



Robot Framework

tutorialspoint

SIMPLY EASY LEARNING



www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

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. This feature makes it very easy to understand. Test cases are written using keyword style in a tabular format. Robot Framework provides good support for external libraries, tools that are open source and can be used for automation. The most popular library used with Robot Framework is Selenium Library used for web development & UI testing.

Audience

This tutorial is designed for software programmers/testers, who want to learn the basics of Robot Framework automation testing in simple and easy ways. This tutorial will give you enough understanding on various functionalities of Robot Framework with suitable examples.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of testing concepts.

Copyright & Disclaimer

© Copyright 2019 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial	i
Audience.....	i
Prerequisites.....	i
Copyright & Disclaimer	i
Table of Contents	ii
1. Robot Framework — Overview	1
Robot Framework Features	1
Robot Framework Advantages	2
Robot Framework Limitations	2
Conclusion	3
2. Robot Framework — Environment Setup.....	4
Install Python.....	4
Setting path for Windows.....	6
3. Robot — Unix and Linux Installation	9
Setting Path at Unix/Linux	9
Install PIP	9
Install Robot Framework	11
Install wxPython	12
Install Ride	14
Conclusion	16
4. Robot Framework — Introduction to Ride	18
Create New Project.....	19
Conclusion	24
5. Robot Framework — First Test Case Using Ride	25
The Settings Format	30
Conclusion	39

6. Robot framework — Writing and Executing Test Cases	40
Project Setup	40
Importing Libraries	45
Write test case in tabular format	48
Using Tags for Executing Test Case.....	50
Use Resource Files for Test Case	51
Conclusion	57
7. Robot framework — Keyword and Data Driven Test Cases	59
Keyword Driven Style	59
Data Driven Style	64
Conclusion	67
8. Robot — Working With Browsers Using Selenium Library	68
Project Setup In Ride	68
Import Selenium Library	71
Test Case Using Chrome Browser.....	79
Test Case Using Firefox Browser	87
Conclusion	87
9. Robot Framework — Working With Textbox	89
Project Setup for Textbox Testing	89
Enter Data in Textbox	93
Conclusion	97
10. Robot Framework — Working With Radio Button	99
Project Setup For Textbox Testing.....	99
Test Case for Radio Button	104
Conclusion	109
11. Robot Framework — Working With Checkbox	110
Project Setup for Checkbox Testing.....	110
Test Case for Checkbox.....	114

Conclusion	119
12. Robot Framework — Working With Dropdown	120
Project Setup for Dropdown Testing	120
Test Case for Dropdown	124
Conclusion	132
13. Robot Framework — Working With Keywords.....	133
Library Keywords	133
User-defined Keywords	139
Conclusion	147
14. Robot Framework — Working With Variables	148
Scalar Variable	148
Test Case for Scalar Variable	152
List Variable	157
Dictionary Variable	160
Conclusion	166
15. Robot Framework – Working With Command Line	167
Conclusion	170
16. Robot Framework — Working With Setup And Teardown	171
Conclusion	175
17. Robot Framework — Working with Built-In Library	176
Conclusion	178
18. Robot Framework — Working With External Database libraries	179
Import Database Library.....	183
Conclusion	190
19. Robot Framework — Testing Login Page Using Robot Framework	191
Conclusion	218

1. 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"**.

2. Robot Framework — Environment Setup

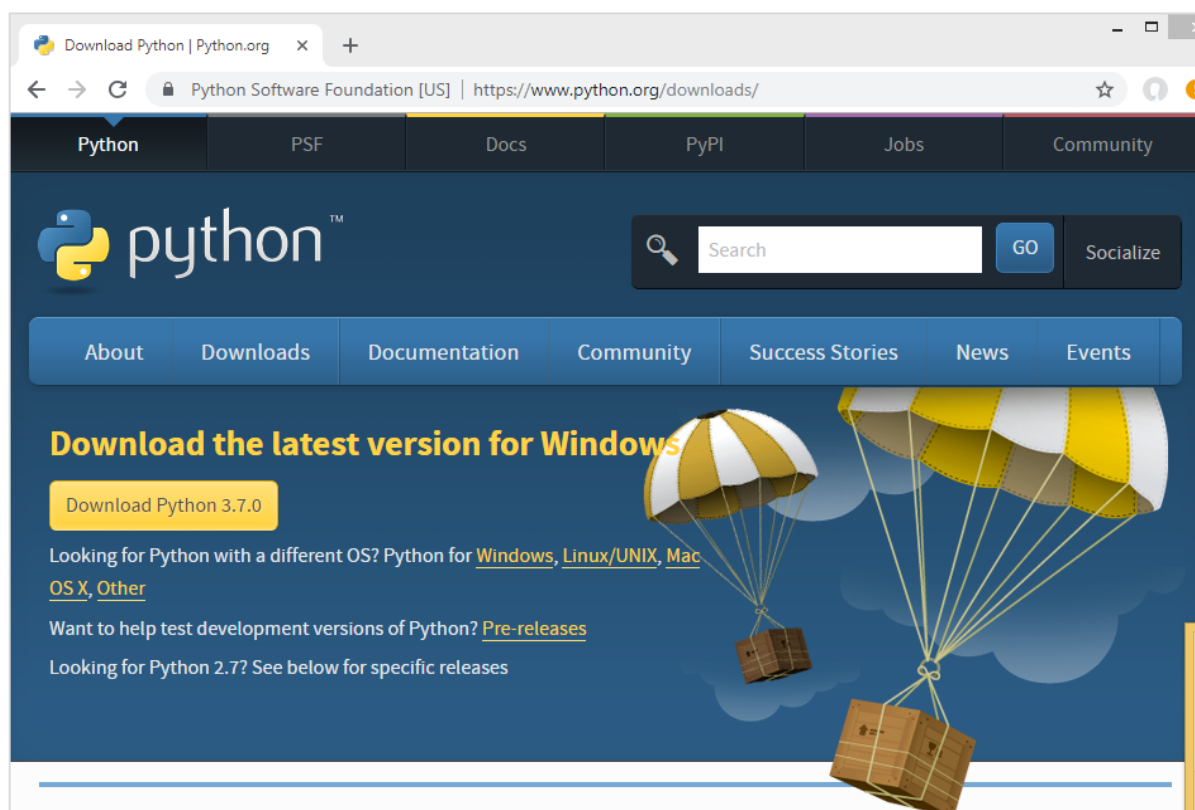
Robot framework is built using python. In this chapter, we will learn how to set up Robot Framework. To work with Robot Framework, we need to install the following:

- 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:

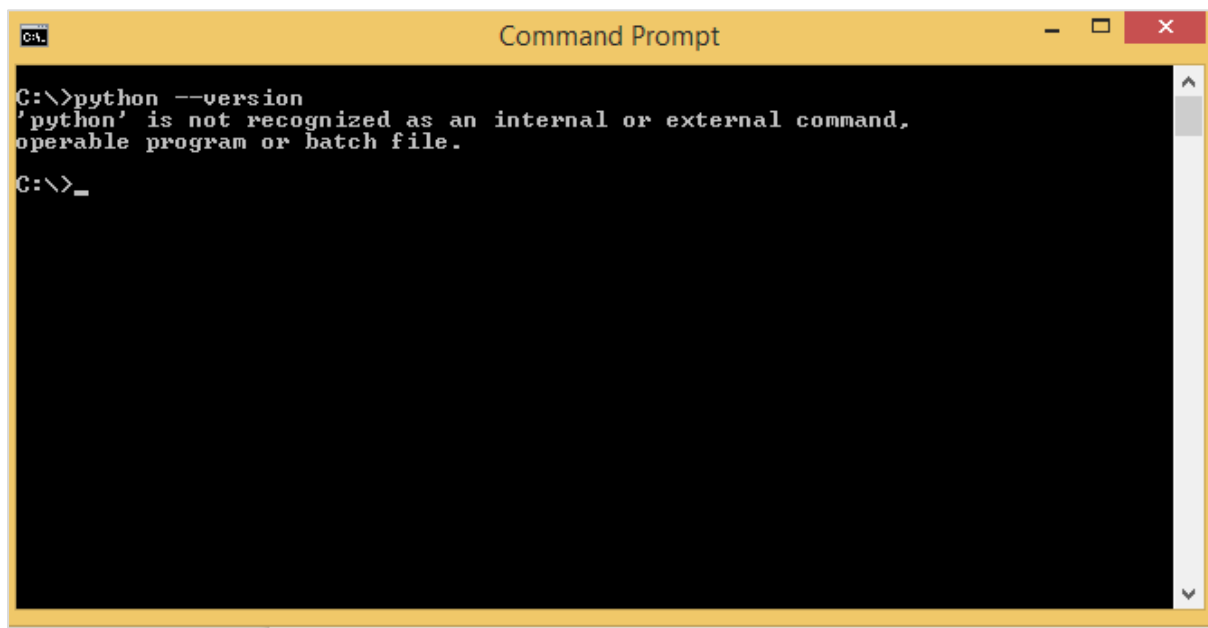
Release version	Release date	Click for more	
Python 3.5.6	2018-08-02	Download	Release Notes
Python 3.4.9	2018-08-02	Download	Release Notes
Python 3.7.0	2018-06-27	Download	Release Notes
Python 3.6.6	2018-06-27	Download	Release Notes
Python 2.7.15	2018-05-01	Download	Release Notes
Python 3.6.5	2018-03-28	Download	Release Notes
Python 3.4.8	2018-02-05	Download	Release Notes
Python 3.5.5	2018-02-05	Download	Release Notes

[View older releases](#)

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
```

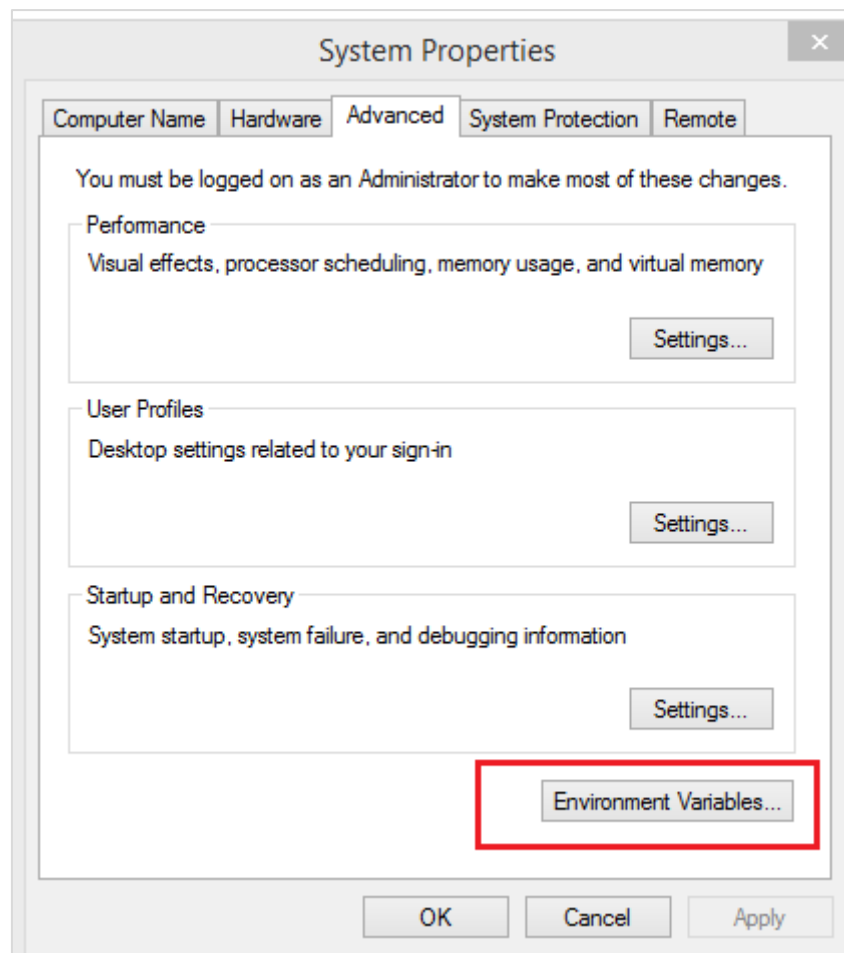


If we get the version of python as output then, we have python installed in our system. Otherwise, you will get a display as shown above.

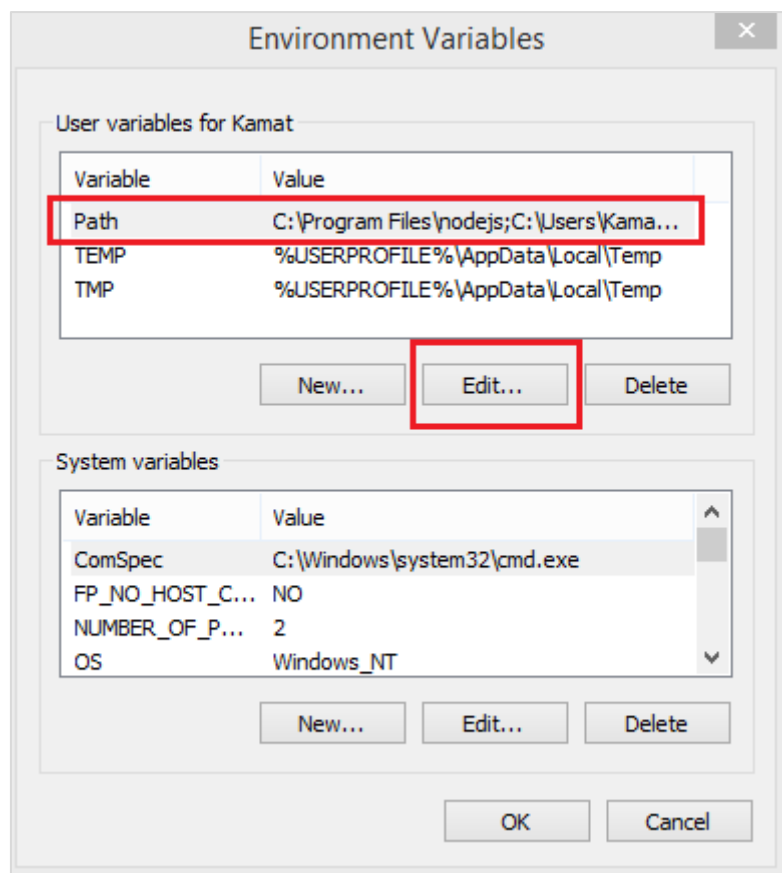
Here, we will download python version 2.7 as it is compatible to the windows 8 we are using right now. Once downloaded, install python on your system by double-clicking on .exe python download. Follow the installation steps to install Python on your system. Once installed, to make python available globally,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

Setting path for Windows

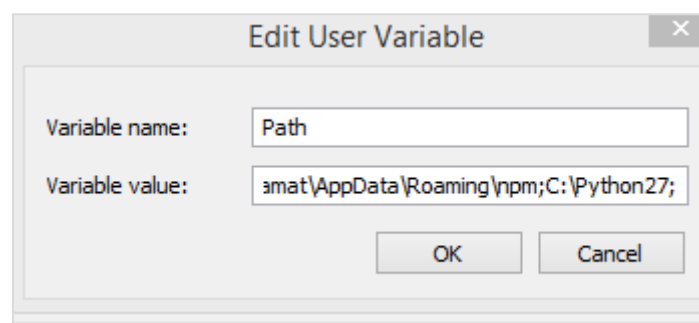
Right-click on My Computer icon and select properties. Click on Advanced System setting and the following screen will be displayed.



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

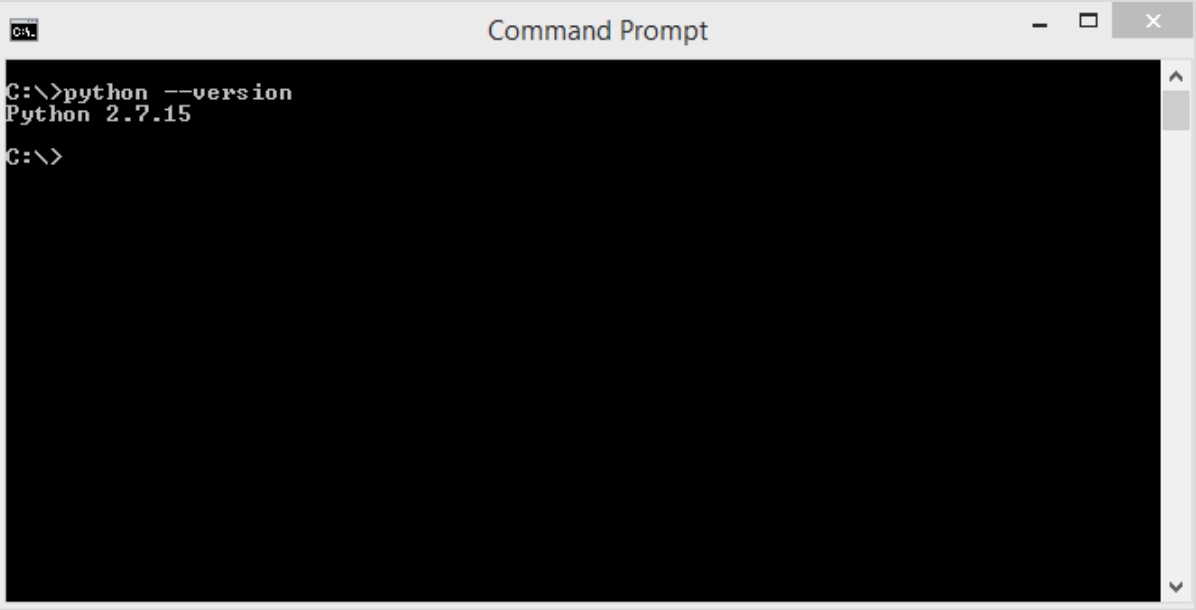


Select the *Variable Path* and click the *Edit* button.



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:\>
```

3. Robot — 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 PATH="$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:

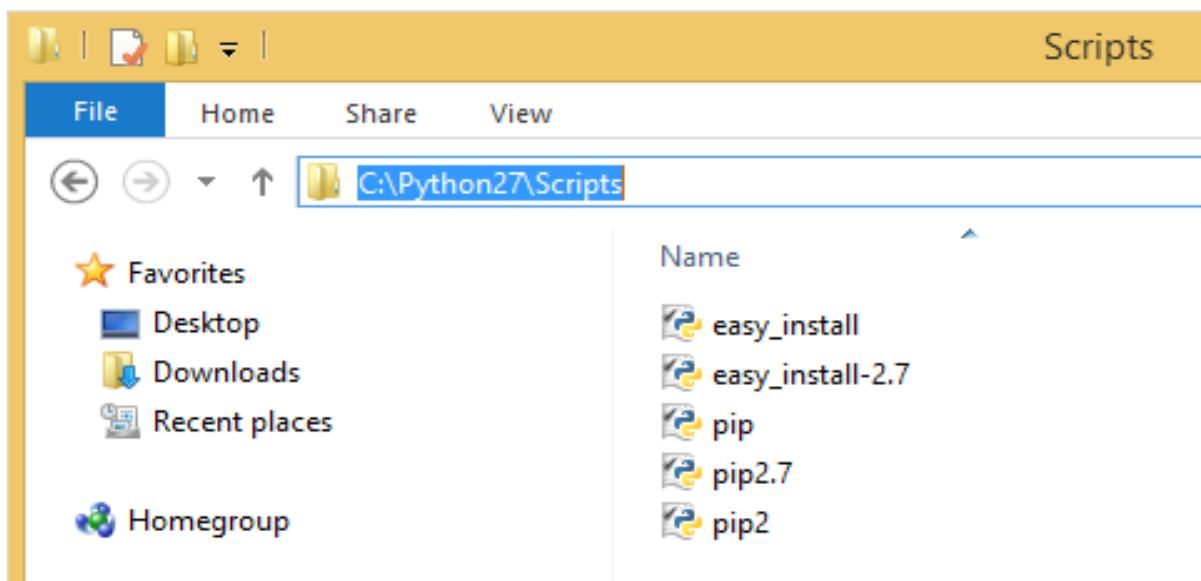
Command

```
pip --version
```

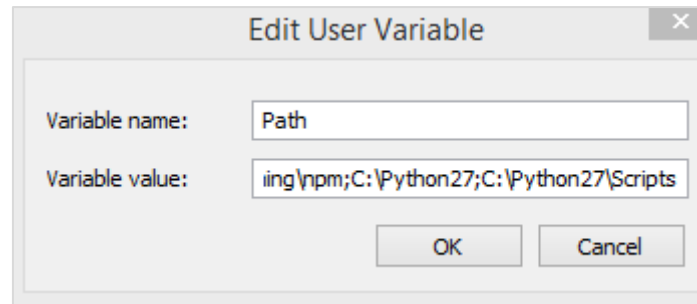


```
G:\>pip --version
'pip' is not recognized as an internal or external command,
operable program or batch file.
G:\>_
```

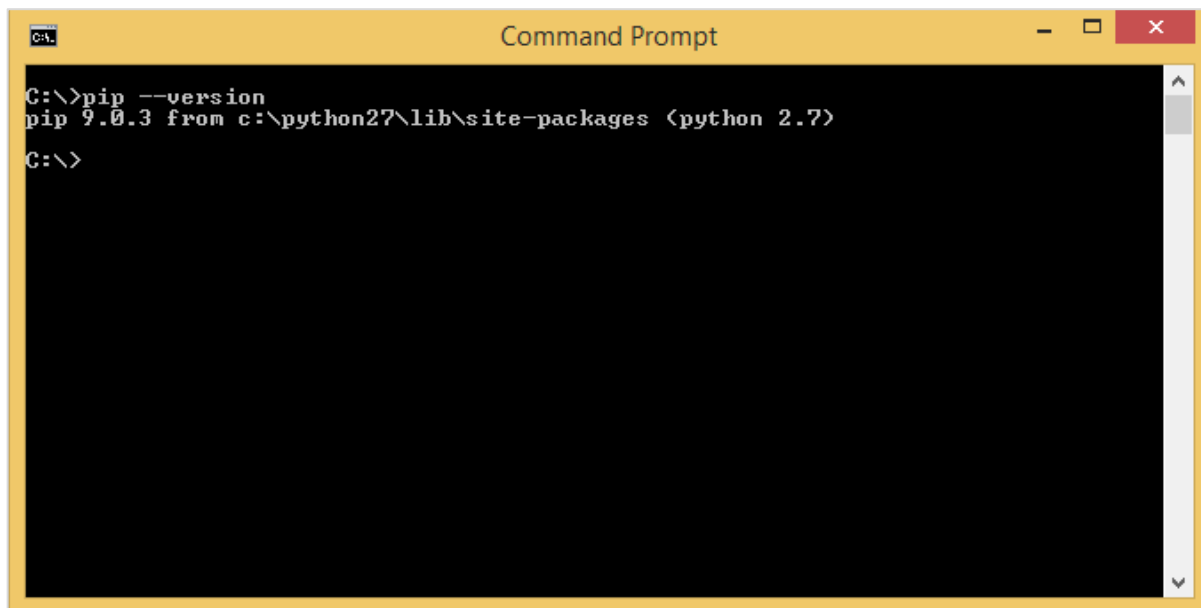
Here we are still not getting the version for pip. We need to add the pip path to Environment variables so that we can use it globally. PIP will be installed in Scripts folder of python as shown below:



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:



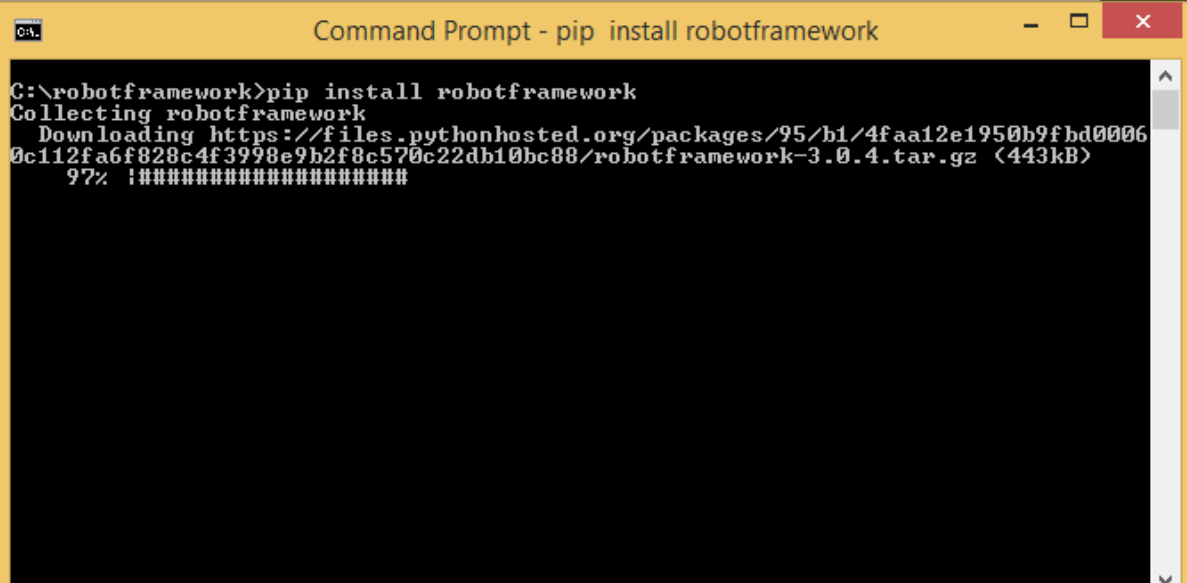
So now, we have python and pip installed.

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
```

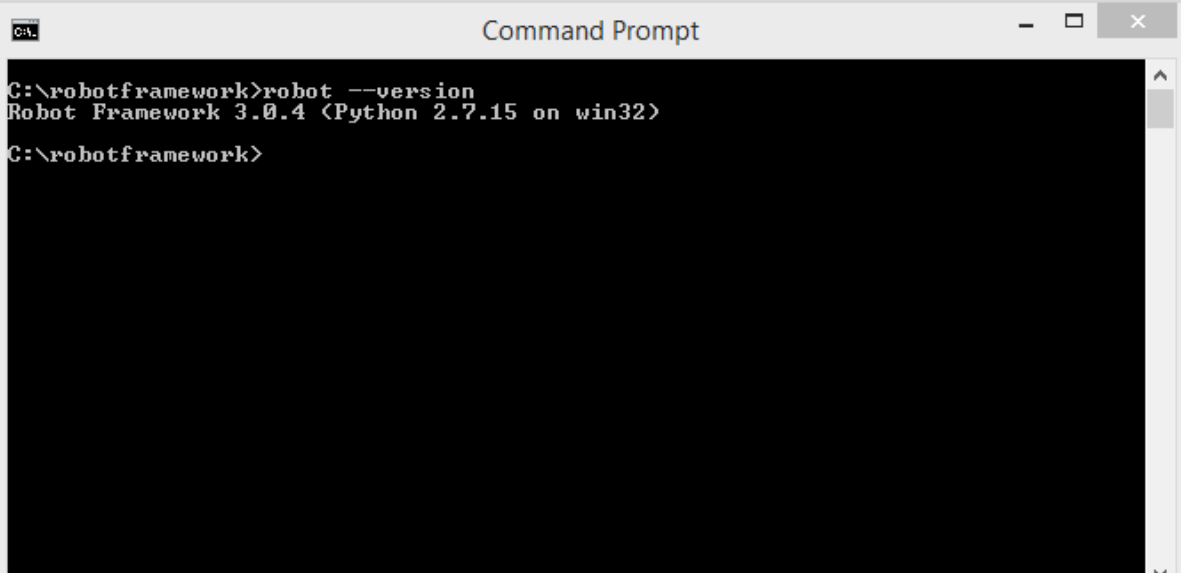


```
C:\robotframework>pip install robotframework
Collecting robotframework
  Downloading https://files.pythonhosted.org/packages/95/b1/4faa12e1950b9fbd0006
0c112fa6f828c4f3998e9b2f8c570c22db10bc88/robotframework-3.0.4.tar.gz (443kB)
    97% |#####|
```

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

Command

