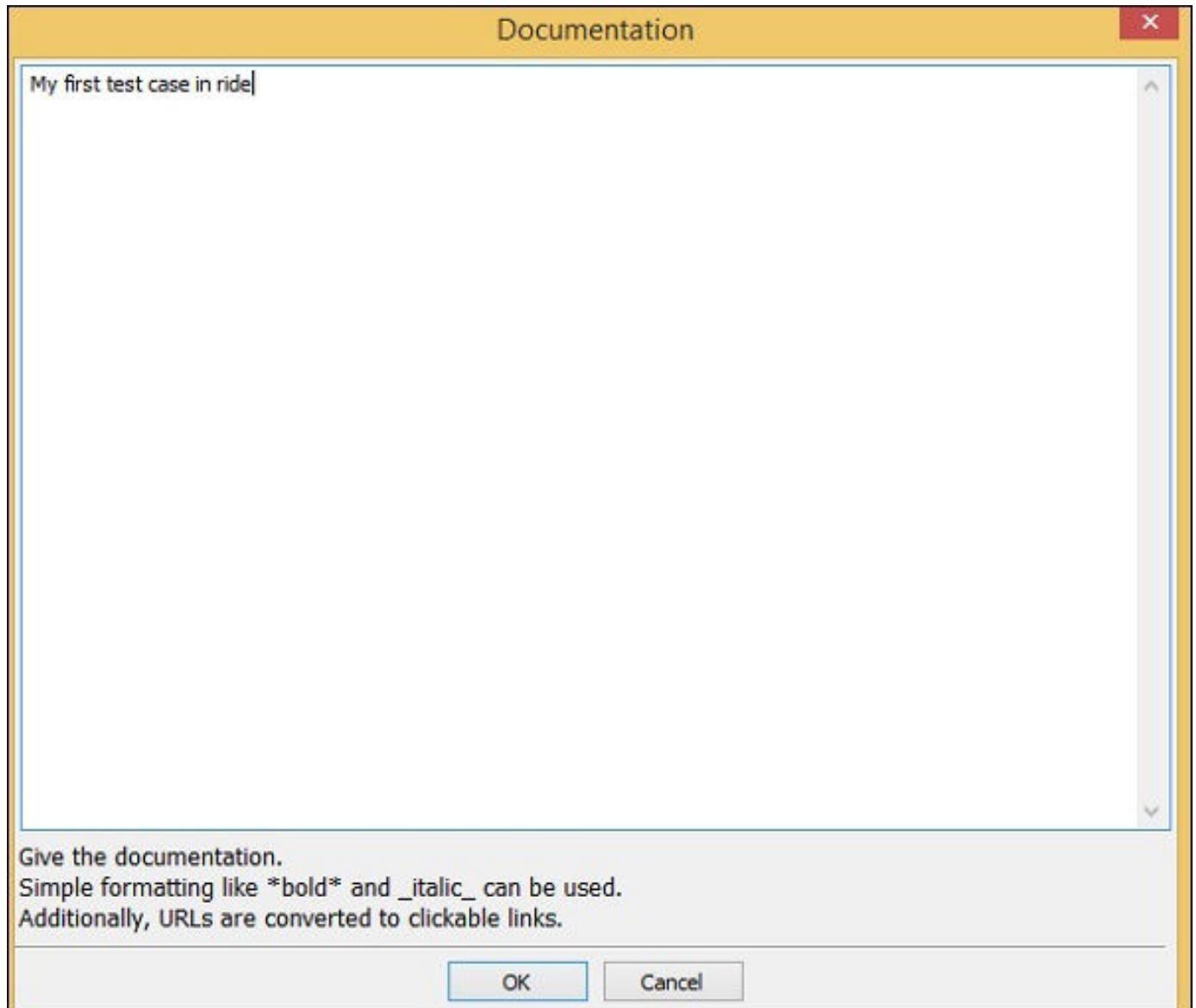
The Edit tab comes with two formats – Settings and Tabular. We will discuss the two formats in our subsequent sections.

The Settings Format

In Settings, we have documentation, setup, teardown, tags, timeout and template.

Documentation

You can add details about your test case so that it becomes easy for future reference.



Click OK to save the documentation.

Setup and Teardown

If there is a setup assigned to a test case, it will be executed before the test case execution and the test setup that will be executed after the test case is done for teardown. We will get into the details of this in our subsequent chapters. We do not need it now for our first test case and can keep it empty.

Tags

This is used for tagging test cases – to include, exclude specific test cases. You can also specify if any of the test cases is critical.

Timeout

This is used to set a timeout on the test case. We will keep it empty for now.

Template

This will have the keywords to be used for the test case. It is mostly used for data driven test case. The high-level user-defined keyword is specified in the template and test cases are used to pass data to the keyword.

In the tabular format, we will write our first test case and execute the same to see the output.

In this test case, we are just going to add some logs and see the output of it. Consider the following screenshot to understand this –

The screenshot shows the Robot Framework Test Case Editor interface. At the top, there are tabs for 'Edit' (which is selected), 'Text Edit', and 'Run'. Below the tabs, the title 'TC1' is displayed. A 'Settings <<' button is located on the left. The 'Documentation' field contains the text 'My first test case in ride'. To the right of the documentation are 'Edit' and 'Clear' buttons. Below the documentation, there are sections for 'Setup', 'Teardown', 'Tags' (with a button '

1	Log	Welcome to our first test case
2	Log	In robot framework
3	Log	End of the test case
4		
5		

We have used the keyword *Log* to log messages as shown above.

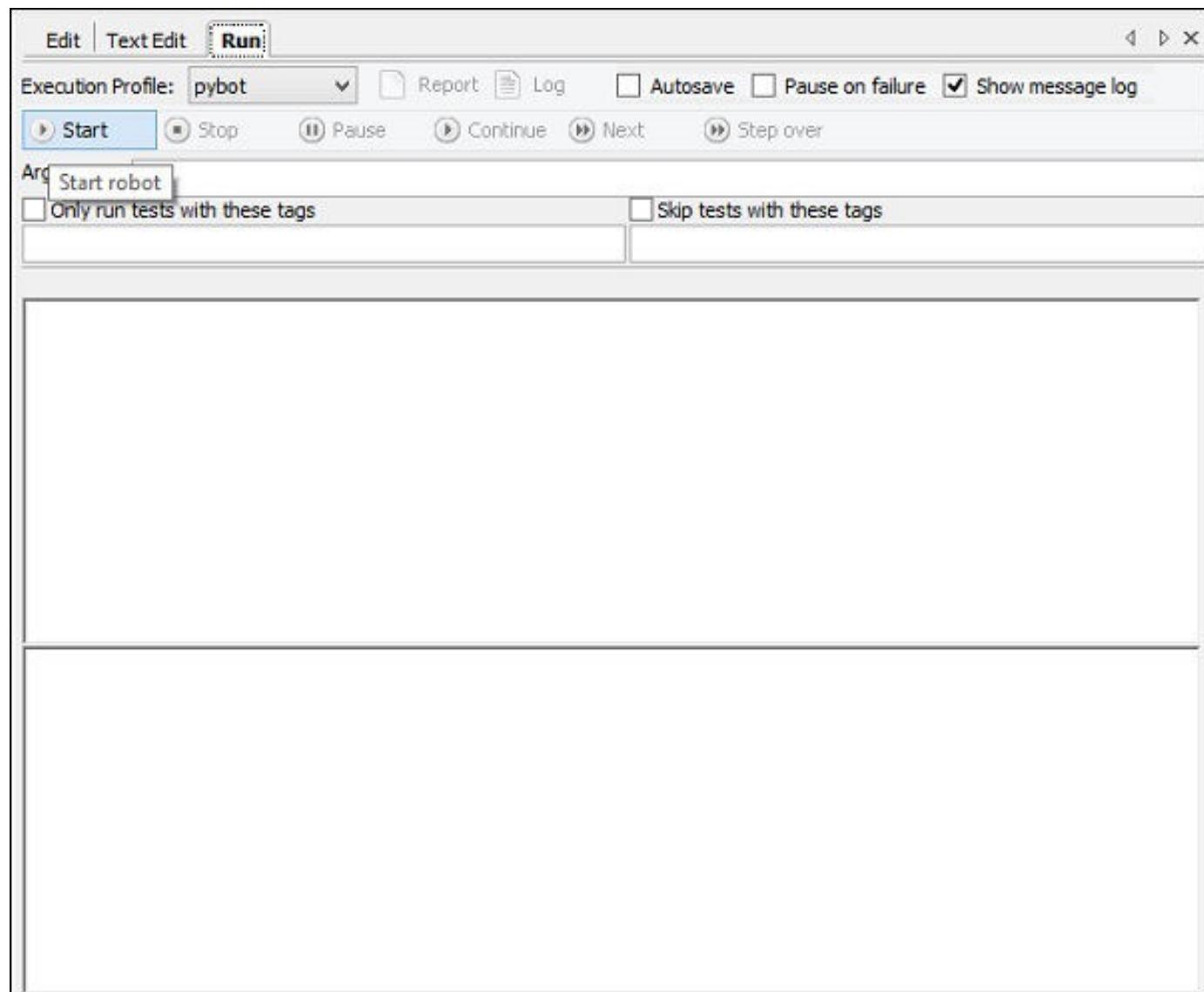
Based on the keywords specified in Edit, we can get the code in Text Edit as shown below –

The screenshot shows the 'Text Edit' tab of the Robot Framework Test Case Editor. At the top, there are buttons for 'Apply Changes', 'Search', and a status message 'Syntax colorization disabled due to missing requirements.' followed by a 'Get help' link. The main area contains the generated Python code for the test case:

```
1 *** Test Cases ***
2 TC1
3 [Documentation] My first test case in ride
4 Log Welcome to our first test case
5 Log In robot framework
6 Log End of the test case
7
```

You may also write the test case in the Text Edit and the same will reflect in the tabular format. Now let us Run the test case and see the output.

To run the test case, we need to click on Start as shown below –



Click on start and here are is the output of the test case –

The screenshot shows the RIDE (Robot IDE) interface. At the top, there's a menu bar with 'Edit', 'Text Edit', and a highlighted 'Run' button. Below the menu is a toolbar with buttons for 'Report', 'Log', 'Autosave', 'Pause on failure', and 'Show message log'. There are also buttons for 'Start', 'Stop', 'Pause', 'Continue', 'Next', and 'Step over'. A section for 'Arguments:' is present with checkboxes for 'Only run tests with these tags' and 'Skip tests with these tags'. The main area displays the test results:
elapsed time: 0:00:05 pass: 1 fail: 0
FirstTestCase

TC1 :: My first test case in ride | PASS |
FirstTestCase | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

Output: c:\users\local\temp\RIDE1yphe8.d\output.xml
Log: c:\users\local\temp\RIDE1yphe8.d\log.html
Report: c:\users\local\temp\RIDE1yphe8.d\report.html
test finished 20181008 10:12:15

Our test case has executed successfully and the details are as shown above. It gives the status as *PASS*.

We can also see the details of the test case execution in Report and Log as highlighted below.



Click on Report and it opens the details in a new tab as follows

← → C file:///C:/Users/AppData/Local/Temp/RIDE1yphe8.d/report.html

LOG Generated
2018/08/10 12:14 GMT+05:30
5 minutes 32 seconds ago

FirstTestCase Test Report

Summary Information

Status:	All tests passed
Start Time:	2018/08/10 12:14:324
End Time:	2018/08/10 12:14:542
Elapsed Time:	00:00:00.218
Log File:	log.html

Test Statistics

Total Statistics		Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests		1	1	0	00:00:00	<div style="width: 100%; background-color: green;"></div>
All Tests		1	1	0	00:00:00	<div style="width: 100%; background-color: green;"></div>

Statistics by Tag		Total	Pass	Fail	Elapsed	Pass / Fail
No Tags						<div style="width: 0%; background-color: lightgray;"></div>

Statistics by Suite		Total	Pass	Fail	Elapsed	Pass / Fail
FirstTestCase		1	1	0	00:00:00	<div style="width: 100%; background-color: green;"></div>

Test Details

Totals Tags Suites Search

Type: Critical Tests
 All Tests

In Report, it gives the details like the start time, end time, path to the log file, status of the test case, etc.

Click on Log at the top right corner in report or from the Run screen.

Here are the details of the log file –

← → C ⓘ file:///C:/Users/AppData/Local/Temp/RIDE1yphe8.d/log.html

REPORT
20181008 10:12:14 GMT+05:30
10 minutes 8 seconds ago

FirstTestCase Test Log

Test Statistics

Total Statistics		Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests		1	1	0	00:00:00	█
All Tests		1	1	0	00:00:00	█

Statistics by Tag		Total	Pass	Fail	Elapsed	Pass / Fail
No Tags						█

Statistics by Suite		Total	Pass	Fail	Elapsed	Pass / Fail
FirstTestCase		1	1	0	00:00:00	█

Test Execution Log

- **SUITE** FirstTestCase 00:00:00.218
 - Full Name: FirstTestCase
 - Source: C:\robotframework\FirstTestCase.robot
 - Start / End / Elapsed: 20181008 10:12:14.324 / 20181008 10:12:14.542 / 00:00:00.218
 - Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
- **TEST** TC1 00:00:00.017
 - Full Name: FirstTestCase.TC1
 - Documentation: My first test case in ride
 - Start / End / Elapsed: 20181008 10:12:14.517 / 20181008 10:12:14.534 / 00:00:00.017
 - Status: PASS (critical)
 - + **KEYWORD** BuiltIn.Log Welcome to our first test case 00:00:00.004
 - + **KEYWORD** BuiltIn.Log In robot framework 00:00:00.003
 - + **KEYWORD** BuiltIn.Log End of the test case 00:00:00.002

The Log file gives the details of the test execution and the details of keywords we gave for the test case.

In the report and the log file, we get green color for the status.

Let us now make some changes that will lead to the failure of the test case fail and see the output.

TC1

Settings <<

Documentation	My first test case in ride	Edit	Clear
Setup		Edit	Clear
Teardown		Edit	Clear
Tags	<Add New>	Edit	Clear
Timeout		Edit	Clear
Template		Edit	Clear

Log

1	Log	Welcome to our first test case	
2	Log	In robot framework	
3	Lo	End of the test case	
4			
5			
6			
7			
...			

In the above test case, the Log keyword is wrong. We will run the test case and see the output –

Execution Profile: pybot

Arguments:

elapsed time: 0:00:01 pass: 0 fail: 1

```

TC1 :: My first test case in ride | FAIL |
No keyword with name 'Lo' found. Did you mean:
  BuiltIn.Log

FirstTestCase | FAIL |
1 critical test, 0 passed, 1 failed
1 test total, 0 passed, 1 failed
-----
Output:  c:\users\appdata\local\temp\RIDE1yphe8.d\output.xml
Log:    c:\users\appdata\local\temp\RIDE1yphe8.d\log.html
Report: c:\users\appdata\local\temp\RIDE1yphe8.d\report.html

test finished 20181008 10:36:40

```

Starting test: FirstTestCase.TC1
20181008 10:36:39.567 : INFO : Welcome to our first test case
20181008 10:36:39.575 : INFO : In robot framework
20181008 10:36:39.584 : FAIL :
No keyword with name 'Lo' found. Did you mean:
 BuiltIn.Log
Ending test: FirstTestCase.TC1

We see that the test case has failed. I have highlighted the error that it tells about the test case.

Now will see the report and log output. From Report –

FirstTestCase Test Report

LOG Generated 2018/08/08 10:36:39 GMT+05:30
2 minutes 30 seconds ago

Summary Information

Status:	1 critical test failed
Start Time:	2018/08/08 10:36:39.445
End Time:	2018/08/08 10:36:39.592
Elapsed Time:	00:00:00.147
Log File:	log.html

Test Statistics

	Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests		1	0	1	00:00:00	<div style="width: 0%; background-color: red;"></div>
All Tests		1	0	1	00:00:00	<div style="width: 0%; background-color: red;"></div>

	Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags						<div style="width: 0%; background-color: lightgray;"></div>

	Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
FirstTestCase		1	0	1	00:00:00	<div style="width: 0%; background-color: red;"></div>

Test Details

Totals Tags Suites Search

Type: Critical Tests
 All Tests

From Log

FirstTestCase Test Log							REPORT Generated 2018/08/10 10:36:39 GMT+05:30 3 minutes 16 seconds ago
Test Statistics							
Total Statistics							
Critical Tests		1	0	1	00:00:00	 	
All Tests		1	0	1	00:00:00	 	
Statistics by Tag							
No Tags						 	
Statistics by Suite							
FirstTestCase		1	0	1	00:00:00	 	
Test Execution Log							
- SUITE FirstTestCase							00:00:00.147
Full Name:	FirstTestCase						
Source:	C:/robotframework/FirstTestCase.robot						
Start / End / Elapsed:	2018/08/10 10:36:39.445 / 2018/08/10 10:36:39.592 / 00:00:00.147						
Status:	1 critical test, 0 passed, 1 failed						
	1 test total, 0 passed, 1 failed						
- TEST TC1							00:00:00.027
Full Name:	FirstTestCase.TC1						
Documentation:	My first test case in ride						
Start / End / Elapsed:	2018/08/10 10:36:39.561 / 2018/08/10 10:36:39.588 / 00:00:00.027						
Status:	FAIL (critical)						
Message:	No keyword with name 'Lo' found. Did you mean: BuiltIn.Log						
+ KEYWORD	BuiltIn.Log Welcome to our first test case						00:00:00.005
+ KEYWORD	BuiltIn.Log In robot framework						00:00:00.003
- KEYWORD	Lo End of the test case						00:00:00.003
Start / End / Elapsed:	2018/08/10 10:36:39.583 / 2018/08/10 10:36:39.586 / 00:00:00.003						
10:36:39.584	FAIL No keyword with name 'Lo' found. Did you mean: BuiltIn.Log						

When the test case fails, the color is changed to Red as shown above.

Conclusion

In this chapter, we covered a simple test case and the results seen during execution are shown. The reports and logs show the details of test case execution.

Writing and Executing Test Cases

In this chapter, we will learn how to write and execute test cases. We would cover the following areas in this chapter –

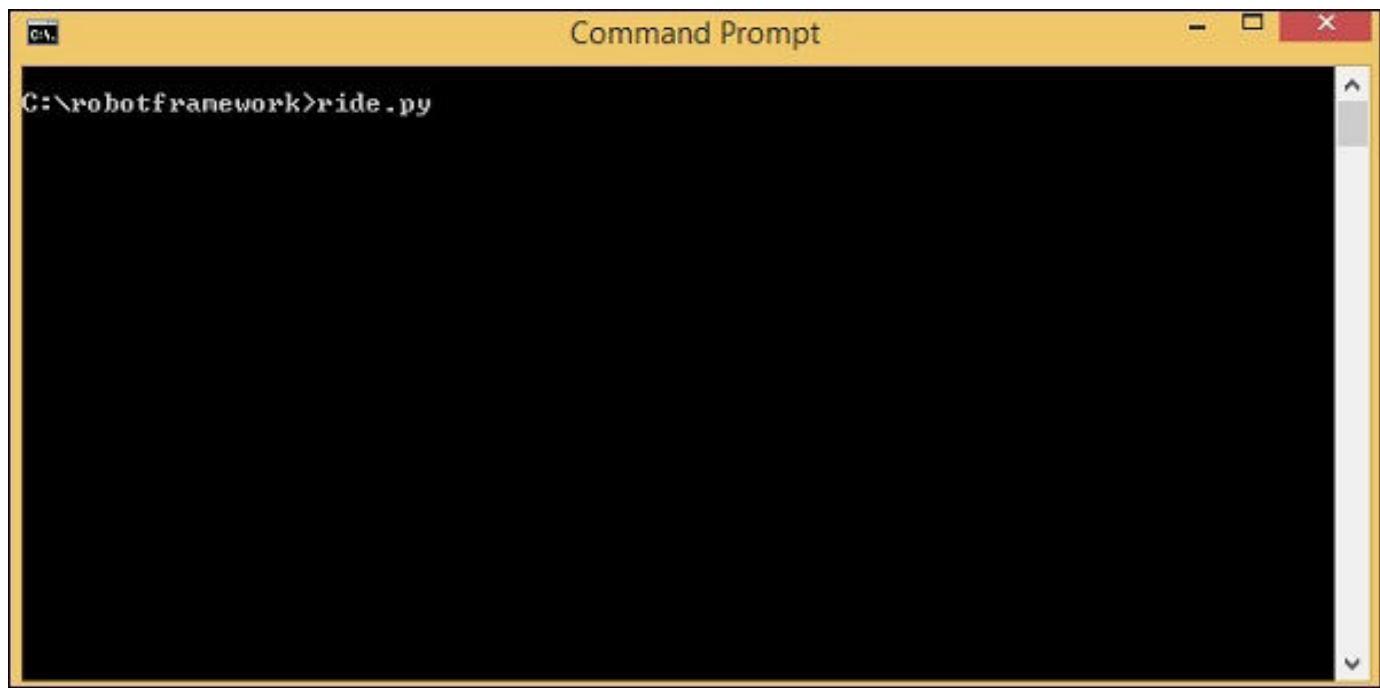
- Project Setup
- Importing Libraries
- Write test case in tabular format
- Using Tags for Executing Test Case
- Use Resource Files for Test Case

Project Setup

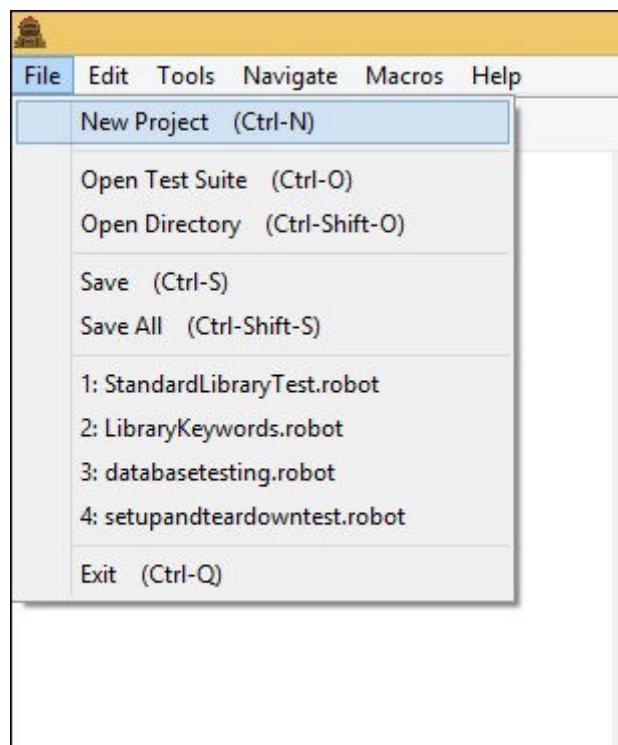
Run the command ride.py to start RIDE IDE.

Command

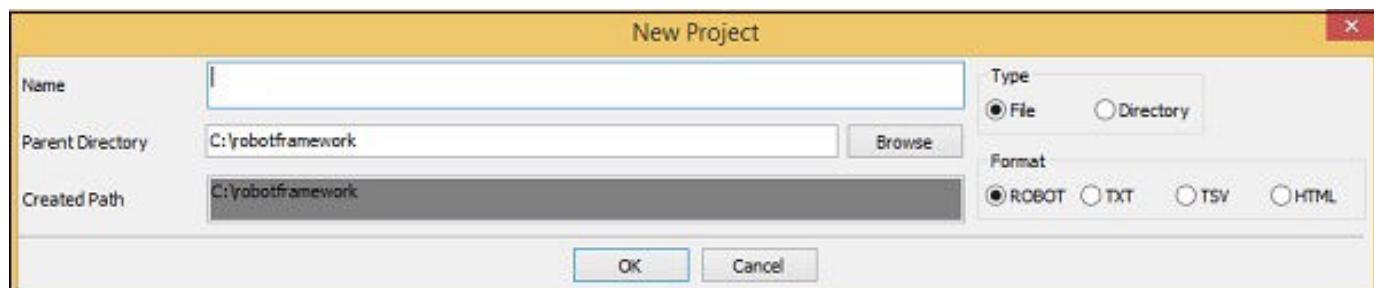
```
ride.py
```



Click on **File -> New Project** as shown below –

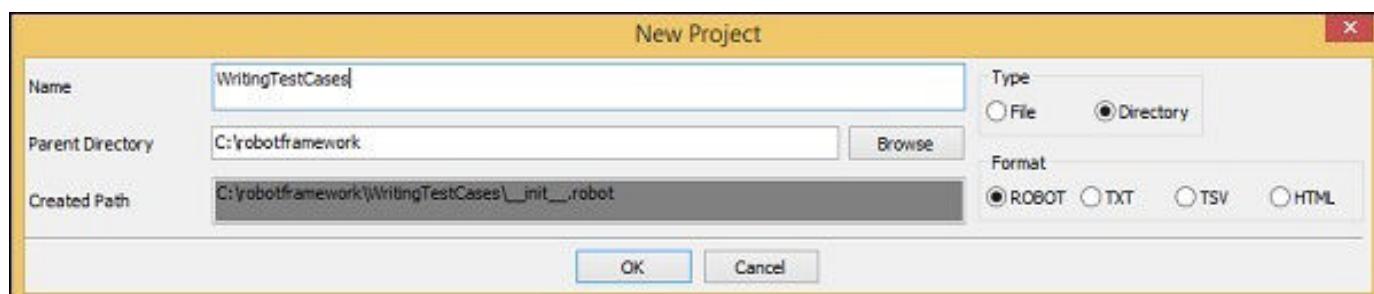


Upon clicking New Project, the screen will appear as shown below –

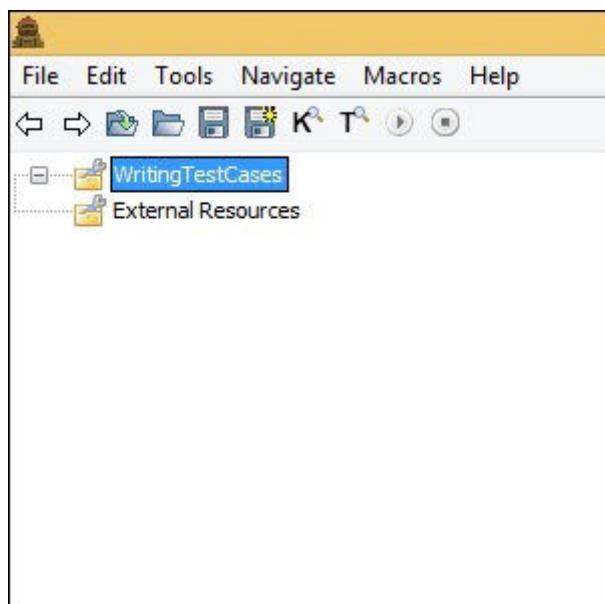


New Project shows the type as file or directory. By default, File is selected. We will click on Directory to create test suite, which can have many test suites in that directory. Each suite will have test-cases.