```
robot --version
```



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

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.

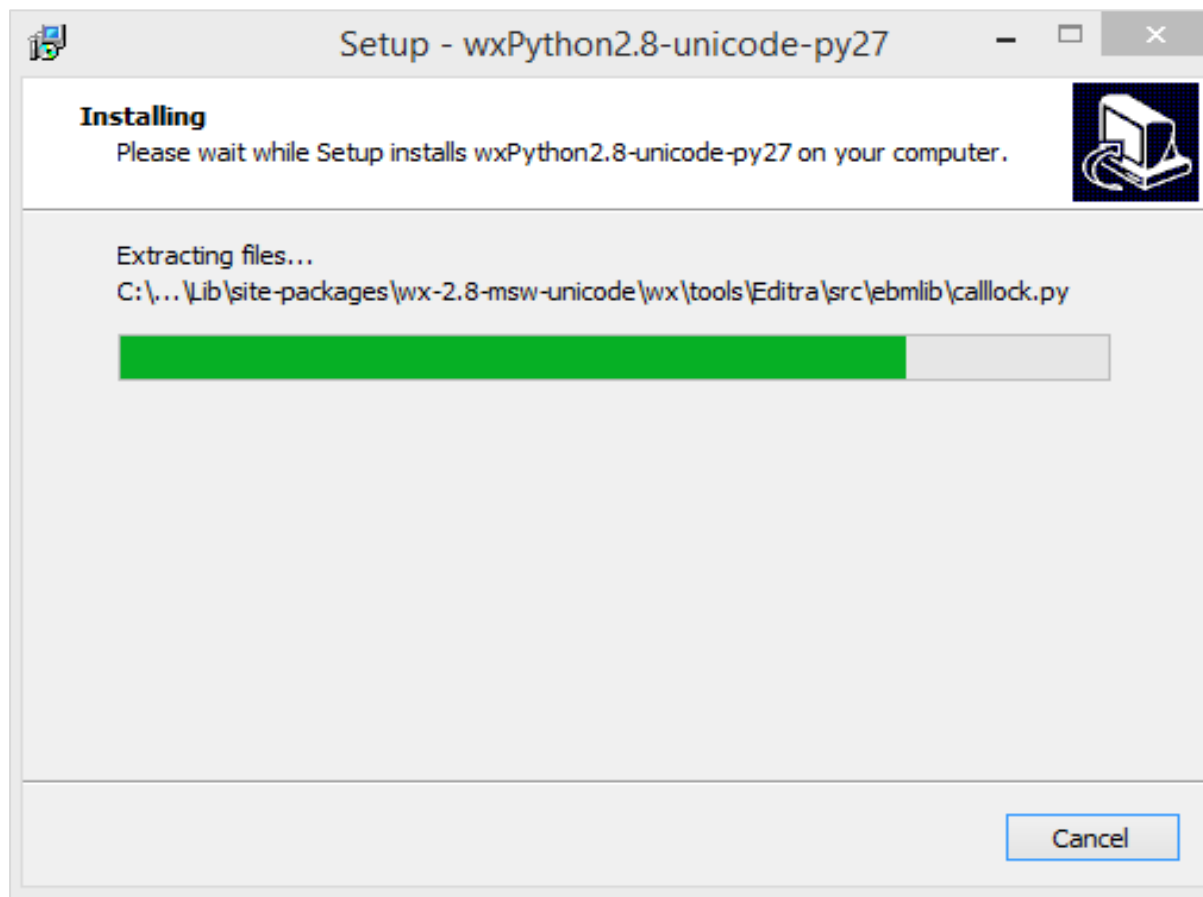
For windows to get the required download for wxPython, go to the following URL:

<http://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.

Name	Modified	Size	Downloads / Week
Parent folder			
wxwidgets2.8_2.8.12.1.orig.tar.gz	2011-07-25	32.6 MB	57
wxPython2.8-win64-unicode-2.8.12.1-py27.exe	2011-07-25	11.7 MB	525
wxPython2.8-win64-unicode-2.8.12.1-py26.exe	2011-07-25	11.7 MB	19
wxPython2.8-win64-devel-2.8.12.1-msvc9x64.tar.bz2	2011-07-25	7.2 MB	12
wxPython2.8-win32-unicode-2.8.12.1-py27.exe	2011-07-25	11.4 MB	525
wxPython2.8-win32-docs-demos-2.8.12.1.exe	2011-07-25	7.8 MB	1
wxPython2.8-win32-devel-2.8.12.1-msvc9.tar.bz2	2011-07-25	10.8 MB	1
wxPython2.8-win32-ansi-2.8.12.1-py27.exe	2011-07-25	11.3 MB	13
wxPython2.8-win32-ansi-2.8.12.1-py26.exe	2011-07-25	11.3 MB	1
wxPython2.8-osx-unicode-2.8.12.1-universal-py2.7.dmg	2011-07-25	42.1 MB	90

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
_msw.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\too
lbar.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 python-wxgtk2.8` 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:

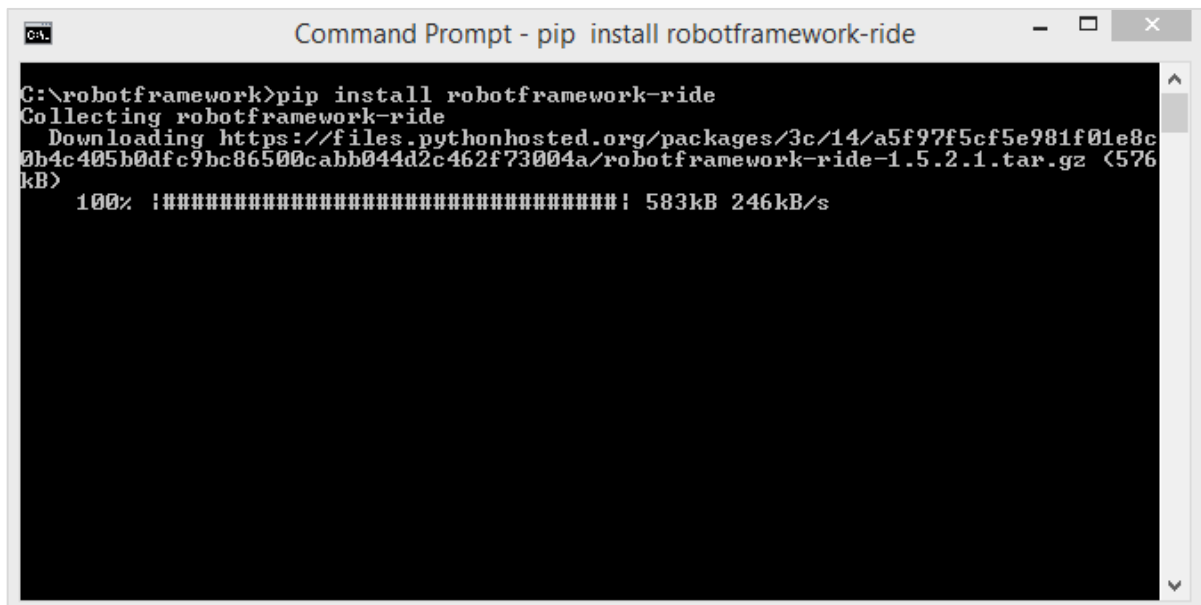
```
> VERSIONER_PYTHON_PREFER_32_BIT=yes ride.py
```

Install Ride

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

Command

```
pip install robotframework-ride
```



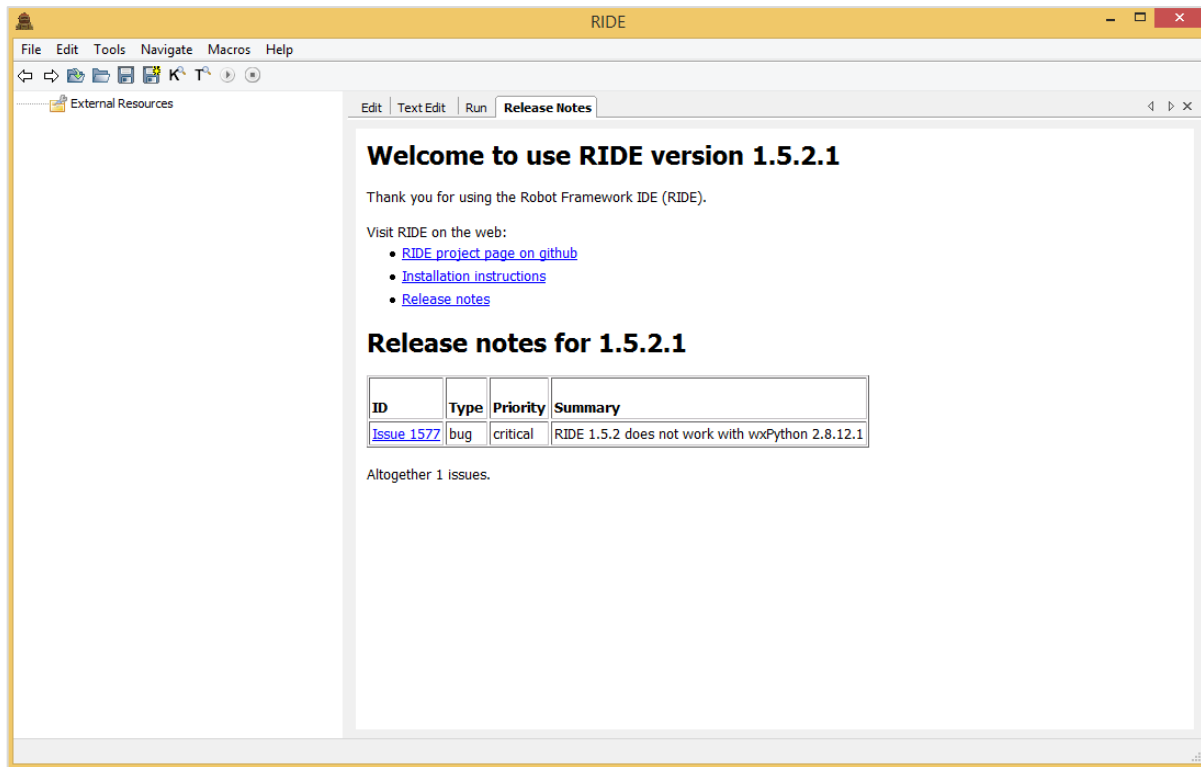
```
C:\robotframework>pip install robotframework-ride
Collecting robotframework-ride
  Downloading https://files.pythonhosted.org/packages/3c/14/a5f97f5cf5e981f01e8c0b4c405b0dfc9bc86500cab044d2c462f73004a/robotframework-ride-1.5.2.1.tar.gz (576kB)
    100% |#####| 583kB 246kB/s
```

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

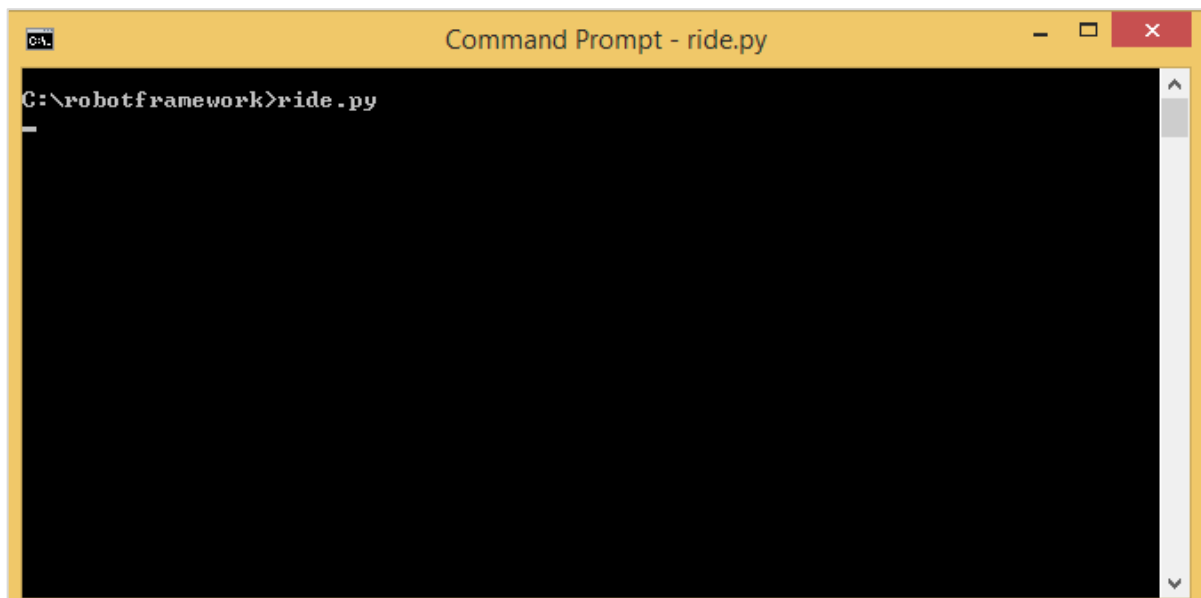
We now know how to install python, pip, robot framework and also get RIDE installed to work with test cases in robot framework.

4. Robot Framework — Introduction to Ride

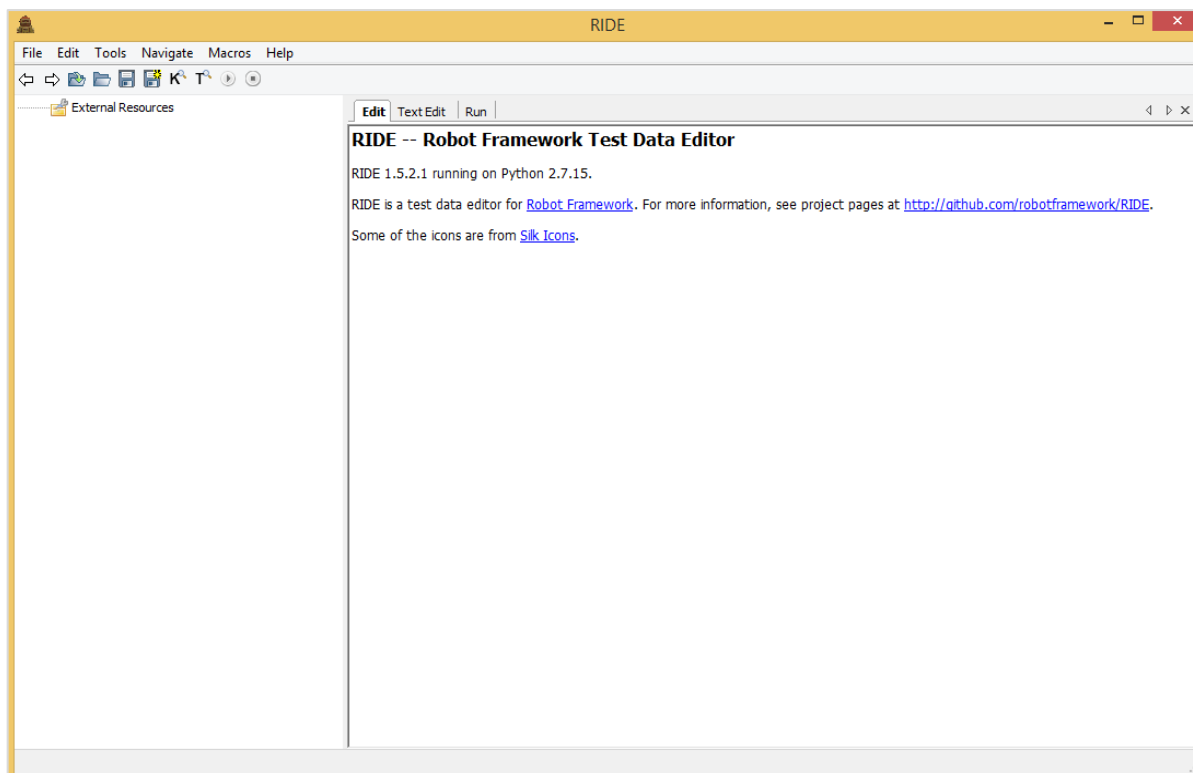
Ride is a testing editor for Robot Framework. Further, we will write test cases in Ride. To start Ride, we need to run the command shown below.

Command

```
ride.py
```



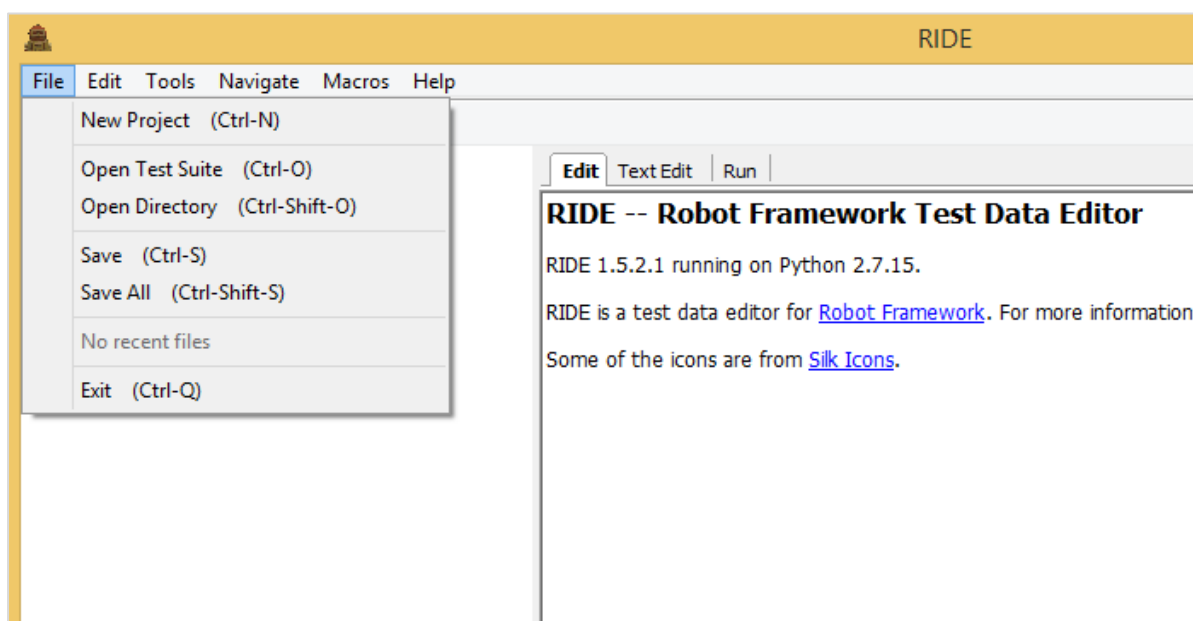
The above command will open the IDE as shown in the following screenshot:



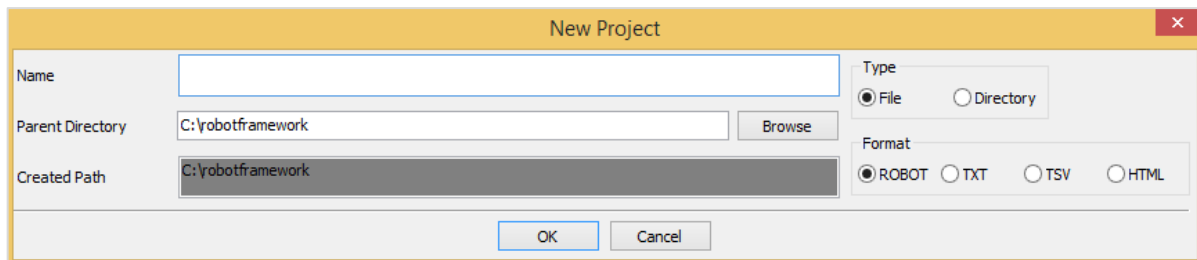
In this chapter, we will walk through the editor to see what options and features are available in the IDE. The options and features will help us in testing our project.

Create New Project

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



The following screen will appear when you click New Project.

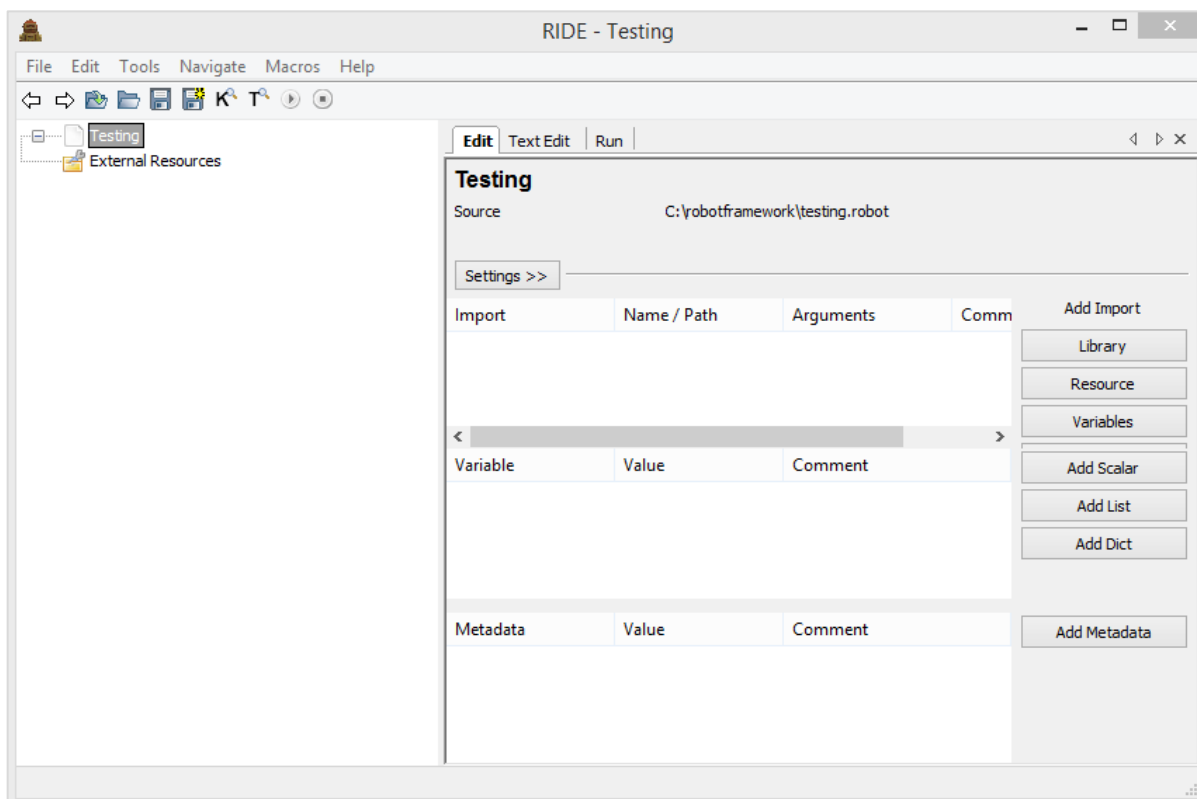


The 'New Project' dialog box has a yellow title bar. It contains the following fields and controls:

- Name:** An empty text input field.
- Parent Directory:** A text field containing 'C:\robotframework' with a 'Browse' button to its right.
- Created Path:** A text field containing 'C:\robotframework'.
- Type:** Two radio buttons: 'File' (selected) and 'Directory'.
- Format:** Four radio buttons: 'ROBOT' (selected), 'TXT', 'TSV', and 'HTML'.
- Buttons:** 'OK' and 'Cancel' buttons at the bottom.

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 'RIDE - Testing' window shows the project 'Testing' in the left sidebar. The main area has three tabs: 'Edit', 'Text Edit', and 'Run'. The 'Edit' tab is active, showing the source file 'C:\robotframework\testing.robot'. Below the source path, there is a 'Settings >>' button. The main content area is divided into two sections:

Import	Name / Path	Arguments	Comm

Variable	Value	Comment

Metadata	Value	Comment

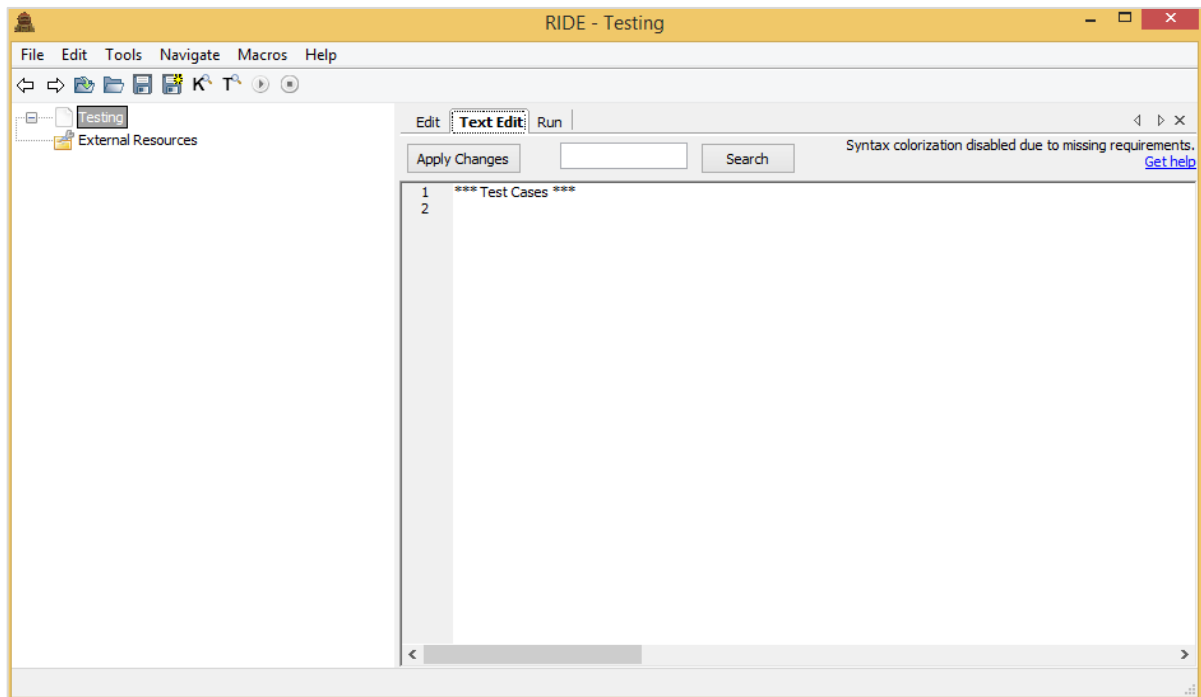
On the right side of the 'Edit' tab, there is a vertical stack of buttons for adding data:

- Add Import
- Library
- Resource
- Variables
- Add Scalar
- Add List
- Add Dict
- Add Metadata

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.

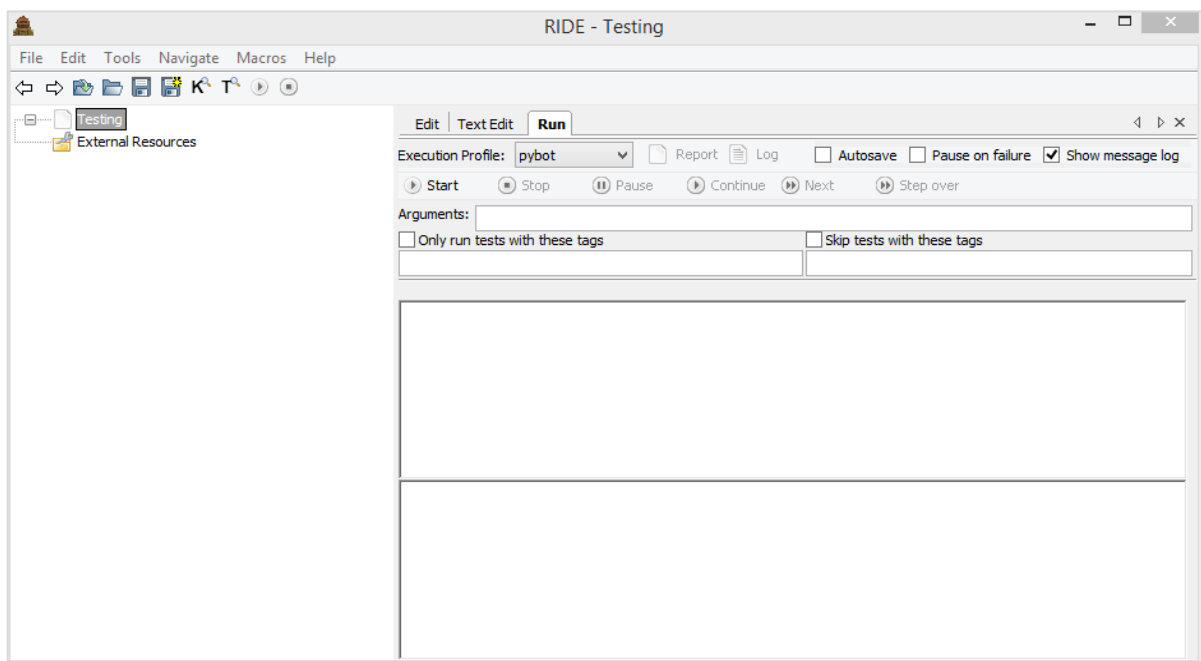
Edit has a lot of options on the UI as shown above. In this section, we can add data required to run our test cases. We can import Library, Resource, Variables, Add scalar, Add list, Add dict and Add Metadata.

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.

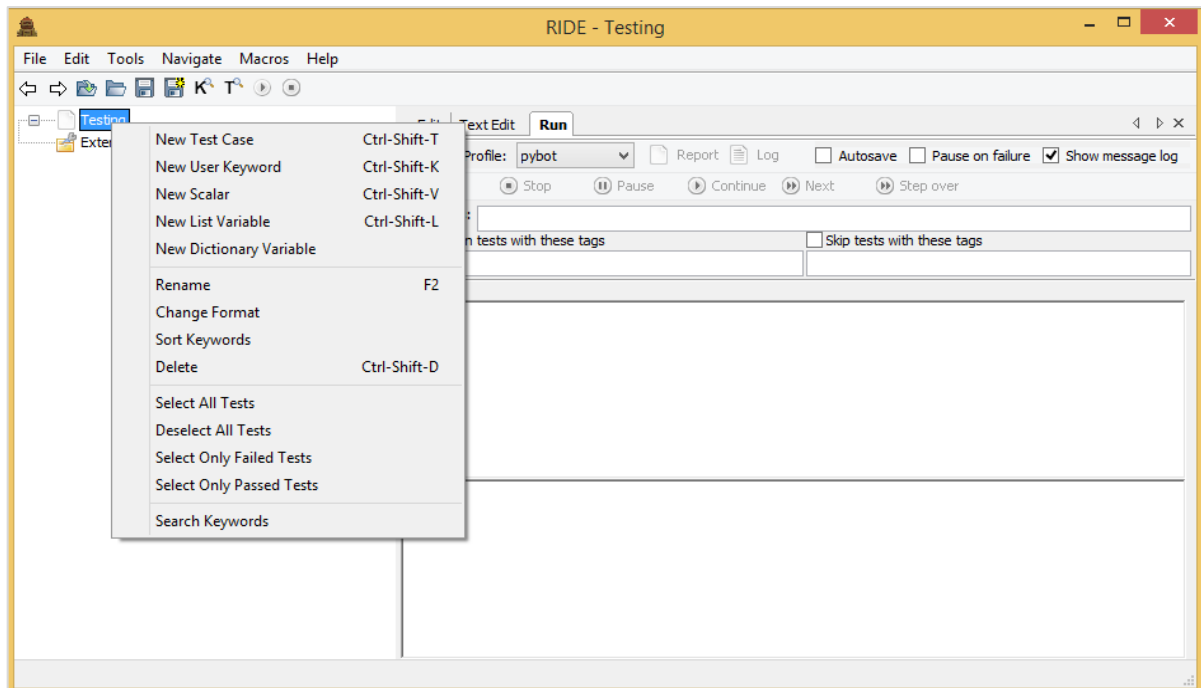
Once the test cases are ready, we can use the third tab Run to execute them.



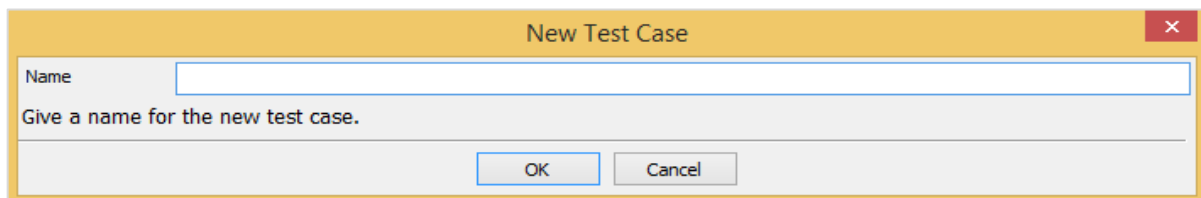
The Run UI is as shown above. It allows to run the test case and comes with options like start, stop, pause continue, next test case, step over, etc. You can also create Report, Log for the test cases you are executing.

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

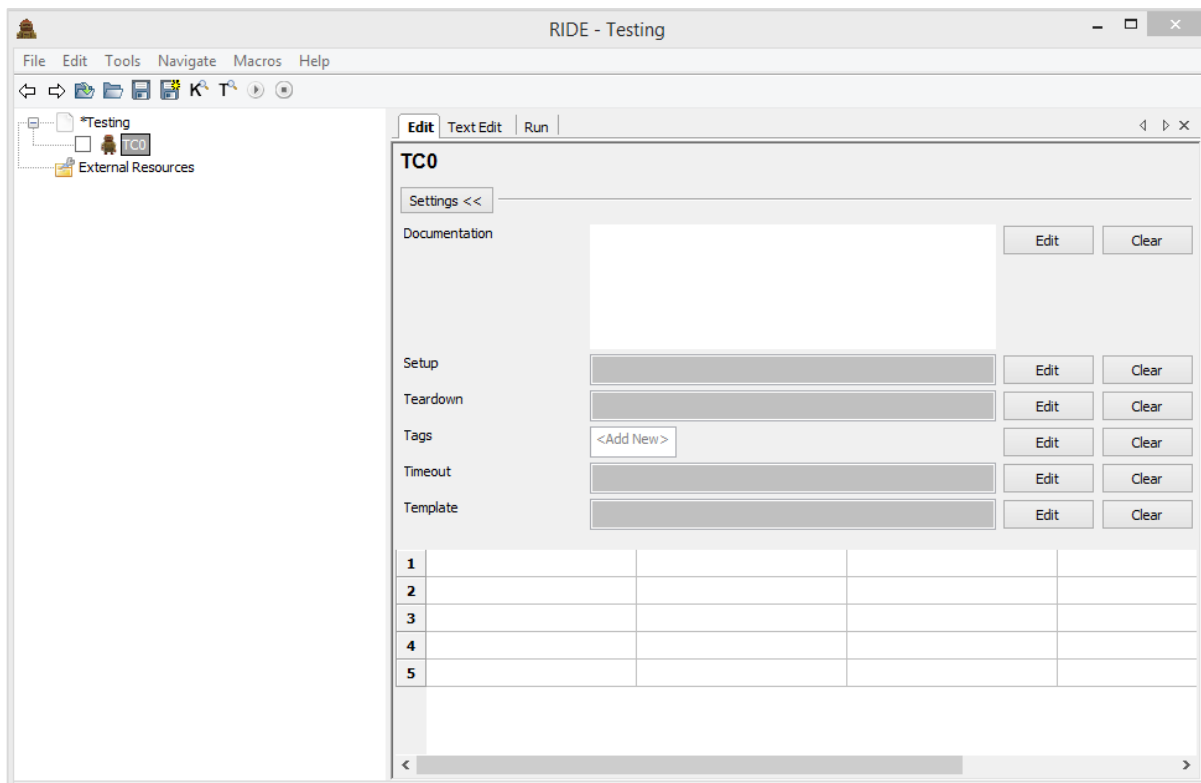
Right-click on the project created and click on new test case as shown below:



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



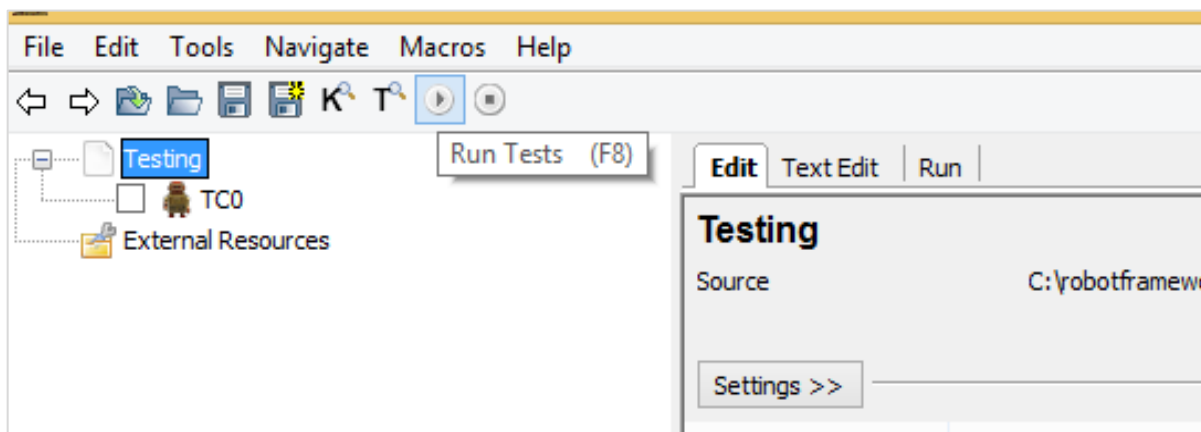
Enter the name of the test case and click **OK**. We have saved the test case as TC0. The following screen appears once the test case is saved.



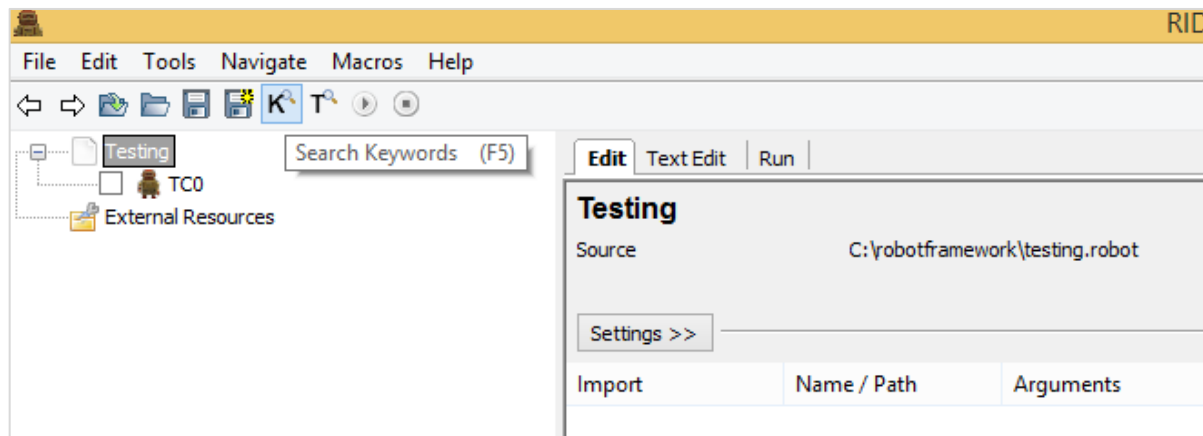
The test case has options like Documentation, setup, teardown, tags, timeout and Template. They have an edit button across it; upon clicking the button a screen appears wherein, you can enter the details for each option. We will discuss the various parameters of these details in our subsequent chapters.

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.

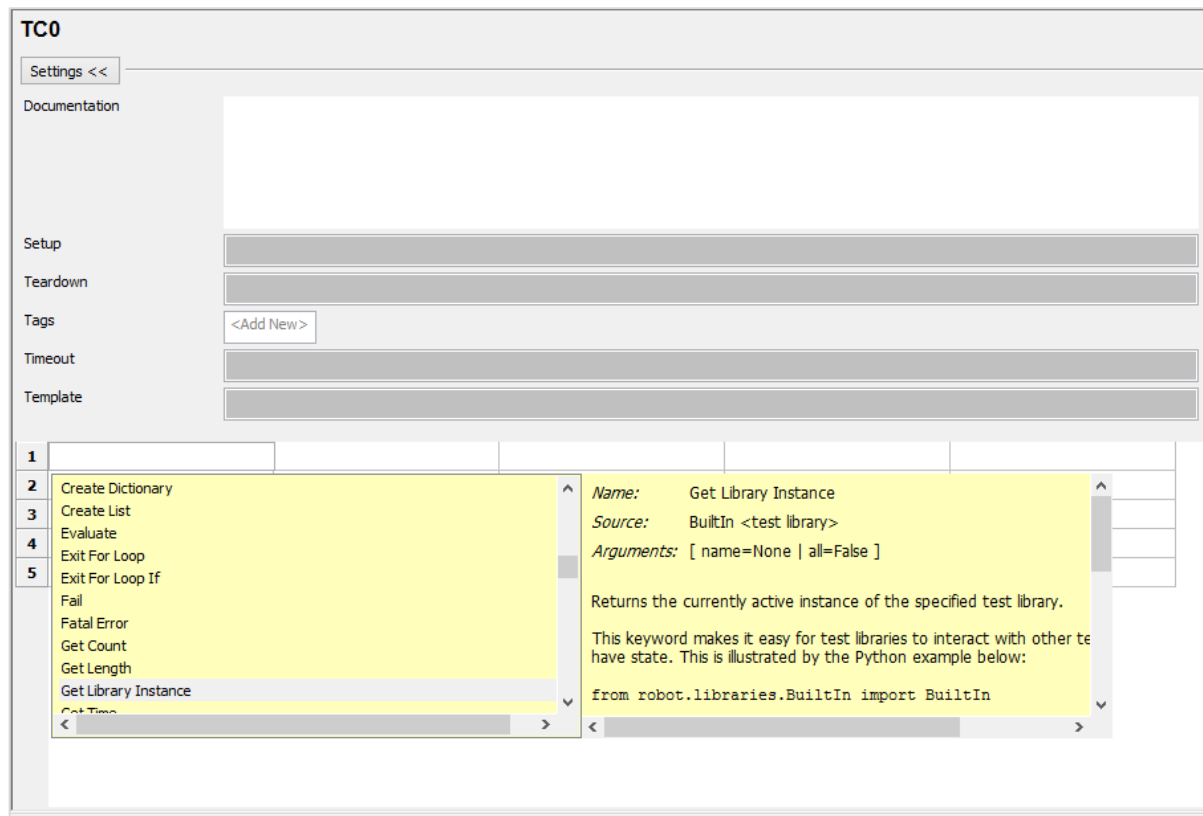
There are shortcuts available in the navigation bar to run/stop test case as shown below:



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.

5. Robot Framework — First Test Case Using Ride

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

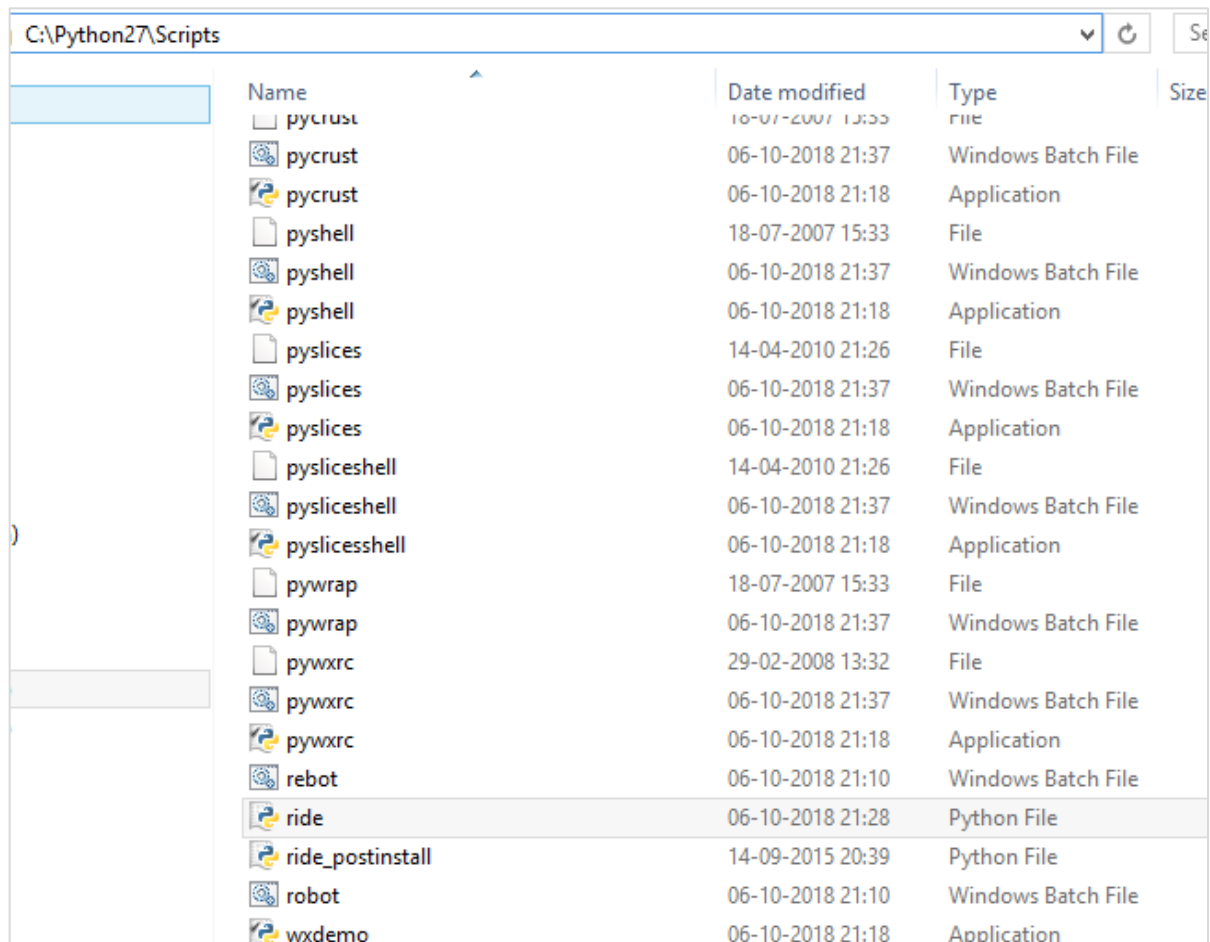
Open Ride from command prompt or you can create a shortcut of ride on your desktop.

From command line

```
ride.py
```

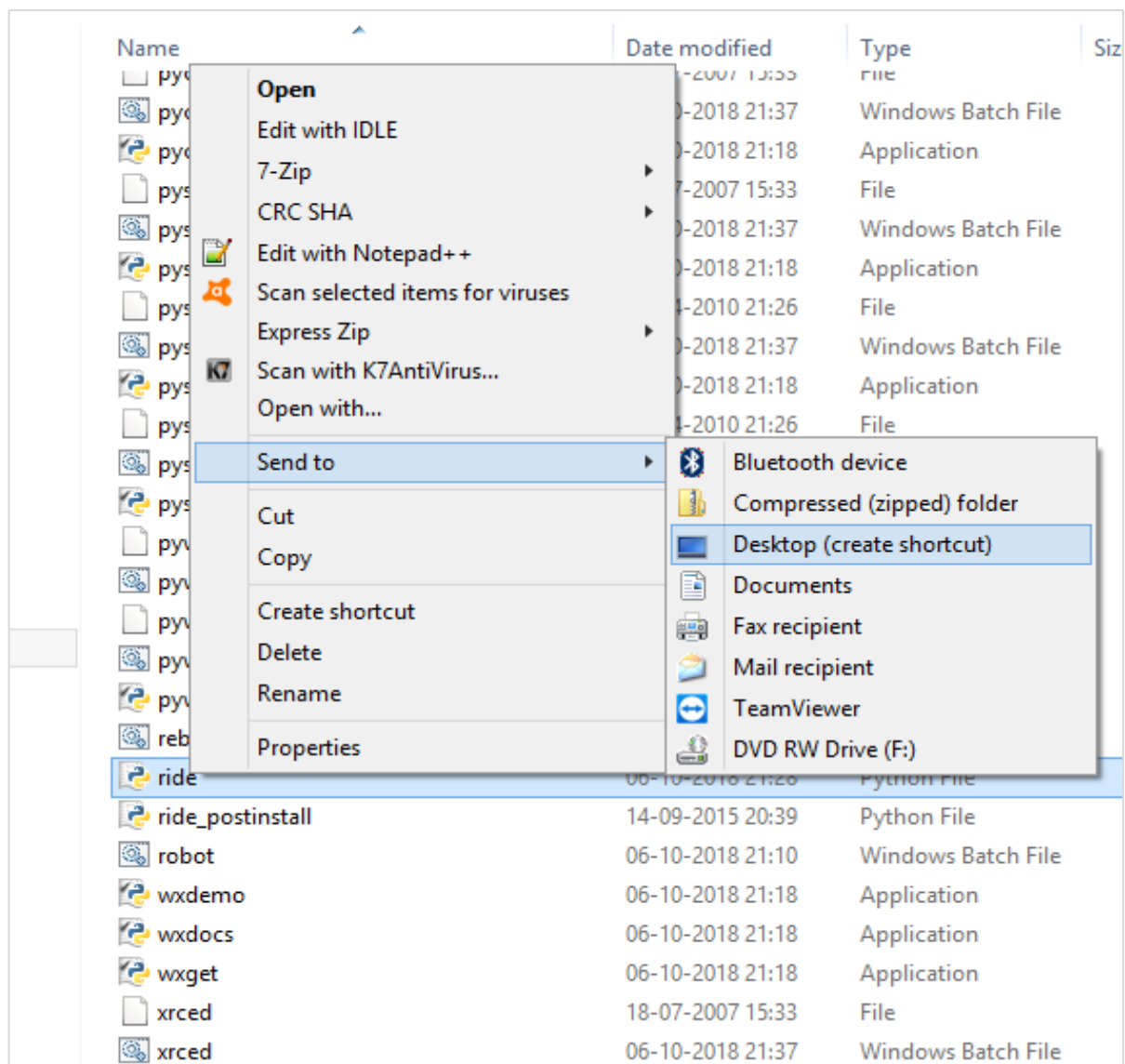
From Desktop

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



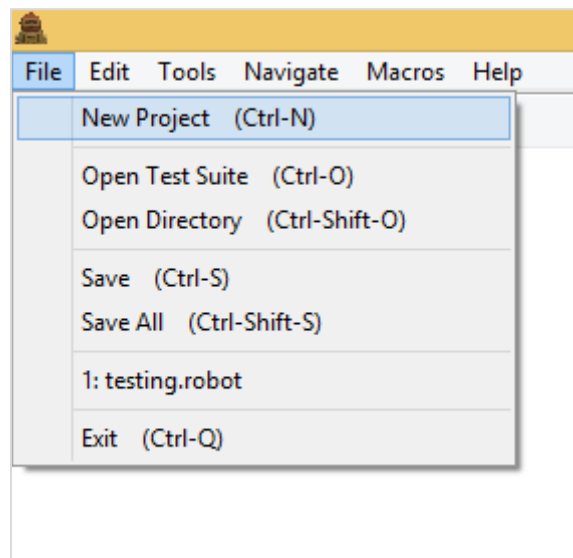
Name	Date modified	Type	Size
pycrust	18-07-2007 15:33	File	
pycrust	06-10-2018 21:37	Windows Batch File	
pycrust	06-10-2018 21:18	Application	
pyshell	18-07-2007 15:33	File	
pyshell	06-10-2018 21:37	Windows Batch File	
pyshell	06-10-2018 21:18	Application	
pyslices	14-04-2010 21:26	File	
pyslices	06-10-2018 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	
pywxc	29-02-2008 13:32	File	
pywxc	06-10-2018 21:37	Windows Batch File	
pywxc	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).

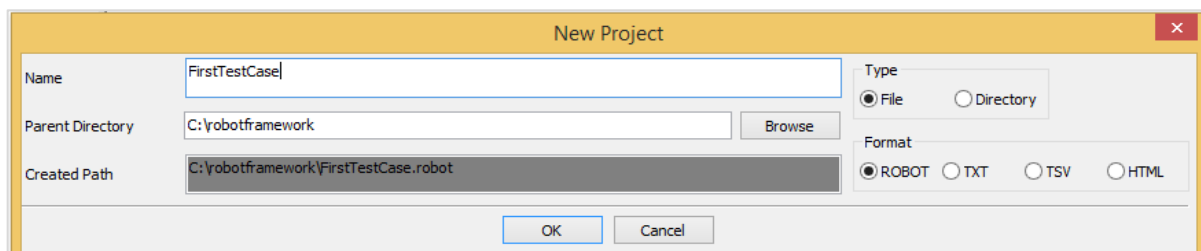


You will now see an icon of ride on your desktop. You can click on it to open the ride editor.

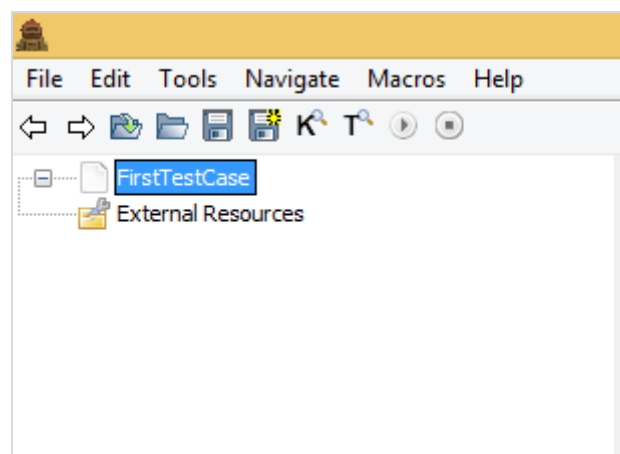
Let us start with our first test case in ride. Open the editor and click on File -> New Project.



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

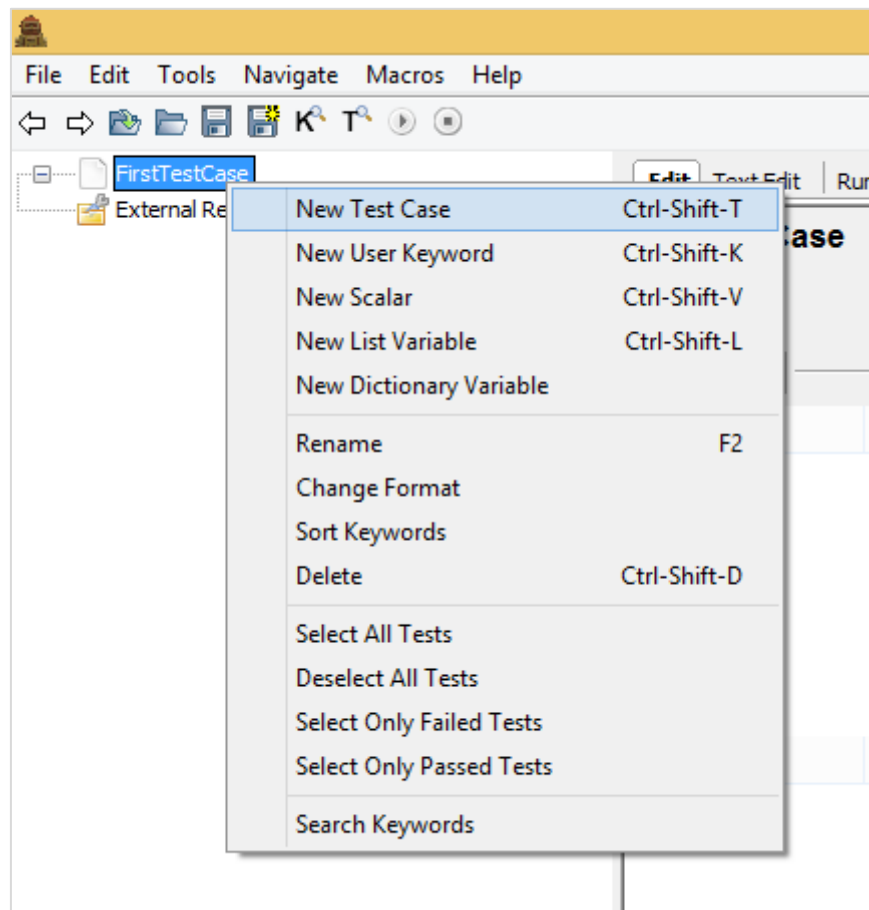


Parent Directory is the path where the project will be saved. You can change the path if required. I have created a folder called robotframework and will save all the files in that folder.

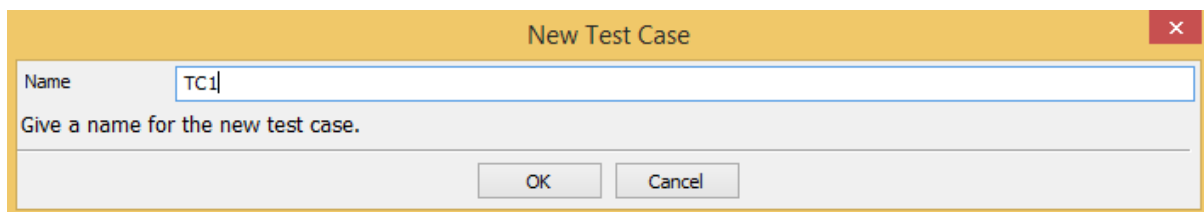


Project *FirstTestCase* is created.

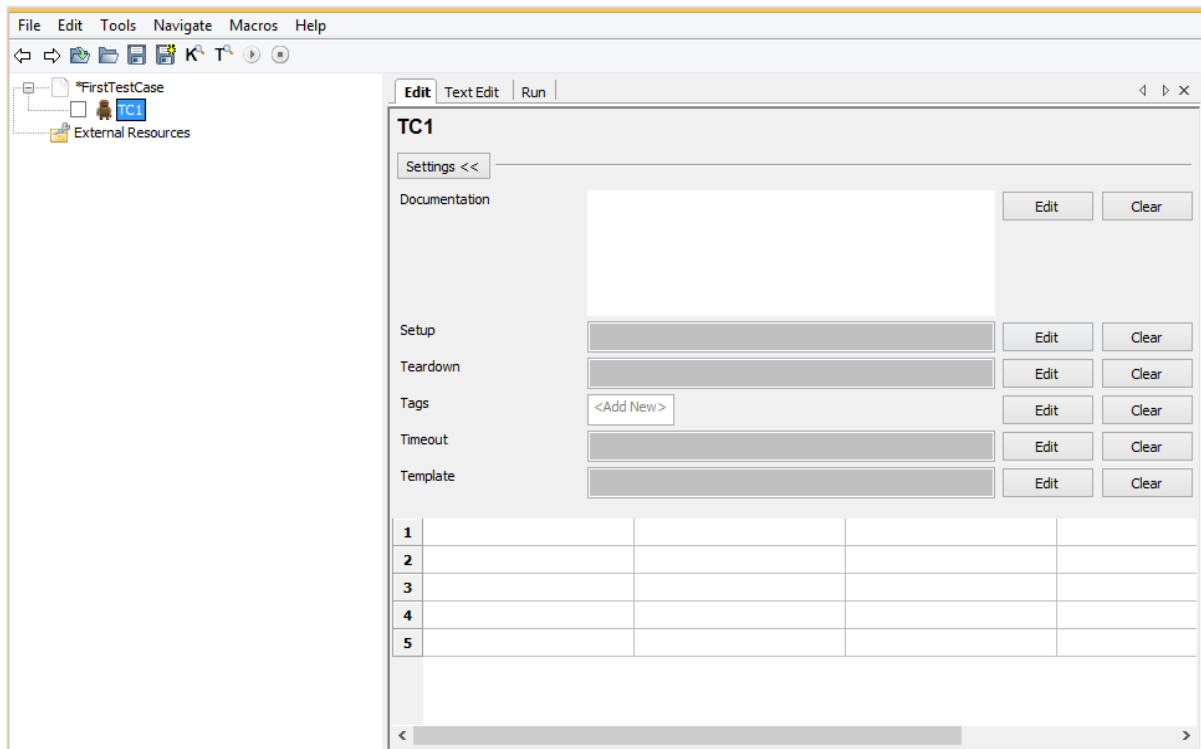
To create test case, right-click on the project.



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*.

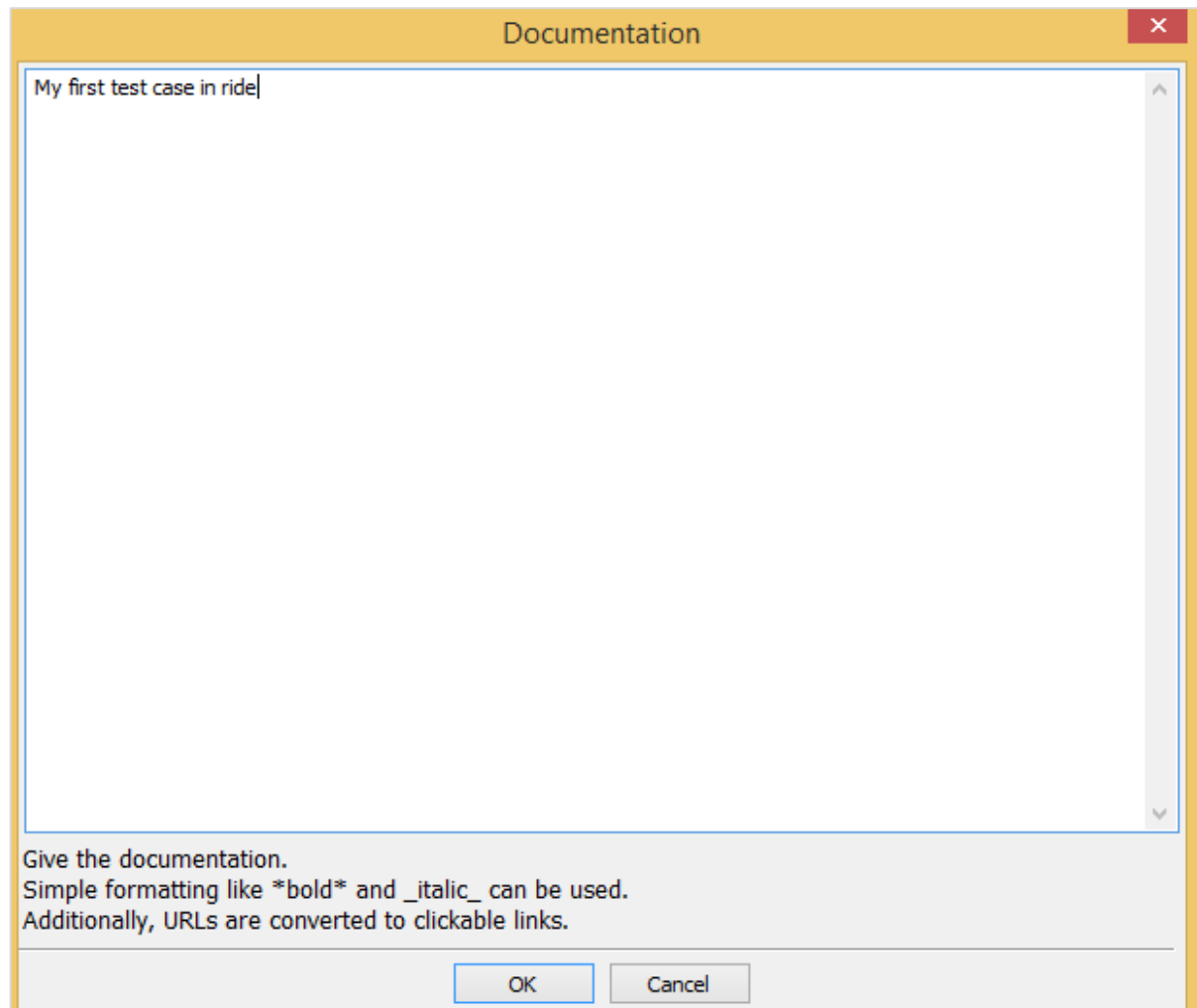
The Edit tab comes with two formats – Settings and Tabular. We will discuss the two formats in our subsequent sections.

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.

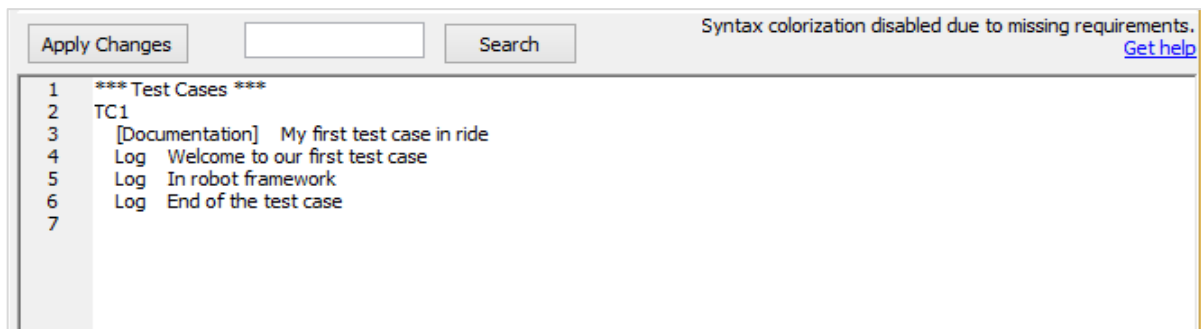
In this test case, we are just going to add some logs and see the output of it. Consider the following screenshot to understand this:

The screenshot shows the Robot Framework Test Case Editor interface for a test case named 'TC1'. The interface includes tabs for 'Edit', 'Text Edit', and 'Run'. Below the tabs, there are sections for 'Documentation', 'Setup', 'Teardown', 'Tags', 'Timeout', and 'Template', each with a text input field and 'Edit' and 'Clear' buttons. The 'Documentation' field contains the text 'My first test case in ride'. Below these sections is a table with 5 rows and 4 columns. The first three rows contain log messages: 'Welcome to our first test case', 'In robot framework', and 'End of the test case'. The third row is highlighted in yellow. The fourth and fifth rows are empty.

	Log			
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.

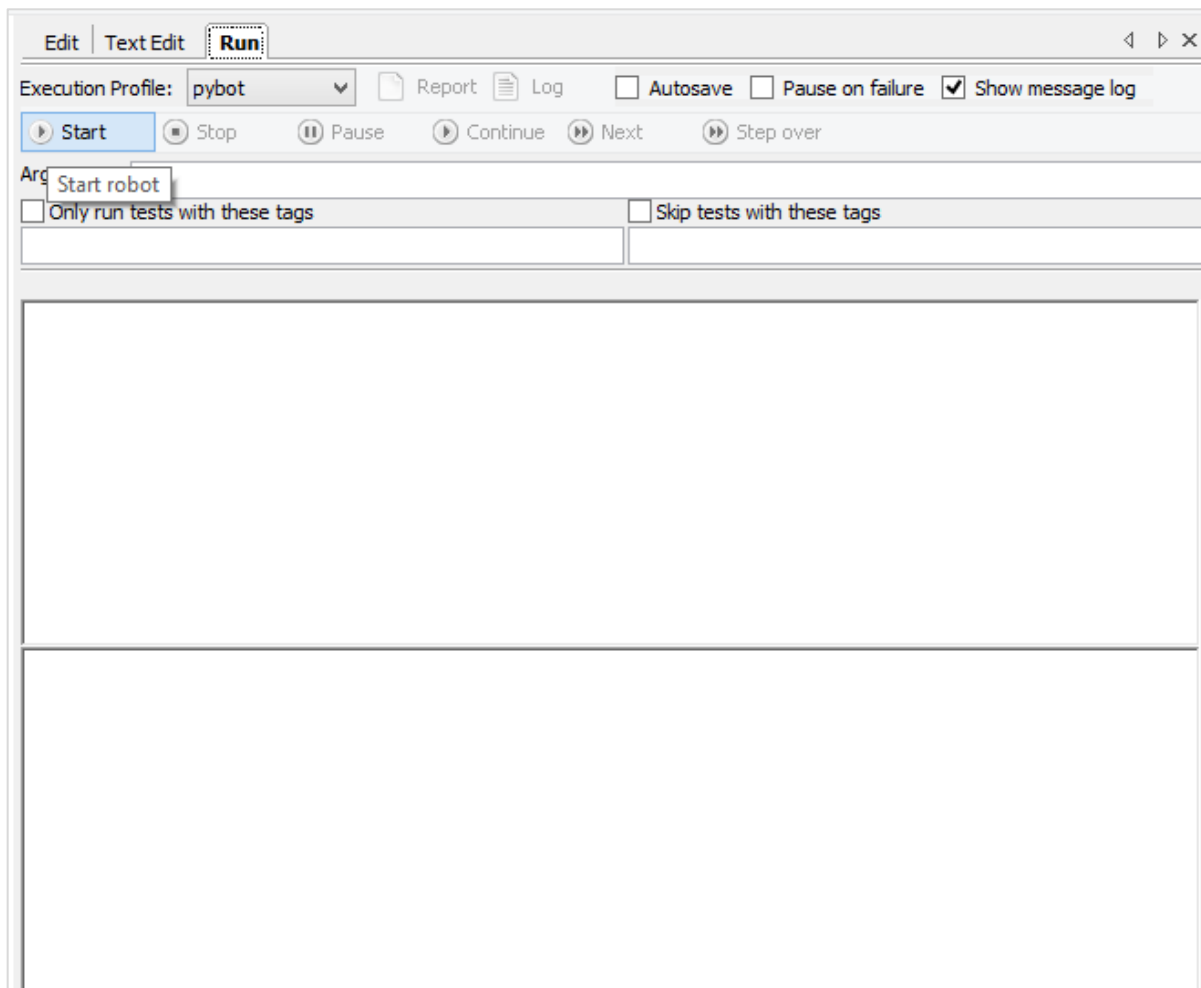
Based on the keywords specified in Edit, we can get the code in Text Edit as shown below:



```
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 Robot Framework Run window. At the top, the 'Run' tab is active. Below it, the 'Execution Profile' is set to 'pybot'. There are buttons for 'Start', 'Stop', 'Pause', 'Continue', 'Next', and 'Step over'. The 'Show message log' checkbox is checked. The 'Arguments' field is empty. Below the arguments, there are checkboxes for 'Only run tests with these tags' and 'Skip tests with these tags'. The main output area shows the following text:

```

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

```

Below the main output, there is a section for the 'Starting test' and 'Ending test' details:

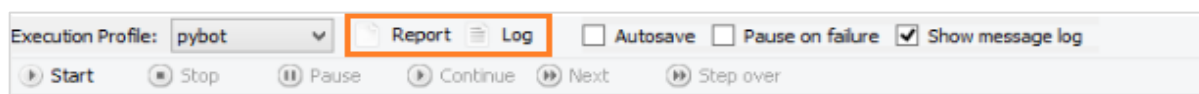
```

Starting test: FirstTestCase.TC1
20181008 10:12:14.524 : INFO : Welcome to our first test case
20181008 10:12:14.528 : INFO : In robot framework
20181008 10:12:14.532 : INFO : End of the test case
Ending test: FirstTestCase.TC1

```

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:

The screenshot shows a web browser window with the address bar displaying a local file path. The report title is 'FirstTestCase Test Report'. In the top right corner, there is a 'LOG' button and a timestamp indicating the report was generated 5 minutes and 32 seconds ago. The 'Summary Information' section lists the status as 'All tests passed', along with start, end, and elapsed times, and a link to the log file. The 'Test Statistics' section contains three tables: 'Total Statistics', 'Statistics by Tag', and 'Statistics by Suite'. All tables show 1 total test, 1 passed, and 0 failed. The 'Test Details' section at the bottom has tabs for 'Totals', 'Tags', 'Suites', and 'Search', with a 'Type' filter set to 'All Tests'.

FirstTestCase Test Report

Generated 20181008 10:12:14 GMT+05:30
5 minutes 32 seconds ago

Summary Information

Status: All tests passed
 Start Time: 20181008 10:12:14.324
 End Time: 20181008 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	
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 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:

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

REPORT

FirstTestCase Test Log

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

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

Tearardown

Edit Clear

Tags

<Add New>

Edit Clear

Timeout

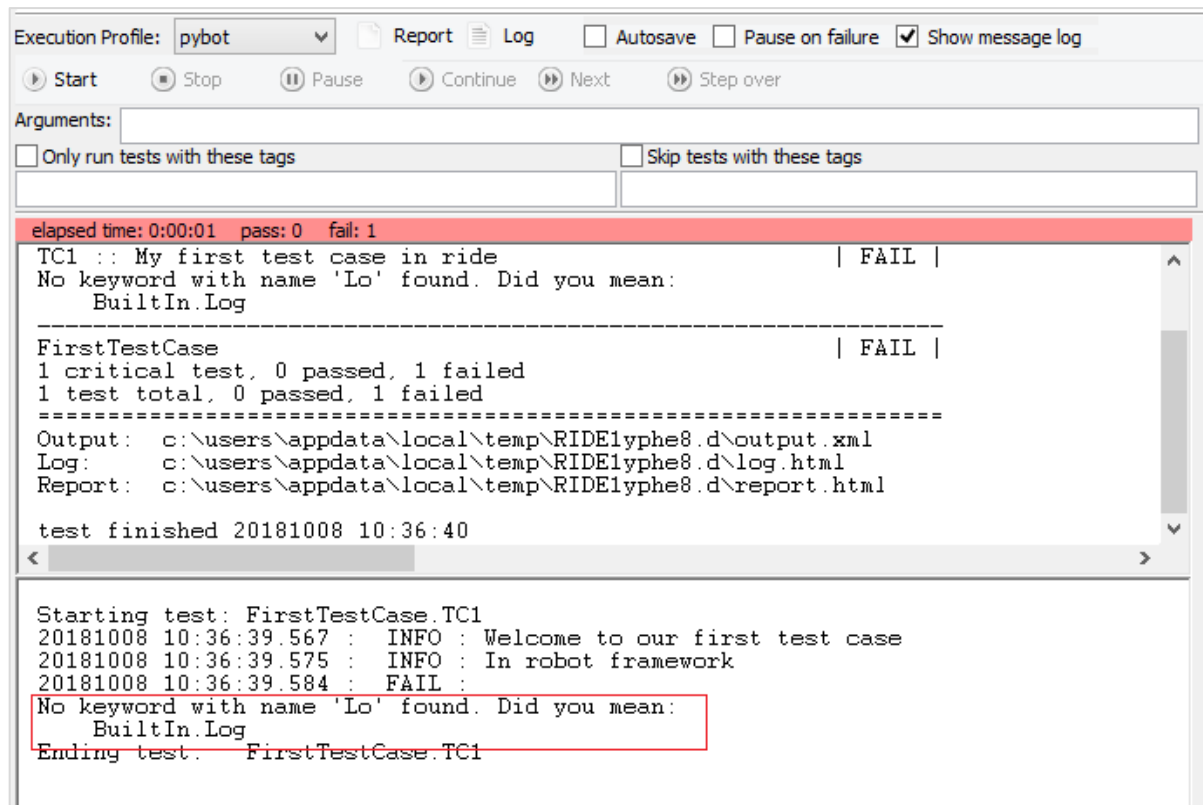
Edit Clear

Template

Edit Clear

1	Log	Welcome to our first test case		
2	Log	In robot framework		
3	Lo	End of the test case		
4				
5				
6				
7				
8				

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



```

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: 0:00:01  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

Summary Information

Status: 1 critical test failed
Start Time: 20181008 10:36:39.445
End Time: 20181008 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	
All Tests	1	0	1	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	0	1	00:00:00	

Test Details

☒ Totals
 ☐ Tags
 ☐ Suites
 ☐ Search

Type:

☐ Critical Tests
☐ All Tests

LOG

Generated
20181008 10:36:39 GMT+05:30
2 minutes 30 seconds ago

From Log

FirstTestCase Test Log

Test Statistics

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

REPORT

Generated
20181008 10:36:39 GMT+05:30
3 minutes 16 seconds ago

Test Execution Log

SUITE FirstTestCase

Full Name: FirstTestCase

Source: [C:\robotframework\FirstTestCase.robot](#)

Start / End / Elapsed: 20181008 10:36:39.445 / 20181008 10:36:39.592 / 00:00:00.147

Status: 1 critical test, 0 passed, **1 failed**
1 test total, 0 passed, **1 failed**

00:00:00.147

TEST TC1

Full Name: FirstTestCase.TC1

Documentation: My first test case in ride

Start / End / Elapsed: 20181008 10:36:39.561 / 20181008 10:36:39.588 / 00:00:00.027

Status: **FAIL** (critical)

Message: No keyword with name 'Lo' found. Did you mean:
BuiltIn.Log

00:00:00.027

KEYWORD BuiltIn.Log Welcome to our first test case

KEYWORD BuiltIn.Log In robot framework

KEYWORD Lo End of the test case

00:00:00.005

00:00:00.003

00:00:00.003

Start / End / Elapsed: 20181008 10:36:39.583 / 20181008 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.

6. Robot framework — 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:

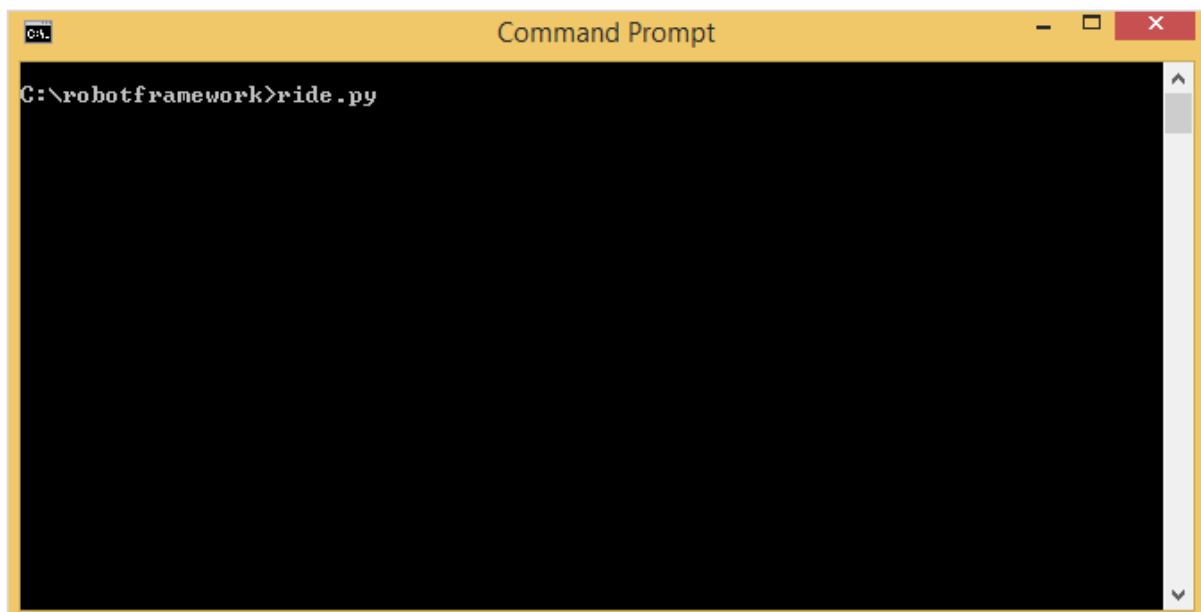
- Project Setup
- Importing Libraries
- Write test case in tabular format
- Using Tags for Executing Test Case
- Use Resource Files for Test Case

Project Setup

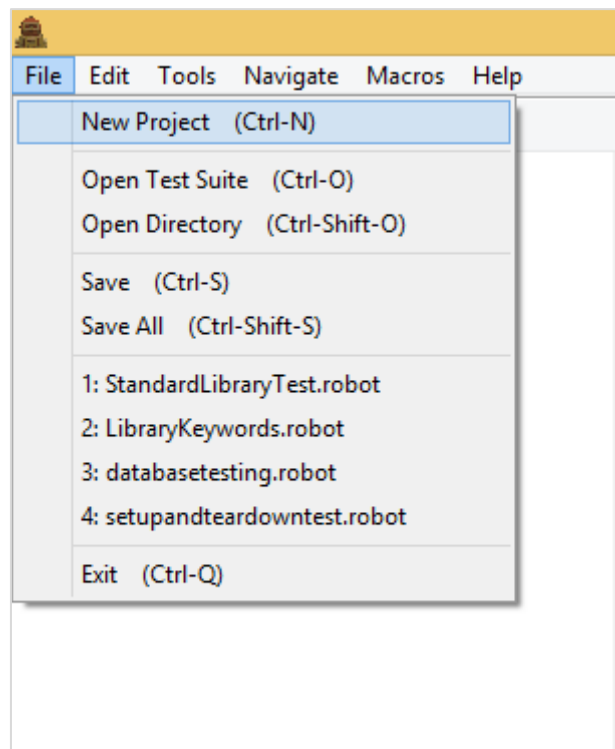
Run the command `ride.py` to start RIDE IDE.

Command

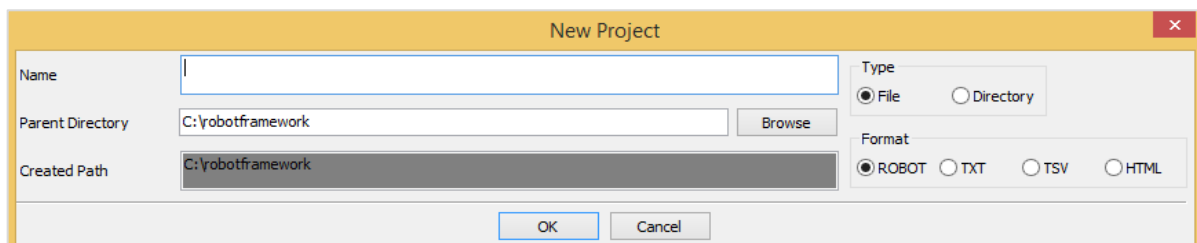
```
ride.py
```



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

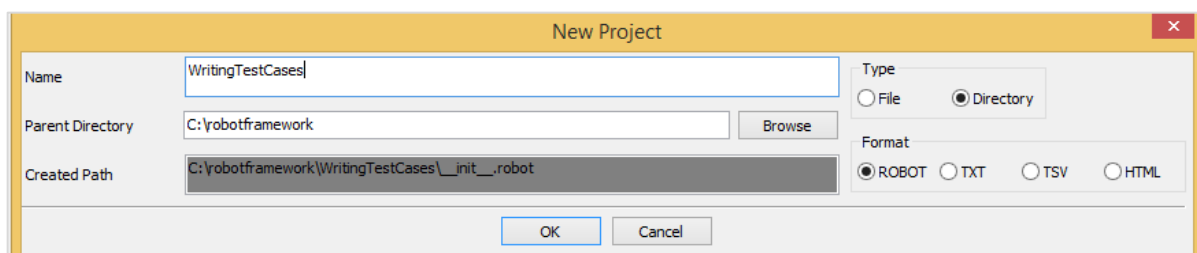


Upon clicking New Project, the screen will appear as shown below:

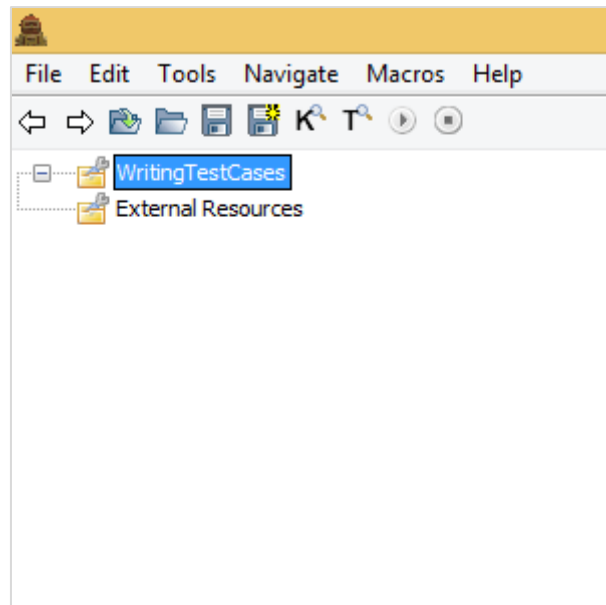


New Project shows the type as file or directory. By default, File is selected. We will click on Directory to create test suite, which can have many test suites in that directory. Each suite will have test-cases.

We will use the ROBOT format for now.

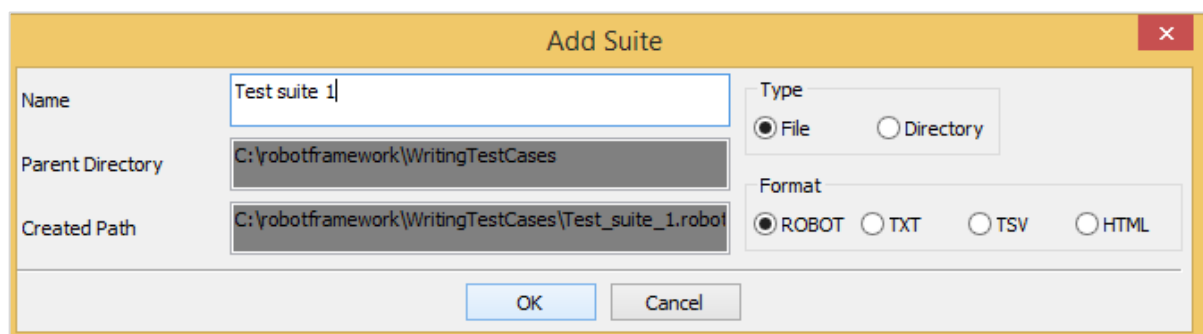
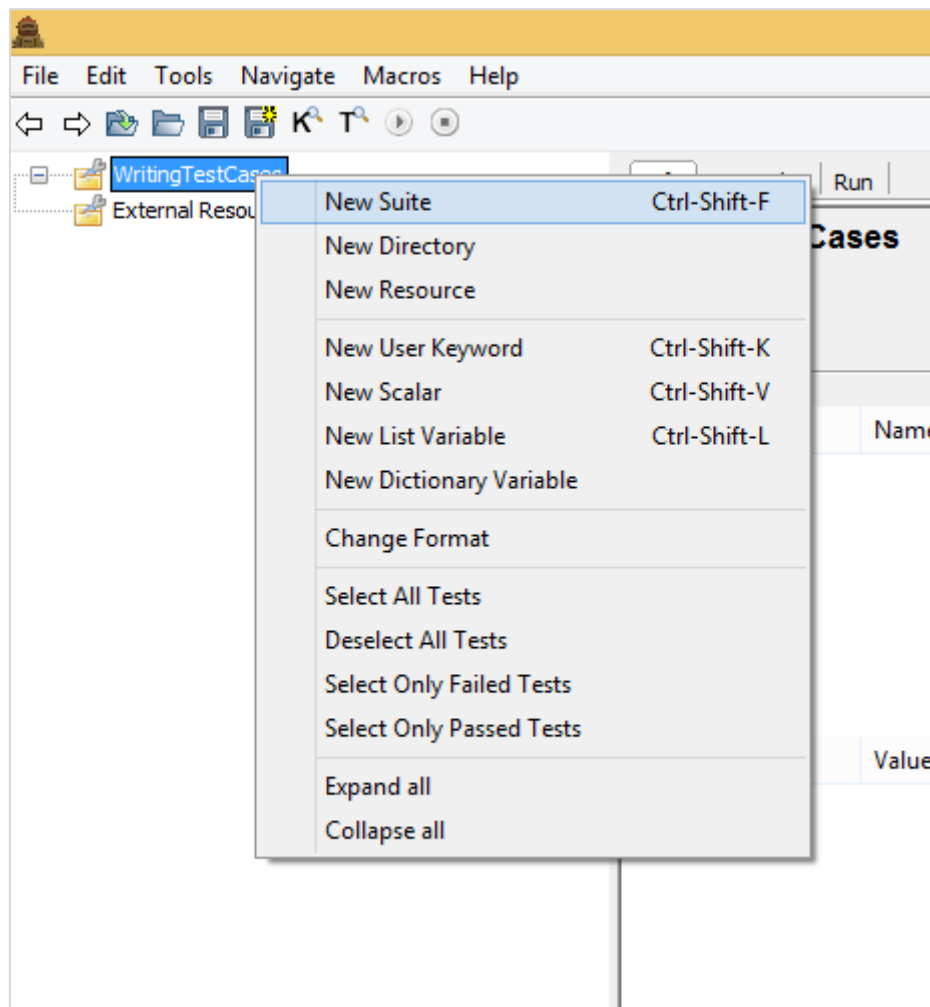


The Parent-Directory is the path where the *WritingTestCases* directory will be created. Click OK to save the test suite directory.



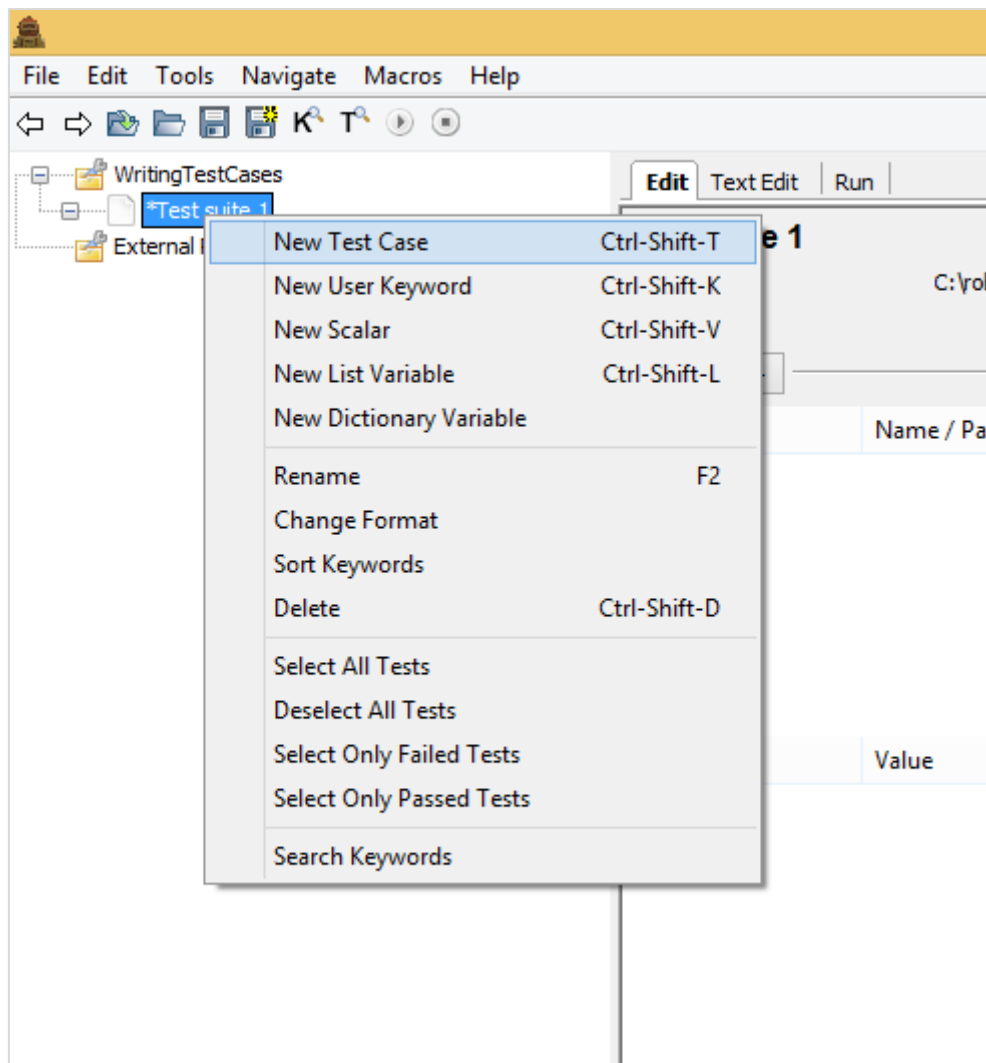
Right-click on the directory created and click on *New Suite*. You can also create sub directories with test suites in that.

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

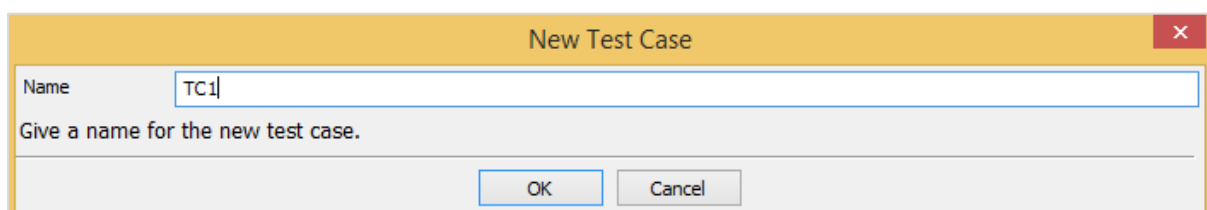


Click OK to save the Test suite.

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

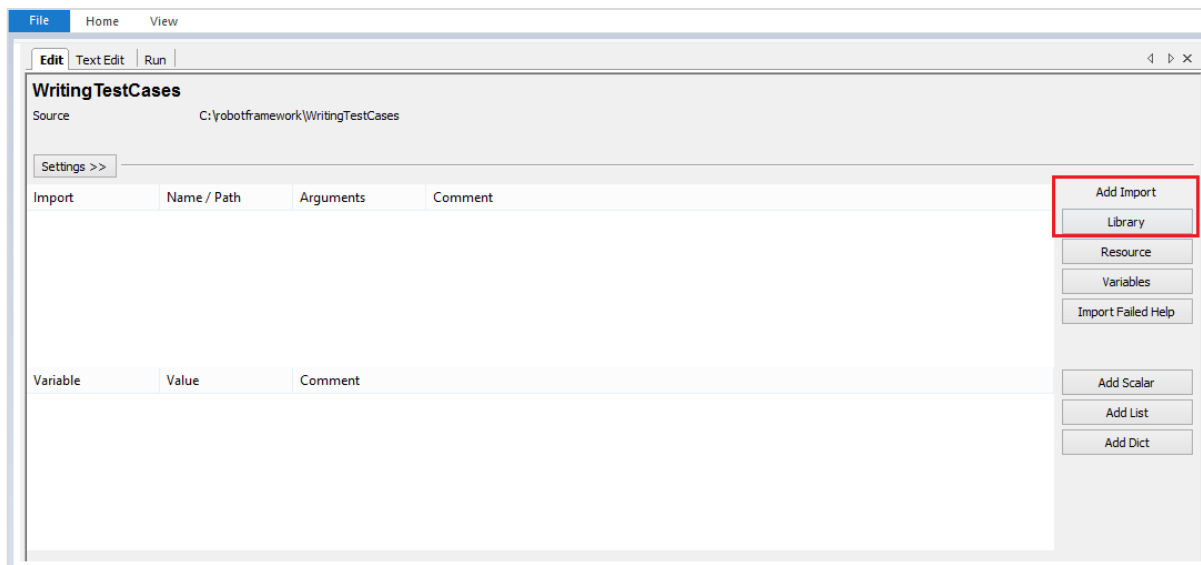
Robot Framework has its own built-in library, which need not be imported. But we need to interact with the browsers, databases, etc. To interact, we need to import the libraries.

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

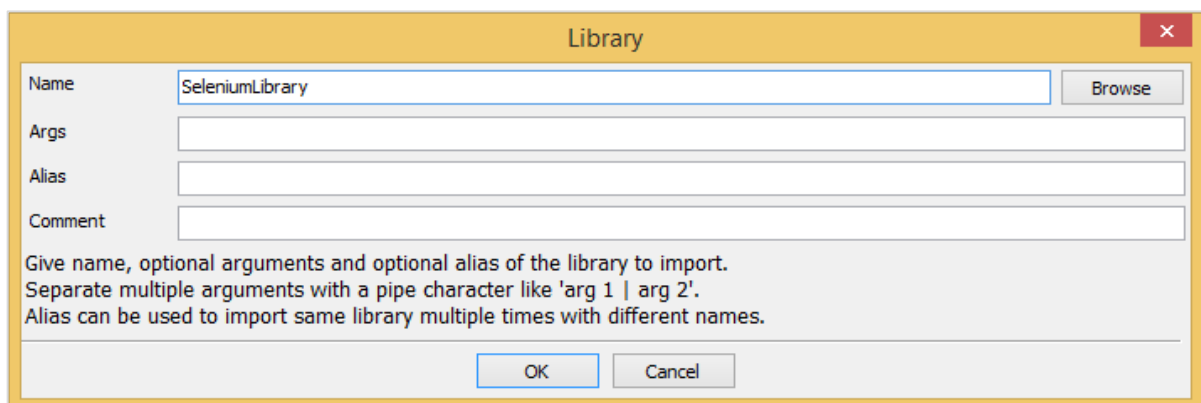
Android library Library for all your Android automation needs. It uses Calabash Android internally.	AnywhereLibrary Library for testing Single-Page Apps (SPA). Uses Selenium Webdriver and Appium internally.	AppiumLibrary Library for Android- and iOS-testing. It uses Appium internally.
Archive library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

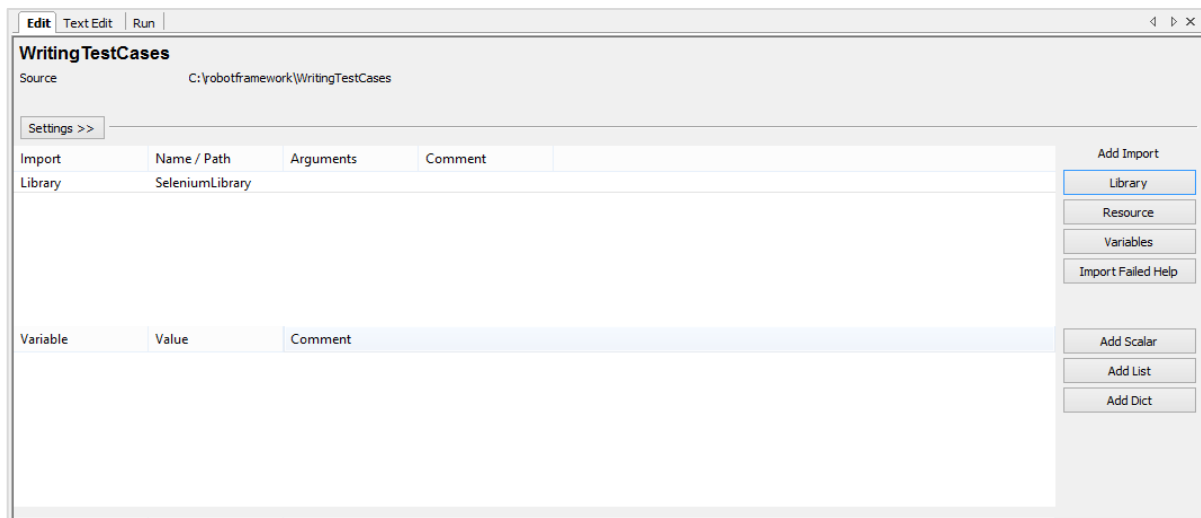


Click Library and enter the name of the library as shown below:

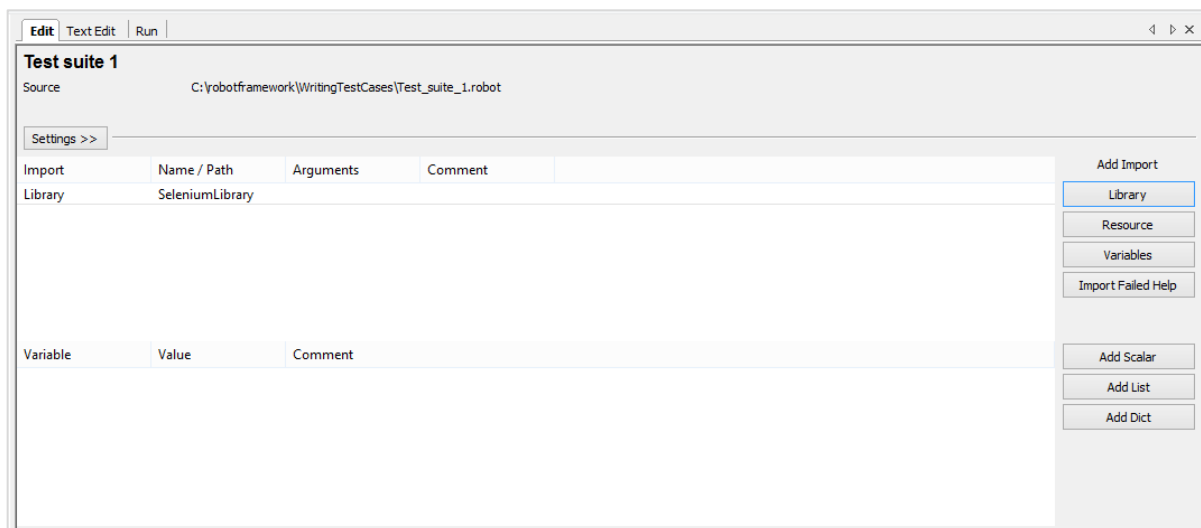


Click Ok to save the library.

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.

	Open Browser	https://www.tutorialspoint.com/	chrome		
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 Edit Clear

Setup Edit Clear

Teardown Edit Clear

Tags <Add New> Edit Clear

Timeout Edit Clear

Template Edit Clear

1	<code>\${a}</code>	Set Variable	Hi			
2	Log	<code>\${a}</code>				
3	<code>\${b}</code>	Set Variable If	<code>\${number}>0</code>	Yes	No	
4	Log	<code>\${b}</code>				
5						

*** Settings ***

Library SeleniumLibrary

*** Variables ***

`${number}` 100

*** Test Cases ***

TC1

Open Browser `https://www.tutorialspoint.com/` chrome

Maximize Browser Window

Close Browser

TC2

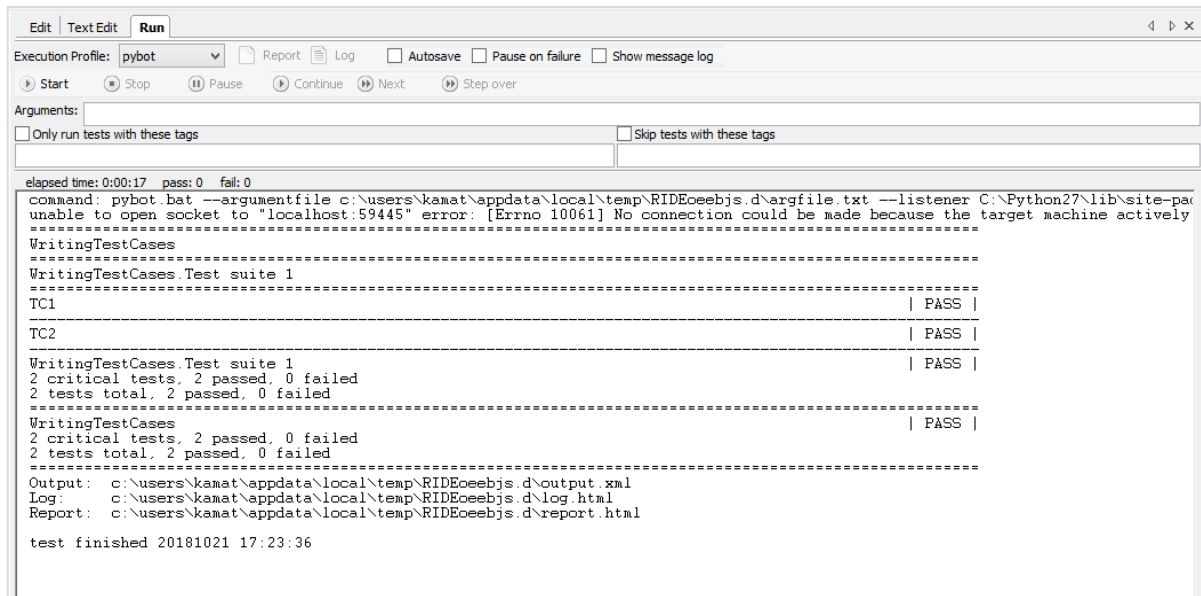
`${a}` Set Variable Hi

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:



The screenshot shows the Robot Framework Run dialog box with the 'Run' button highlighted. Below the dialog, the execution output is displayed in a text area. The output shows the command being executed, the listener, and the results of the test suite. The test suite 'Test suite 1' contains two test cases, TC1 and TC2, both of which passed. The output also shows the number of critical tests, passed tests, and failed tests. The test finished on 20181021 17:23:36.

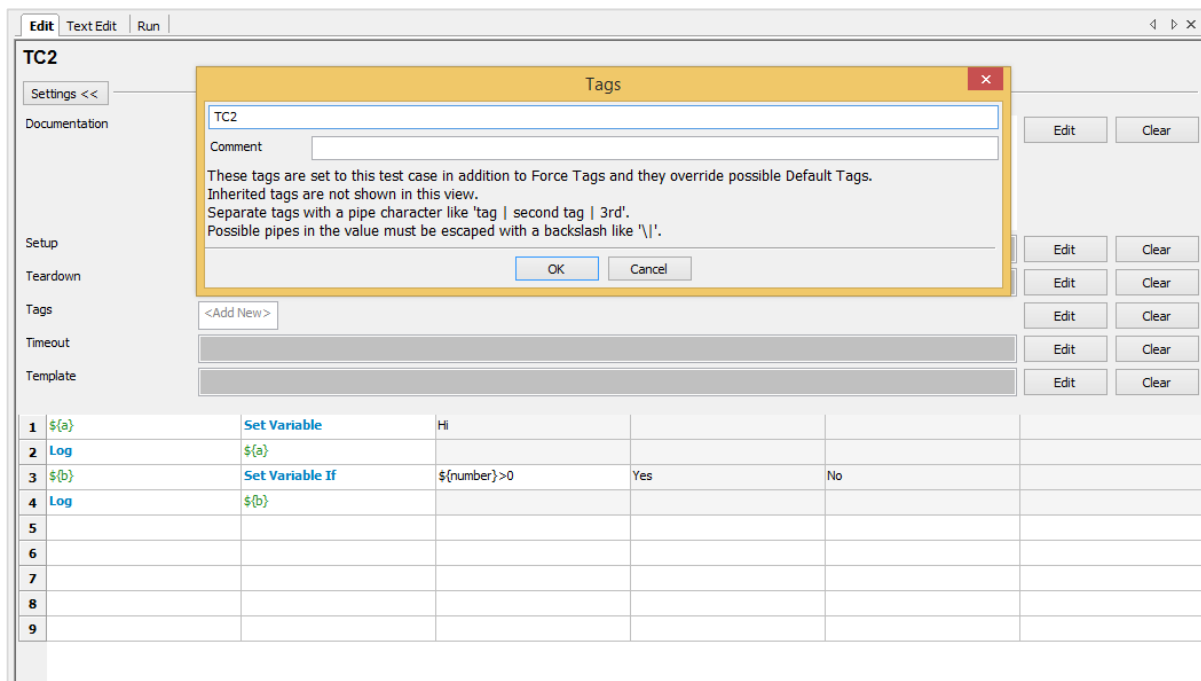
```