We will use the ROBOT format for now.

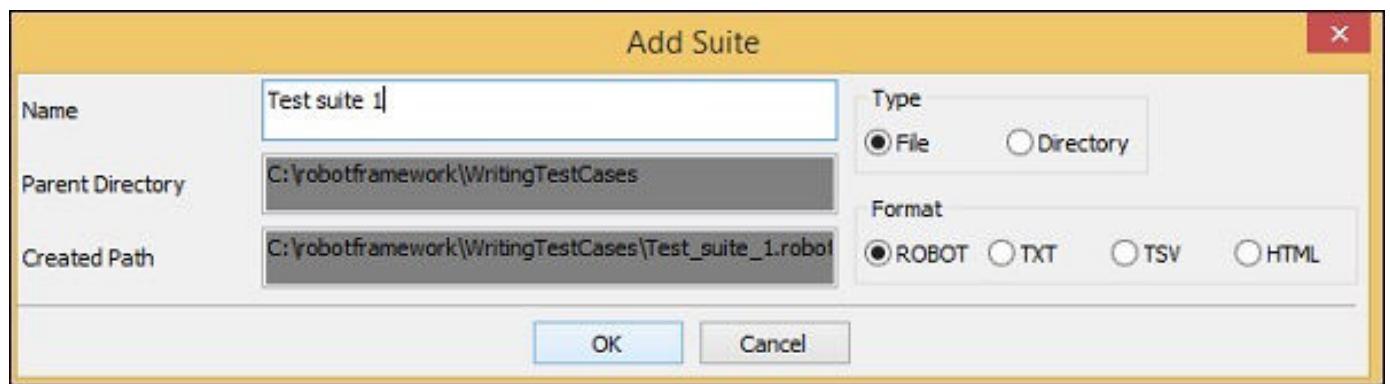
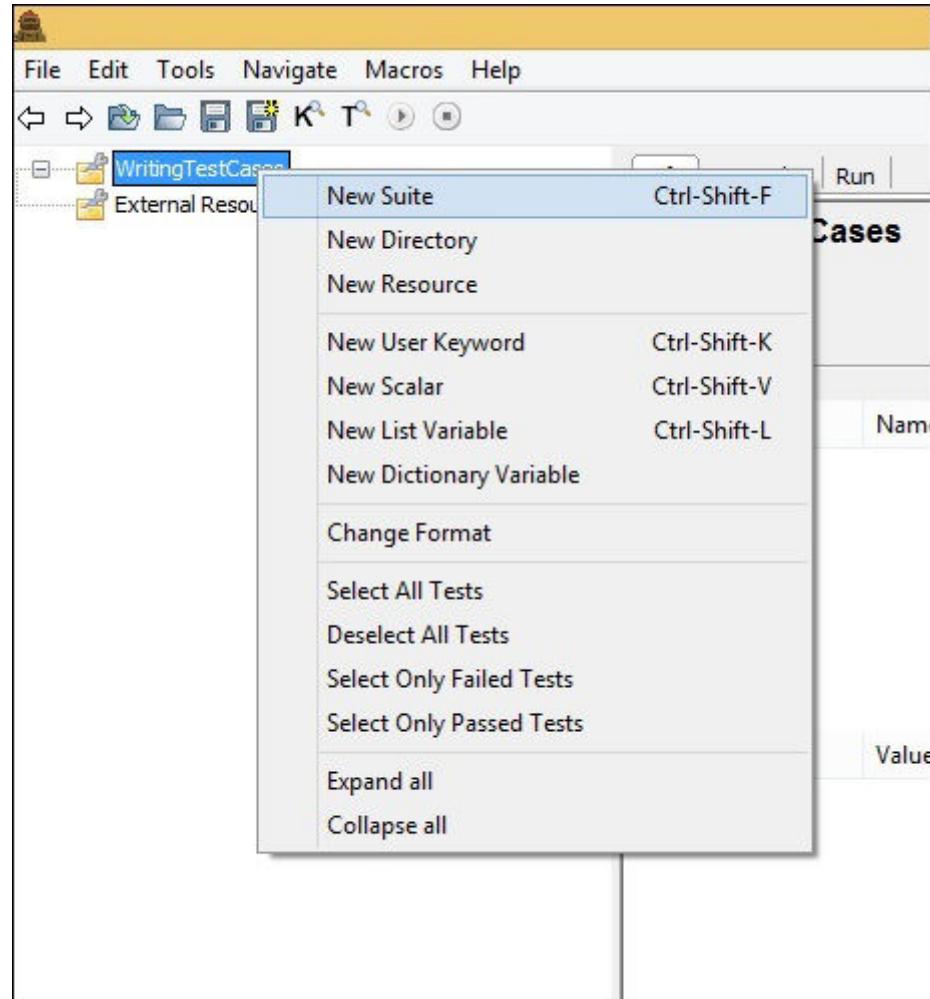


The Parent-Directory is the path where the *WritingTestCases* directory will be created. Click OK to save the test suite directory.



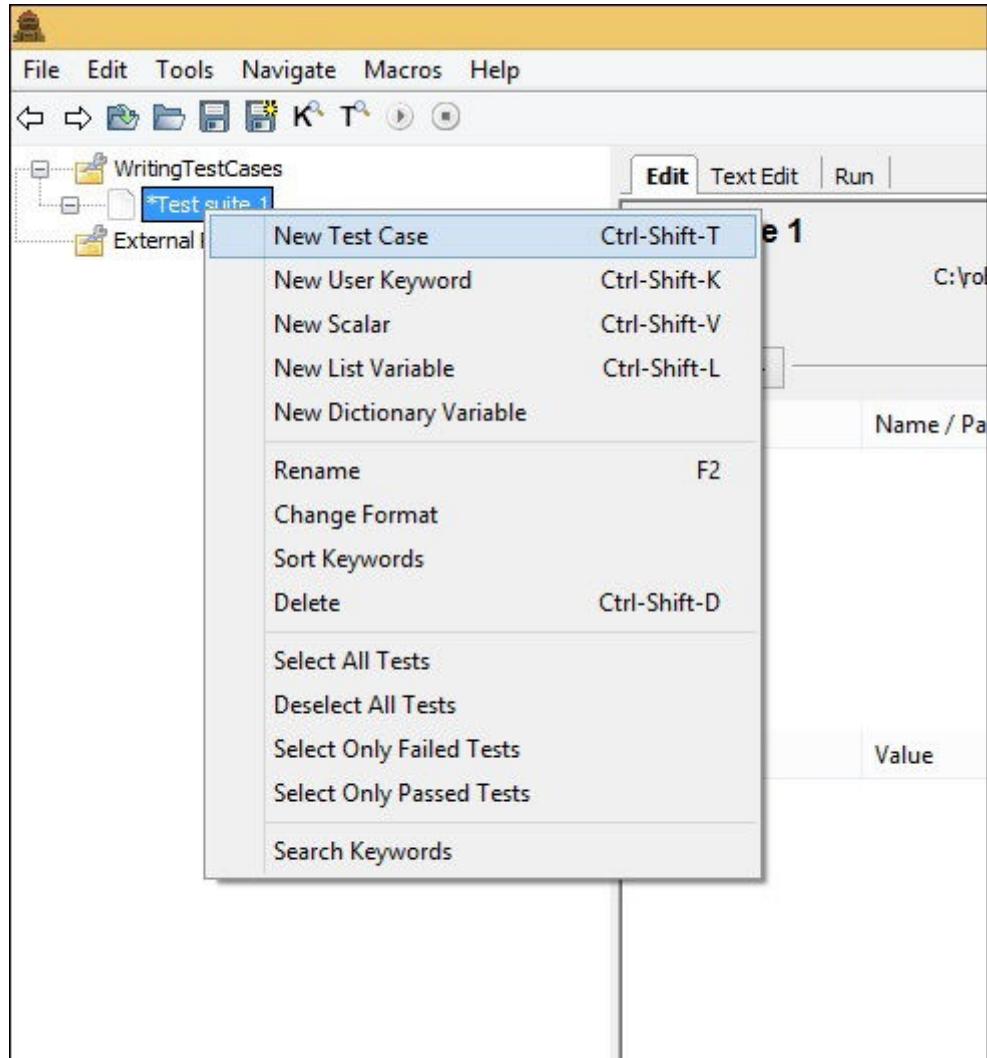
Right-click on the directory created and click on *New Suite*. You can also create sub directories with test suites in that.

For now, we will start with Test Suite creation as shown below –

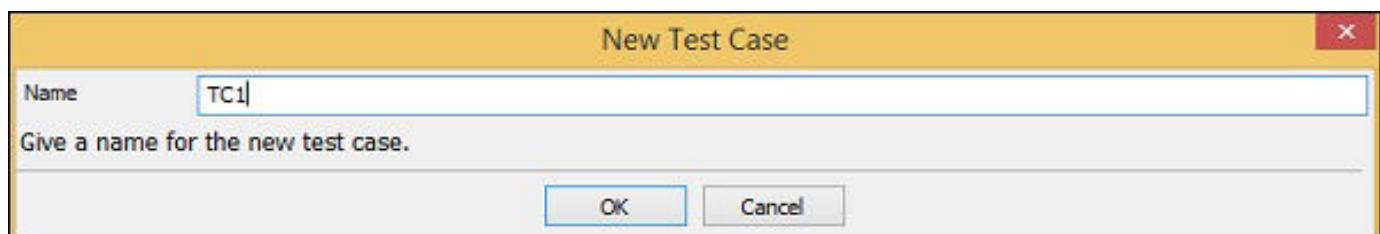


Click OK to save the Test suite.

Now you can add test case to the suite. Right-click on the Test suite created as shown below –



Click *New Test Case*. It will display the screen to add name of the test case as shown below –



Click OK to save the test case. We have the project setup ready.

Importing Libraries

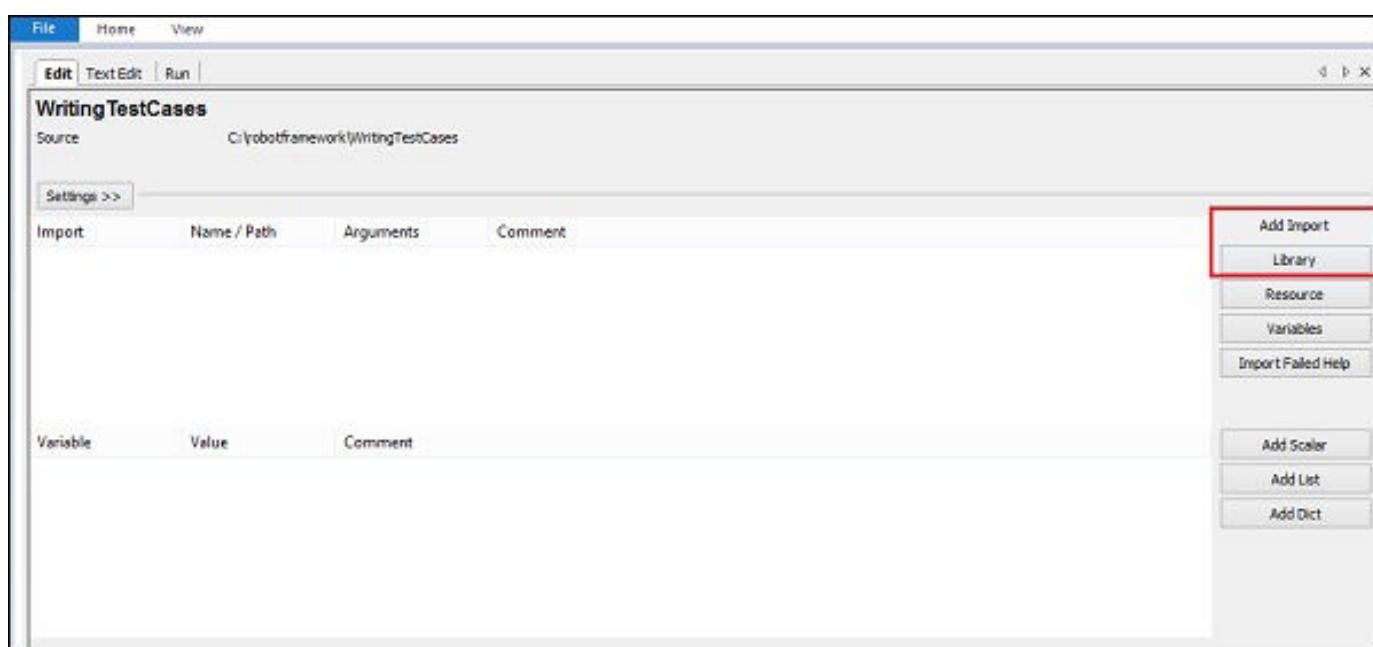
Robot Framework has its own built-in library, which need not be imported. But we need to interact with the browsers, databases, etc. To interact, we need to import the libraries.

The list of external libraries supported by robot framework are listed on robot framework official site as shown below –

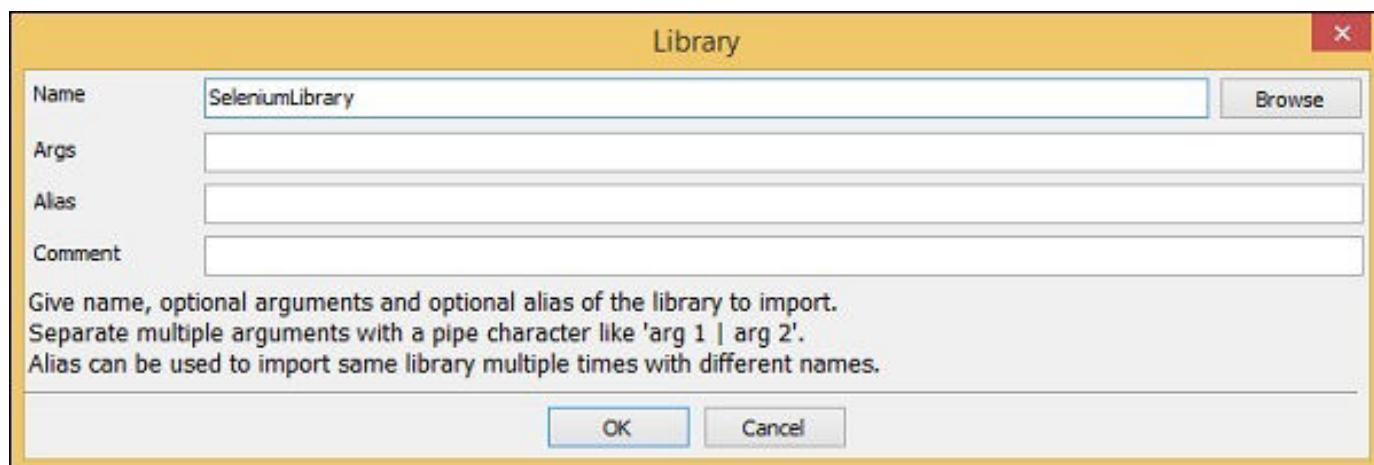
Android library	AnywhereLibrary	AppiumLibrary
Library for all your Android automation needs. It uses Calabash Android internally.	Library for testing Single-Page Apps (SPA). Uses Selenium Webdriver and Appium internally.	Library for Android- and iOS-testing. It uses Appium internally.
Archive library	AutoItLibrary	CncLibrary
Library for handling zip- and tar-archives.	Windows GUI testing library that uses AutoIt freeware tool as a driver.	Library for driving a CNC milling machine.
Database Library (Java)	Database Library (Python)	Debug Library
Java-based library for database testing. Usable with Jython. Available also at Maven central .	Python based library for database testing. Works with any Python interpreter, including Jython.	A debug library for RobotFramework, which can be used as an interactive shell(REPL) also.
Diff Library	Django Library	Eclipse Library
Library to diff two files together.	Library for Django , a Python web framework.	Library for testing Eclipse RCP applications using SWT widgets.
robotframework-faker	FTP library	HTTP library (livetest)
Library for Faker , a fake test data generator.	Library for testing and using FTP server with Robot Framework.	Library for HTTP level testing using livetest tool internally.
HTTP library (Requests)	HttpRequestLibrary (Java)	iOS library
Library for HTTP level testing using Request internally.	Library for HTTP level testing using Apache HTTP client. Available also at Maven central .	Library for all your iOS automation needs. It uses Calabash iOS Server internally.
ImageHorizonLibrary	JavaFXMLibrary	MongoDB library
Cross-platform pure Python library for GUI.	Library for testing JavaFX applications based	Library for interacting with MongoDB using

For working with browsers and web application, we are going to import Selenium Library. The installation is discussed in the chapter **Working with Browsers using Selenium Library**.

To import a library, we need to click main project. To the right, the setting will display the Add Import option as shown below –

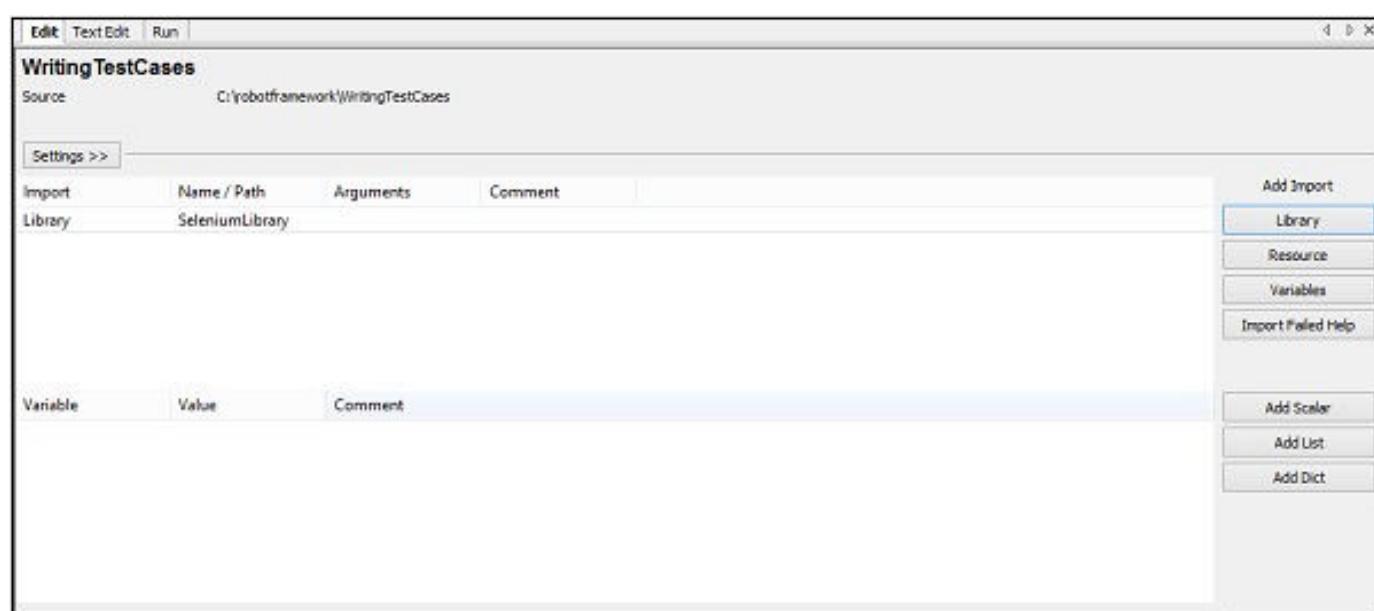


Click Library and enter the name of the library as shown below –

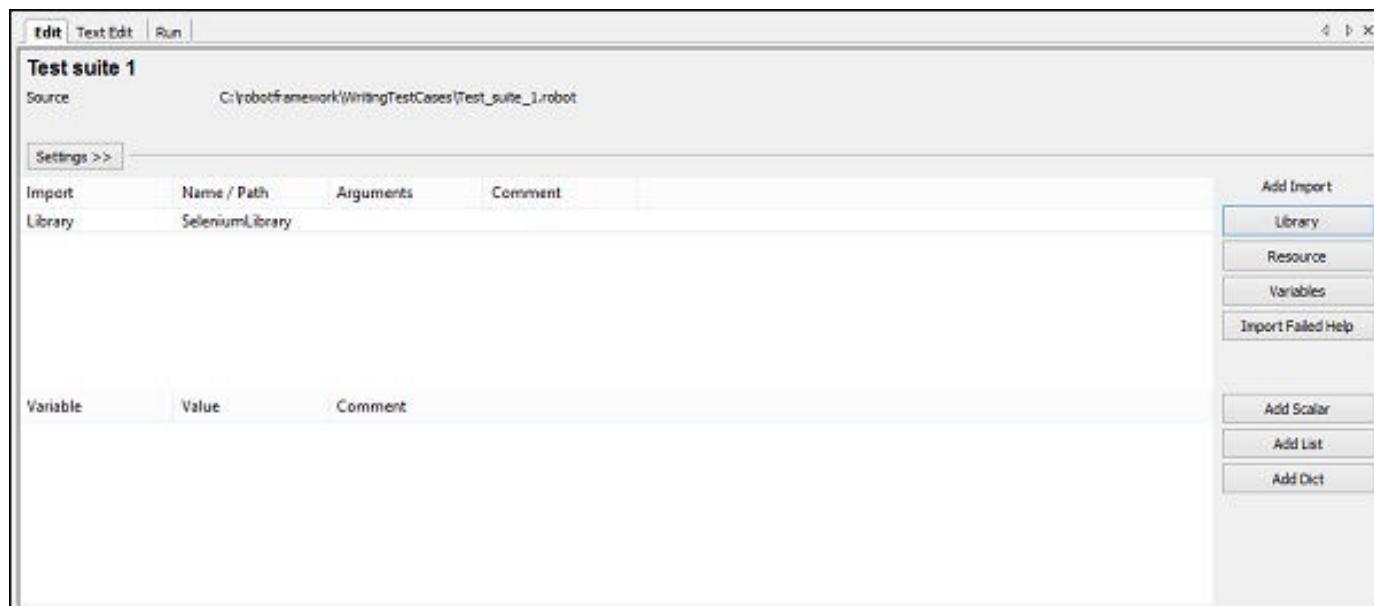


Click Ok to save the library.

The settings will be displayed in the settings as shown below –



We need to repeat the same step and add library for the test suite created. Click on the test suite created and import the library as shown below –



When you click on the test case on the left side, it will display the tabular format where you can enter the keywords. Now, you can use the built-in keywords and the keywords available from the selenium library.

Write test case in tabular format

Here is a simple test case, which opens the URL in chrome browser.

Edit		TextEdit		Run			
TC1							
Settings <<		Documentation				Edit	Clear
Setup						Edit	Clear
Teardown						Edit	Clear
Tags	<Add New>					Edit	Clear
Timeout						Edit	Clear
Template						Edit	Clear
1	Open Browser	https://www.tutorialspoint.com/	chrome				
2	Maximize Browser Window						
3	Close Browser						
4							
5							
6							
7							

The following shows the details of the test cases –

```
*** Settings ***
Library SeleniumLibrary

*** Test Cases ***
TC1
    Open Browser https://www.tutorialspoint.com/ chrome
    Maximize Browser Window
    Close Browser
```

We will add one more test case: TC2 in the same project.