command: pybot.bat --argumentfile c:\users\kamat\appdata\local\temp\RIDEeebjs.d\argfile.txt --listener C:\Python27\lib\site-pa
unable to open socket to "localhost:59445" error: [Errno 10061] No connection could be made because the target machine actively
WritingTestCases
WritingTestCases.Test suite 1
TC1
TC2
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\RIDEeebjs.d\output.xml
Log: c:\users\kamat\appdata\local\temp\RIDEeebjs.d\log.html
Report: c:\users\kamat\appdata\local\temp\RIDEeebjs.d\report.html
test finished 20181021 17:23:36

```

Using Tags for Executing Test Case

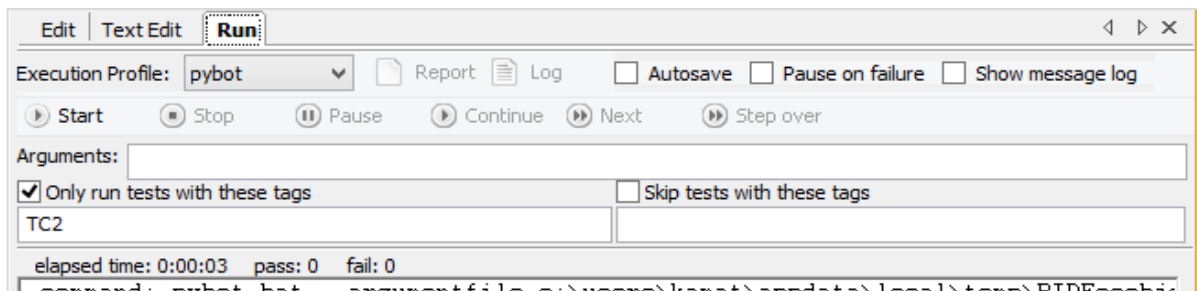
In case you want to run only test case TC2, you can tag the same. Click on the test case and click Edit across Tags as shown below:



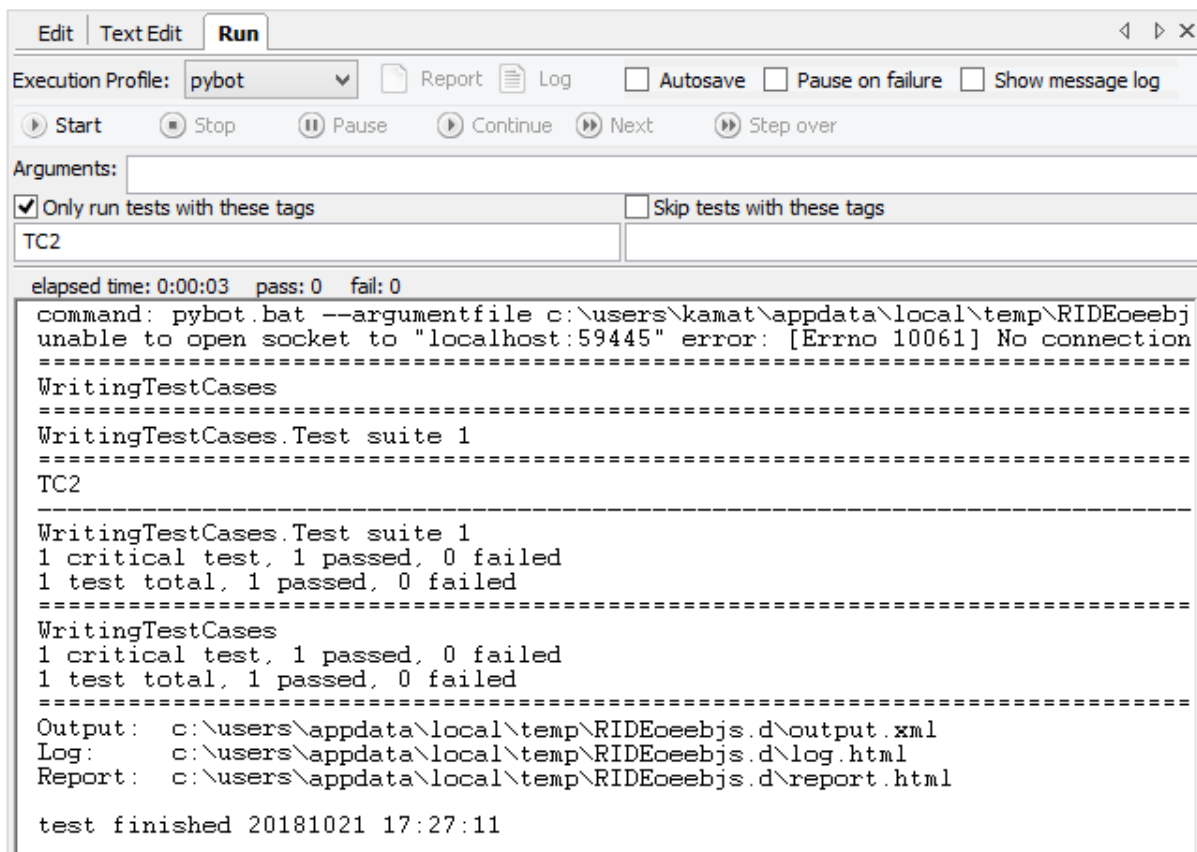
The screenshot shows the Robot Framework interface with the 'Tags' dialog box open for test case TC2. The dialog box has a title bar 'Tags' and a close button. It contains a text area for 'Comment' and a text area for 'Tags'. The 'Tags' text area contains the text 'TC2'. The 'Comment' text area contains the text 'These tags are set to this test case in addition to Force Tags and they override possible Default Tags. Inherited tags are not shown in this view. Separate tags with a pipe character like 'tag | second tag | 3rd'. Possible pipes in the value must be escaped with a backslash like '\\'. The dialog box has 'OK' and 'Cancel' buttons. The background shows the 'Tags' section of the test case editor with a table of tags.

1	2	3	4	5	6	7	8	9
1	\$(a)	Set Variable	Hi					
2	Log	\$(a)						
3	\$(b)	Set Variable If	\$(number)>0	Yes	No			
4	Log	\$(b)						
5								
6								
7								
8								
9								

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.



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:

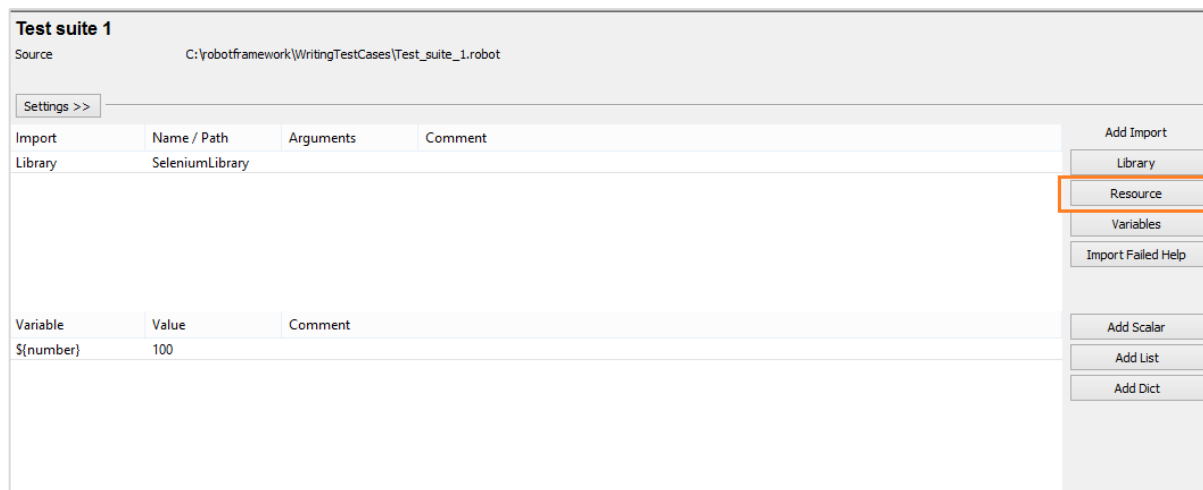
```
*** Settings ***
Library           SeleniumLibrary

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

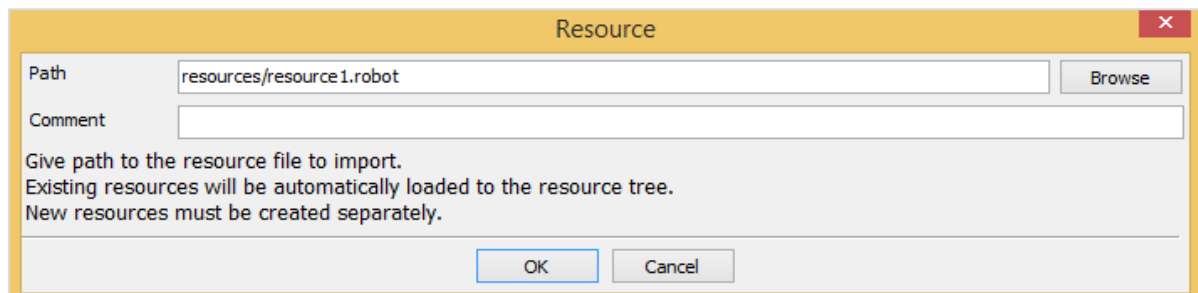
*** Keywords ***
Test Browser
    Open Browser    ${url}        ${browser}
    Maximize Browser Window
```

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.

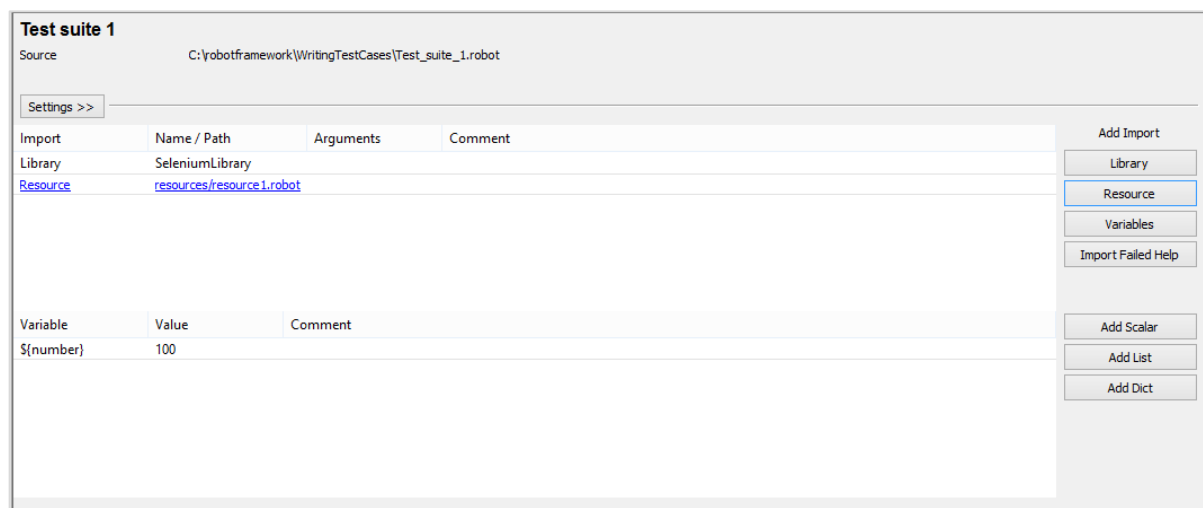
Select the test suite. On the left side, click on resource option as shown below:



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:



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 the following structure:

- WritingTestCases
 - Test suite 1
 - Test Case Icon \${number}
 - Test Case Icon TC1
 - Test Case Icon TC2
 - Resources
 - browseropen.robot
 - resource1.robot
 - resource1.robot (selected)
 - Test Case Icon \${url}
 - Test Case Icon \${browser}
 - Test Browser
 - External Resources

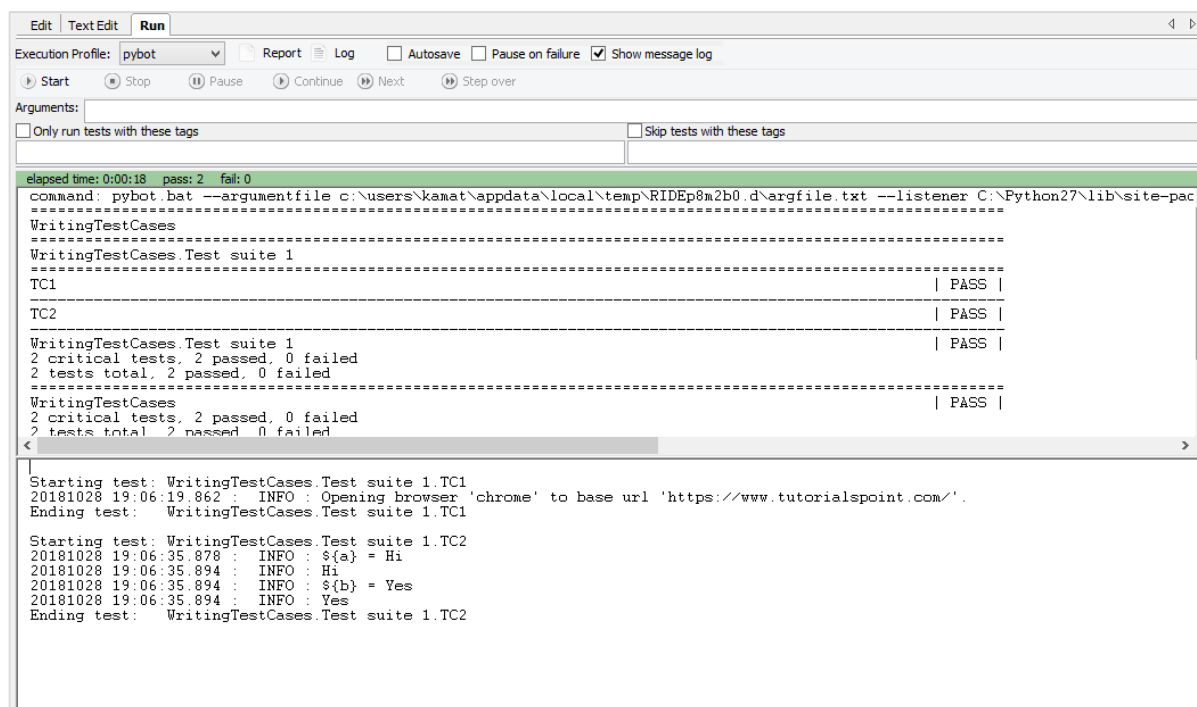
On the right, the 'Text Edit' window shows the content of the selected resource file:

```

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:



```

command: pybot.bat --argumentfile c:\users\kamat\appdata\local\temp\RIDEp8m2b0.d\argfile.txt --listener C:\Python27\lib\site-pac
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 19:06:19.862 : 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 19:06:35.878 : INFO : ${a} = Hi
20181028 19:06:35.894 : INFO : Hi
20181028 19:06:35.894 : INFO : ${b} = Yes
20181028 19:06:35.894 : 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

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 19:06:19.038 / 20181028 19:06:35.925 / 00:00:16.887

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 19:06:19.830 / 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

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

Conclusion

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

7. Robot framework — Keyword and Data Driven Test Cases

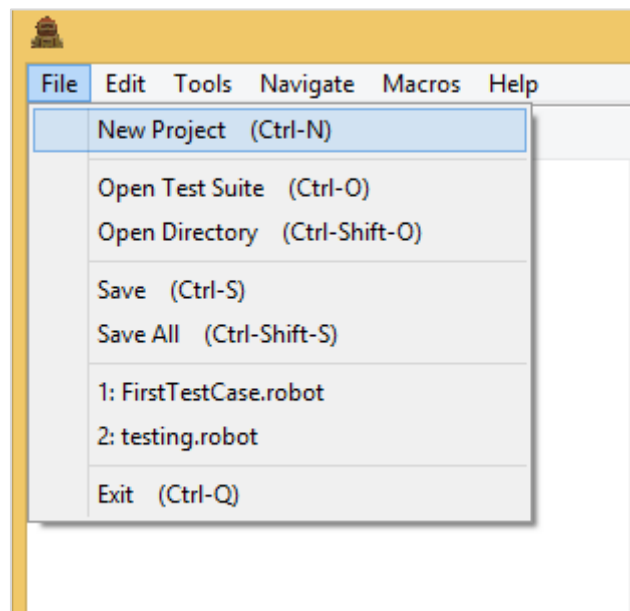
The workflow of a test-case can be tested using keyword or data driven style. In case you want to test the workflow with different inputs, the same can be done using data driven test cases. We will work on an example to go through the following test case approaches:

- Keyword Driven style
- Data Driven style

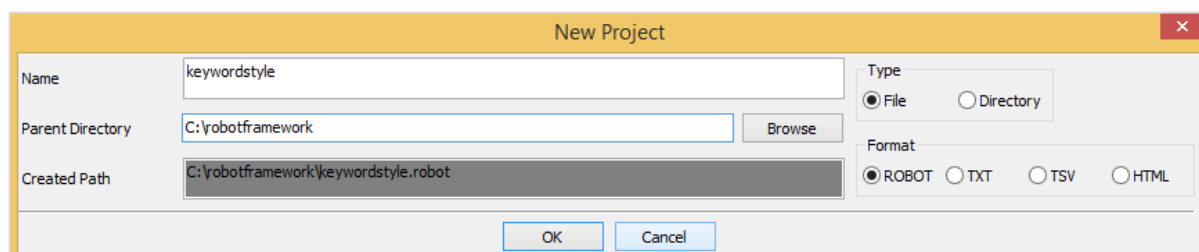
Keyword Driven Style

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

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

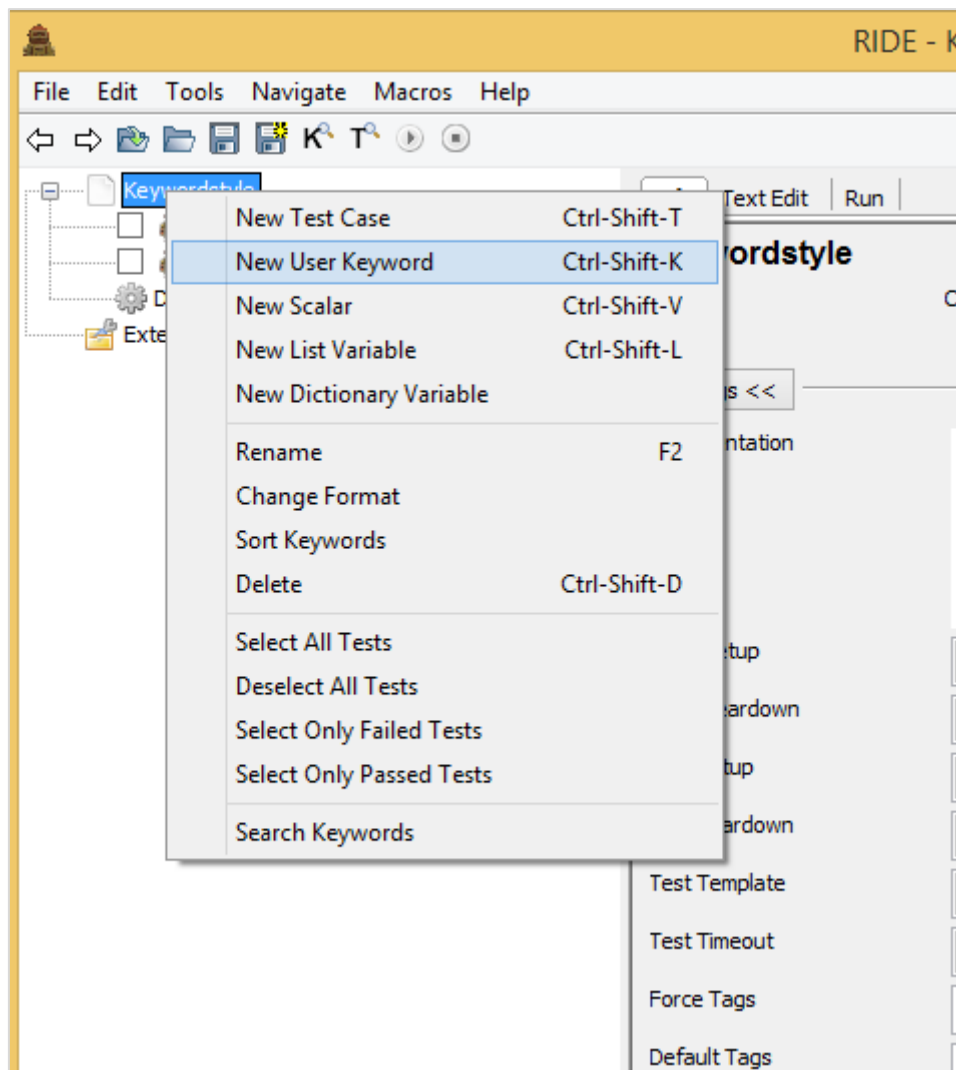


Click on *New Project* and give a name to your project.

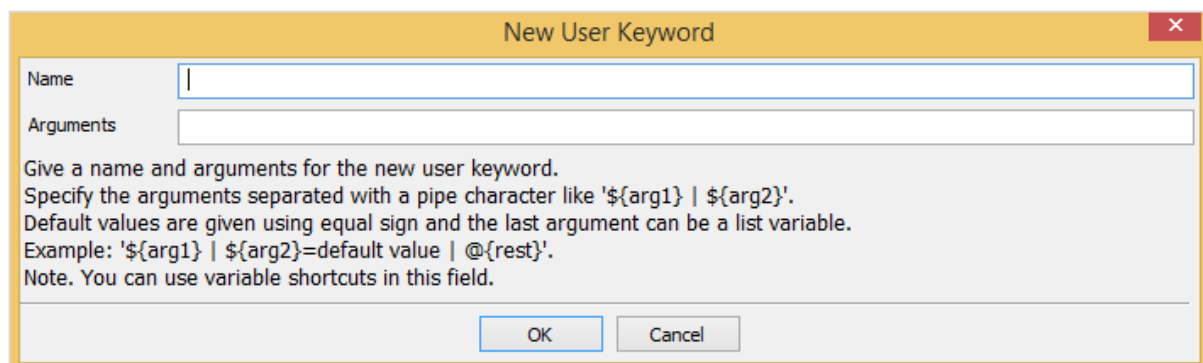


The name given to the project is *keywordstyle*. Click *OK* to save the project. In this project, we will create a user keyword as shown below.

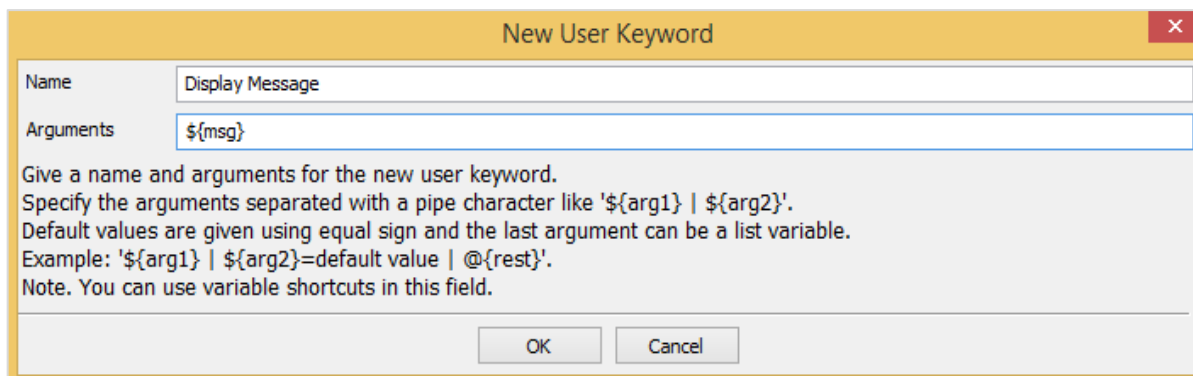
Right-click on the name of the project and click on *New User Keyword* as shown below:



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}`.



New User Keyword

Name:

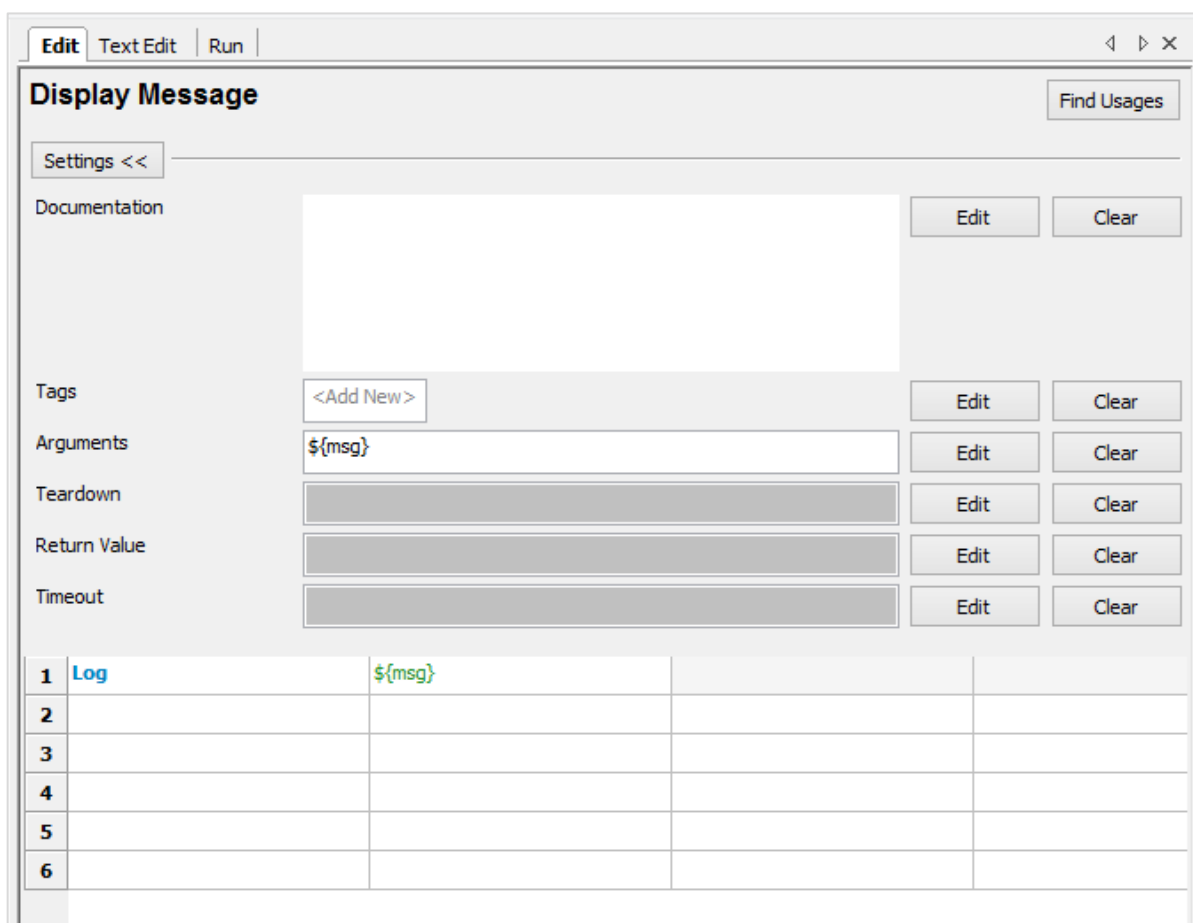
Arguments:

Give a name and arguments for the new user keyword.
Specify the arguments separated with a pipe character like '`${arg1} | ${arg2}`'.
Default values are given using equal sign and the last argument can be a list variable.
Example: '`${arg1} | ${arg2}=default value | @rest`'.
Note. You can use variable shortcuts in this field.

OK Cancel

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:



Display Message Find Usages

Settings <<

Documentation

Tags: <Add New> Edit Clear

Arguments: Edit Clear

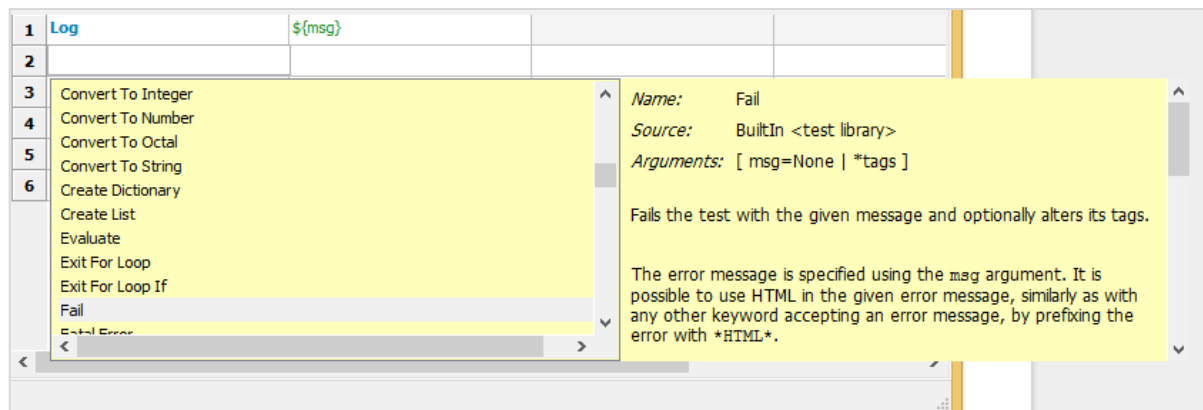
Teardown: Edit Clear

Return Value: Edit Clear

Timeout: Edit Clear

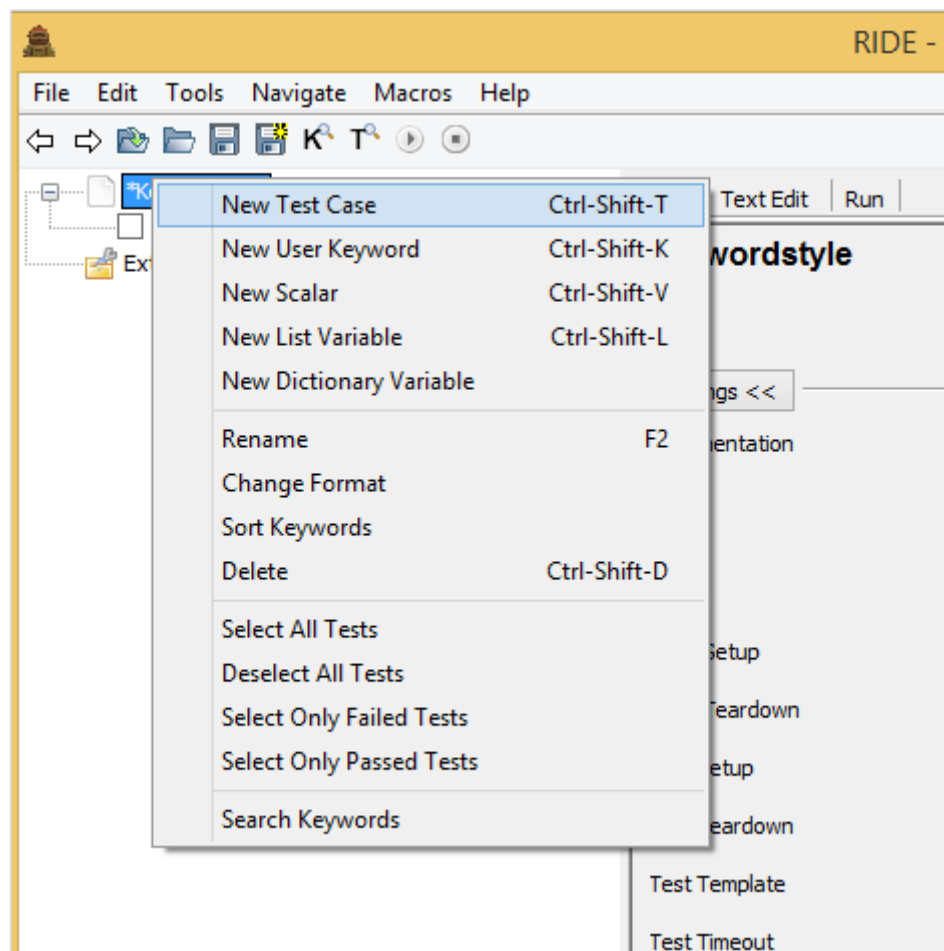
1	Log	<code>\${msg}</code>		
2				
3				
4				
5				
6				

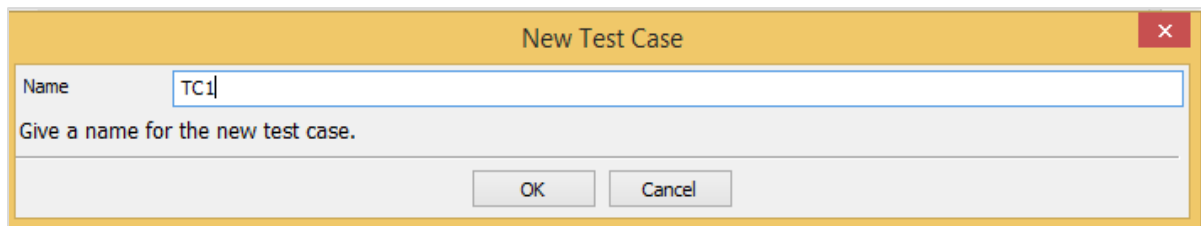
To get more keywords available with Robot framework, press ctrl + space bar in the table column as shown below:



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*:



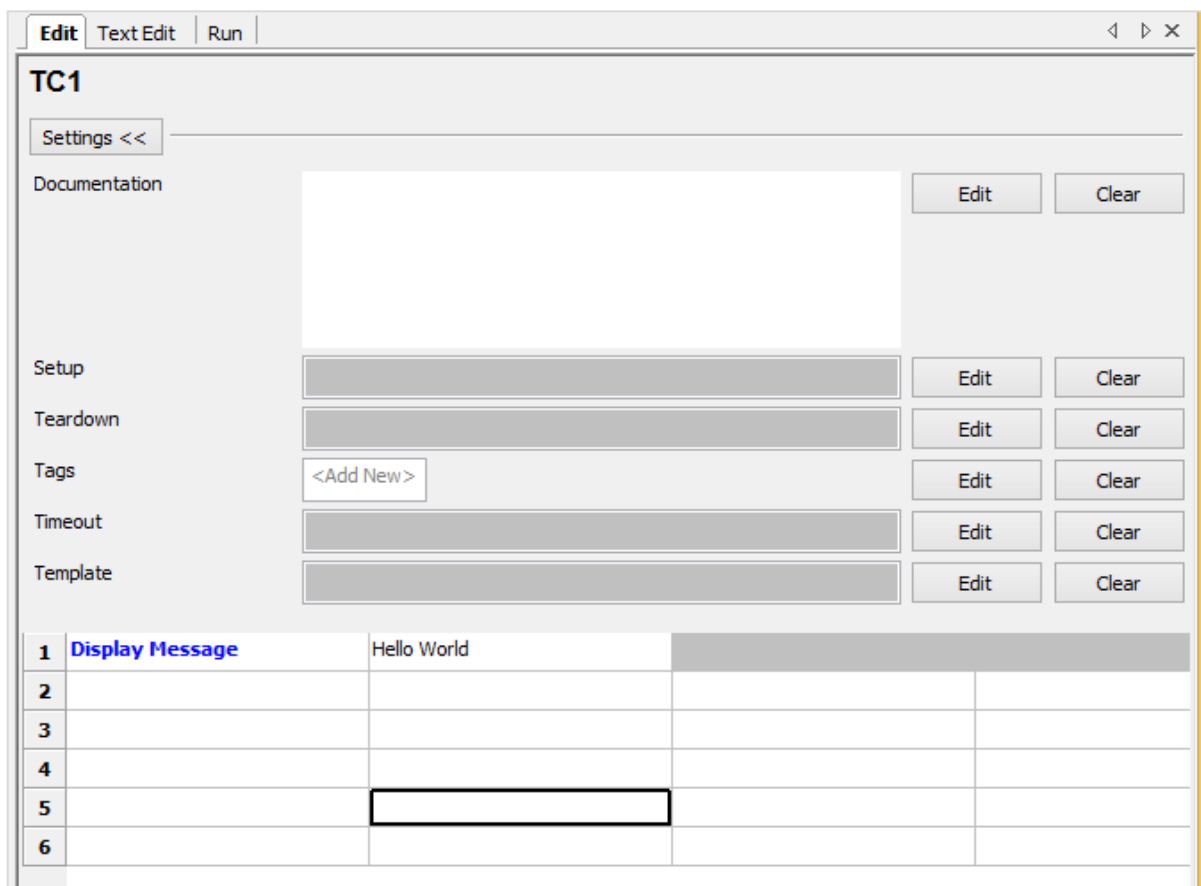


A dialog box titled "New Test Case" with a close button (X) in the top right corner. It contains a text input field labeled "Name" with the text "TC1" entered. Below the input field is a label "Give a name for the new test case." At the bottom are two buttons: "OK" and "Cancel".

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:



The Test Case Editor window shows a test case titled "TC1". It has tabs for "Edit", "Text Edit", and "Run". The "Edit" tab is active. On the left is a "Settings <<" button. The main area is divided into sections: "Documentation", "Setup", "Teardown", "Tags", "Timeout", and "Template". Each section has a text input field and "Edit" and "Clear" buttons. The "Tags" section has a "<Add New>" button. Below these sections is a table with 6 rows and 4 columns. The first row is highlighted in blue and contains the keyword "Display Message" in the first column and the value "Hello World" in the second column. The other rows are empty.

	Keyword	Value	
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\RIDEcdk0i.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\RIDEcdk0i.d\output.xml
Log:      c:\users\appdata\local\temp\RIDEcdk0i.d\log.html
Report:   c:\users\appdata\local\temp\RIDEcdk0i.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:

TC2

Settings <<

Documentation

Setup

Teardown

Tags

Timeout

Template

1			
2			
3			
4			
5			
6			
7			

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

Template

Comment

Specifies the template keyword to use.
The test itself will contain only data to use as arguments to that keyword.

OK Cancel

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

Template

Display Message

Comment

Specifies the template keyword to use.
The test itself will contain only data to use as arguments to that keyword.

OK Cancel

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*.

TC2

Settings <<

Documentation

Setup

Teardown

Tags

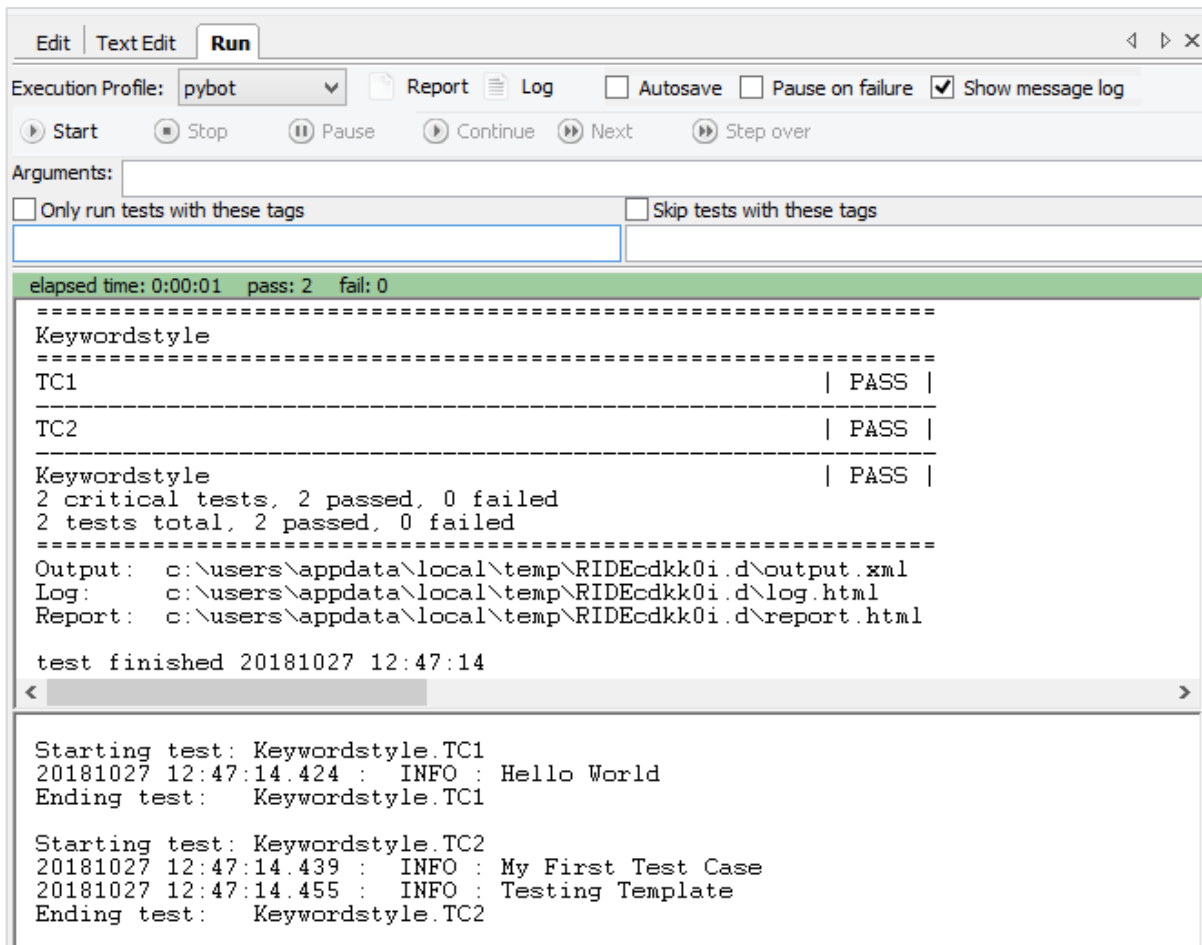
Timeout

Template

1	My First Test Case			
2	Testing Template			
3				
4				
5				
6				
7				

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.



```

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: 0:00:01 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

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.

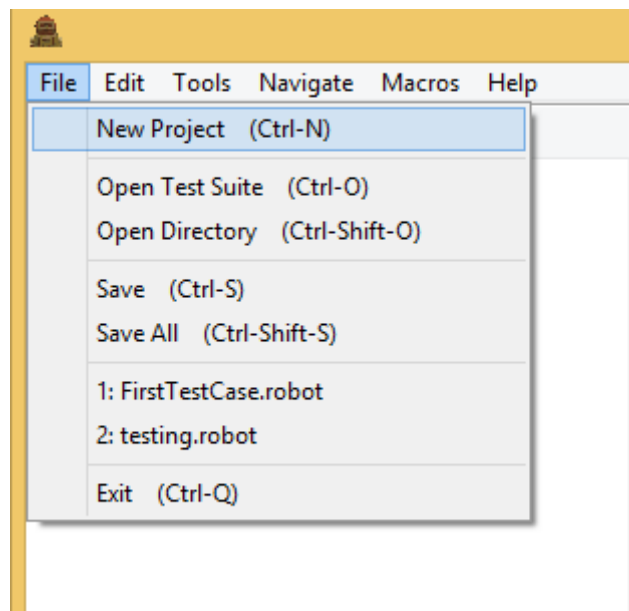
8. Robot — Working With Browsers Using Selenium Library

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

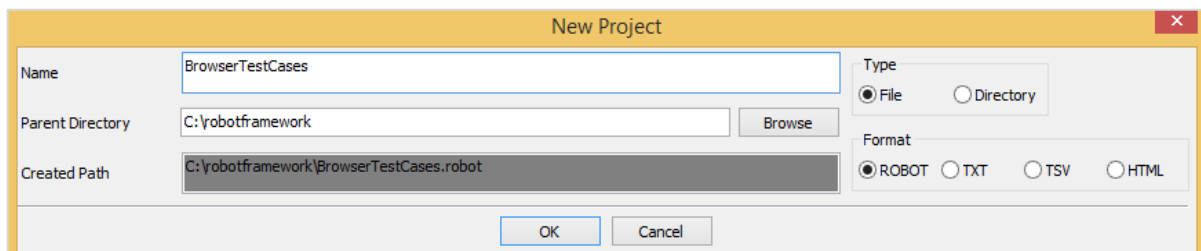
- Project setup in Ride
- Import Selenium Library
- Test case using Chrome Browser
- Test case using Firefox Browser

Project Setup In Ride

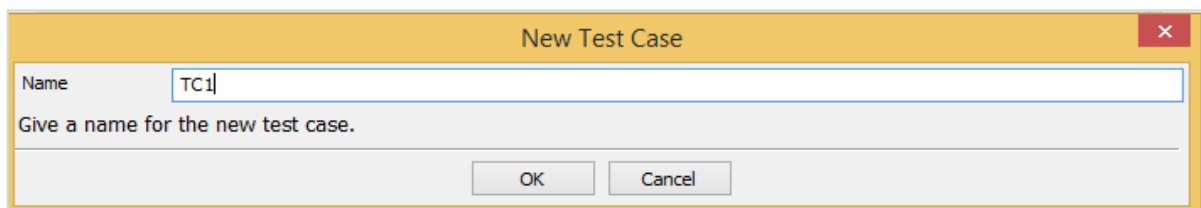
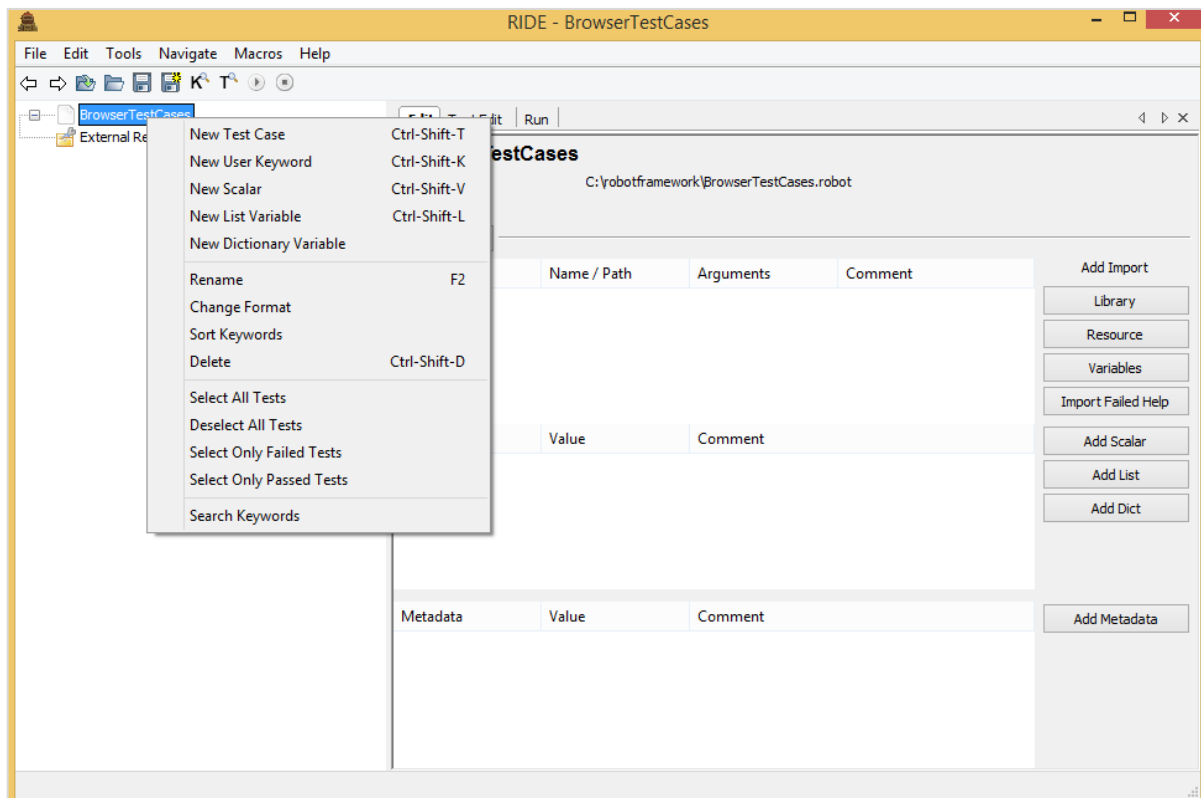
We will first create a project in Ride to work with browsers. Open ride using ride.py from the command line.



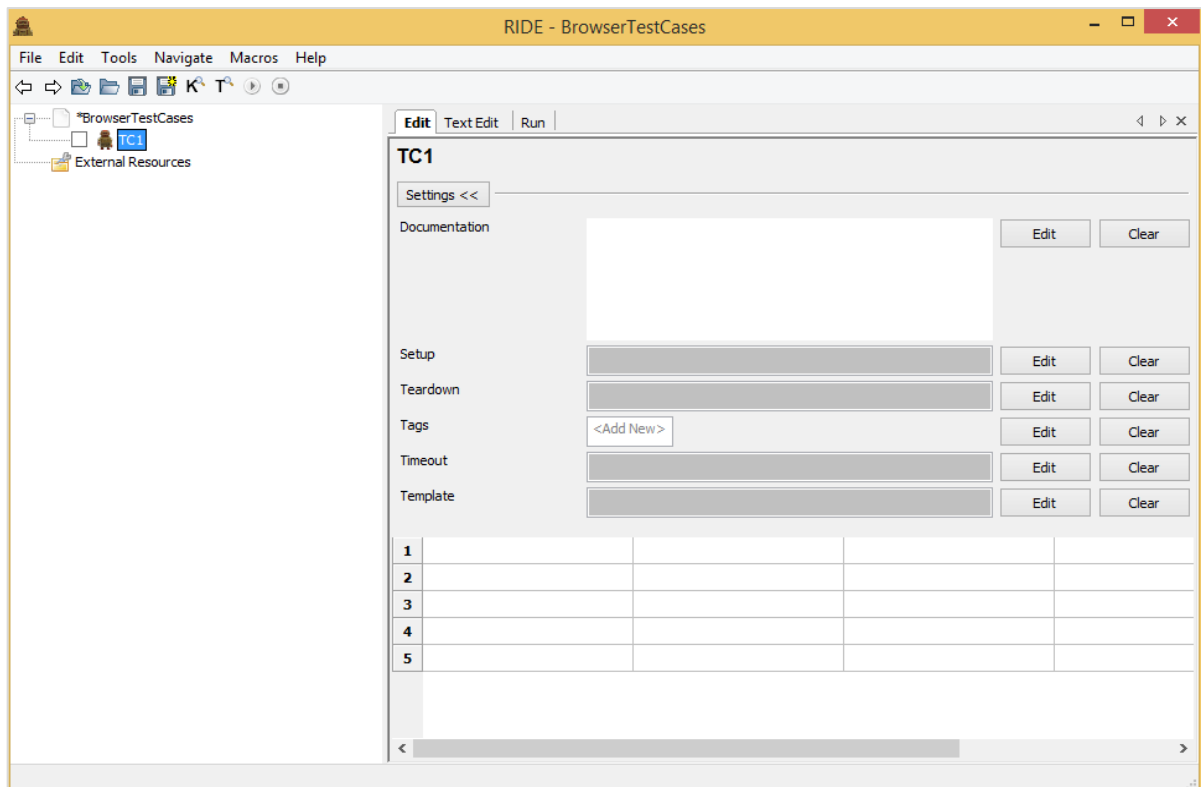
Click on *New Project* and give name to your project.



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.

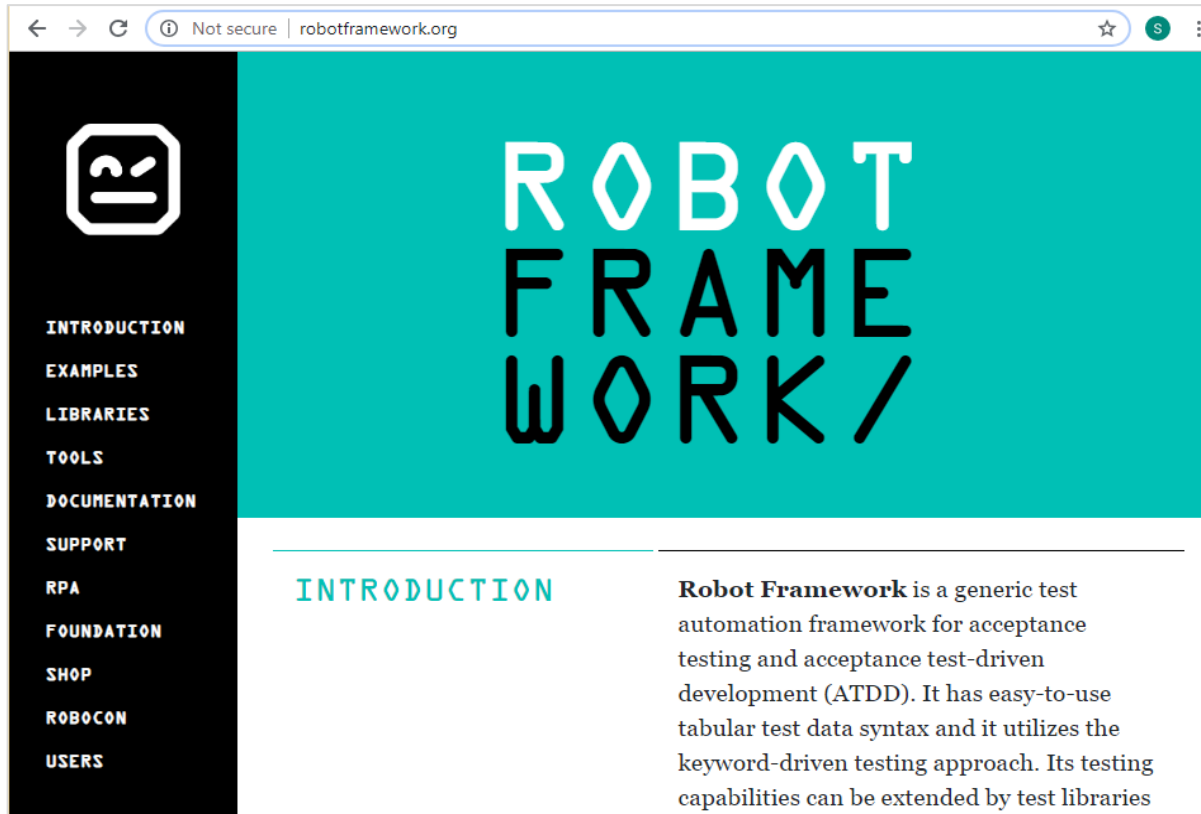


We are done with the project setup. Now, we will write test cases for the browser.

Import Selenium Library

To work with browsers, we need selenium library to be imported in robot. We can do that as follows:

Go to <http://robotframework.org/>



On the left side, select the LIBRARIES option.



- INTRODUCTION
- EXAMPLES
- LIBRARIES
- TOOLS
- DOCUMENTATION
- SUPPORT
- RPA
- FOUNDATION
- SHOP
- ROBOCON
- USERS

LIBRARIES

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.

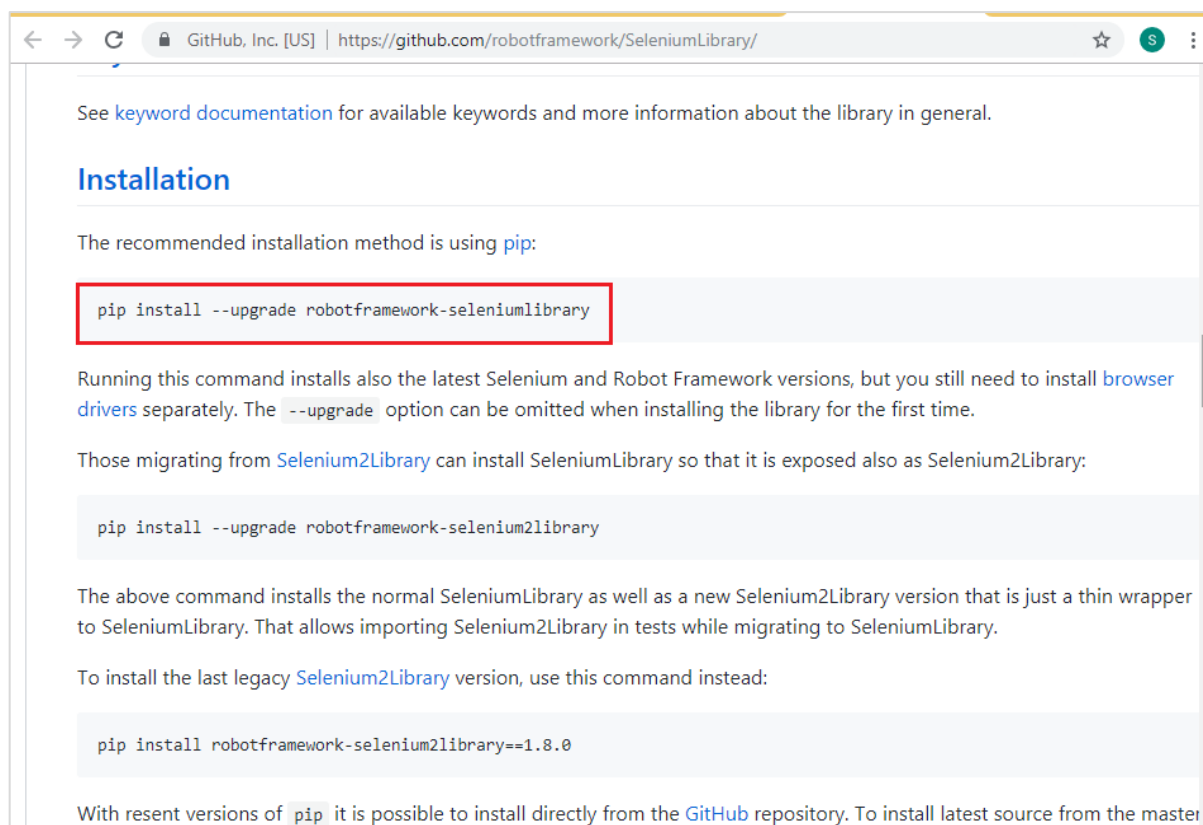
STANDARD	EXTERNAL	OTHER
Builtin Provides a set of often needed generic keywords. Always automatically available without imports. OperatingSystem	Dialogs Provides means for pausing the test execution and getting input from users. Remote	Collections Provides a set of keywords for handling Python lists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

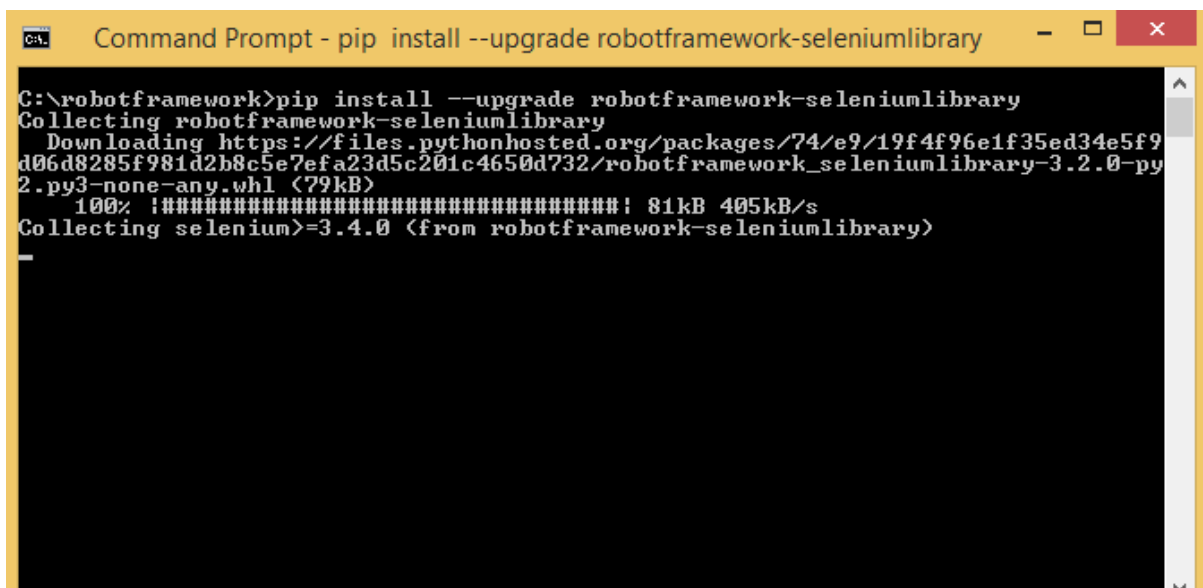
```
pip install robotframework-selenium2library==1.8.0
```

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
```

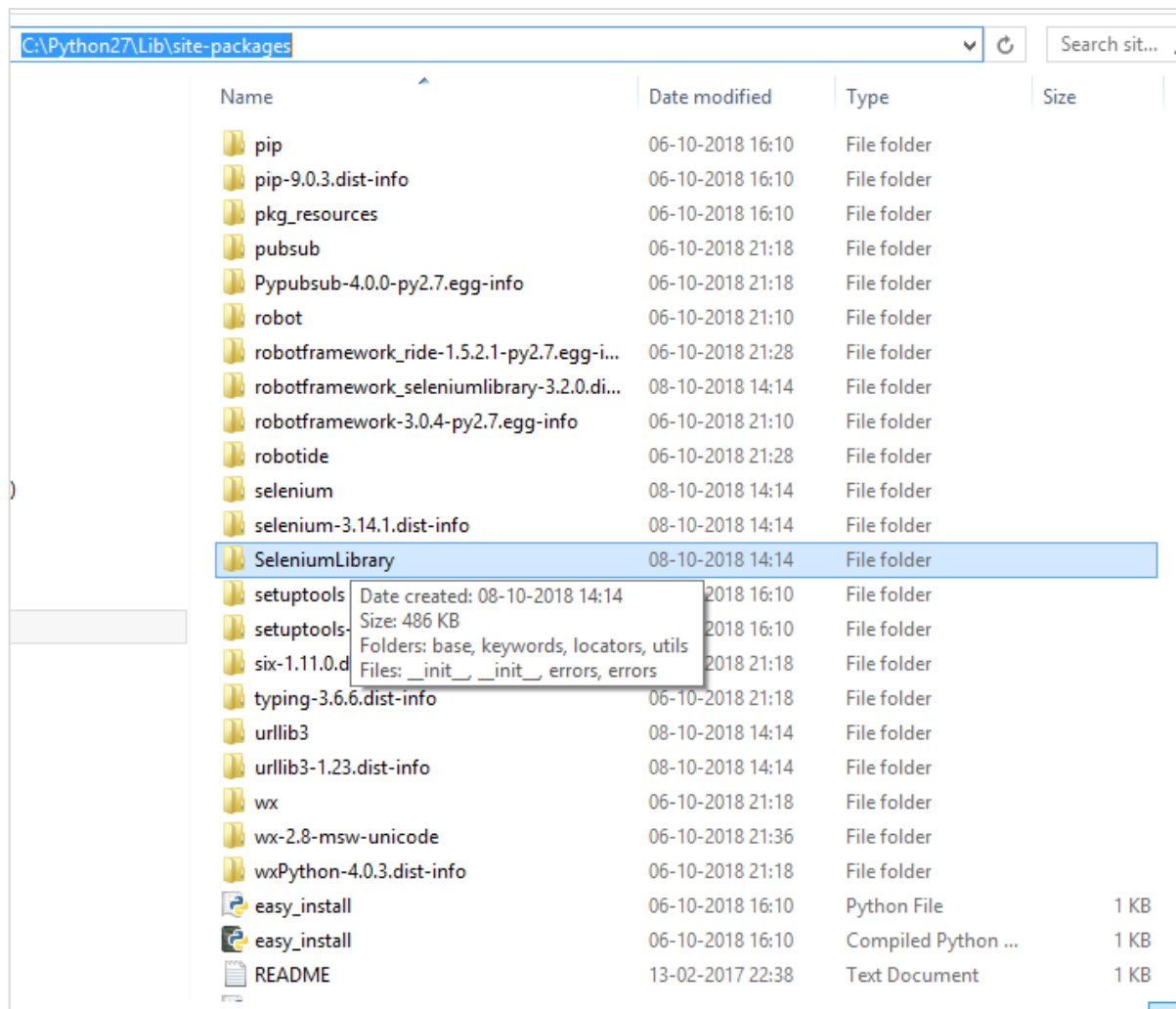


```

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)

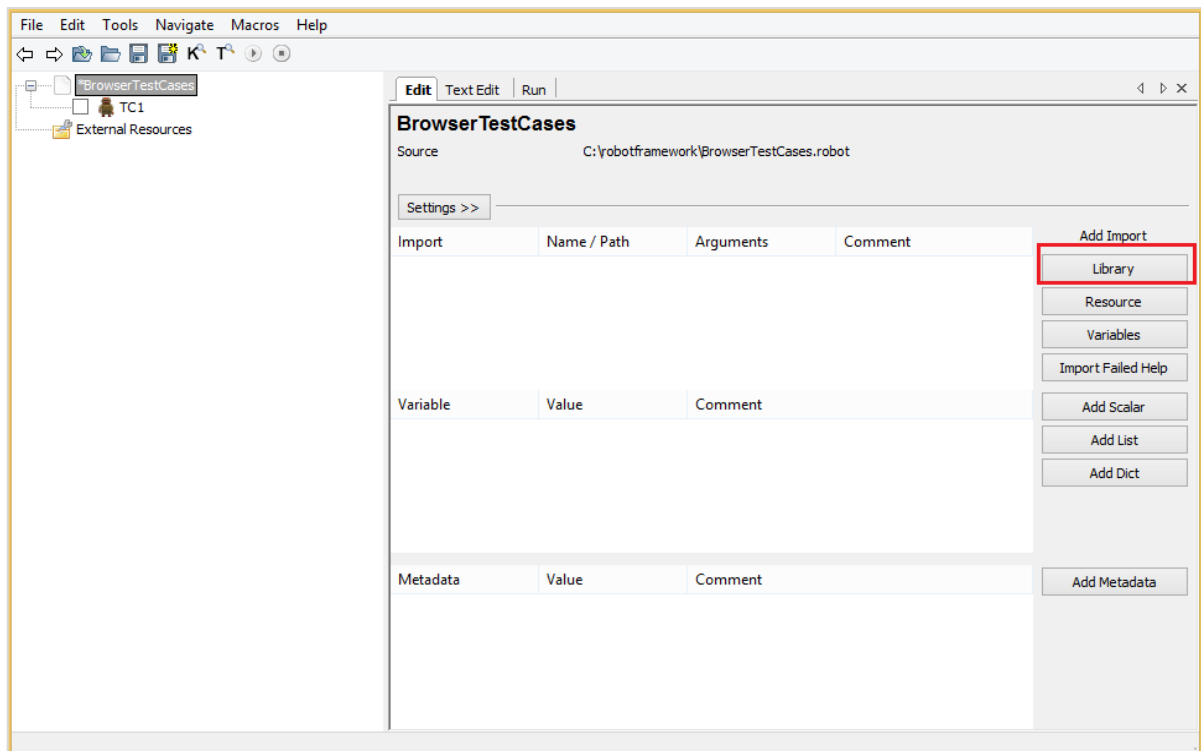
```


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

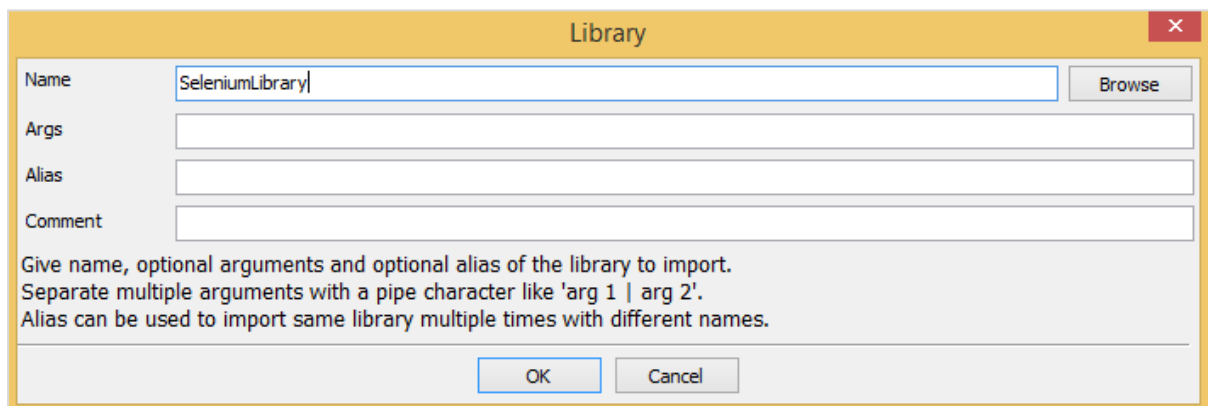


Once the installation is done, we have to import the library in Ride as shown in the below steps.

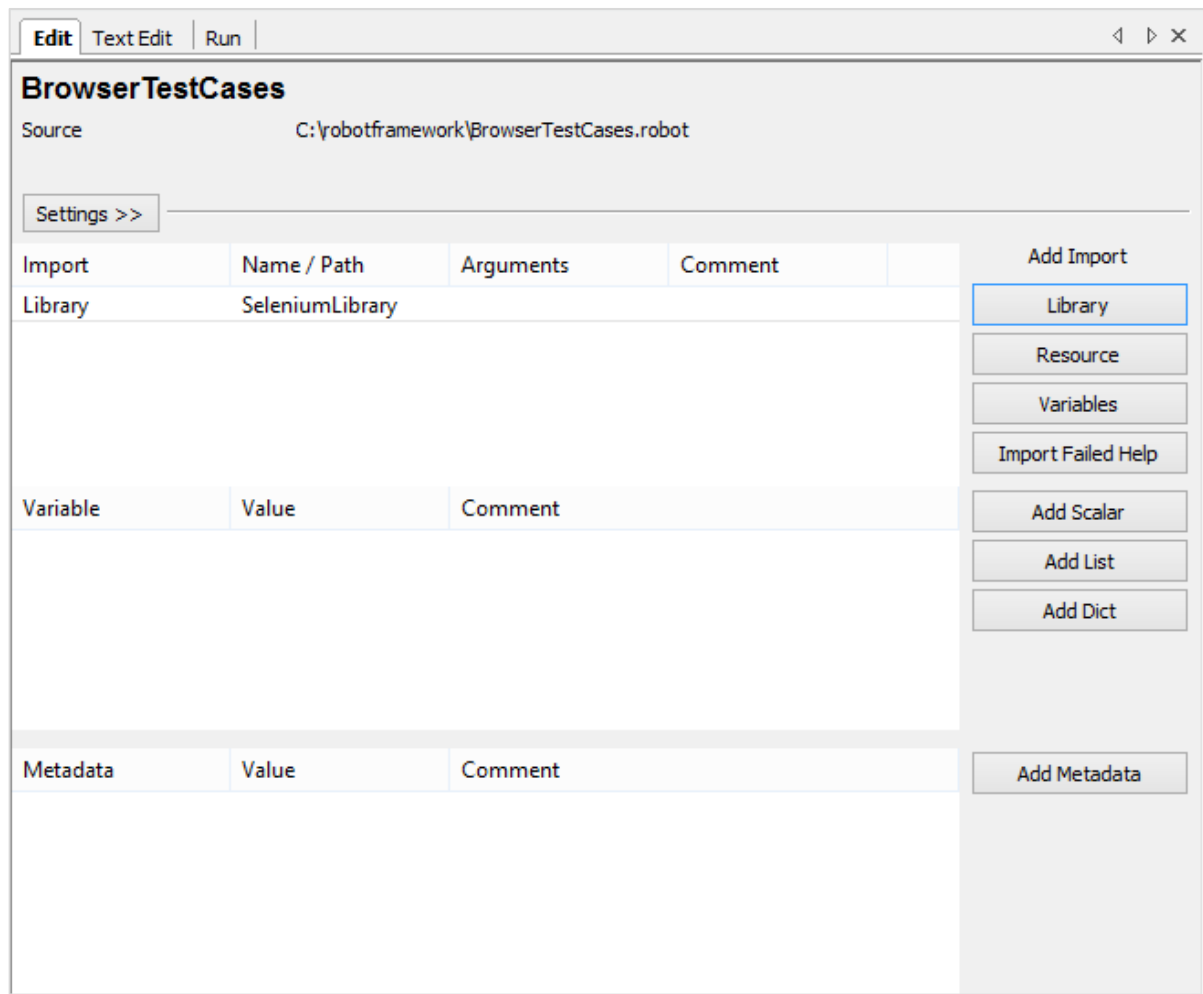
Click on your project on the left side and use Library from Add Import:



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.



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:

BrowserTestCases

Source C:\robotframework\BrowserTestCases.robot

Settings >>

Import	Name / Path	Arguments	Comment
Library	SeleniumLibrary		
Library	seleniumlibrary		

Variable	Value	Comment
----------	-------	---------

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

Add Import

Library

Resource

Variables

Import Failed Help

Add Scalar

Add List

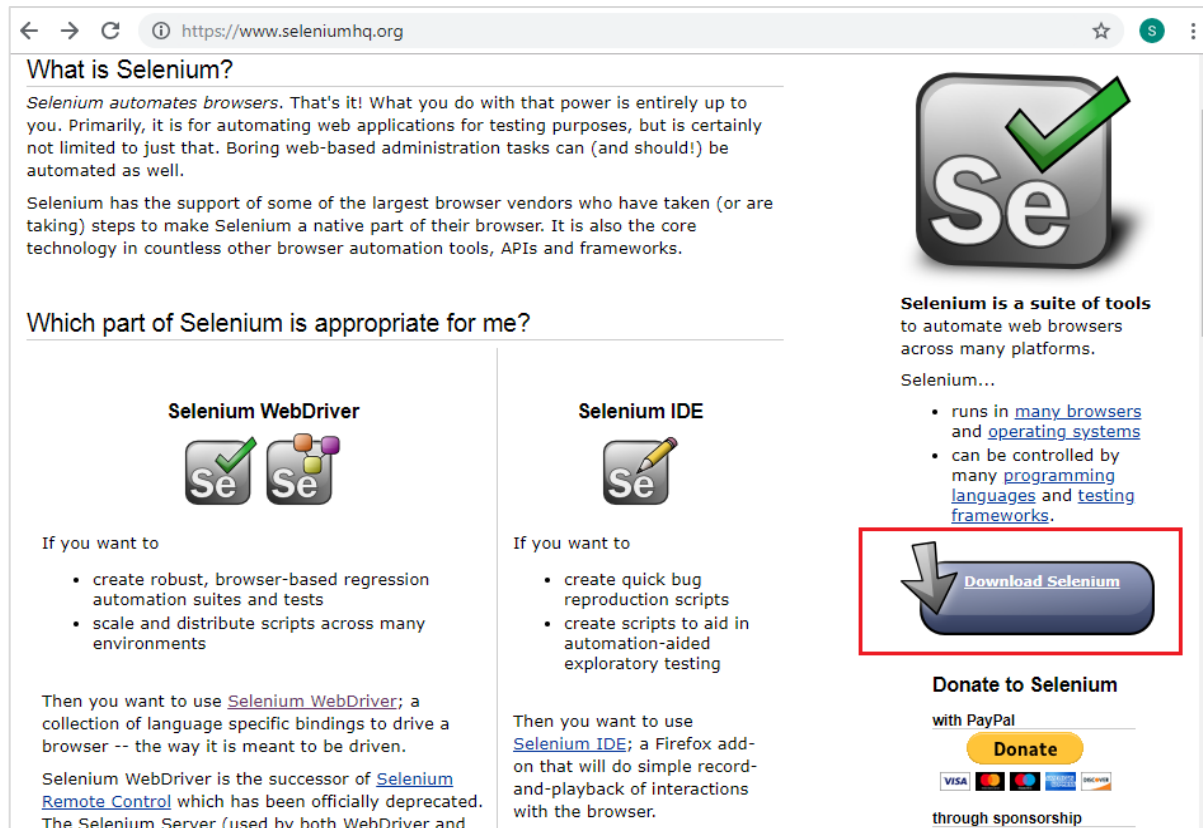
Add Dict

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/>.



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 (used 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

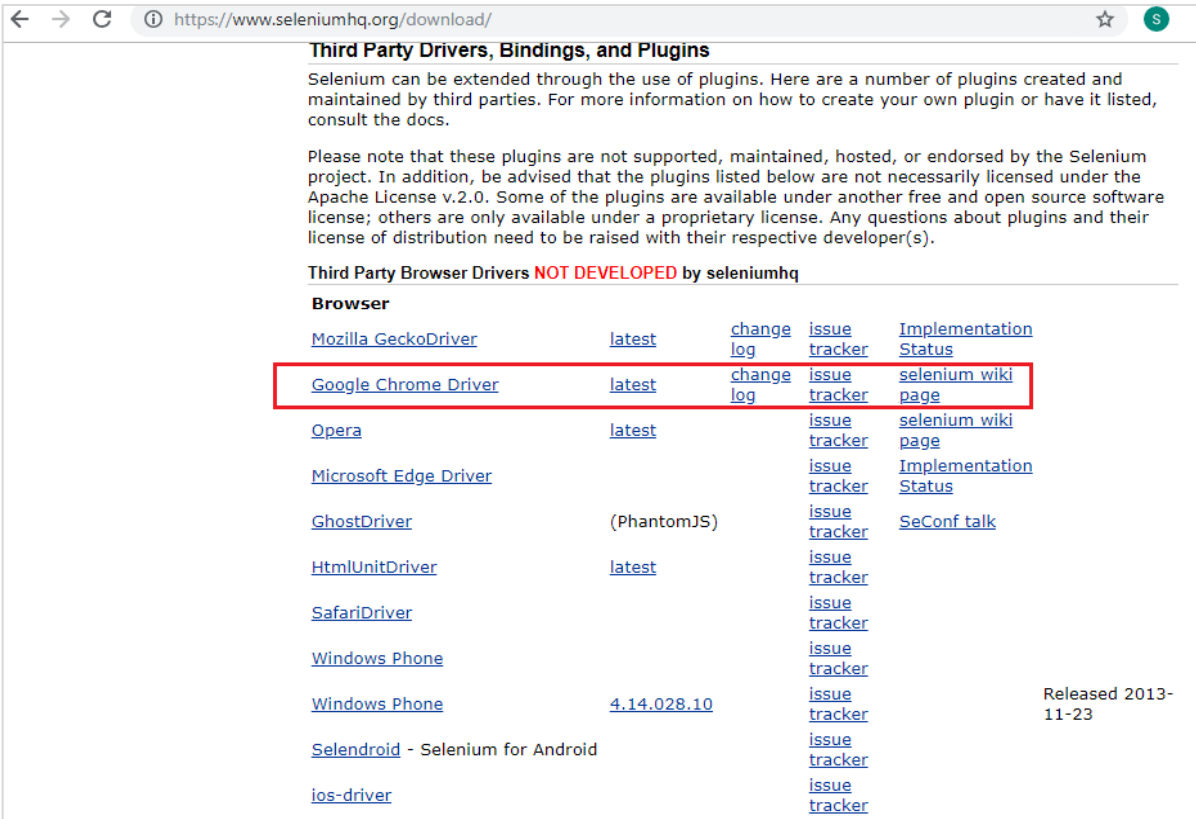
Donate

VISA, Mastercard, PayPal, American Express, Discover

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:



Third Party Drivers, Bindings, and Plugins

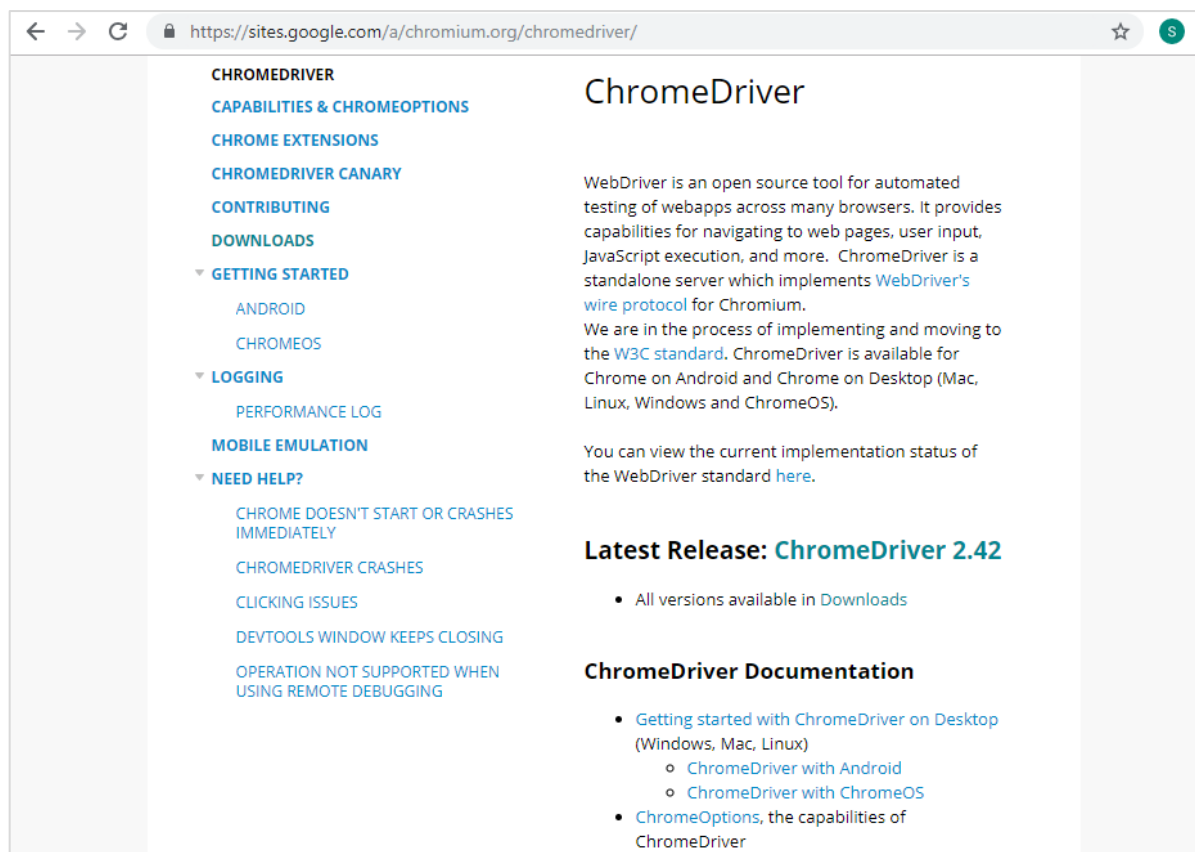
Selenium can be extended through the use of plugins. Here are a number of plugins created and maintained by third parties. For more information on how to create your own plugin or have it listed, consult the docs.

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.



The screenshot shows the ChromeDriver website. The left sidebar contains a navigation menu with the following items: CHROMEDRIVER, CAPABILITIES & CHROME OPTIONS, CHROME EXTENSIONS, CHROMEDRIVER CANARY, CONTRIBUTING, DOWNLOADS, GETTING STARTED (expanded), ANDROID, CHROME OS, LOGGING (expanded), PERFORMANCE LOG, MOBILE EMULATION, and NEED HELP? (expanded). Under NEED HELP?, there are links for CHROME DOESN'T START OR CRASHES IMMEDIATELY, CHROMEDRIVER CRASHES, CLICKING ISSUES, DEVTOOLS WINDOW KEEPS CLOSING, and OPERATION NOT SUPPORTED WHEN USING REMOTE DEBUGGING. The main content area is titled 'ChromeDriver' and contains a description of the tool, a link to the latest release (ChromeDriver 2.42), and a list of documentation links including 'Getting started with ChromeDriver on Desktop' and 'ChromeOptions, the capabilities of ChromeDriver'.

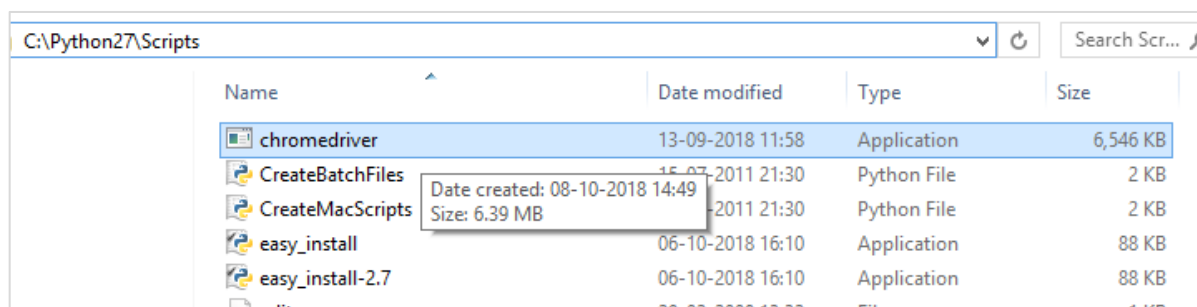
Click on the latest release. It will display the downloads as per the operating system – windows, linux and mac.

Index of /2.42/

	<u>Name</u>	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.

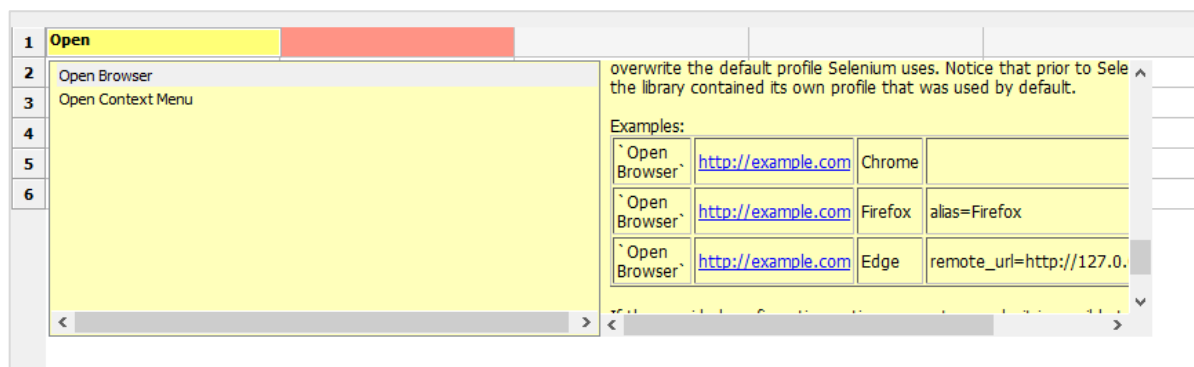
We are copying the file to **C:\Python27\Scripts**.



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:



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:

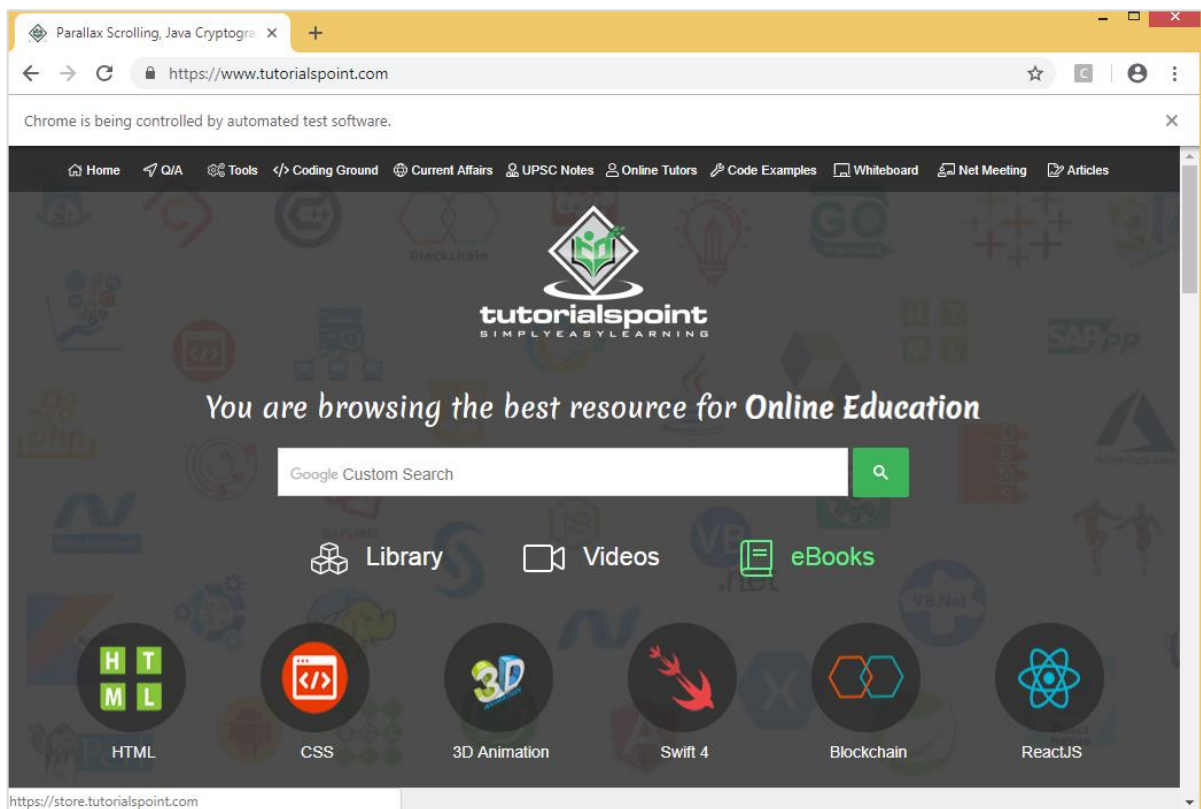
```