TC2	
Settings <<	
Documentation	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
Setup	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
Teardown	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
Tags	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
Timeout	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
Template	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
1 \${a}	Set Variable Hi
2 Log	\${a}
3 \${b}	Set Variable If \${number}>0 Yes No
4 Log	\${b}
5	

*** Settings ***

Library SeleniumLibrary

*** Variables ***

\${number} 100

*** Test Cases ***

TC1

```
Open Browser https://www.tutorialspoint.com/ chrome
Maximize Browser Window
Close Browser
```

TC2

```
${a} Set Variable Hi
Log ${a}
${b} Set Variable If ${number}>0 Yes No
Log ${b}
```

We can add multiple test cases under the test suite created. Click Run to execute the test cases. The execution will take place based on the number of test cases added –

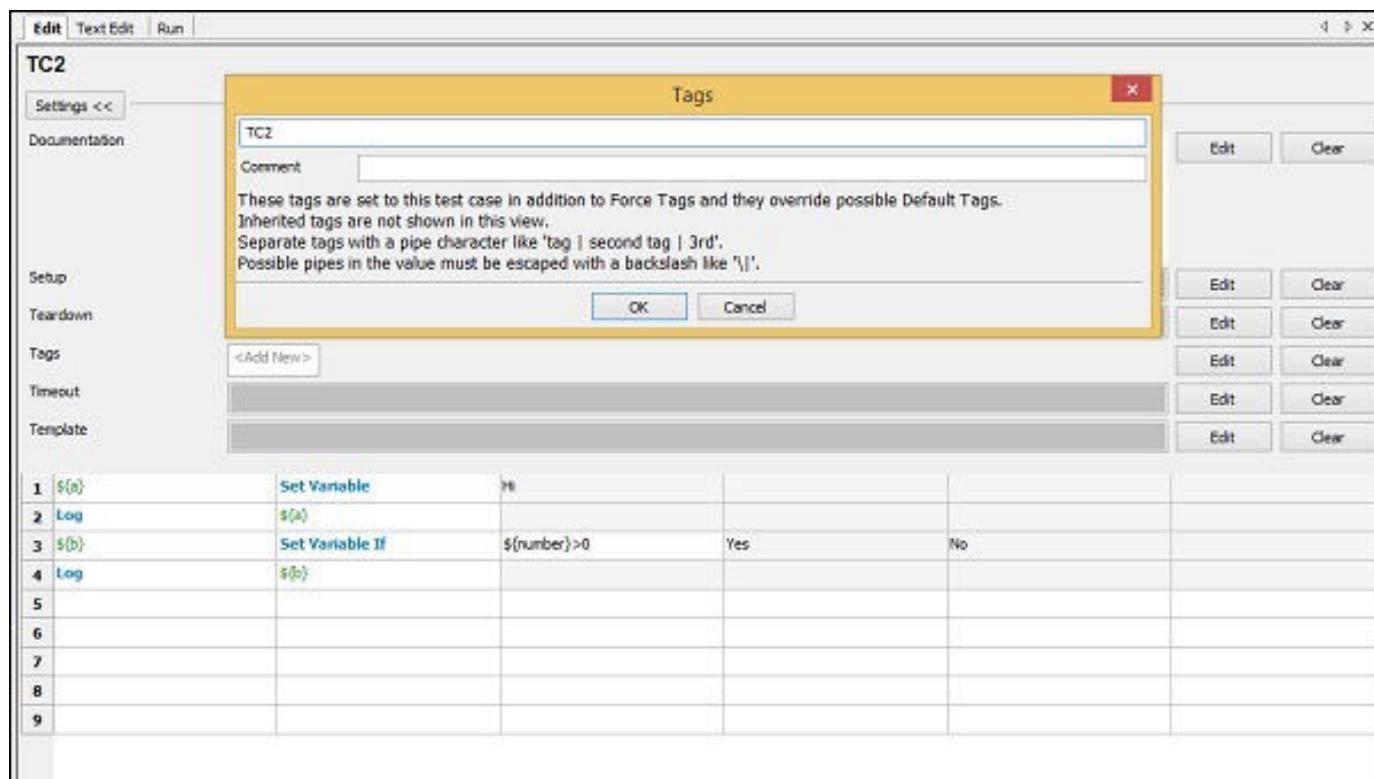
```

Elapsed time: 0:00:17  pass: 0  fail: 0
command: pybot.bat --argumentfile c:\users\kamat\appdata\local\temp\RIDEceebjs.d\argfile.txt --listener C:\Python27\lib\site-packages\riderunner\listeners\html.py
unable to open socket to "localhost:59445" error: [Errno 10061] No connection could be made because the target machine actively refused it
=====
WritingTestCases
=====
WritingTestCases: Test suite 1
=====
TC1
=====
| PASS |
=====
TC2
=====
| PASS |
=====
WritingTestCases: Test suite 1
2 critical tests, 2 passed, 0 failed
2 tests total. 2 passed, 0 failed
=====
WritingTestCases
2 critical tests, 2 passed, 0 failed
2 tests total. 2 passed, 0 failed
=====
Output: c:\users\kamat\appdata\local\temp\RIDEceebjs.d\output.xml
Log:   c:\users\kamat\appdata\local\temp\RIDEceebjs.d\log.html
Report: c:\users\kamat\appdata\local\temp\RIDEceebjs.d\report.html
test finished 20181021 17:23:36

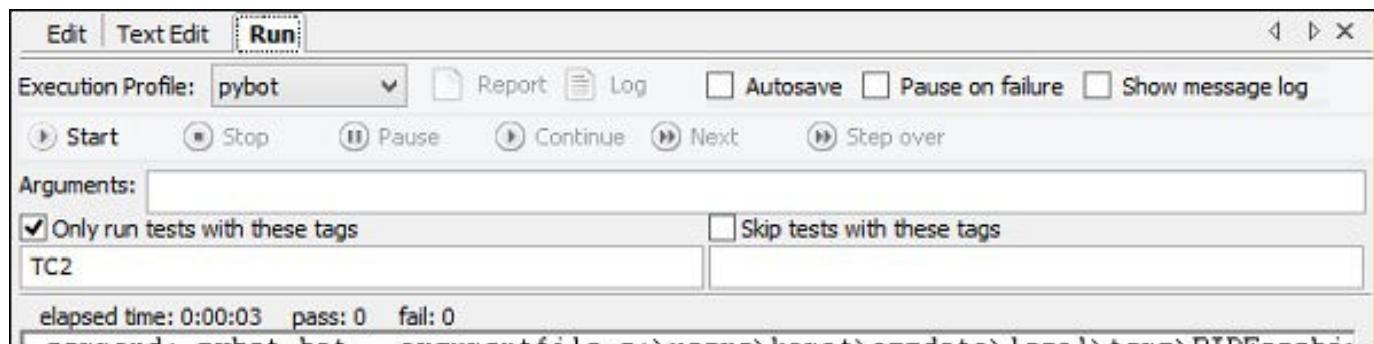
```

Using Tags for Executing Test Case

In case you want to run only test case TC2, you can tag the same. Click on the test case and click Edit across Tags as shown below –



Click Ok to save the tag. Add the tag name in Run as shown below –



We have selected option -> **Only run tests with these tags** and added tag name in it. Now, it will run only those test cases that have tag names. You can give any name and group the test cases based on tag name and run the same. You can also use tag to skip the test case.

```
Elapsed time: 0:00:03  pass: 0  fail: 0
command: pybot.bat --argumentfile c:\users\kamat\appdata\local\temp\RIDEoeebj
unable to open socket to "localhost:59445" error: [Errno 10061] No connection
=====
WritingTestCases
=====
WritingTestCases.Test suite 1
=====
TC2
=====
WritingTestCases.Test suite 1
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
=====
WritingTestCases
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
=====
Output:  c:\users\appdata\local\temp\RIDEoeebjs.d\output.xml
Log:    c:\users\appdata\local\temp\RIDEoeebjs.d\log.html
Report: c:\users\appdata\local\temp\RIDEoeebjs.d\report.html
test finished 20181021 17:27:11
```

Now we can see only TC2 running when executed.

Use Resource Files for Test Case

Robot framework has option for resource, where you can import robot files to be used with the test cases.

Test case TC1 that we have created uses the following keywords –

1	Open Browser	https://www.tutorialspoint.com/	chrome		
2	Maximize Browser Window				
3	Close Browser				
4					
5					
6					
7					
8					

We have used Keywords like –

- Open Browser
- Maximize Browser Window
- Close Browser

We will use a user-defined keyword for the above test case. The user-defined keyword will be available in the robot file which will be used as a resource.

We will create a file in the same directory and write our keyword as follows –

Please note details of keywords, i.e., how to create user-defined keywords are explained in *Robot Framework – Working with Keywords* chapter.

We have created a user-defined keyword called **Test Browser** as shown in the browseropen.robot file –

```
*** Settings ***
Library SeleniumLibrary

*** Variables ***
${url} https://www.tutorialspoint.com/
${browser} chrome

*** Keywords ***
Test Browser
  Open Browser ${url} ${browser}
  Maximize Browser Window
```

The file contains various options such as Settings, Variables, and Keywords. Please note, we cannot write test case inside the file to be used as resource. We will upload the above file as resource for the test suite as shown below.

Select the test suite. On the left side, click on resource option as shown below –

Test suite 1

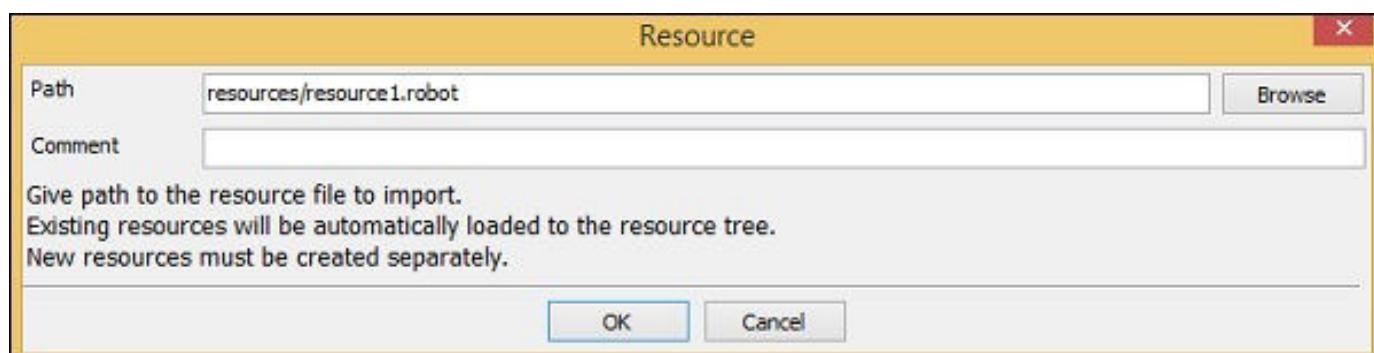
Settings >>			
Import	Name / Path	Arguments	Comment
Library	SeleniumLibrary		

Variable	Value	Comment
\$number	100	

Add Import
Library
Resource
Variables
Import Failed Help

Add Scalar
Add List
Add Dict

Click on Resource and it will ask the path to import robot file –



Mention the path where the file is stored as shown above and click OK to add resource. It will be displayed as shown below –

Test suite 1

Settings >>			
Import	Name / Path	Arguments	Comment
Library	SeleniumLibrary		
Resource	resources/resource1.robot		

Variable	Value	Comment
\$number	100	

Add Import
Library
Resource
Variables
Import Failed Help

Add Scalar
Add List
Add Dict

Now, we will change test case TC1 which has keywords as shown below –

1	Open Browser	https://www.tutorialspoint.com/	chrome		
2	Maximize Browser Window				
3	Close Browser				
4					
5					
6					
7					
8					

We will add the user-defined keyword to TC1 from the resource file, i.e., Test Browser keyword –

1	Test Browser				
2	Close Browser				
3					
4					
5					
6					
7					

The resource file uploaded is as shown below –

The screenshot shows the Robot Framework IDE interface. On the left, the project tree displays a folder named 'WritingTestCases' containing a 'Test suite 1' folder. Inside 'Test suite 1' are two test cases: 'TC1' and 'TC2'. A 'Resources' folder contains three files: 'browseropen.robot', 'resource1.robot', and 'resource1.robot' (a duplicate). The 'resource1.robot' file is currently selected and its content is visible in the main pane:

```

1  *** Settings ***
2  Library    SeleniumLibrary
3
4  *** Variables ***
5  ${url}      https://www.tutorialspoint.com/
6  ${browser}   chrome
7
8  *** Keywords ***
9  Test Browser
10 Open Browser ${url} ${browser}
11 Maximize Browser Window
12

```

The user-defined Keyword is used in test case TC1.

We will now execute the test case –

Execution Profile: pybot Report Log Autosave Pause on failure Show message log

Start Stop Pause Continue Next Step over

Arguments: Only run tests with these tags Skip tests with these tags

```

elapsed time: 0:00:18 pass: 2 fail: 0
command: pybot bat --argumentfile c:\users\kanat\appdata\local\temp\RIDE8u2b0.d\argfile.txt --listener C:\Python27\lib\site-packages\robot\reporters\html.py
WritingTestCases
WritingTestCases.Test suite 1
TC1 | PASS |
TC2 | PASS |
WritingTestCases.Test suite 1
2 critical tests, 2 passed, 0 failed
2 tests total. 2 passed. 0 failed
WritingTestCases
2 critical tests, 2 passed, 0 failed
2 tests total. 2 passed. 0 failed
Starting test: WritingTestCases.Test suite 1.TC1
20181028 19:06:19.062 : INFO : Opening browser 'chrome' to base url 'https://www.tutorialspoint.com/'.
Ending test: WritingTestCases.Test suite 1.TC1
Starting test: WritingTestCases.Test suite 1.TC2
20181028 19:06:35.078 : INFO : ${a} = Hi
20181028 19:06:35.094 : INFO : ${b} =
20181028 19:06:35.094 : INFO : ${b} = Yes
20181028 19:06:35.094 : INFO : Yes
Ending test: WritingTestCases.Test suite 1.TC2

```

We have both test cases being passed. Let us now see the report and log details.

Report

WritingTestCases Test Report

Generated
20181028 19:06:35 GMT+05:30
1 minute 35 seconds ago

Summary Information

Status:	All tests passed
Start Time:	20181028 19:06:19.038
End Time:	20181028 19:06:35.925
Elapsed Time:	00:00:16.887
Log File:	log.html

Test Statistics

Total Statistics		Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests		2	2	0	00:00:16	
All Tests		2	2	0	00:00:16	

Statistics by Tag		Total	Pass	Fail	Elapsed	Pass / Fail
TC2		1	1	0	00:00:00	

Statistics by Suite		Total	Pass	Fail	Elapsed	Pass / Fail
WritingTestCases		2	2	0	00:00:17	
WritingTestCases.Test suite 1		2	2	0	00:00:16	

Test Details

Totals Tags Suites Search

Type: Critical Tests All Tests

Log

WritingTestCases Test Log

Generated
20181028 19:06:35 GMT+05:30
1 minute 59 seconds ago

Test Statistics

Total Statistics		Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests		2	2	0	00:00:16	
All Tests		2	2	0	00:00:16	

Statistics by Tag		Total	Pass	Fail	Elapsed	Pass / Fail
TC2		1	1	0	00:00:00	

Statistics by Suite		Total	Pass	Fail	Elapsed	Pass / Fail
WritingTestCases		2	2	0	00:00:17	
WritingTestCases.Test suite 1		2	2	0	00:00:16	

Test Execution Log

- **SUITE** WritingTestCases
 - Full Name: WritingTestCases
 - Source: C:/robotframework/WritingTestCases
 - Start / End / Elapsed: 20181028 19:06:19.030 / 20181028 19:06:35.925 / 00:00:16.897
 - Status: 2 critical test, 2 passed, 0 failed
2 test total, 2 passed, 0 failed
- **SUITE** Test suite 1
 - Full Name: WritingTestCases.Test suite 1
 - Source: C:/robotframework/WritingTestCases/Test_suite_1.robot
 - Start / End / Elapsed: 20181028 19:06:19.630 / 20181028 19:06:35.909 / 00:00:16.079
 - Status: 2 critical test, 2 passed, 0 failed
2 test total, 2 passed, 0 failed
- + **TEST** TC1
- + **TEST** TC2

Conclusion

This chapter gives details on how to write test case, execute it, how to tag a test-case, use resources, etc.

Keyword and Data Driven Test Cases

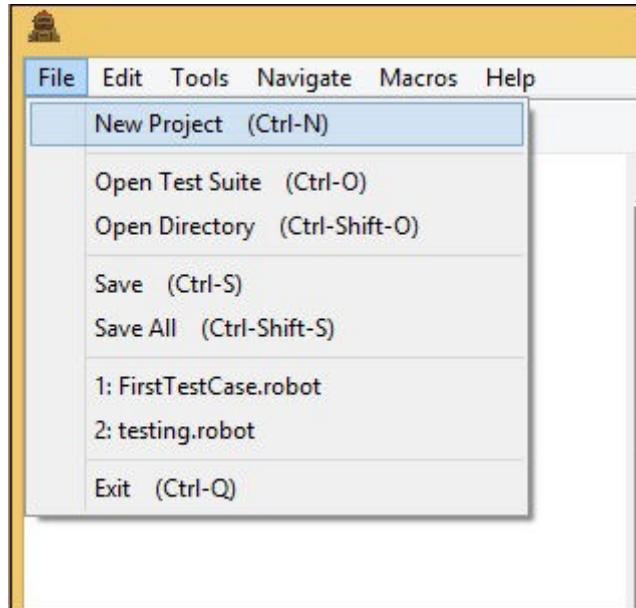
The workflow of a test-case can be tested using keyword or data driven style. In case you want to test the workflow with different inputs, the same can be done using data driven test cases. We will work on an example to go through the following test case approaches –

- Keyword Driven style
- Data Driven style

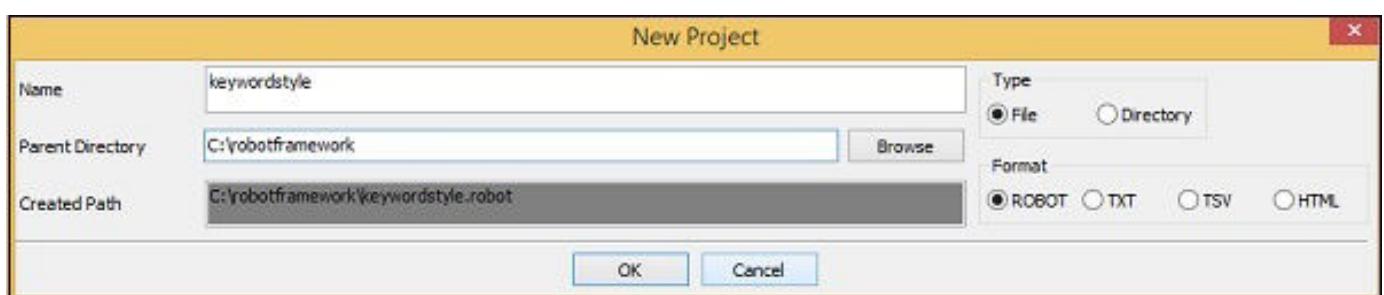
Keyword Driven Style

We will do a project setup to show the working of Keyword driven style.

Open ride using **ride.py** from the command line.

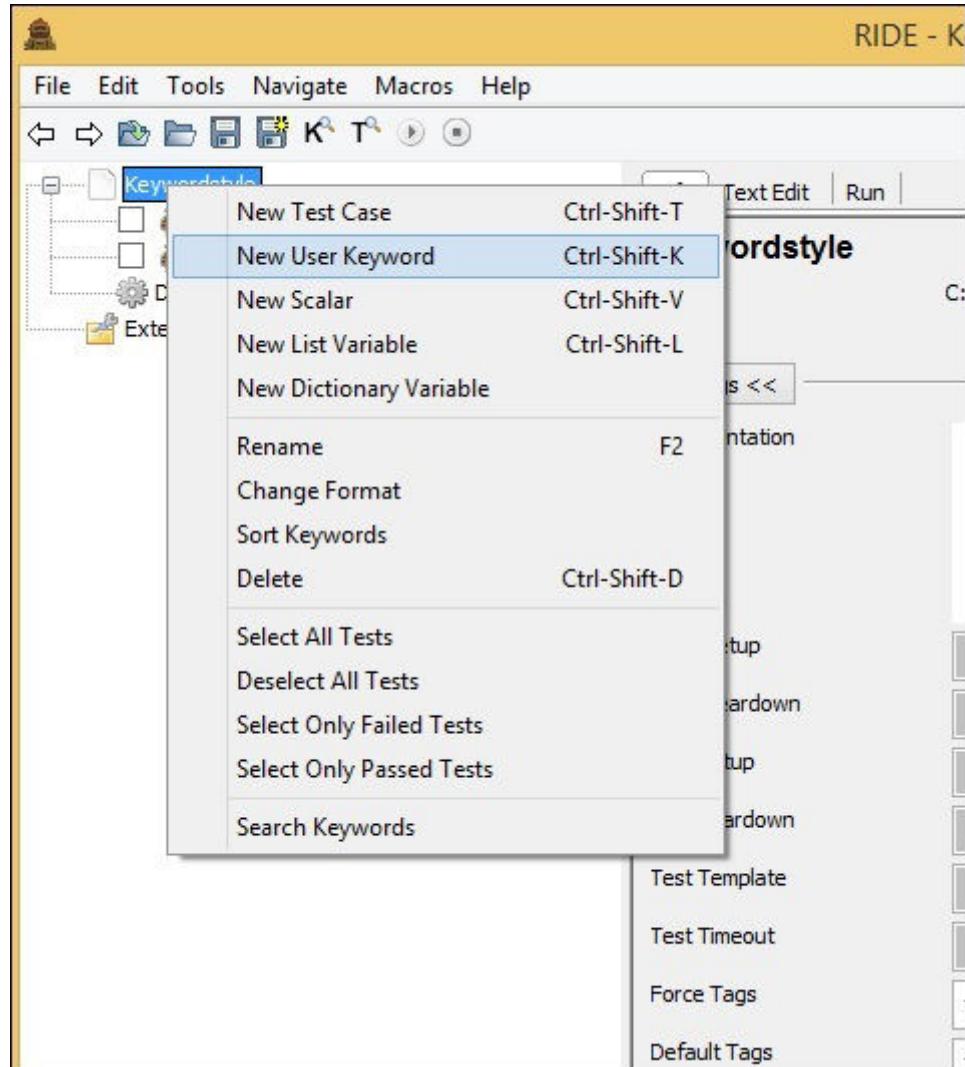


Click on New Project and give a name to your project.

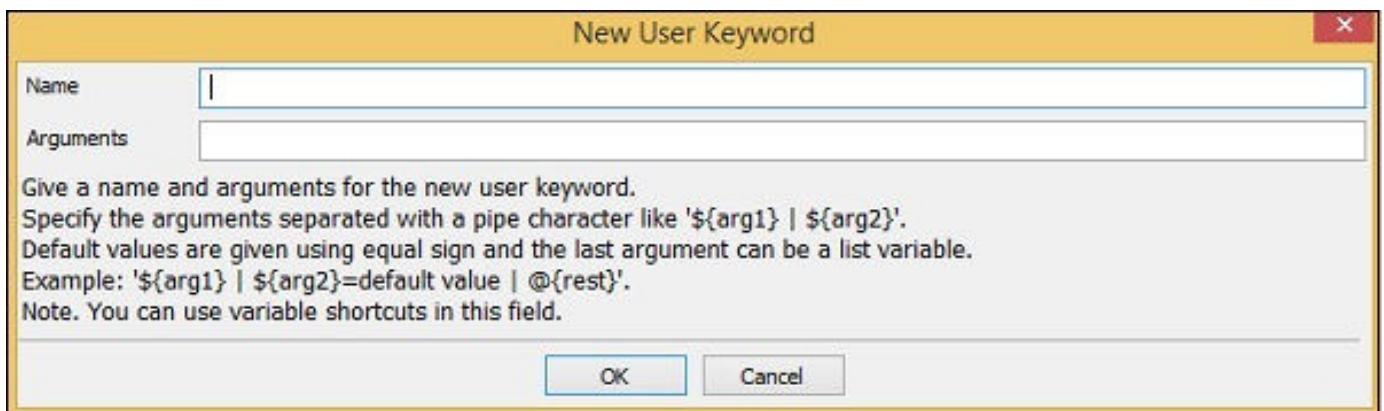


The name given to the project is *keywordstyle*. Click OK to save the project. In this project, we will create a user keyword as shown below.

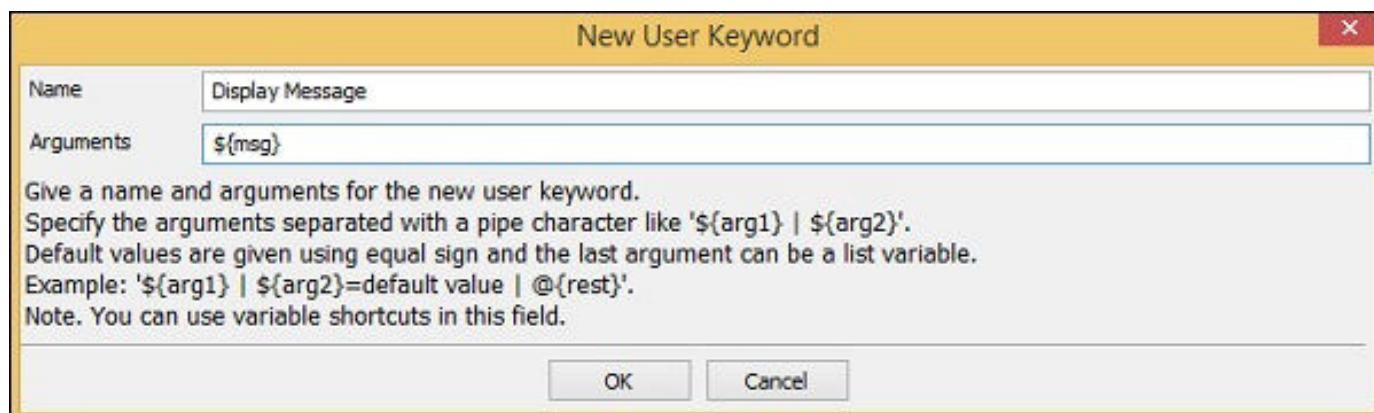
Right-click on the name of the project and click on *New User Keyword* as shown below



It will display screen as follows –



Enter the name of the keyword and the arguments it will take. Here we will give name of the keyword as Display Message. The role of Keyword Display Message is, when it is called, it will log a message. So we need to give an argument to it. Therefore, in the above example the argument will be a scalar variable \${msg}.



Click OK to save the user keyword. Now we need to write the action the keywords need to do. So, it will have tabular format as shown below where we can give the Library keywords or built-in keywords available with Robot Framework.

Here, we will use a simple Log keyword available with Robot Framework as shown below –

The screenshot shows the configuration screen for the "Display Message" keyword. At the top, there are tabs for "Edit", "Text Edit", and "Run". The main title is "Display Message" with a "Find Usages" button. Below the title, there's a "Settings <<" button and a "Documentation" section with "Edit" and "Clear" buttons. The configuration table has columns for "Tags", "Arguments", "Teardown", "Return Value", and "Timeout". The "Arguments" column contains "\${msg}". The table below lists actions:

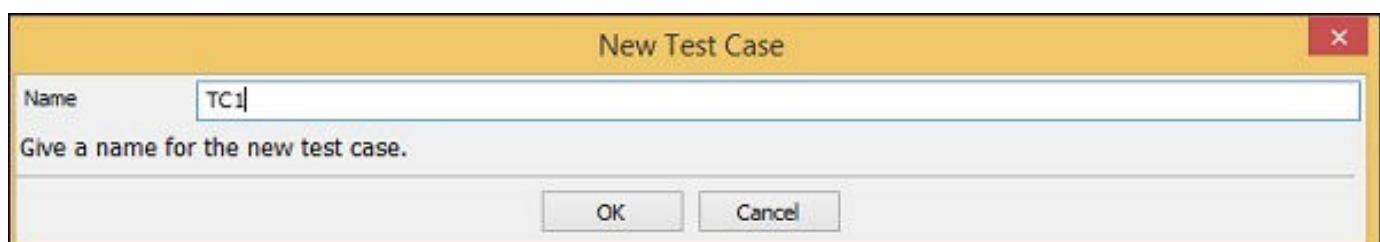
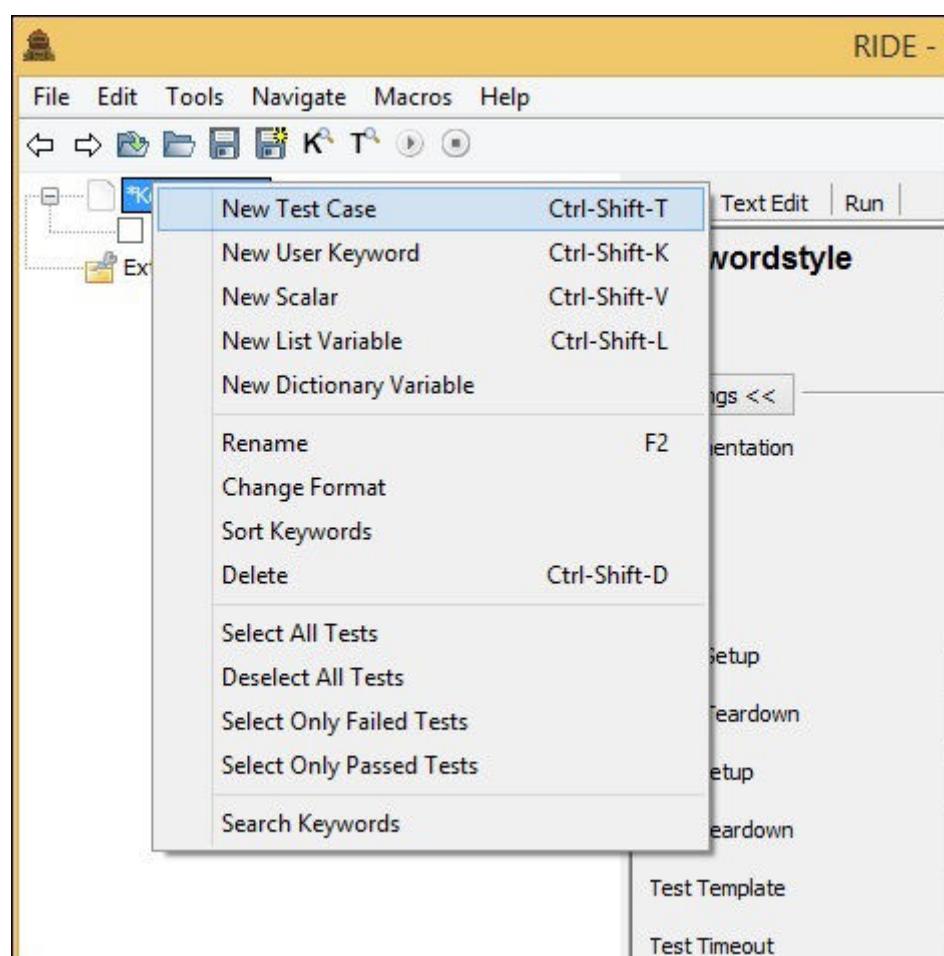
1	Log	\${msg}
2		
3		
4		
5		
6		

To get more keywords available with Robot framework, press **ctrl + space bar** in the table column as shown below –

The screenshot shows the RIDE (Robot IDE) interface. On the left, a tree view lists various keywords under the 'Log' category. One keyword, 'Fail', is selected and highlighted with a yellow background. To the right of the tree view, detailed information about the 'Fail' keyword is displayed in a large yellow box. The box contains the following text:
Name: Fail
Source: BuiltIn <test library>
Arguments: [msg=None | *tags]
Description: Fails the test with the given message and optionally alters its tags.
The error message is specified using the msg argument. It is possible to use HTML in the given error message, similarly as with any other keyword accepting an error message, by prefixing the error with *HTML*.

So the keyword we want to use with our testcase is ready. The name of the user keyword is *Display Message* and it takes one argument called \${msg}.

Let us now use this keyword in simple keyword driven style test-case. To do that we need to create test case. Right-click on the name of the project created. Now, click New Test Case –



Give name to the test case and click OK.

We are done with the project setup and now will write test cases for the keyword driven style.

In the test case, we have used the user-defined keyword Display Message in the tabular format as shown below –

TC1		
Settings <<		
Documentation	<input type="text"/>	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
Setup	<input type="text"/>	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
Teardown	<input type="text"/>	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
Tags	<Add New>	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
Timeout	<input type="text"/>	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
Template	<input type="text"/>	<input type="button" value="Edit"/> <input type="button" value="Clear"/>
1	Display Message	Hello World
2		
3		
4		
5		
6		

We have used the keyword we have created as shown above and passed the value Hello World.

We will execute the test case TC1 and check the output –

```

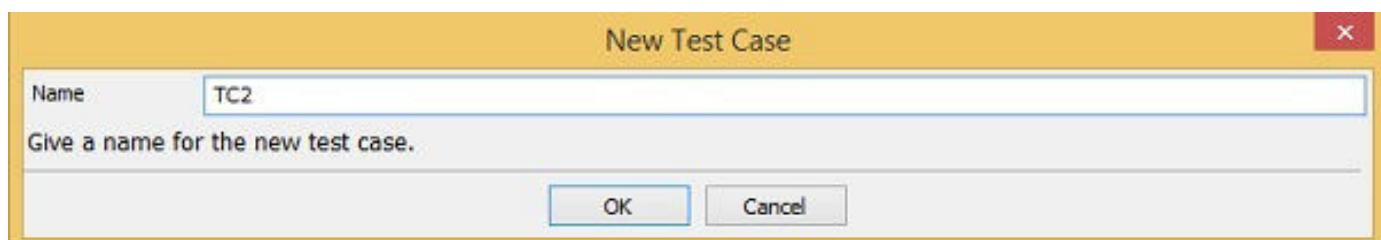
elapsed time: 0:00:01  pass: 1  fail: 0
command: pybot.bat --argumentfile c:\users\kamat\appdata\local\temp\RIDEcdkk0i.d
=====
Keywordstyle
=====
TC1 | PASS |
=====
Keywordstyle | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
=====
Output: c:\users\appdata\local\temp\RIDEcdkk0i.d\output.xml
Log:   c:\users\appdata\local\temp\RIDEcdkk0i.d\log.html
Report: c:\users\appdata\local\temp\RIDEcdkk0i.d\report.html
test finished 20181027 12:35:41
< >
Starting test: Keywordstyle.TC1
20181027 12:35:41,490 : INFO : Hello World
Ending test:   Keywordstyle.TC1

```

In the above example, we have written a simple test-case which logs message and the test case is executed with output *Hello World*. We can see the output Hello World printed in the log. The test case is also passed here.

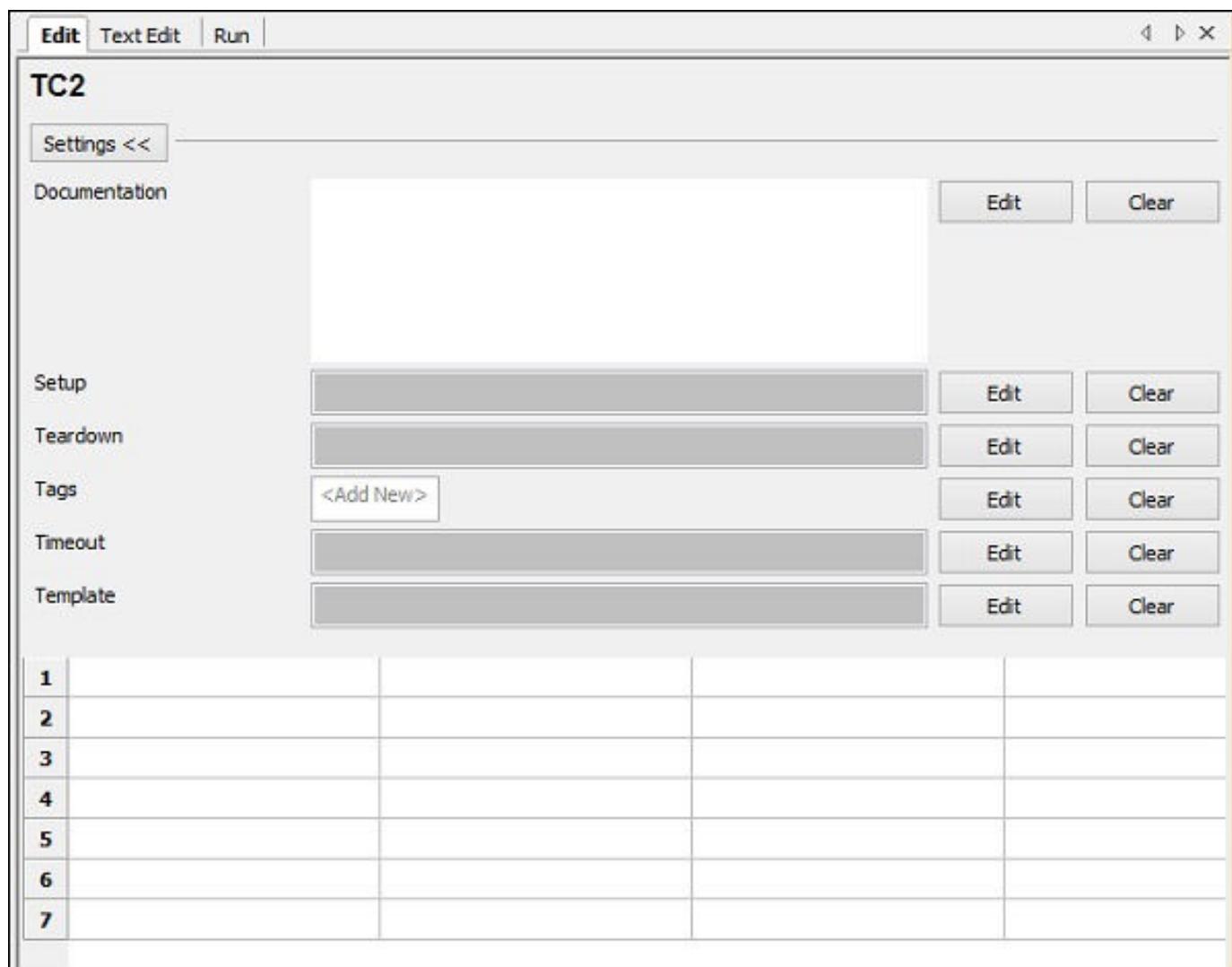
Data Driven Style

We will create one more test case in the same project. We will give the name of the test-case as TC2.

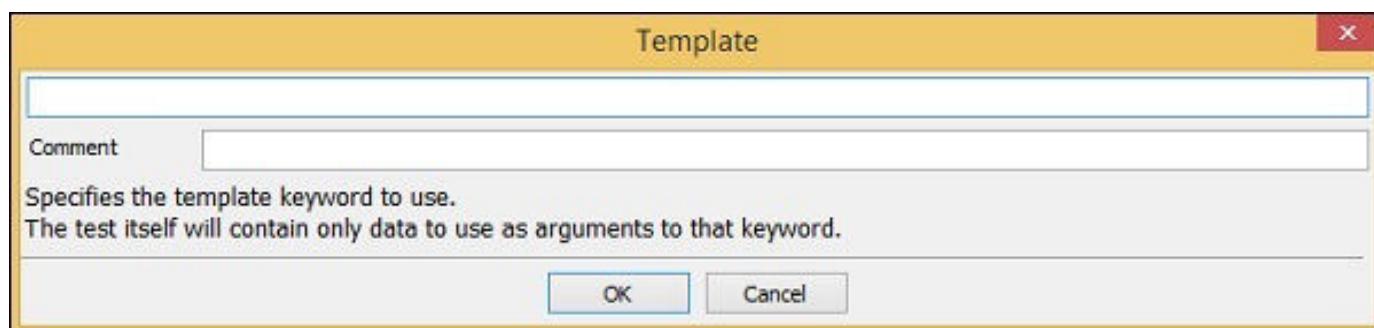


To work with data driven style, we need to create template. Template will take the name of the high level keyword, which is a user-defined keyword like the one we created at the start called Display Message. The arguments to that template will be sent in the form of test-cases. We can pass different values to that template keyword. The data driven approach is mostly used when you want to test the scenario with different data to it.

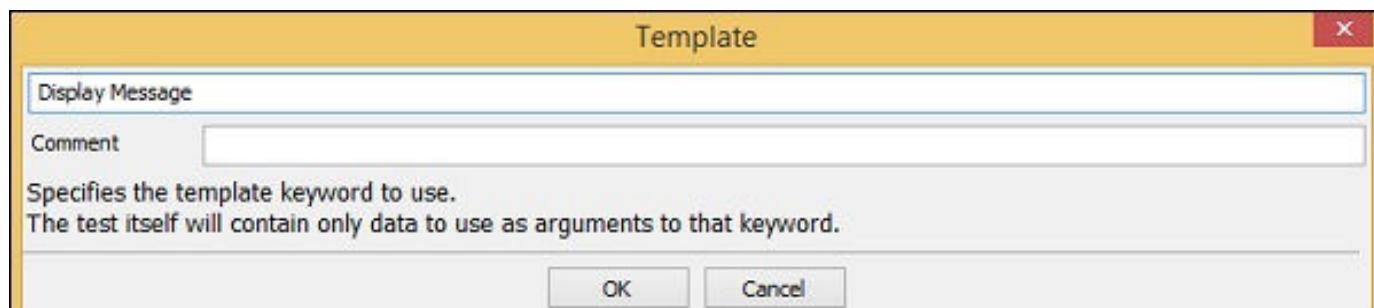
Once the test case is saved. Click on the test case and the display will be as follows –



Click on Edit button for Template and add the user-defined keyword.



Enter the user keyword for the template and click OK to save the template.



Display Message keyword takes one argument called \${msg}. This is a scalar variable. The details passed in this test case will act as arguments to the user-defined keyword *Display*

Message.

The screenshot shows the TC2 software interface. At the top, there is a menu bar with 'Edit' (highlighted in blue), 'Text Edit', and 'Run'. Below the menu is a title 'TC2'. On the left, there is a 'Settings <<' button. Under 'Documentation', there are 'Edit' and 'Clear' buttons. The main area contains configuration sections for 'Setup', 'Teardown', 'Tags' (with a link to '[Add New](#)'), 'Timeout', and 'Template' (set to '[Display Message](#)'). To the right of each configuration section are 'Edit' and 'Clear' buttons. Below these sections is a table with 7 rows, indexed from 1 to 7. Row 1 contains 'My First Test Case'. Row 2 contains 'Testing Template'. Rows 3 through 6 are empty. Row 7 contains a redacted rectangular box.

1	My First Test Case		
2	Testing Template		
3			
4			
5			
6			
7	<input type="text"/>		

In TC2, we have added Template Display Message (user-defined keyword). We have given messages in the tabular format.

Let us now execute the test case.

The screenshot shows the Ride IDE interface with the 'Run' tab selected. The execution profile is set to 'pybot'. The arguments section contains two checkboxes: 'Only run tests with these tags' and 'Skip tests with these tags'. The main output window displays the following test results:

```
elapsed time: 0:00:01  pass: 2  fail: 0
=====
Keywordstyle
=====
TC1 | PASS |
TC2 | PASS |
=====
Keywordstyle | PASS |
2 critical tests, 2 passed, 0 failed
2 tests total, 2 passed, 0 failed
=====
Output: c:\users\appdata\local\temp\RIDEcdkk0i.d\output.xml
Log:   c:\users\appdata\local\temp\RIDEcdkk0i.d\log.html
Report: c:\users\appdata\local\temp\RIDEcdkk0i.d\report.html
test finished 20181027 12:47:14
```

Below the main output, there is a detailed log of test executions:

```
Starting test: Keywordstyle.TC1
20181027 12:47:14.424 : INFO : Hello World
Ending test: Keywordstyle.TC1

Starting test: Keywordstyle.TC2
20181027 12:47:14.439 : INFO : My First Test Case
20181027 12:47:14.455 : INFO : Testing Template
Ending test: Keywordstyle.TC2
```

We can see Run executes both the Test Cases. The output shown for TC1 is Hello World. This was the message we had given to the User Keyword Display Message.

For TC2, we used Display Message as a Template. We passed *My First Test Case* and *Testing Template* as values in TC2. As the user keyword Display Message uses internally Log Keyword, it displays the message in the log as shown above.

Conclusion

We have used keyword style and data driven style in this chapter and seen the working of both. Data Driven style takes high-level user-defined keyword as a template and all the test cases act as values to the template.

Working With Browsers Using Selenium Library

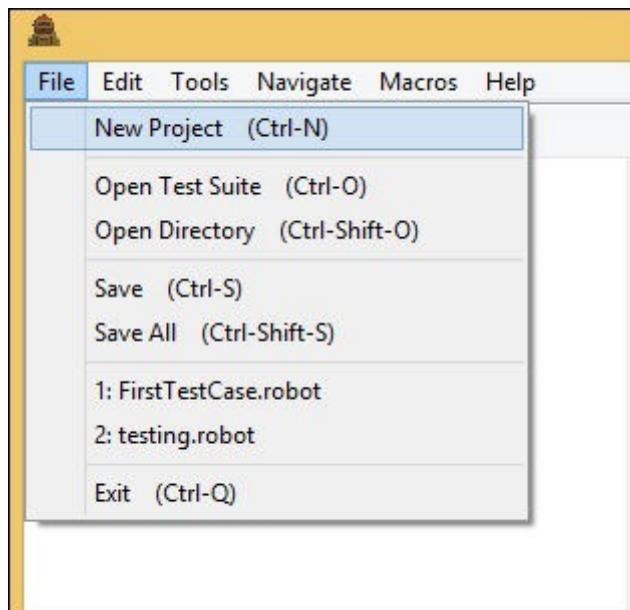
In this chapter, we will learn how to work with browsers using Robot Framework and Selenium Library in ride.

- Project setup in Ride
- Import Selenium Library
- Test case using Chrome Browser

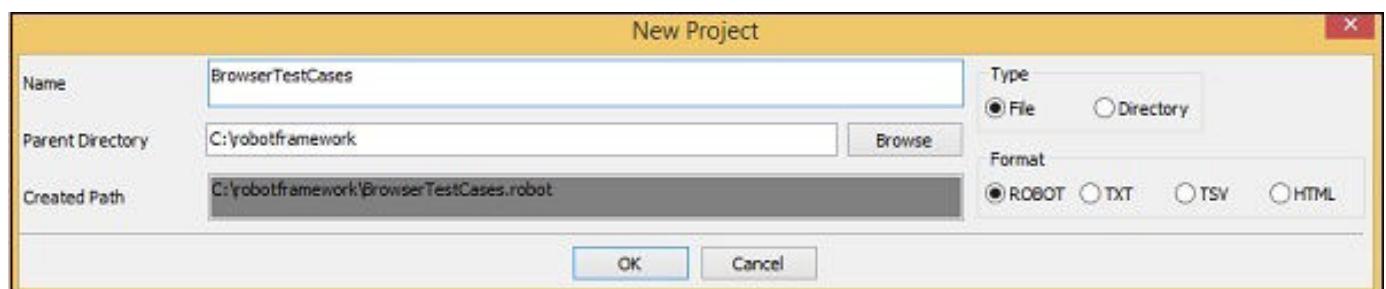
- Test case using Firefox Browser

Project Setup In Ride

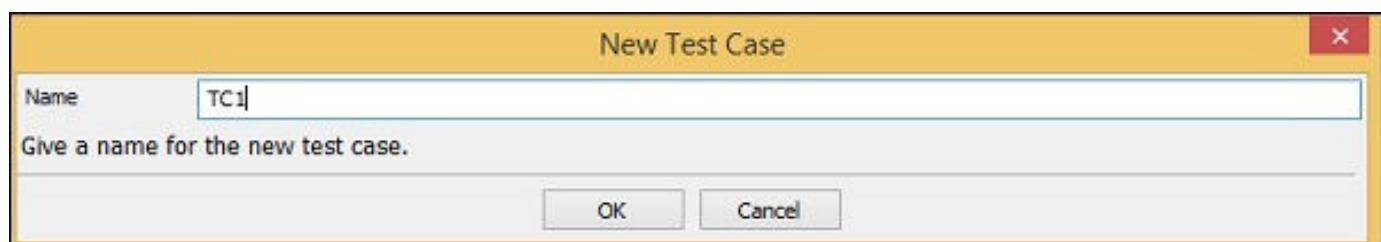
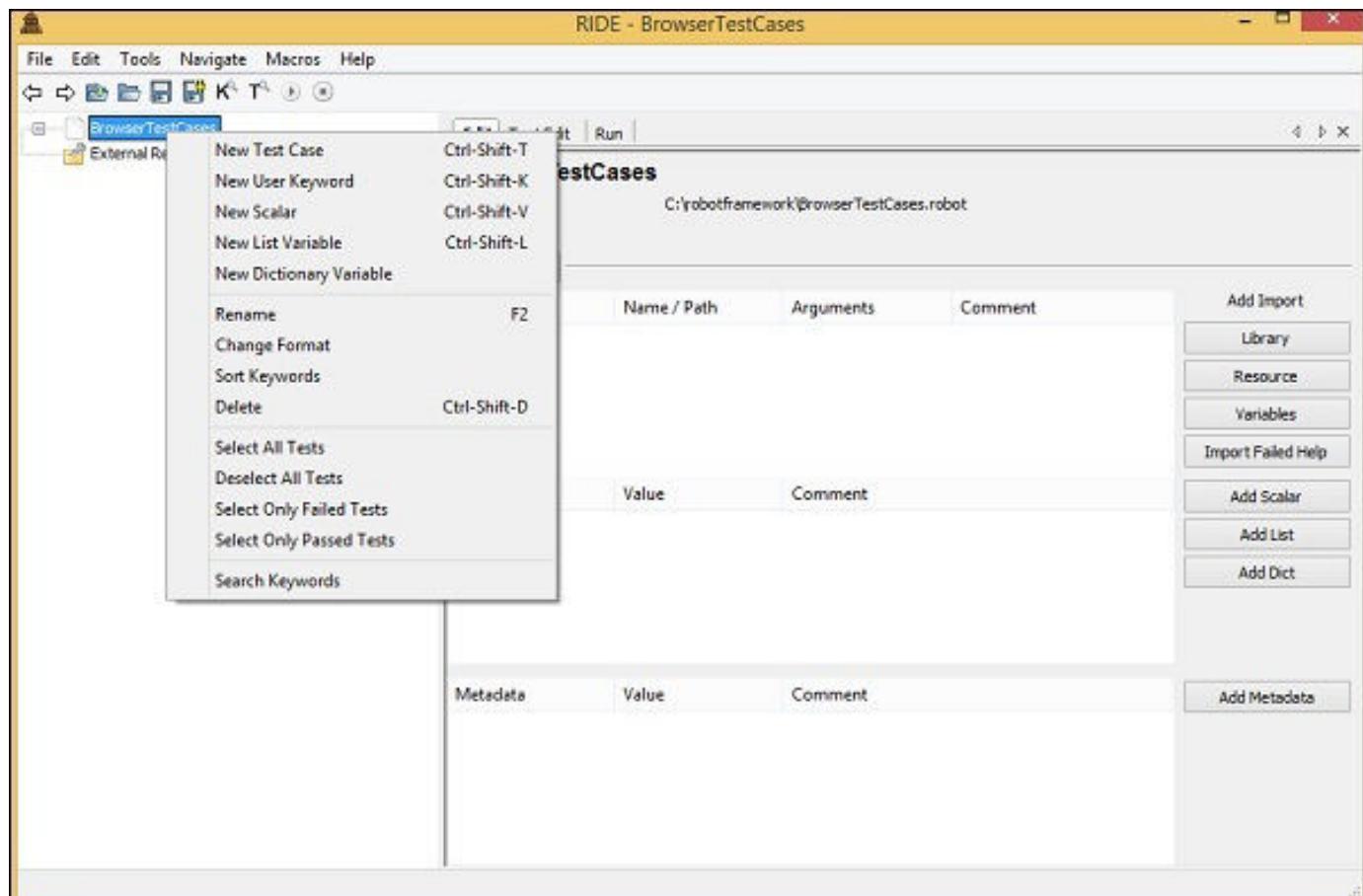
We will first create a project in Ride to work with browsers. Open ride using ride.py from the command line.



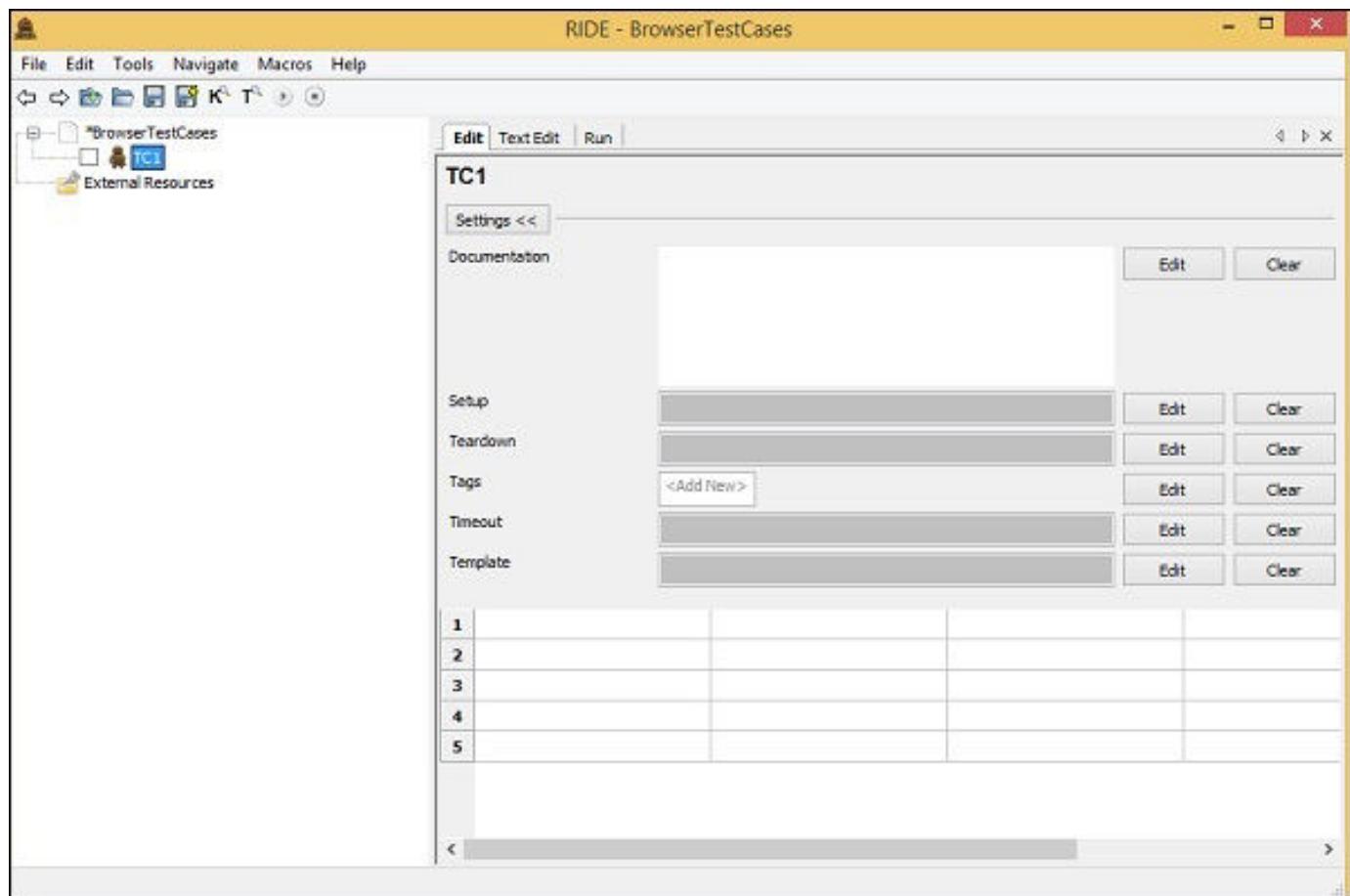
Click on *New Project* and give name to your project.



The name given is *BrowserTestCases*. Click *OK* to save the project. Right-click on the name of the project created and click on *New Test Case* –



Give name to the test case and click OK.



The screenshot shows the official Robot Framework website at robotframework.org. The header includes a logo of a stylized 'R' inside a square, the text 'Not secure | robotframework.org', and a search bar with a magnifying glass icon. The main title 'ROBOT FRAMEWORK' is displayed in large, bold, white and black letters against a teal background. On the left sidebar, there is a vertical menu with options: INTRODUCTION, EXAMPLES, LIBRARIES (which is highlighted in blue), TOOLS, DOCUMENTATION, SUPPORT, RPA, FOUNDATION, SHOP, ROBOCON, and USERS.

INTRODUCTION

Robot Framework is a generic test automation framework for acceptance testing and acceptance test-driven development (ATDD). It has easy-to-use tabular test data syntax and it utilizes the keyword-driven testing approach. Its testing capabilities can be extended by test libraries

On the left side, select the LIBRARIES option.

The screenshot shows the 'LIBRARIES' page from the Robot Framework website. The left sidebar remains the same as the homepage. The main content area has a heading 'LIBRARIES'. Below it is a paragraph of text: 'Test libraries provide the actual testing capabilities to Robot Framework by providing keywords. There are several standard libraries that are bundled in with the framework, and galore of separately developed external libraries that can be installed based on your needs. Creating your own test libraries **is a breeze**. Let us know if there are useful libraries missing from the list.' At the bottom, there is a table with three tabs: 'STANDARD', 'EXTERNAL', and 'OTHER'. The 'STANDARD' tab is selected, showing two rows of library information:

	Builtin	Dialogs	Collections
Provides a set of often needed generic keywords. Always automatically available without imports.	Provides means for pausing the test execution and getting input from users.	Provides a set of keywords for handling Python lists and dictionaries.	
OperatingSystem	Remote	Screenshot	

Select External option from above and it will list you all the libraries available to be used.

STANDARD	EXTERNAL	OTHER
Android library Library for all your Android automation needs. It uses Calabash Android internally.	AnywhereLibrary Library for testing Single-Page Apps (SPA). Uses Selenium Webdriver and Appium internally.	AppiumLibrary Library for Android- and iOS-testing. It uses Appium internally.
Archive library Library for handling zip- and tar-archives.	AutoItLibrary Windows GUI testing library that uses AutoIt freeware tool as a driver.	CncLibrary Library for driving a CNC milling machine.
RESTinstance Robot Framework test library for HTTP JSON APIs.	SapGuiLibrary Library for testing the SAPGUI client using the internal SAP Scripting Engine	SeleniumLibrary Web testing library that uses popular Selenium tool internally.
Selenium2Library Web testing library that uses Selenium 2. Library is deprecated; users should upgrade to SeleniumLibrary described above.	Selenium2Library for Java Java port of the Selenium2Library.	ExtendedSelenium2Library Web testing library that uses Selenium2Library internally, providing AngularJS support on top of it.

Click [SeleniumLibrary](#).

You will be redirected to the github repo as shown below –

See keyword documentation for available keywords and more information about the library in general.

Installation

The recommended installation method is using pip:

```
pip install --upgrade robotframework-seleniumlibrary
```

Running this command installs also the latest Selenium and Robot Framework versions, but you still need to [install browser drivers](#) separately. The `--upgrade` option can be omitted when installing the library for the first time.

Those migrating from [Selenium2Library](#) can install SeleniumLibrary so that it is exposed also as Selenium2Library:

```
pip install --upgrade robotframework-selenium2library
```

The above command installs the normal SeleniumLibrary as well as a new Selenium2Library version that is just a thin wrapper to SeleniumLibrary. That allows importing Selenium2Library in tests while migrating to SeleniumLibrary.

To install the last legacy Selenium2Library version, use this command instead:

```
pip install robotframework-selenium2library==1.8.0
```

With recent versions of pip it is possible to install directly from the [GitHub](#) repository. To install latest source from the master

For Installation of seleniumlibrary, we can use the command from the github and install it using pip.

Command

```
pip install --upgrade robotframework-seleniumlibrary
```

Command Prompt - pip install --upgrade robotframework-seleniumlibrary

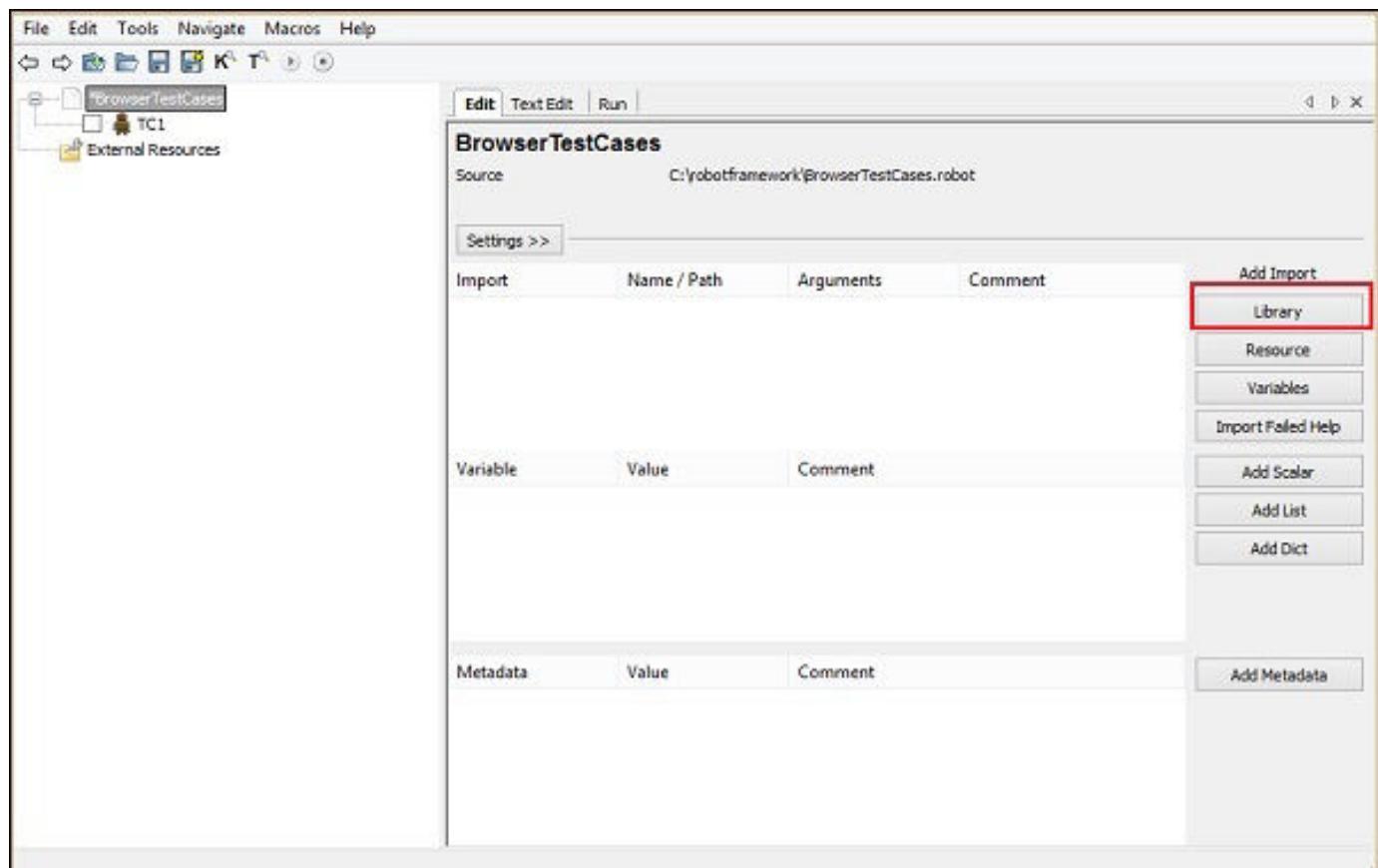
```
C:\>robotframework>pip install --upgrade robotframework-seleniumlibrary
Collecting robotframework-seleniumlibrary
  Downloading https://files.pythonhosted.org/packages/74/e9/19f4f96e1f35ed34e5f9d06d8285f981d2b8c5e7efa23d5c201c4650d732/robotframework_seleniumlibrary-3.2.0-py2.py3-none-any.whl (79kB)
    100% #####: 81kB 405kB/s
Collecting selenium>=3.4.0 (from robotframework-seleniumlibrary)
-
```

Selenium library gets installed inside the lib folder in python as follows –

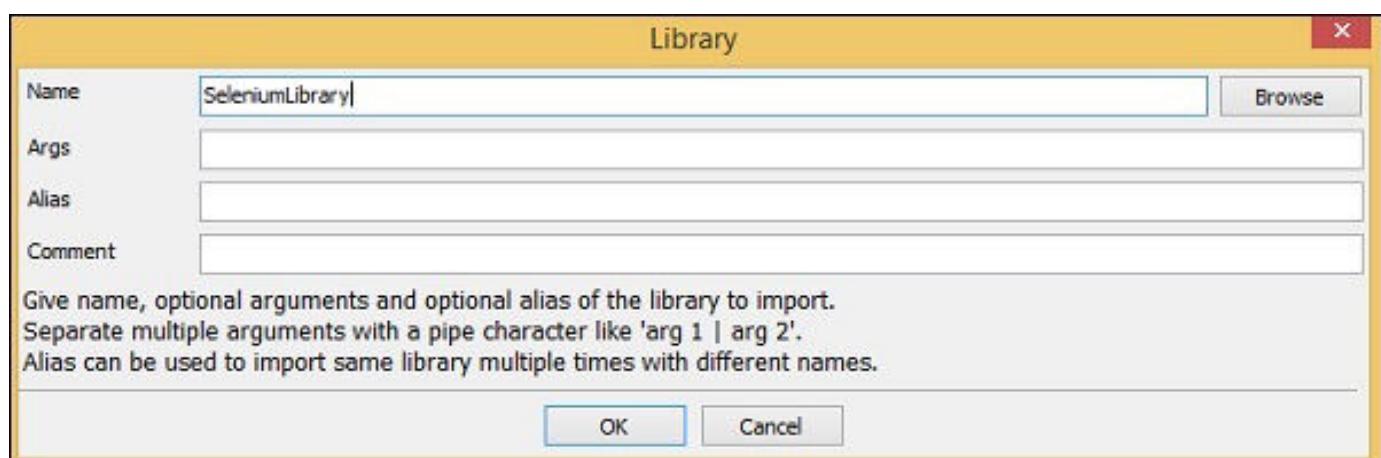
C:\Python27\Lib\site-packages			
	Name	Date modified	Type
📁	pip	06-10-2018 16:10	File folder
📁	pip-9.0.3.dist-info	06-10-2018 16:10	File folder
📁	pkg_resources	06-10-2018 16:10	File folder
📁	pubsub	06-10-2018 21:18	File folder
📁	Pypubsub-4.0.0-py2.7.egg-info	06-10-2018 21:18	File folder
📁	robot	06-10-2018 21:10	File folder
📁	robotframework_ride-1.5.2.1-py2.7.egg-i...	06-10-2018 21:28	File folder
📁	robotframework_seleniumlibrary-3.2.0.di...	08-10-2018 14:14	File folder
📁	robotframework-3.0.4-py2.7.egg-info	06-10-2018 21:10	File folder
📁	robotide	06-10-2018 21:28	File folder
📁	selenium	08-10-2018 14:14	File folder
📁	selenium-3.14.1.dist-info	08-10-2018 14:14	File folder
📁	SeleniumLibrary	08-10-2018 14:14	File folder
📁	setuptools	Date created: 08-10-2018 14:14 Size: 486 KB Folders: base, keywords, locators, utils Files: __init__, __init__.errors, errors	2018 16:10
📁	setuptools		2018 16:10
📁	six-1.11.0.dist-info		2018 21:18
📁	typing-3.6.6.dist-info		06-10-2018 21:18
📁	urllib3	08-10-2018 14:14	File folder
📁	urllib3-1.23.dist-info	08-10-2018 14:14	File folder
📁	wx	06-10-2018 21:18	File folder
📁	wx-2.8-msw-unicode	06-10-2018 21:36	File folder
📁	wxPython-4.0.3.dist-info	06-10-2018 21:18	File folder
🐍	easy_install	06-10-2018 16:10	Python File
🐍	easy_install	06-10-2018 16:10	Compiled Python ...
📄	README	13-02-2017 22:38	Text Document

Once the installation is done, we have to import the library in Ride as shown in the below steps.

Click on your project on the left side and use Library from Add Import –



Upon clicking Library, a screen will appear wherein you need to enter the library name –



Click OK and the library will get displayed in the settings.

Edit Text Edit Run

BrowserTestCases

Source C:\robotframework\BrowserTestCases.robot

Settings >>

Import	Name / Path	Arguments	Comment	Add Import
Library	SeleniumLibrary			Library

Variable	Value	Comment	Add Scalar
			Add List
			Add Dict

Metadata	Value	Comment	Add Metadata

The name given has to match with the name of the folder installed in site-packages. In case the names do not match, the library name will be in red as shown below –

Edit Text Edit Run

BrowserTestCases

Source C:\robotframework\BrowserTestCases.robot

Settings >>

Import	Name / Path	Arguments	Comment	Add Import
Library	SeleniumLibrary			Library
Library	seleniumlibrary			Resource
				Variables
				Import Failed Help

Variable	Value	Comment	Add Scalar
			Add List
			Add Dict

Metadata	Value	Comment	Add Metadata

Library import in red is as good as the library does not exist inside python. Now, we have completed selenium library import.

Test Case Using Chrome Browser

To work with Chrome browser in Robot, we need to first install the drivers for chrome to work with Selenium. The drives are available on Selenium site – <https://www.seleniumhq.org/>.

<https://www.seleniumhq.org>

What is Selenium?

Selenium automates browsers. That's it! What you do with that power is entirely up to you. Primarily, it is for automating web applications for testing purposes, but is certainly not limited to just that. Boring web-based administration tasks can (and should!) be automated as well.

Selenium has the support of some of the largest browser vendors who have taken (or are taking) steps to make Selenium a native part of their browser. It is also the core technology in countless other browser automation tools, APIs and frameworks.

Which part of Selenium is appropriate for me?

Selenium WebDriver



If you want to

- create robust, browser-based regression automation suites and tests
- scale and distribute scripts across many environments

Then you want to use [Selenium WebDriver](#); a collection of language specific bindings to drive a browser -- the way it is meant to be driven.

Selenium WebDriver is the successor of [Selenium Remote Control](#) which has been officially deprecated. The Selenium Server fuses by both WebDriver and

Selenium IDE



If you want to

- create quick bug reproduction scripts
- create scripts to aid in automation-aided exploratory testing

Then you want to use [Selenium IDE](#); a Firefox add-on that will do simple record-and-playback of interactions with the browser.



Selenium is a suite of tools to automate web browsers across many platforms.

Selenium...

- runs in [many browsers](#) and [operating systems](#)
- can be controlled by many [programming languages](#) and [testing frameworks](#).

Download Selenium

Donate to Selenium
with PayPal

through sponsorship



Click *Download Selenium* as in the above screenshot.

In download section, go to *Third Party Browser Drivers NOT DEVELOPED by seleniumhq* and select Google Chrome driver as shown in highlighted section below

<https://www.seleniumhq.org/download/>

Third Party Drivers, Bindings, and Plugins				
Selenium can be extended through the use of plugins. Here are a number of plugins created and maintained by third parties. For more information on how to create your own plugin or have it listed, consult the docs.				
Please note that these plugins are not supported, maintained, hosted, or endorsed by the Selenium project. In addition, be advised that the plugins listed below are not necessarily licensed under the Apache License v.2.0. Some of the plugins are available under another free and open source software license; others are only available under a proprietary license. Any questions about plugins and their license of distribution need to be raised with their respective developer(s).				
Third Party Browser Drivers NOT DEVELOPED by seleniumhq				
Browser				
Mozilla GeckoDriver	latest	change log	issue tracker	Implementation Status
Google Chrome Driver	latest	change log	issue tracker	selenium wiki page
Opera	latest		issue tracker	selenium wiki page
Microsoft Edge Driver			issue tracker	Implementation Status
GhostDriver		(PhantomJS)	issue tracker	SeConf talk
HtmlUnitDriver	latest		issue tracker	
SafariDriver			issue tracker	
Windows Phone			issue tracker	
Windows Phone	4.14.028.10		issue tracker	Released 2013-11-23
Selendroid - Selenium for Android			issue tracker	
ios-driver			issue tracker	

Here we have a list of the various drivers available for browsers. For Chrome, click [Google Chrome Driver](#) and download the latest driver as per you operating system.

https://sites.google.com/a/chromium.org/chromedriver/

CHROMEDRIVER

CAPABILITIES & CHROMEOPTIONS

CHROME EXTENSIONS

CHROMEDRIVER CANARY

CONTRIBUTING

DOWNLOADS

GETTING STARTED

ANDROID

CHROMEOS

LOGGING

PERFORMANCE LOG

MOBILE EMULATION

NEED HELP?

CHROME DOESN'T START OR CRASHES IMMEDIATELY

CHROMEDRIVER CRASHES

CLICKING ISSUES

DEVTOOLS WINDOW KEEPS CLOSING

OPERATION NOT SUPPORTED WHEN USING REMOTE DEBUGGING

ChromeDriver

WebDriver is an open source tool for automated testing of webapps across many browsers. It provides capabilities for navigating to web pages, user input, JavaScript execution, and more. ChromeDriver is a standalone server which implements WebDriver's wire protocol for Chromium.

We are in the process of implementing and moving to the W3C standard. ChromeDriver is available for Chrome on Android and Chrome on Desktop (Mac, Linux, Windows and ChromeOS).

You can view the current implementation status of the WebDriver standard [here](#).

Latest Release: ChromeDriver 2.42

- All versions available in Downloads

ChromeDriver Documentation

- [Getting started with ChromeDriver on Desktop \(Windows, Mac, Linux\)](#)
 - [ChromeDriver with Android](#)
 - [ChromeDriver with ChromeOS](#)
- [ChromeOptions, the capabilities of ChromeDriver](#)

Click on the latest release. It will display the downloads as per the operating system – windows, linux and mac.

Index of /2.42/

Name	Last modified	Size	ETag
Parent Directory		-	
 chromedriver_linux64.zip	2018-09-13 19:30:37	3.85MB	acfcc29fb03df9e913ef4c360a121ad1
 chromedriver_mac64.zip	2018-09-13 18:14:11	5.75MB	3fc0e4a97cbf2c8c2a9b824d95e25351
 chromedriver_win32.zip	2018-09-13 21:11:33	3.42MB	28d91b31311146250e7ef1afbcd6d026
 notes.txt	2018-09-13 21:23:09	0.02MB	18bdf6fc9f9d8dd668fa444b77d06bdd

Download the version as per your operating system from the above list. It downloads the zip file. Once the file downloads, unzip it and copy the .exe driver file to python folder.

We are copying the file to **C:\Python27\Scripts**.

Name	Date modified	Type	Size
chromedriver	13-09-2018 11:58	Application	6,546 KB
CreateBatchFiles	15-07-2011 21:30	Python File	2 KB
CreateMacScripts	Date created: 08-10-2018 14:49 Size: 6.39 MB	2011 21:30	Python File
easy_install	06-10-2018 16:10	Application	88 KB
easy_install-2.7	06-10-2018 16:10	Application	88 KB
idle	10-07-2000 12:22	PUI	2 MB

Now we are done installing the driver for chrome. We can get started with writing test case that will open browser and close browser.

Go back to ride and enter the keywords for opening the browser.

Ride helps you with keywords to be used with its built-in tool. Enter the command and press **ctrl+spacebar**. You will get all the details of the command as shown below

1 Open

2 Open Browser

3 Open Context Menu

4

5

6

overwrite the default profile Selenium uses. Notice that prior to Sele the library contained its own profile that was used by default.

Examples:

'Open Browser'	http://example.com	Chrome	
'Open Browser'	http://example.com	Firefox	alias=Firefox
'Open Browser'	http://example.com	Edge	remote_url=http://127.0.

It gives the details of the command and also examples on how to use it. In the test case, we will open the site <https://www.tutorialspoint.com/> in chrome and the test case details will be as follows –

1	Open Browser	https://www.tutorialspoint.com/	chrome	
2				
3				
4				
5				
6				

Let us now run this test case to see the output –

Edit | Text Edit | Run | Autosave Pause on failure Show message log

Execution Profile: pybot Report | Log Autosave Pause on failure Show message log

(i) Start (s) Stop (p) Pause (c) Continue (n) Next (s) Step over

Arguments: Only run tests with these tags Skip tests with these tags

elapsed time: 0:00:16 pass: 1 fail: 0

command: pybot.bat --argumentfile c:\users\kenet\appdata\local\temp\RIDEuboo14.d\xargfile.txt --listener C:\Python27\lib\site-packages\robot\reporter\html.py

BrowerTestCases

TC1

BrowserTestCases

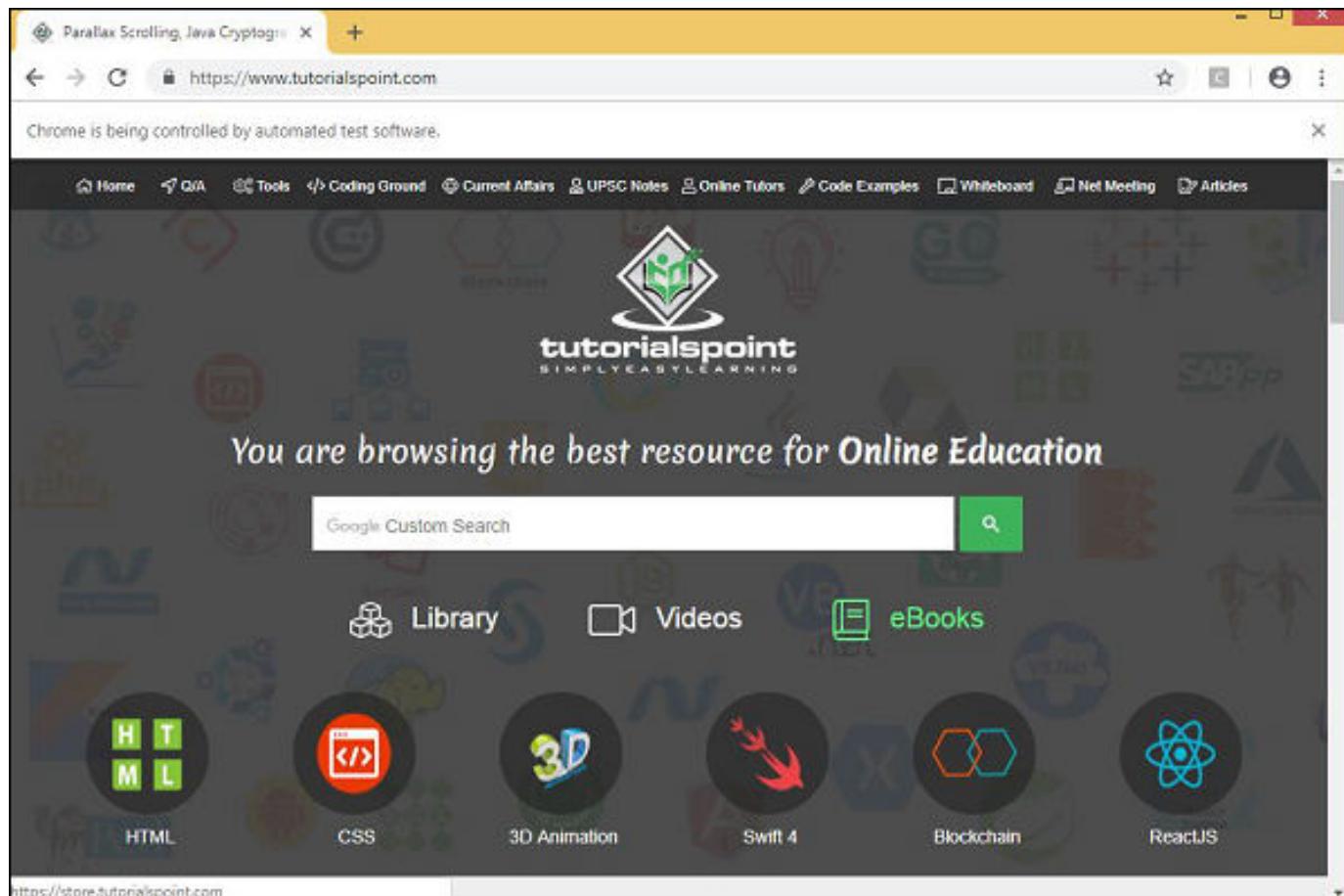
1 critical test. 1 passed. 0 failed
1 test total. 1 passed. 0 failed

Output: c:\users\appdata\local\temp\RIDEuboo14.d\output.xml
Log: c:\users\appdata\local\temp\RIDEuboo14.d\log.html
Report: c:\users\appdata\local\temp\RIDEuboo14.d\report.html

test finished 20181008 14:59:11

< >

Starting test: BrowerTestCases.TC1
20181008 14:58:57.686 : INFO : Opening browser 'chrome' to base url 'https://www.tutorialspoint.com/'.
Ending test: BrowerTestCases.TC1



The test case has passed; we can see the site is opened in chrome browser.

We will add more test cases as follows –

1	Open Browser	https://www.tutorialspoint.com/	chrome			
2	Capture Page Screenshot	page.png				
3	close browser					
4						
5						
6						
7						
8						

- Open Browser – URL – https://www.tutorialspoint.com/ in Chrome browser
- Capture Page Screenshot – name of the image is page.png
- Close browser

Here are the details of the report and log for above test cases executed.

Report

BrowserTestCases Test Report

Generated
2018/08/08 15:22:56 GMT-05:30
4 minutes 14 seconds ago

Summary Information

Status:	All tests passed
Start Time:	2018/08/08 15:22:41.302
End Time:	2018/08/08 15:22:56.177
Elapsed Time:	00:00:14.875
Log File:	log.html

Test Statistics

Total Statistics		Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests		1	1	0	00:00:14	
All Tests		1	1	0	00:00:14	

Statistics by Tag		Total	Pass	Fail	Elapsed	Pass / Fail
No Tags						

Statistics by Suite		Total	Pass	Fail	Elapsed	Pass / Fail
BrowserTestCases		1	1	0	00:00:15	

Test Details

Totals	Tags	Suites	Search
Type:	<input checked="" type="radio"/> Critical Tests	<input type="radio"/> All Tests	

Log

BrowserTestCases Test Log

Generated
20181008 15:22:58 GMT+05:30
9 minutes 12 seconds ago

Test Statistics

Total Statistics		Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests		1	1	0	00:00:14	<div style="width: 100%; background-color: green;"></div>
All Tests		1	1	0	00:00:14	<div style="width: 100%; background-color: green;"></div>
Statistics by Tag		Total	Pass	Fail	Elapsed	Pass / Fail
No Tags						<div style="width: 0%; background-color: lightgray;"></div>
Statistics by Suite		Total	Pass	Fail	Elapsed	Pass / Fail
BrowserTestCases		1	1	0	00:00:15	<div style="width: 100%; background-color: green;"></div>

Test Execution Log

- SUITE BrowserTestCases	00:00:14.875
Full Name:	BrowserTestCases
Source:	C:\robotframework\BrowserTestCases.robot
Start / End / Elapsed:	20181008 15:22:41.302 / 20181008 15:22:56.177 / 00:00:14.875
Status:	1 critical test, 1 passed, 0 failed 1 test total, 1 passed, 0 failed
+ TEST TC1	00:00:14.082

Details of test cases from log

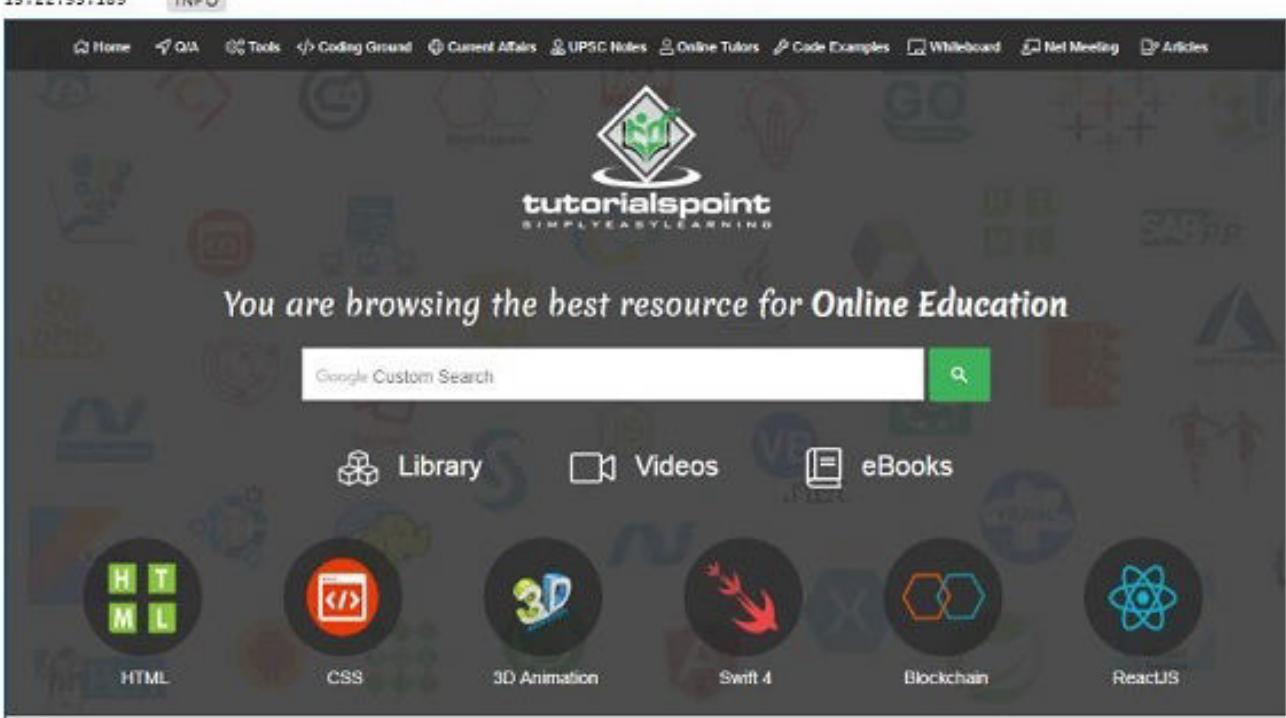
- KEYWORD SeleniumLibrary.Open Browser https://www.tutorialspoint.com/, chrome	00:00:00.790
Documentation:	Opens a new browser instance to the given url.
Start / End / Elapsed:	20181008 15:22:42.095 / 20181008 15:22:52.386 / 00:00:10.291
15:22:42.095	INFO Opening browser 'chrome' to base url 'https://www.tutorialspoint.com/'.

- **KEYWORD** SeleniumLibrary.Capture Page Screenshot page.png 00:00:00.790

Documentation: Takes screenshot of the current page and embeds it into log file.

Start / End / Elapsed: 20181008 15:22:52.389 / 20181008 15:22:53.185 / 00:00:00.796

15:22:53.185 INFO



The screenshot shows the tutorialspoint.com website. The header features the 'tutorialspoint SIMPLY EASY LEARNING' logo. Below the header, a banner reads 'You are browsing the best resource for Online Education'. The main navigation menu includes links like Home, QA, Tools, Coding Ground, Current Affairs, UPSC Notes, Online Tutors, Code Examples, Whiteboard, Net Meeting, and Articles. Below the menu, there are several circular icons representing different subjects: HTML, CSS, 3D Animation, Swift 4, Blockchain, and ReactJS. A search bar with the placeholder 'Google Custom Search' is also visible.

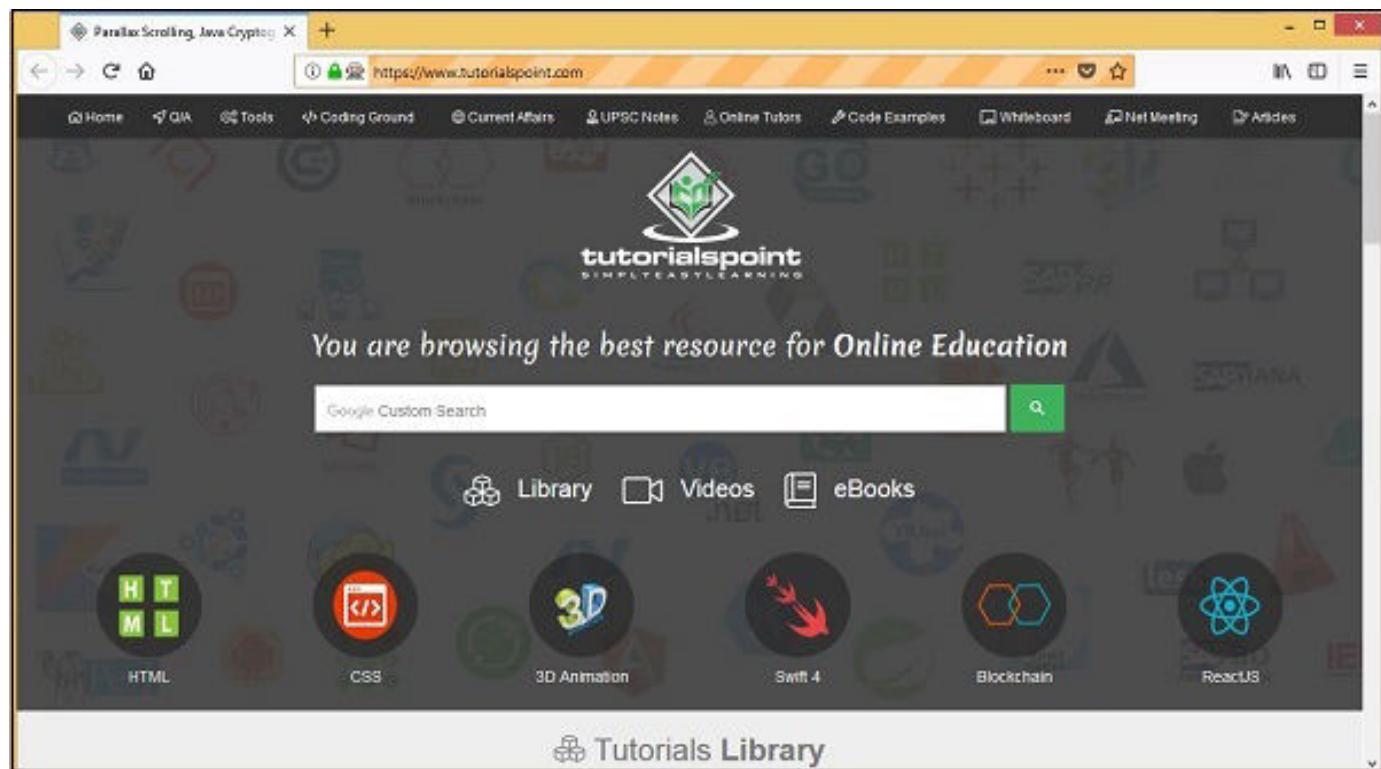
KEYWORD	SeleniumLibrary.Close Browser
Documentation:	Closes the current browser.
Start / End / Elapsed:	20181008 15:22:53.185 / 20181008 15:22:56.177 / 00:00:02.992

Test Case Using Firefox Browser

Install the driver for Firefox and save it in python scripts folder.

Test case for Firefox

1	Open Browser	https://www.tutorialspoint.com/	firefox	
2				
3				
4				
5				
6				



Conclusion

We have seen how to install Selenium library and the browser drivers to work with browsers in Robot framework. Using the selenium library keywords, we can open any given link in the browsers and interact with it. The details of the test-case execution are available in the form of reports and logs, which give the time taken for execution.

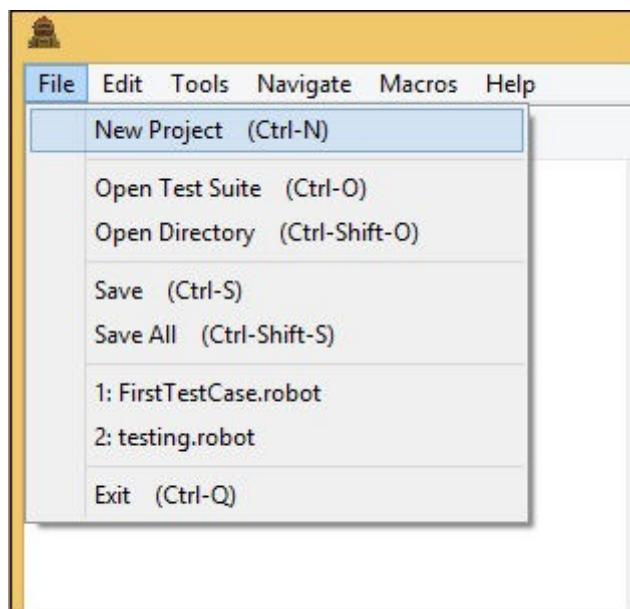
For testing, it becomes important to understand how to interact with the browser and locate the html elements. It is very easy to work with input fields with robot framework. In this chapter, we will learn how to work with textbox using Selenium Library. To work with input field – textbox, we need the locator, which is the main unique identifier for that textbox and it can be id, name, class, etc.

In this chapter, we will discuss the following areas –

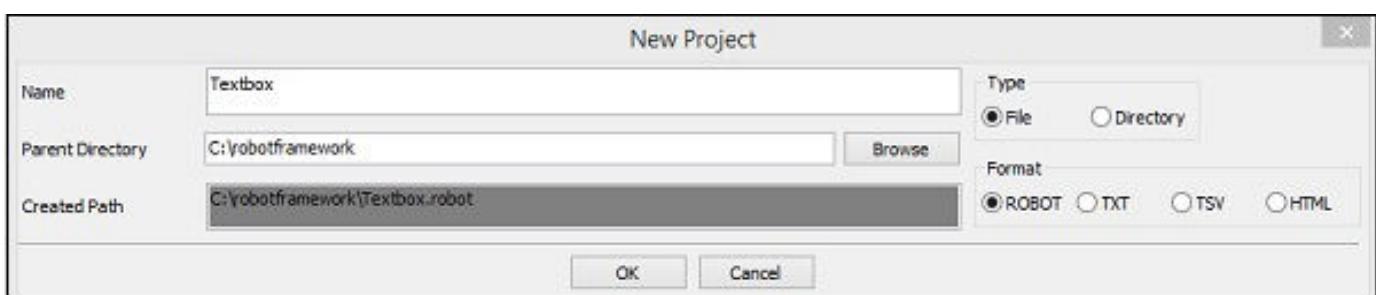
- Project Setup for Textbox Testing
- Enter Data in Search Textbox
- Click on Search Button

Project Setup for Textbox Testing

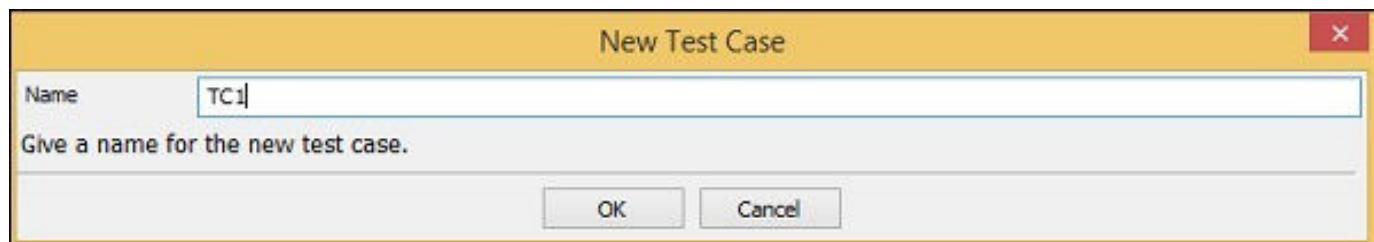
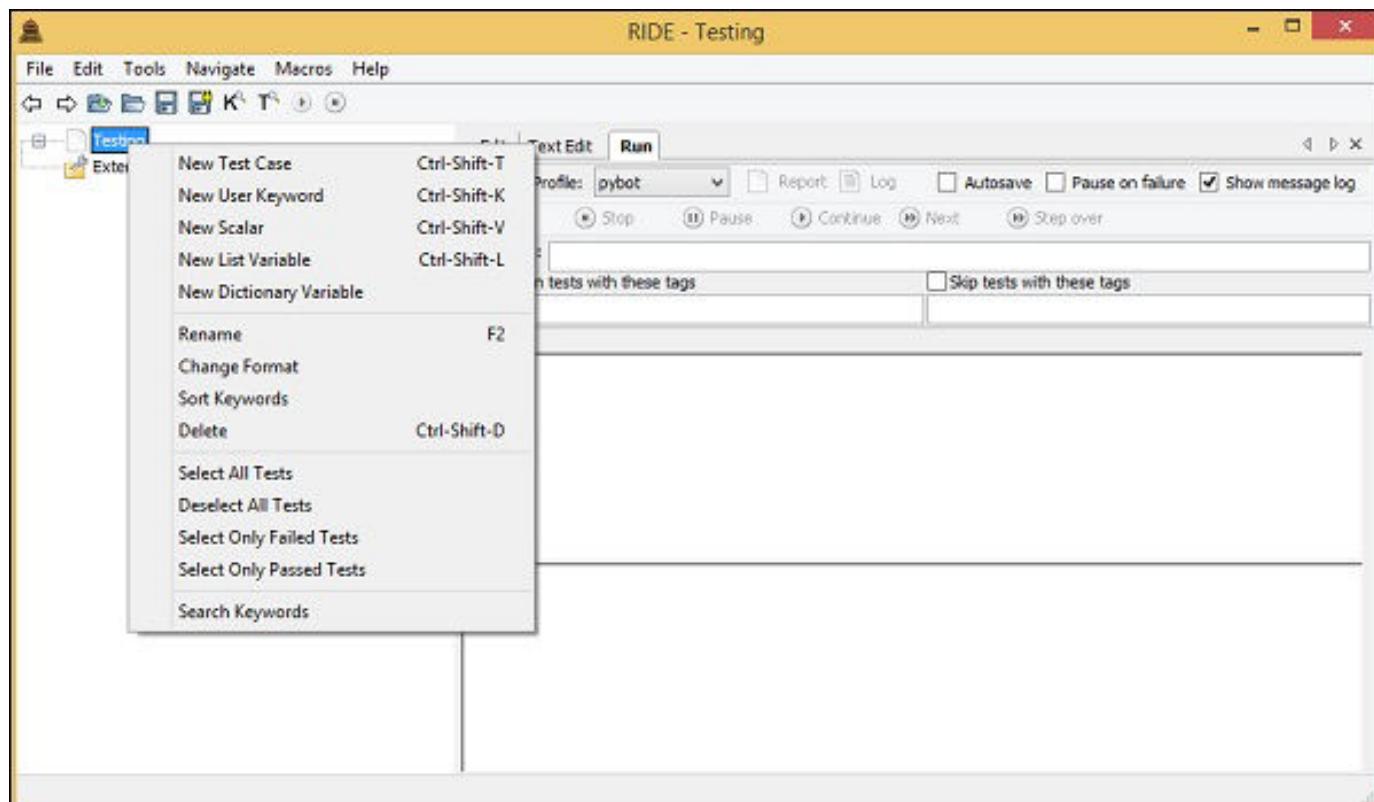
We will first create a project in Ride to work with browsers. Open ride using ride.py from the command line.



Click *New Project* and enter *Name* of your project as shown below.



The name given for the project is *Textbox*. Click OK to save the project. Right-click on the name of the project created and click on *New Test Case* –



Name your test case and click OK to save it. We are now done with the project setup. Further, we will write test cases for the textbox. Since we need Selenium library, we need to import the same in our project.

Click on your project on the left side and use Library from Add Import.

Textbox

Source C:\robotframework\Textbox.robot

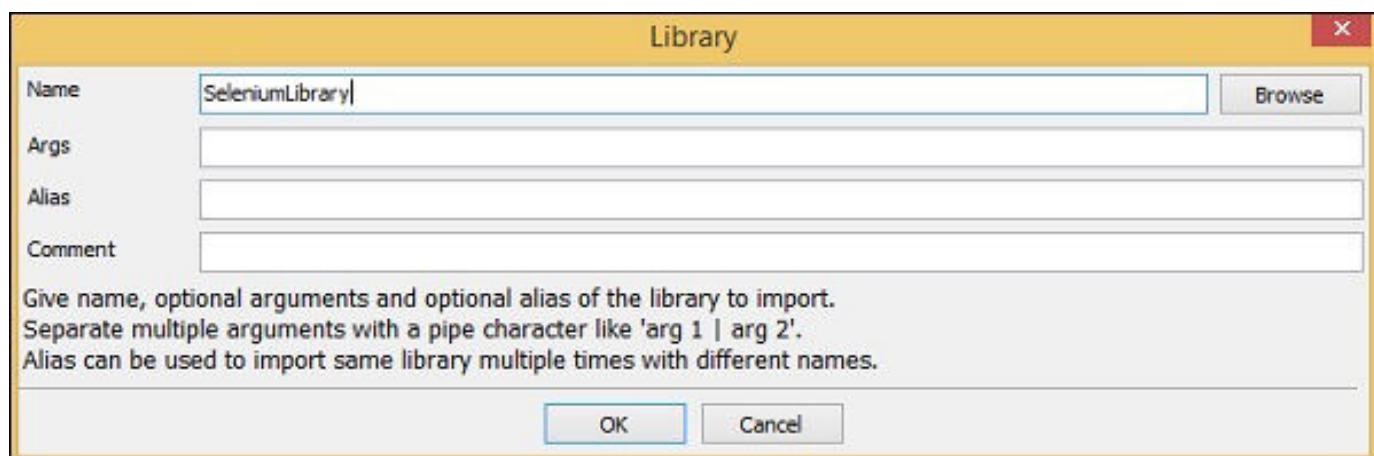
Settings >>

Import	Name / Path	Arguments	Comment	Add Import
				Library Resource Variables Import Failed Help

Variable	Value	Comment	Add Scalar Add List Add Dict

Metadata	Value	Comment	Add Metadata

Upon clicking Library, a screen will appear where you need to enter the library name –



Click OK and the library will get displayed in the settings.

Edit Text Edit Run

BrowserTestCases

Source C:\robotframework\BrowserTestCases.robot

Settings >>

Import	Name / Path	Arguments	Comment	Add Import
Library	SeleniumLibrary			Library Resource Variables Import Failed Help Add Scalar Add List Add Dict

Variable	Value	Comment	Add Metadata

Metadata	Value	Comment	Add Metadata

The name given has to match with the name of the folder installed in site-packages.

In case the names do not match, the library name will show in red as in the following screenshot

-

The screenshot shows the 'Edit' tab selected in the top menu bar. The source file is set to 'C:\robotframework\BrowserTestCases.robot'. A 'Settings >>' button is visible. On the right, there's a context menu with the following options:

- Add Import
- Library (selected)
- Resource
- Variables
- Import Failed Help
- Add Scalar
- Add List
- Add Dict

Below the context menu, there are three tables:

Import	Name / Path	Arguments	Comment	
Library	SeleniumLibrary			
Library	seleniumlibrary			

Variable	Value	Comment	

Metadata	Value	Comment	

Enter Data in Textbox

We are now going to write test cases. The test case details will be as follows –

- Open browser – URL – <https://www.tutorialspoint.com/> in Chrome
- Enter data in the search textbox in <https://www.tutorialspoint.com/>
- Click Search

To work with textbox, we need a locator. A locator is the identifier for the textbox like id, name, class, etc. For example, if you are using the –

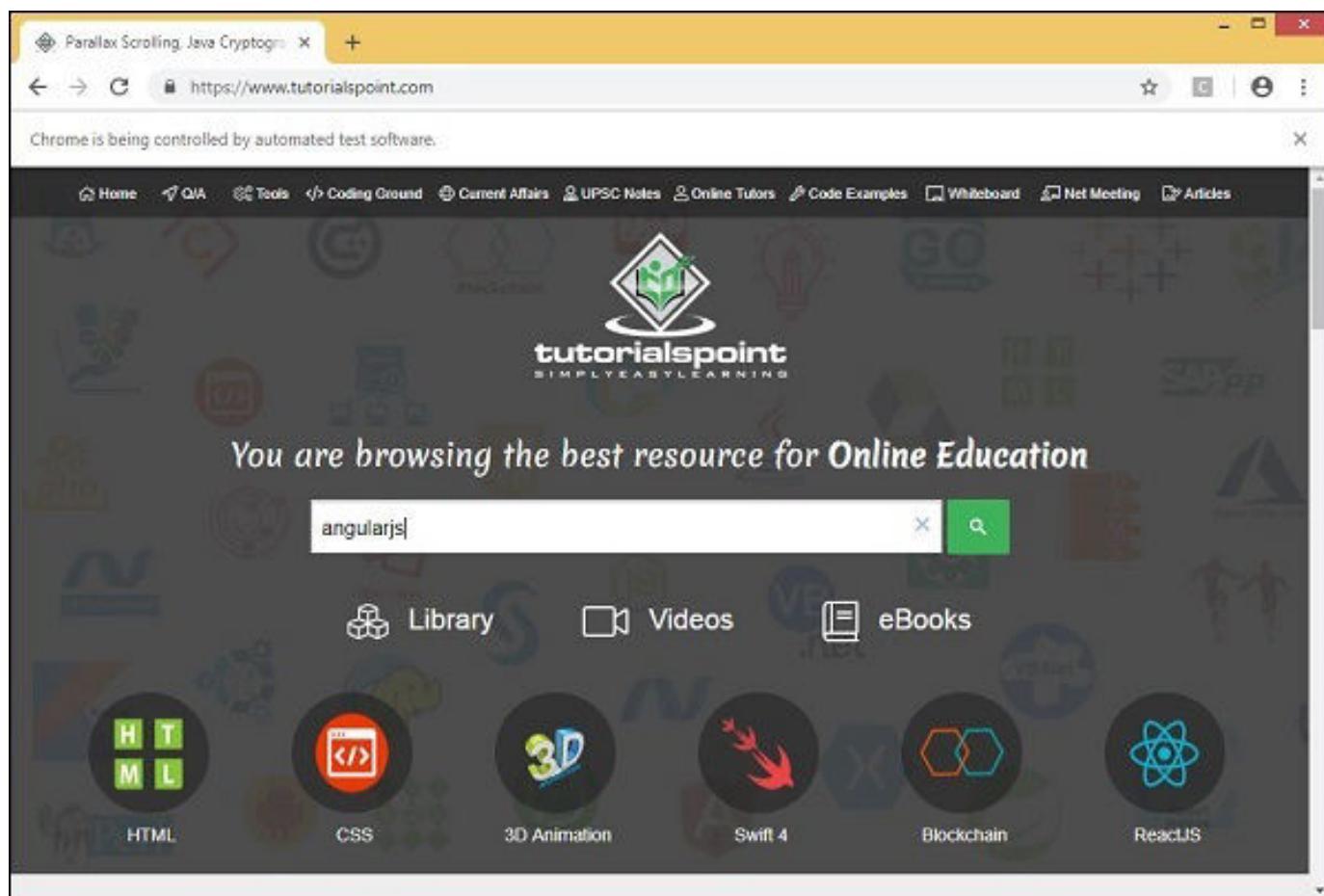
- **name** attribute of the textbox, it has to be name – Nameofthetextbox or name=Nameofthetextbox
- **id** of the textbox, it will be id:idoftextbox or id=idoftextbox
- **class** of the textbox, it will be class – classfortextbox or class=classfortextbox

Now, we will add the details of the test case for textbox in ride. Here are the keywords entered for textbox test case –

1	Open Browser	https://www.tutorialspoint.com/	chrome	
2	Input Text	name:search	angularjs	
3	Click Button	class:gsc-search-button-v2		
4				
5				

- **Open Browser** – The keyword opens the browser for the given URL and the browser specified.
- **Input Text** – This keyword works on the input type and will look for the locator name:search on the site https://www.tutorialspoint.com/ and angularjs is the value we want to type in the textbox.
- **Click button** is used to click on the button with location class:gsc-search-button-v2.

We will now execute the same –



Upon clicking the Search icon, a screen will appear as shown in the following screenshot –

The screenshot shows a web browser window with the URL <https://www.tutorialspoint.com>. The page displays search results for "AngularJS". The results include links to "GDG DevFest Season 2018 - Learn More About DevFest", "AngularJS Tutorial - Design web using AngularJS", "Best Angular Mobile Training - Full Video Courses", and "Simple scripting languages - Simple Scripting Languages". Each result includes a snippet of text and a "Visit Website" button.

Let us now see the reports and the log details –

Report

Textbox Test Report Generated
2018/08/08 18:54:16 GMT+05:30
52 seconds ago

Summary Information

Status:	All tests passed
Start Time:	2018/08/08 18:54:08.143
End Time:	2018/08/08 18:54:16.970
Elapsed Time:	00:00:08.827
Log File:	log.html

Test Statistics

	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:09	<div style="width: 100%; background-color: green;"></div>
All Tests	1	1	0	00:00:09	<div style="width: 100%; background-color: green;"></div>

	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

	Total	Pass	Fail	Elapsed	Pass / Fail
Textbox	1	1	0	00:00:09	<div style="width: 100%; background-color: green;"></div>

Test Details

Totals Tags Suites Search

Type: Critical Tests All Tests

Log

Textbox Test Log

REPORT Generated 20181008 18:54:16 GMT+05:30 1 minute 21 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:09	Success
All Tests	1	1	0	00:00:09	Success

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Textbox	1	1	0	00:00:09	Success

Test Execution Log

- SUITE Textbox	00:00:08.827
Full Name: Textbox	
Source: C:\robotframework\Textbox.robot	
Start / End / Elapsed: 20181008 18:54:08.143 / 20181008 18:54:16.970 / 00:00:08.827	
Status: 1 critical test, 1 passed, 0 failed 1 test total, 1 passed, 0 failed	
+ TEST TC1	00:00:08.500
Full Name: Textbox.TC1	
Start / End / Elapsed: 20181008 18:54:08.468 / 20181008 18:54:16.968 / 00:00:08.500	
Status: PASS (critical)	
- KEYWORD SeleniumLibrary.Open Browser https://www.tutorialspoint.com/, chrome	00:00:07.499
Documentation: Opens a new browser instance to the given url.	
Start / End / Elapsed: 20181008 18:54:08.483 / 20181008 18:54:15.982 / 00:00:07.499	
18:54:08.483 [INFO] Opening browser 'chrome' to base url 'https://www.tutorialspoint.com/'.	
- KEYWORD SeleniumLibrary.Input Text name:search, angularjs	00:00:00.217
Documentation: Types the given text into text field identified by locator.	
Start / End / Elapsed: 20181008 18:54:15.982 / 20181008 18:54:16.199 / 00:00:00.217	
18:54:15.983 [INFO] Typing text 'angularjs' into text field 'name:search'.	
- KEYWORD SeleniumLibrary.Click Button class:gsc-search-button-v2	00:00:00.768
Documentation: Clicks button identified by locator.	
Start / End / Elapsed: 20181008 18:54:16.200 / 20181008 18:54:16.968 / 00:00:00.768	
18:54:16.200 [INFO] Clicking button 'class:gsc-search-button-v2'.	

Conclusion

We have seen how to interact with the textbox using selenium library in robot framework. Using the keywords available with robot framework and the library imported we can locate the textbox and enter data and test the same.

Robot Framework - Working With Radio Button

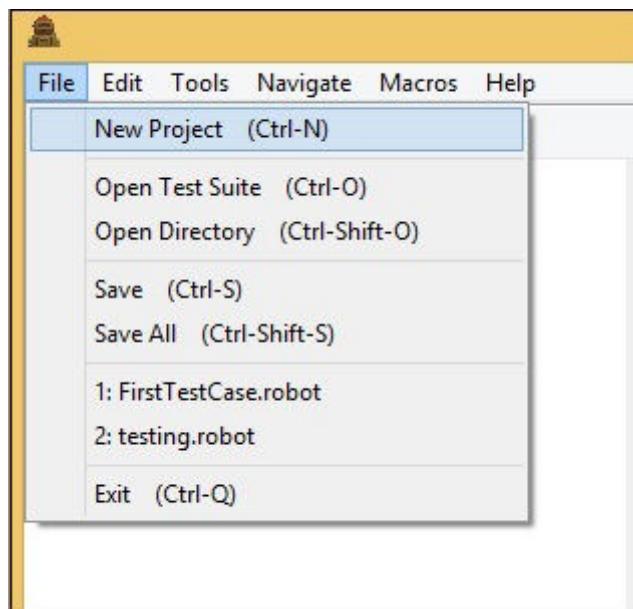
For testing, it becomes important to understand how to interact with the browser and locate the html elements. It is very easy to work with input fields with robot framework. In this chapter, we will learn how to work with radio button using Selenium Library. To work with radio button, we need the locator – the main unique identifier for that radio button.

We are going to discuss the following over here –

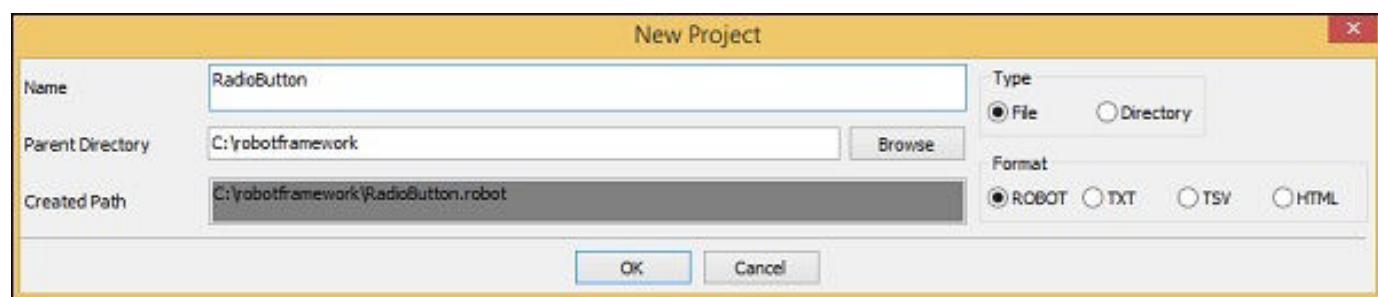
- Project Setup for Radio Button Testing
- Test case for Radio Button

Project Setup For Radio Button Testing

We will first create a project in Ride to work with browsers. Open ride using ride.py from the command line.

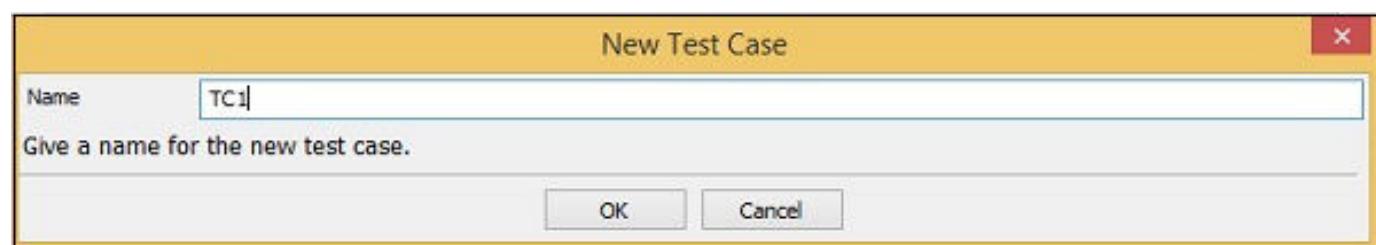
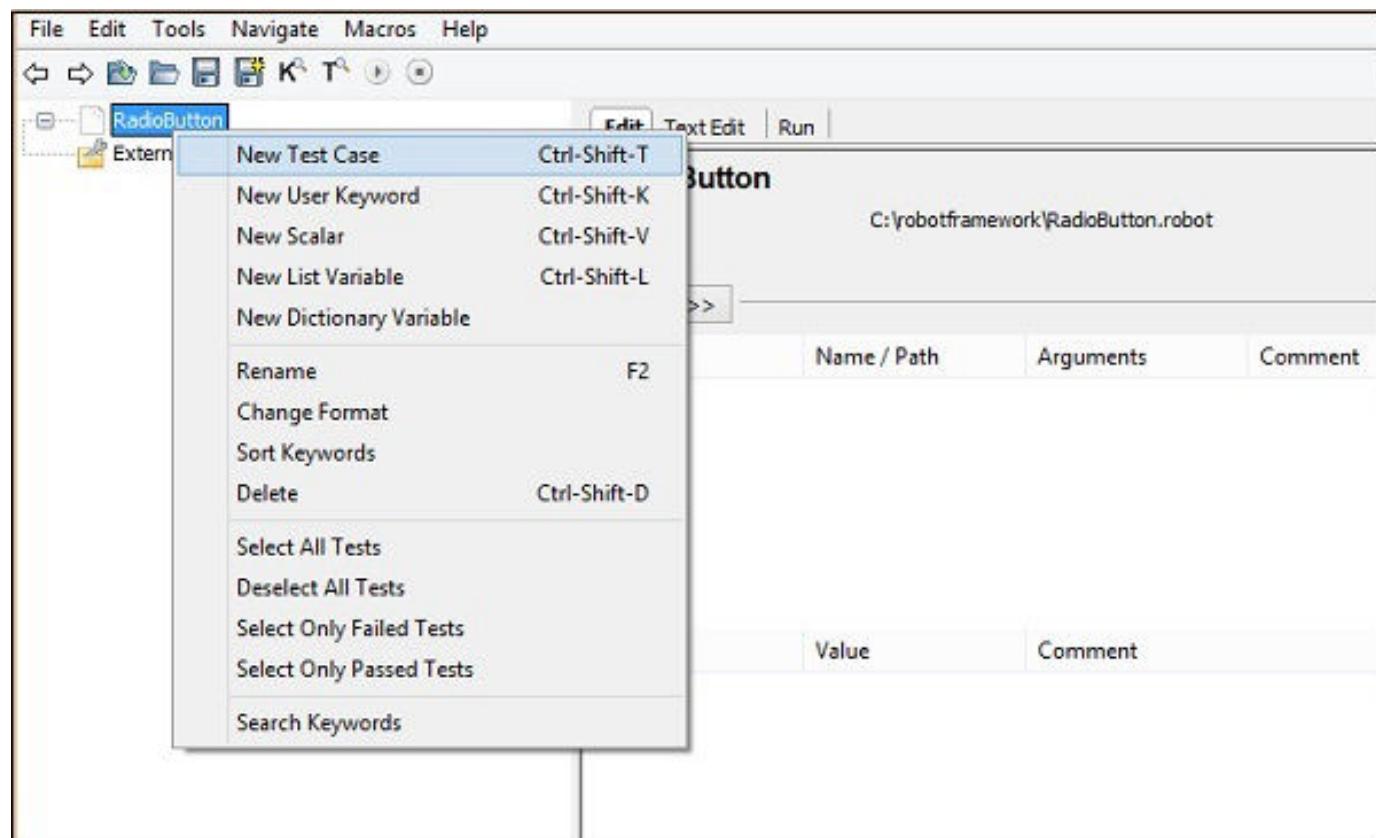


Click *New Project* and enter *Name* of your project as shown in the screenshot below.



The name given is RadioButton. Click on OK button to save the project.

Right-click on the name of the project created and click on *New Test Case* –



Give name to the test case and click OK to save it. We are done with the project setup and now will write test cases for the radio button. Since we need Selenium library, we need to import the same in our project.

Click on your project on the left side and use *Library* from *Add Import*.

Edit Text Edit Run

RadioButton

Source C:\robotframework\RadioButton.robot

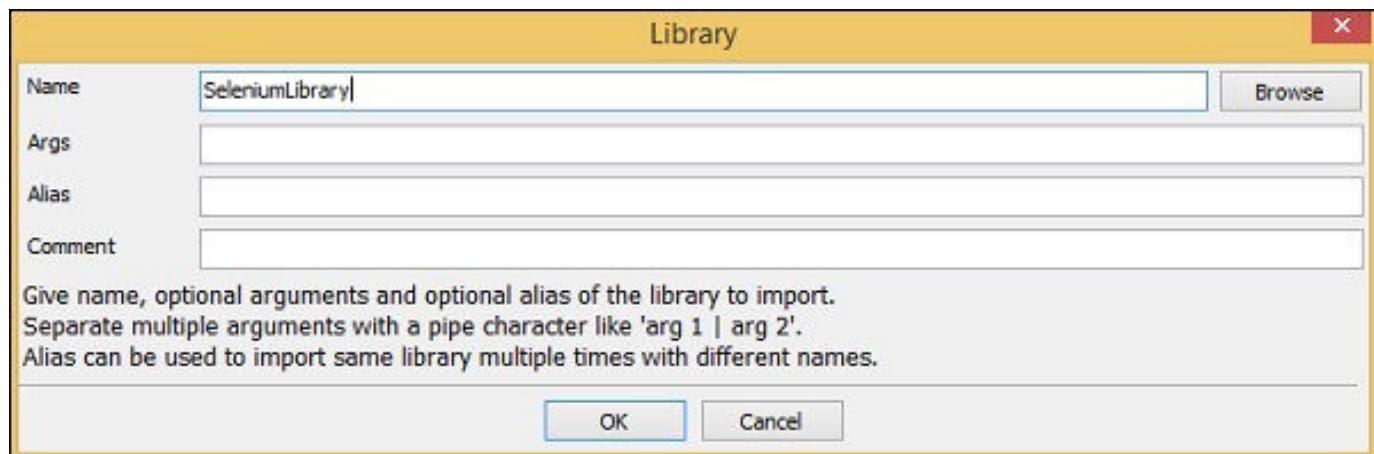
Settings >>

Import	Name / Path	Arguments	Comment	Add Import
				Library Resource Variables Import Failed Help

Variable	Value	Comment	Add Scalar Add List Add Dict

Metadata	Value	Comment	Add Metadata

Upon clicking Library, a screen will appear where you need to enter the library name –



Click OK and the library will be displayed in the settings.

Edit Text Edit Run

RadioButton

Source C:\robotframework\RadioButton.robot

Settings >>

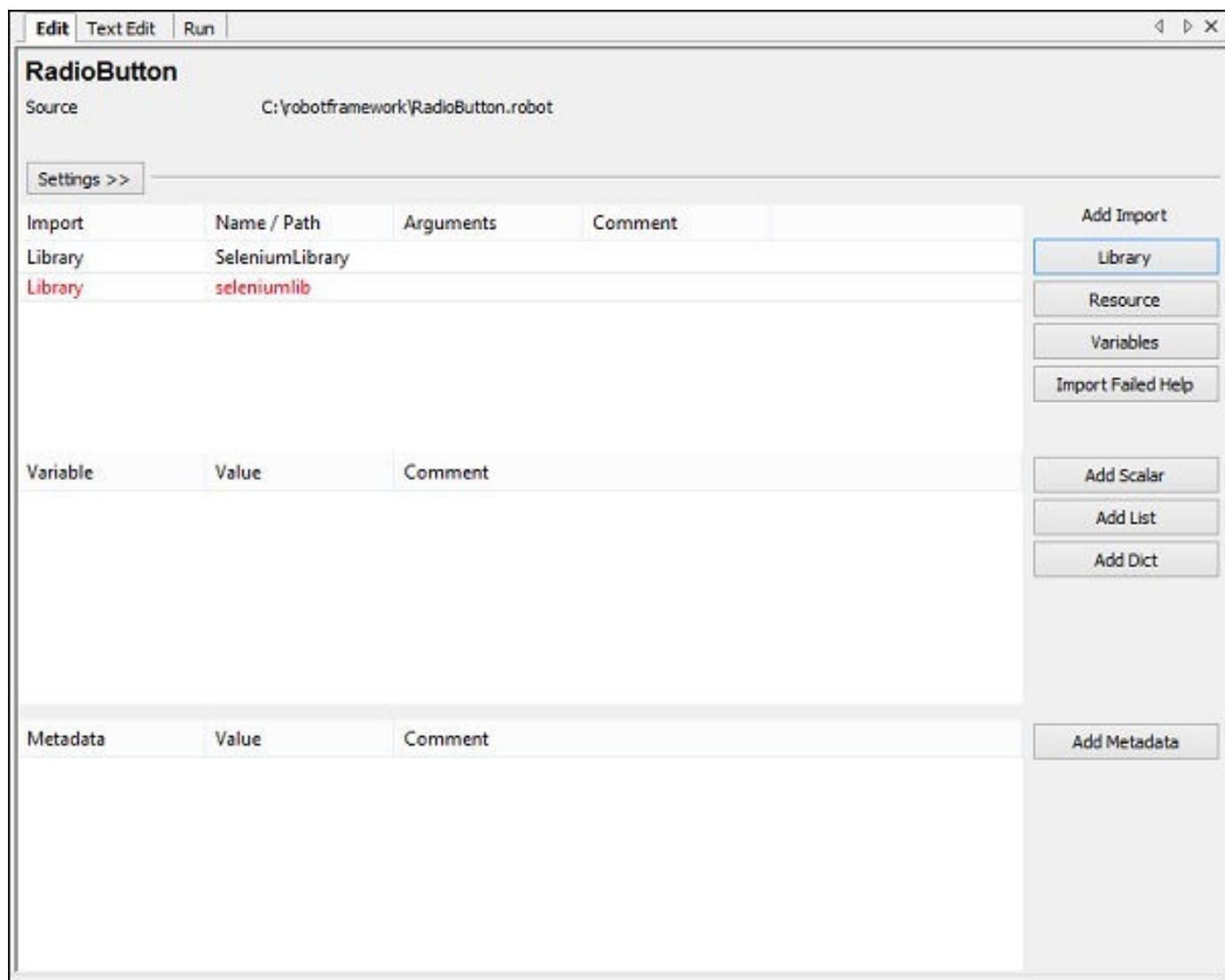
Import	Name / Path	Arguments	Comment	Add Import
Library	SeleniumLibrary			Library

Variable	Value	Comment	Add Scalar
			Add List
			Add Dict

Metadata	Value	Comment	Add Metadata

Import Failed Help

The name given has to match with the name of the folder installed in site-packages. If the name does not match, it will be in red as shown below –



Test Case for Radio Button

The radio button test case will select a radio button, with the help of a locator.

Consider the following html display for radio button –

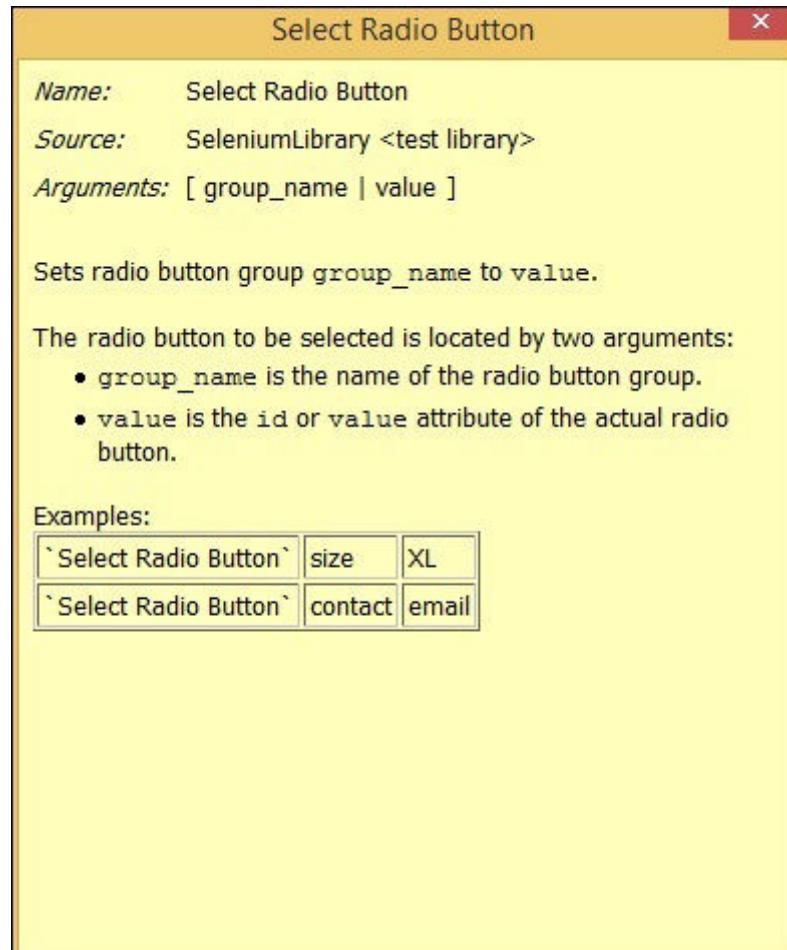
```
<input type="radio" name="gender" value="male" checked> Male<br/>
<input type="radio" name="gender" value="female"> Female<br/>
```

For radio button, *name* is the locator. In the above example, the *name is gender*. We also need the value so that we can select the radio button we want. The values in the above example are *Male and Female*.

Now, we will create a test-page with radio button and open the same in the browser. Now, select the value of the radio button. The test case details will be as follows –

- Open browser – URL – <http://localhost/robotframework/radiobutton.html> in chrome
- Enter details of radio button
- Execute the test case

While writing the keyword for test cases, press Ctrl + Spacebar. You will get the details of the command. Details of Radio button



For the radio button, the arguments are group name and value. Here are the details of the test case for Radio button selection –

1	Open Browser	http://localhost/robotframework/radiobutton.html	
2	Select Radio Button	gender	Female
3			
4			
5			
6			
7			

Following is the Test Page for radio button –

Html code for Radiobutton.html

```
<html>
<head>
<title>Radio Button</title>
```

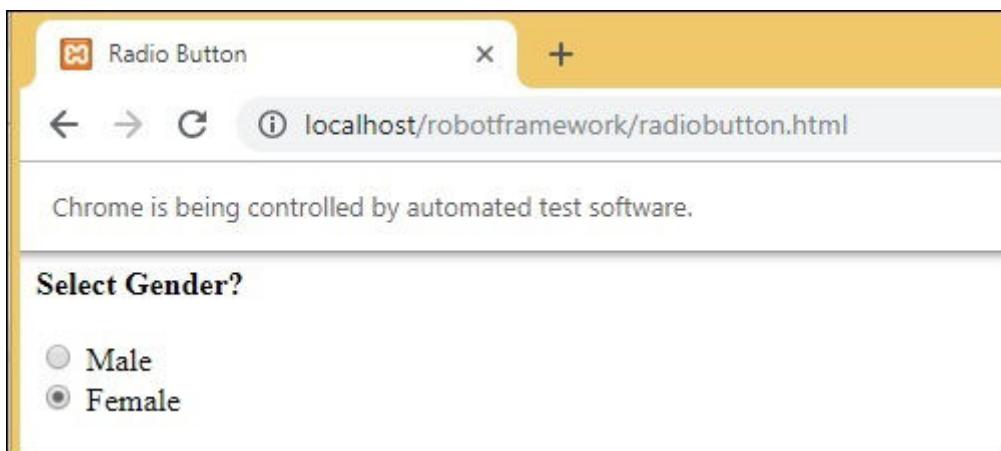
```

</head>
<body>
    <form name="myform" method="POST">
        <b>Select Gender?</b>
        <div><br/>
            <input type="radio" name="gender" value="male" checked> Male<br/>
            <input type="radio" name="gender" value="female"> Female<br/>
        </div>
    </form>
</body>
</html>

```

In the above form, we are planning to select female, which is a radio button. The name and value are given in the test case. Now, we will execute the test case and check the selection of the checkbox for the above form.

Let us execute the test case and see the display in the browser –



When the test case is executed, it opens the URL **http://localhost/robotframework/radiobutton.html** and selects the Female radio button whose name and value we have given in the test case.

Here are the execution details in Ride –

```

Edit Text Edit Run
Execution Profile: pybot Report Log Autosave Pause on failure Show message log
Start Stop Pause Continue Next Step over
Arguments:
Only run tests with these tags Skip tests with these tags
elapsed time: 0:00:08 pass: 0 fail: 0
command: pybot.bat --argumentfile c:\users\kamat\appdata\local\temp\RIDEiruis.d\argfile.txt --listener C:
-----
RadioButton
-----
TC1 | PASS |
-----
RadioButton
1 critical test. 1 passed. 0 failed
1 test total. 1 passed. 0 failed
-----
Output: c:\users\appdata\local\temp\RIDEiruis.d\output.xml
Log: c:\users\appdata\local\temp\RIDEiruis.d\log.html
Report: c:\users\appdata\local\temp\RIDEiruis.d\report.html
test finished 20181014 10:37:14

```

Let us now look at Report and Log for more details.

Report Details

RadioButton Test Report

Generated
20181014 10:37:13 GMT+05:30
1 minute 50 seconds ago

Summary Information

Status:	All tests passed
Start Time:	20181014 10:37:06.590
End Time:	20181014 10:37:13.950
Elapsed Time:	00:00:07.360
Log File:	log.html

Test Statistics

Total Statistics		Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests		1	1	0	00:00:06	
All Tests		1	1	0	00:00:06	

Statistics by Tag		Total	Pass	Fail	Elapsed	Pass / Fail
No Tags						

Statistics by Suite		Total	Pass	Fail	Elapsed	Pass / Fail
RadioButton		1	1	0	00:00:07	

Test Details

Totals Tags Suites Search

Type: Critical Tests
 All Tests

Log Details

RadioButton Test Log

Generated
20181014 10:37:13 GMT+05:30
2 minutes 7 seconds ago

Test Statistics

Total Statistics		Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests		1	1	0	00:00:06	
All Tests		1	1	0	00:00:06	

Statistics by Tag		Total	Pass	Fail	Elapsed	Pass / Fail
No Tags						

Statistics by Suite		Total	Pass	Fail	Elapsed	Pass / Fail
RadioButton		1	1	0	00:00:07	

Test Execution Log

- SUITE RadioButton

Full Name:	RadioButton
Source:	C:\robotframework\RadioButton.robot
Start / End / Elapsed:	20181014 10:37:06.590 / 20181014 10:37:13.950 / 00:00:07.360
Status:	1 critical test, 1 passed, 0 failed 1 test total, 1 passed, 0 failed

+ TEST TC1

Details of test cases

Test Execution Log

-	SUITE	RadioButton
Full Name:	RadioButton	
Source:	C:\robotframework\RadioButton.robot	
Start / End / Elapsed:	20181014 10:37:06.590 / 20181014 10:37:13.950 / 00:00:07.360	
Status:	1 critical test, 1 passed, 0 failed	
	1 test total, 1 passed, 0 failed	
-	TEST	TC1
Full Name:	RadioButton.TC1	
Start / End / Elapsed:	20181014 10:37:07.456 / 20181014 10:37:13.946 / 00:00:06.490	
Status:	PASS (critical)	
-	KEYWORD	SeleniumLibrary.Open Browser http://localhost/robotframework/radiobutton.html, chrome
Documentation:	Opens a new browser instance to the given url.	
Start / End / Elapsed:	20181014 10:37:07.459 / 20181014 10:37:13.498 / 00:00:06.039 10:37:07.461 INFO opening browser 'chrome' to base url ' http://localhost/robotframework/radiobutton.html '.	
-	KEYWORD	SeleniumLibrary.Select Radio Button gender, female
Documentation:	Sets radio button group group_name to value.	
Start / End / Elapsed:	20181014 10:37:13.500 / 20181014 10:37:13.944 / 00:00:00.444 10:37:13.502 INFO selecting 'female' from radio button 'gender'.	

Conclusion

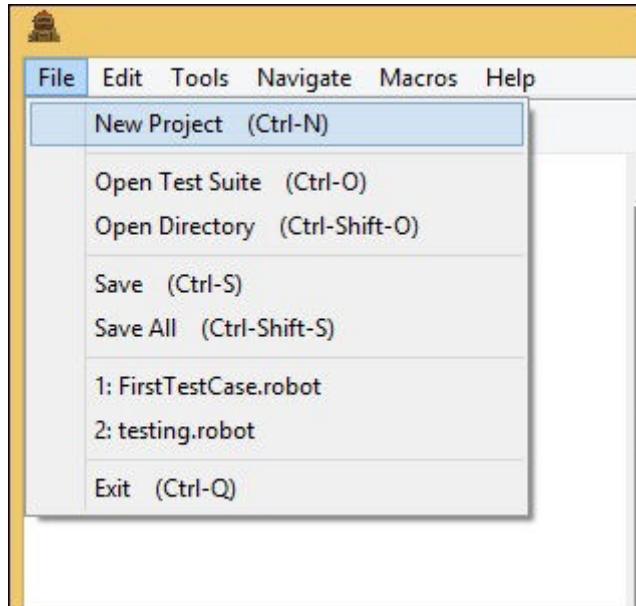
We have seen how to select value of radio button by giving the group name of the radio button to the test case. Using the keywords available with robot framework and the library imported, we can locate the radio button and select the value of the radio button. We do get the details of the test-case executed using robot framework logs and report.

Robot Framework - Working With Checkbox

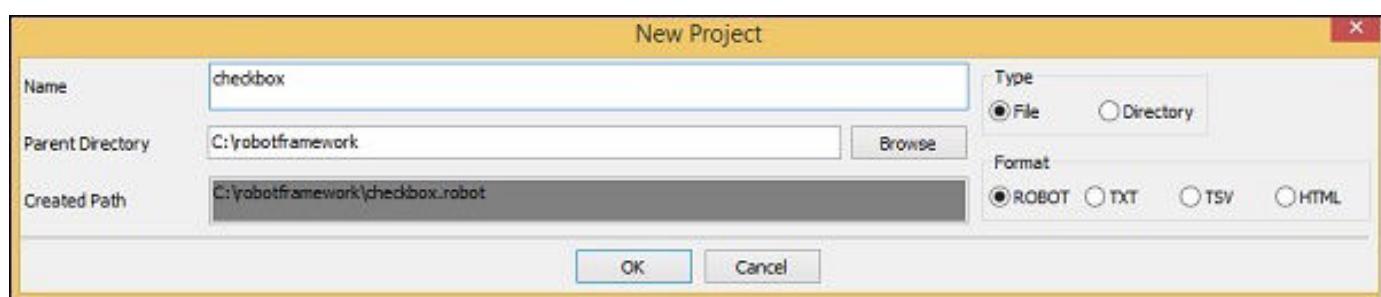
For testing, it becomes important to understand how to interact with the browser and locate the html elements. It is very easy to work with input fields with robot framework. In this chapter, we will learn how to work with checkbox using Selenium Library. To work with checkbox, we need the locator, which is the main unique identifier for that checkbox. The locator can be id, name, class, etc.

Project Setup for Checkbox Testing

We will first create a project in Ride to work with browsers. Open ride using **ride.py** from the command line.

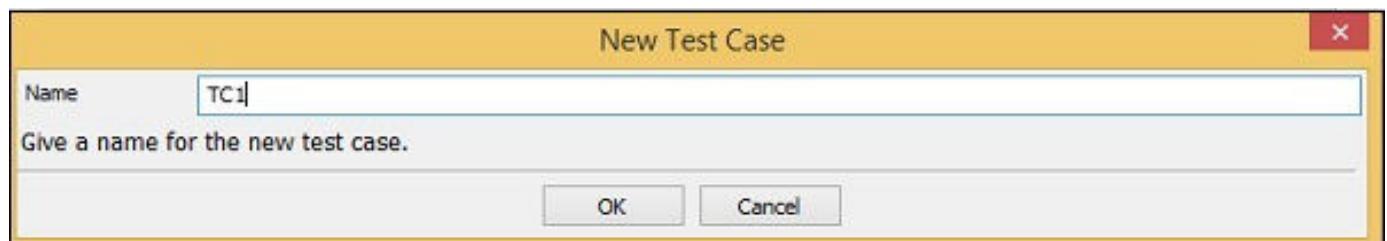
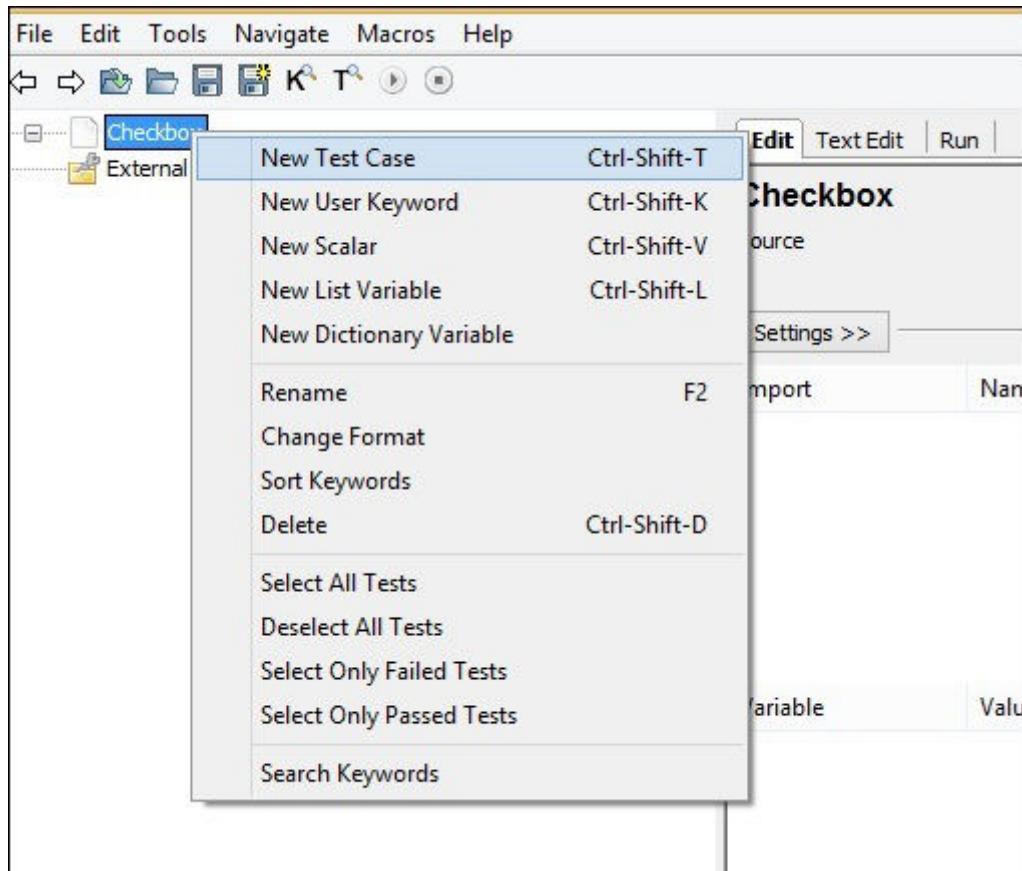


Click on *New Project* and enter *Name* of your project as shown in the screenshot below.



The name given for the project is Checkbox. Click OK to save the project.

Right-click on the name of the project created and click *New Test Case* –



Give name to the test case and click OK. We are done with the project setup. Now we will write test cases for checkbox. Since we need Selenium library, we need to import the same in our project.

Click on your project on the left side and use *Library* from *Add Import*.

Edit Text Edit Run

Checkbox

Source C:\robotframework\checkbox.robot

Settings >>

Import	Name / Path	Arguments	Comment	Add Import
--------	-------------	-----------	---------	------------

Variable	Value	Comment	Add Scalar
----------	-------	---------	------------

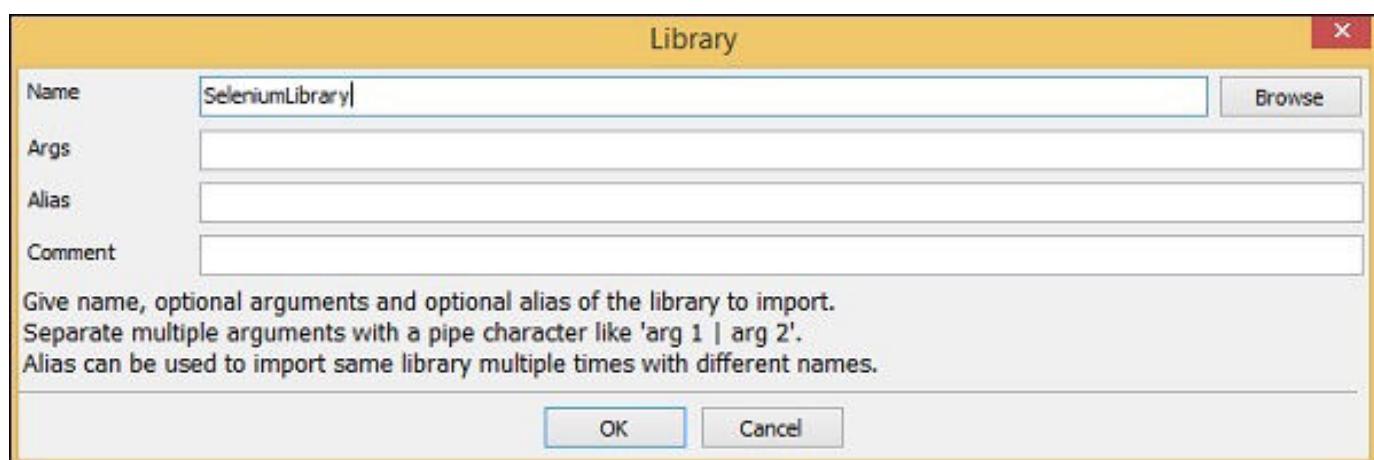
Metadata	Value	Comment	Add List
----------	-------	---------	----------

Metadata	Value	Comment	Add Dict
----------	-------	---------	----------

Metadata	Value	Comment	Add Metadata
----------	-------	---------	--------------

Library
Resource
Variables
Import Failed Help

Now, click Library. A screen will appear where you need to enter the library name –



Click OK and the library will get displayed in the settings.

Edit Text Edit Run

Checkbox

Source C:\robotframework\checkbox.robot

Settings >>

Import Library	Name / Path SeleniumLibrary	Arguments	Comment	Add Import
----------------	-----------------------------	-----------	---------	------------

Variable	Value	Comment	Add Scalar
----------	-------	---------	------------

Metadata	Value	Comment	Add List
----------	-------	---------	----------

Metadata	Value	Comment	Add Dict
----------	-------	---------	----------

Metadata	Value	Comment	Add Metadata
----------	-------	---------	--------------

Library

Resource

Variables

Import Failed Help

Add Scalar

Add List

Add Dict

Add Metadata

The screenshot shows the Robot Framework Test Case Editor interface. At the top, there are tabs for 'Edit', 'Text Edit', and 'Run'. Below the tabs, the title 'Checkbox' is displayed. Underneath the title, it says 'Source C:\robotframework\checkbox.robot'. A 'Settings >>' button is present. The main area contains three tables: 'Imports', 'Variables', and 'Metadata'. The 'Imports' table has one row with 'Import Library' and 'Name / Path SeleniumLibrary'. The 'Variables' and 'Metadata' tables are currently empty. To the right of the tables is a vertical sidebar with buttons for adding different types of imports and metadata. The 'Library' button is highlighted.

The name given has to match with the name of the folder installed in site-packages. If the names do not match, the library name will show in red –

The screenshot shows the Robot Framework Test Case Editor interface. At the top, there's a menu bar with 'Edit', 'TextEdit', and 'Run'. Below the menu is a title 'Checkbox' and a 'Source' field containing 'C:\robotframework\checkbox.robot'. A 'Settings >>' button is located next to the source path. On the right side, there's a vertical toolbar with buttons for 'Add Import' (highlighted in blue), 'Library', 'Resource', 'Variables', and 'Import Failed Help'. Below this are buttons for 'Add Scalar', 'Add List', and 'Add Dict'. At the bottom of the toolbar is a 'Add Metadata' button. The main area contains three tables: 'Import', 'Variable', and 'Metadata', each with columns for 'Name / Path', 'Arguments', 'Comment', and 'Value'. In the 'Import' table, there are two entries: 'Library' with value 'SeleniumLibrary' and 'Library' with value 'seleniumlibrary' (in red). The 'Variable' and 'Metadata' tables are currently empty.

Test Case for Checkbox

In the test case, we will select the checkbox. To select the checkbox, we need the identifier locator.

Now consider the following html display for checkbox –

```
<input type="checkbox" name="option1" value="Car"> Car
```

For checkbox, we have the *name* as the locator. In the above example, the *name* is *option1*. We also need the value so that we can select the same. **Car** holds the value in the above example.

Now, we will create a test page with checkbox. Open the checkbox in the browser and select the value.

The test case details will be as follows –

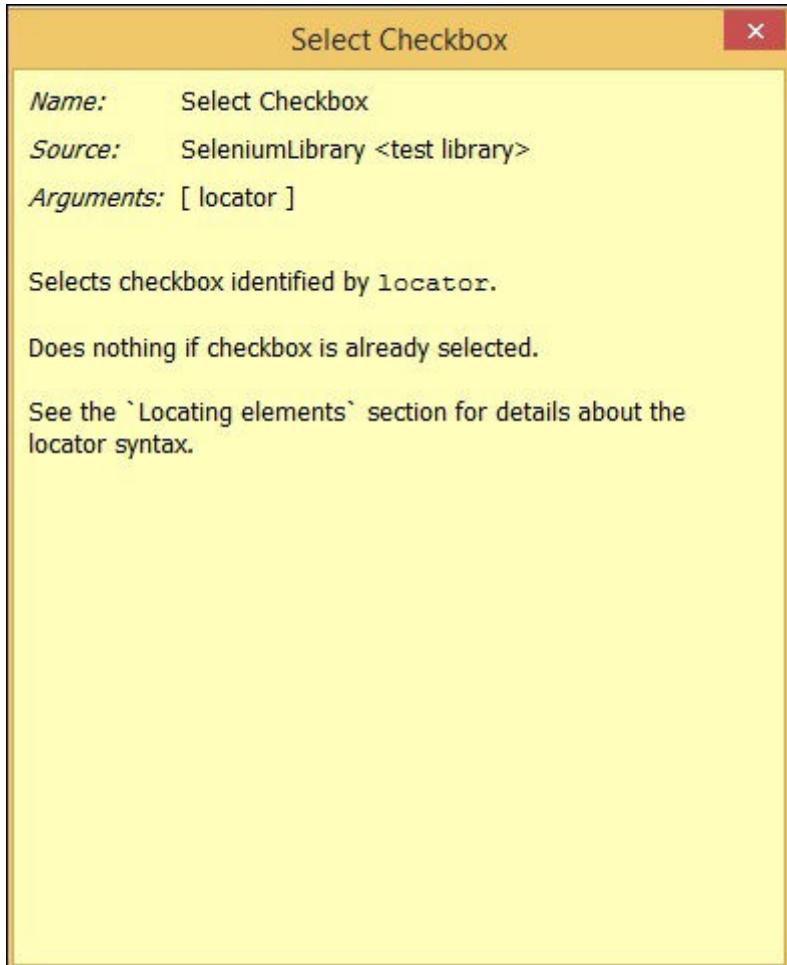
- Open browser – URL – <http://localhost/robotframework/checkbox.html> in Chrome
- Enter details of checkbox.
- Execute the test case.

While writing the keyword for test cases, press Ctrl + Spacebar. It gives all the details of the command. Details of checkbox.

The keywords to be used for checkbox is –

```
Select checkbox name:nameofcheckbox value
```

The command details from ride is as follows –



So, arguments is the locator for the checkbox. Here are the details of the test case for Checkbox selection –

1	Open Browser	http://localhost/robotframework/d	chrome		
2	Select Checkbox	name:option1			
3					
4					
5					
6					
7					

This is how the URL is –

The screenshot shows a web browser window with the URL 'localhost/robotframework/checkbox.html'. The page title is 'How would you like to travel?'. Below the title is a list of four options, each preceded by a checkbox:

- Car
- Bus
- Train
- Airways

checkbox.html

```
<html>
  <head>
    <title>Checkbox Test Page</title>
  </head>
  <body>
    <form name="myform" method="POST">
      <b>How would you like to travel?</b>
      <div><br>
        <input type="checkbox" name="option1" value="Car"> Car<br>
        <input type="checkbox" name="option2" value="Bus"> Bus<br>
        <input type="checkbox" name="option3" value="Train"> Train<br>
        <input type="checkbox" name="option4" value="Air"> Airways<br>
        <br>
      </div>
    </form>
  </body>
</html>
```

In the above form, we are planning to select Car, which is a checkbox. The details are given in the test case. Now, we will execute the test case and check the selection of the checkbox for the above form.