Execution Profile: pybot
Start Stop Pause Continue Next 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\kamat\appdata\local\temp\RIDEubool4.d\argfile.txt --listener C:\Python27\1
BrowserTestCases
=====
TC1 | PASS |
BrowserTestCases | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
Output: c:\users\appdata\local\temp\RIDEubool4.d\output.xml
Log: c:\users\appdata\local\temp\RIDEubool4.d\log.html
Report: c:\users\appdata\local\temp\RIDEubool4.d\report.html
test finished 20181008 14:59:11

Starting test: BrowserTestCases.TC1
20181008 14:58:57.686 : INFO : Opening browser 'chrome' to base url 'https://www.tutorialspoint.com/'.
Ending test: BrowserTestCases.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 20181008 15:22:56 GMT+05:30
4 minutes 14 seconds ago

Summary Information

Status: All tests passed
Start Time: 20181008 15:22:41.302
End Time: 20181008 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:
☐ Critical Tests
☐ All Tests

Log

BrowserTestCases Test Log

Generated
20181008 15:22:56 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	
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 Execution Log

SUITE
BrowserTestCases

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

00:00:14.875

TEST
TC1

00:00:14.082

85



tutorialspoint
SIMPLY EASY LEARNING

Details of test cases from log

KEYWORD SeleniumLibrary . Open Browser <https://www.tutorialspoint.com/>, chrome

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

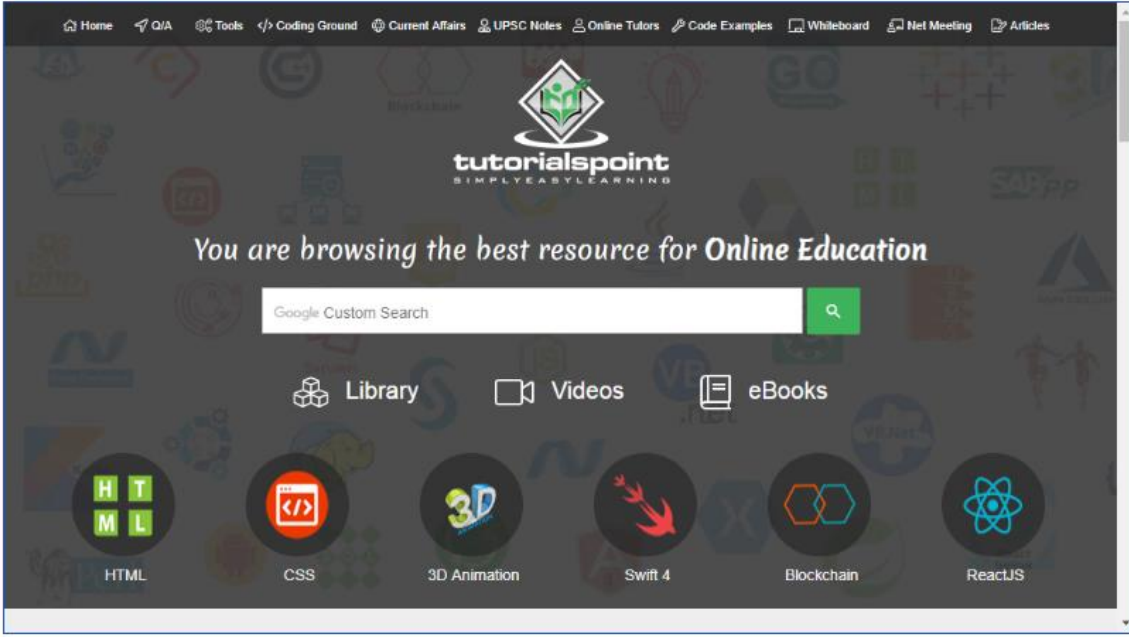
15:22:42.095 **INFO** opening browser 'chrome' to base url 'https://www.tutorialspoint.com/'.

KEYWORD SeleniumLibrary . Capture Page Screenshot page.png 00:00:00.796

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**



KEYWORD SeleniumLibrary . Close Browser

Documentation: Closes the current browser.

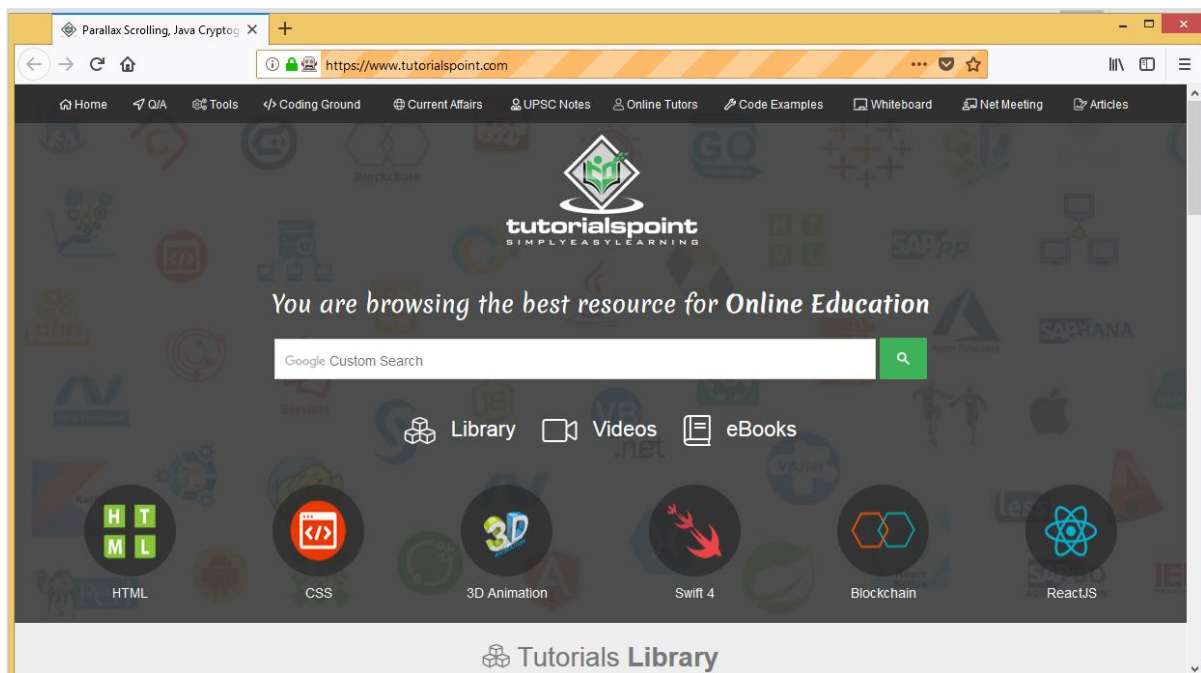
Start / End / Elapsed: 20181008 15:22:53.185 / 20181008 15:22:56.177 / 00:00:02.992

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.

9. Robot Framework — Working With Textbox

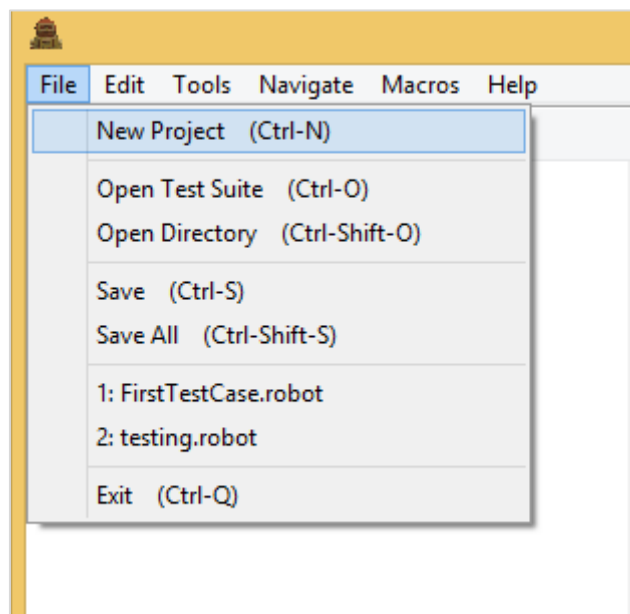
For testing, it becomes important to understand how to interact with the browser and locate the html elements. It is very easy to work with input fields with robot framework. In this chapter, we will learn how to work with textbox using Selenium Library. To work with input field – textbox, we need the 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:

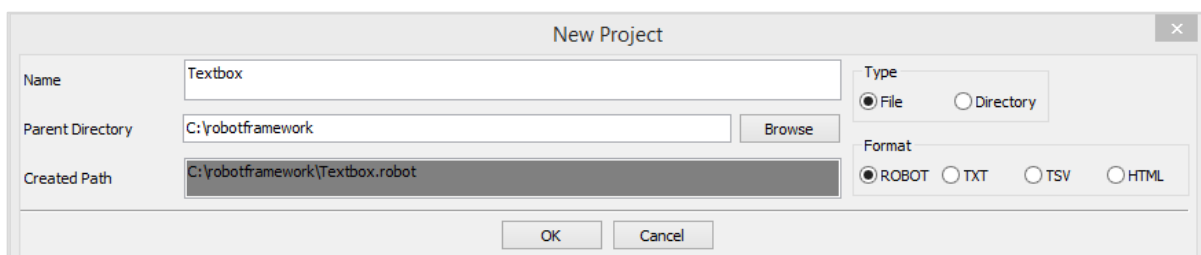
- Project Setup for Textbox Testing
- Enter Data in Search Textbox
- Click on Search Button

Project Setup for Textbox Testing

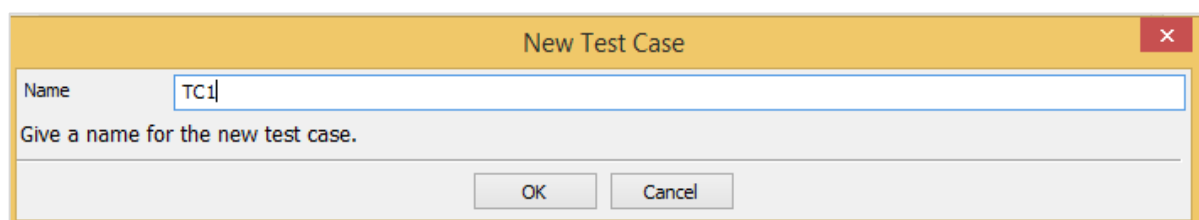
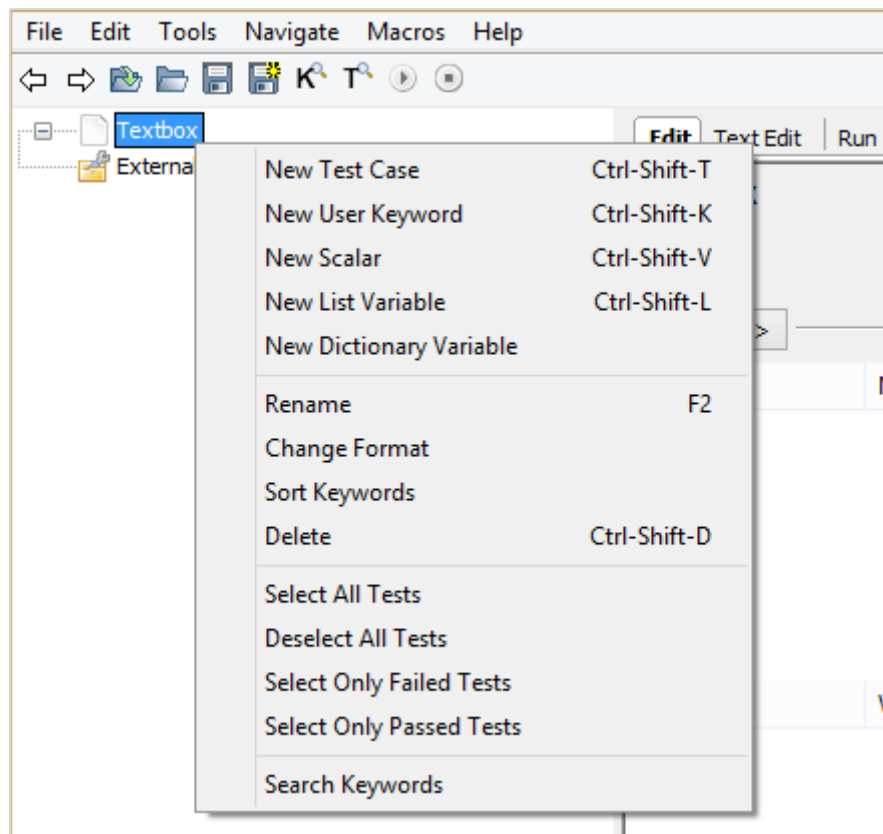
We will first create a project in Ride to work with browsers. Open ride using ride.py from the command line.



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.

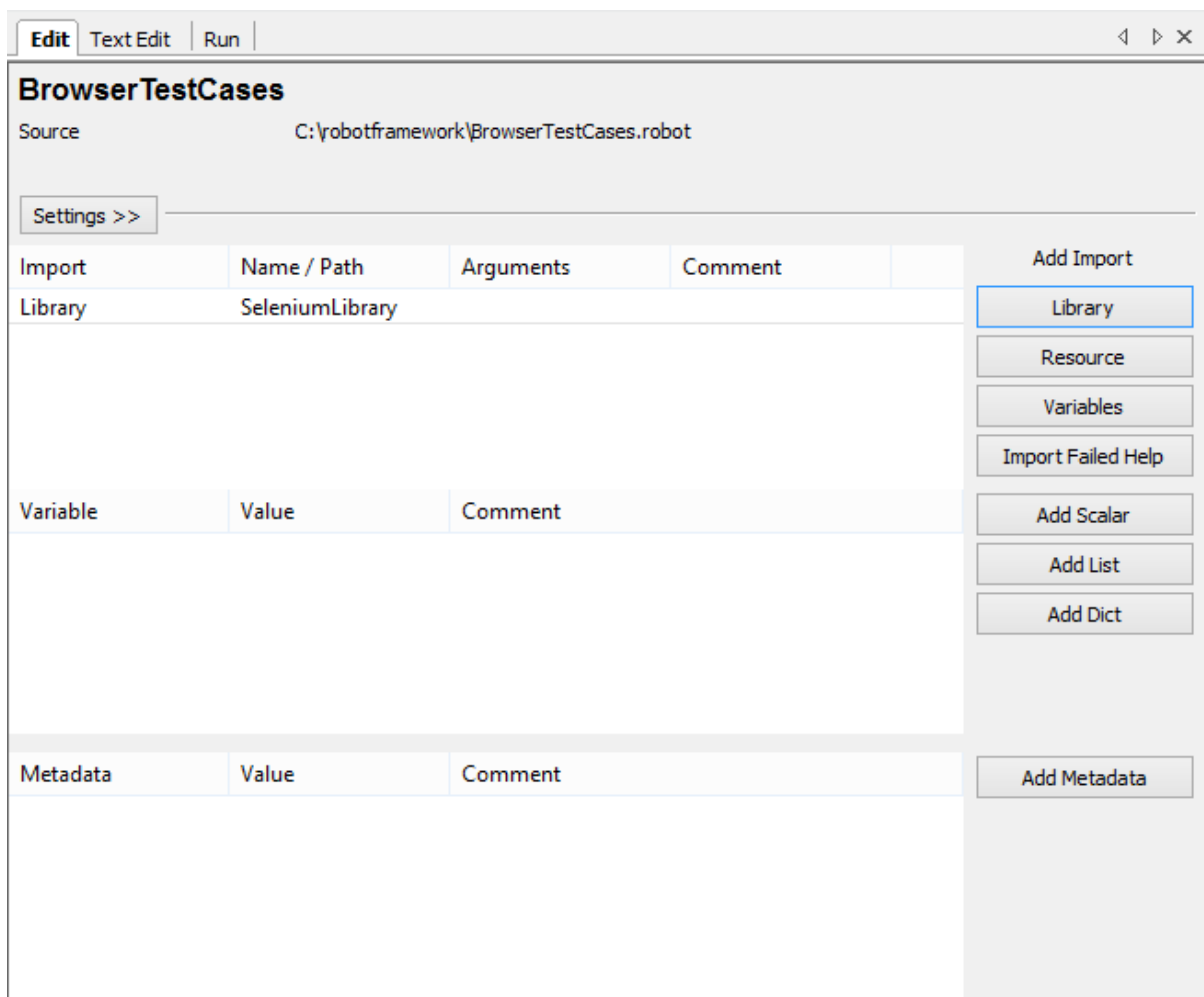
The screenshot shows the Robot Framework IDE interface for a project named 'Textbox'. The source path is 'C:\robotframework\Textbox.robot'. The 'Add Import' menu is open, showing options: 'Library', 'Resource', 'Variables', 'Import Failed Help', 'Add Scalar', 'Add List', 'Add Dict', and 'Add Metadata'. The 'Library' option is highlighted.

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

The screenshot shows the 'Library' dialog box. The 'Name' field contains 'SeleniumLibrary'. The 'Args', 'Alias', and 'Comment' fields are empty. The 'Browse' button is next to the 'Name' field. The 'OK' and 'Cancel' buttons are at the bottom. The dialog box contains the following text:

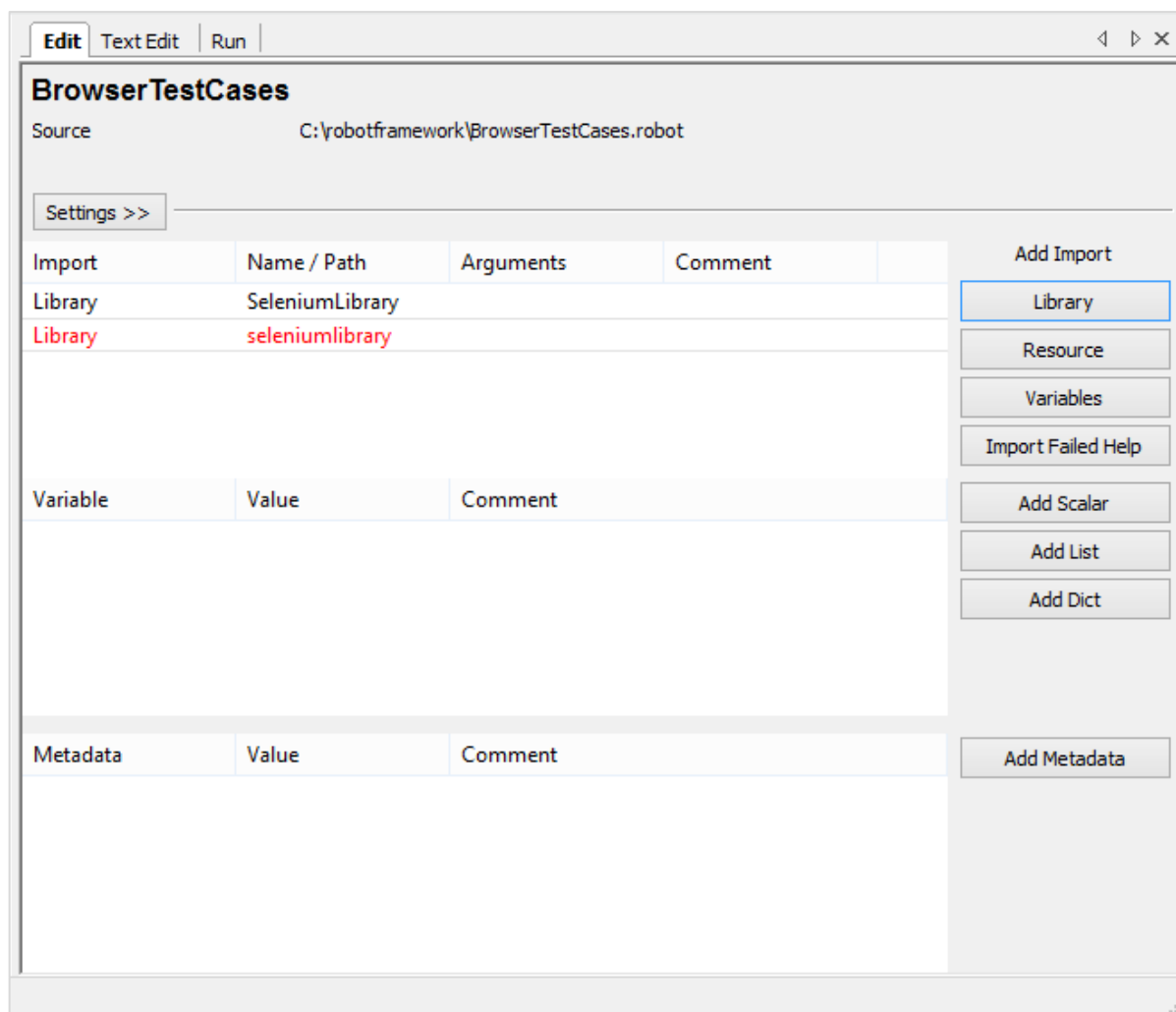
Give name, optional arguments and optional alias of the library to import.
 Separate multiple arguments with a pipe character like 'arg 1 | arg 2'.
 Alias can be used to import same library multiple times with different names.

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



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:



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 -

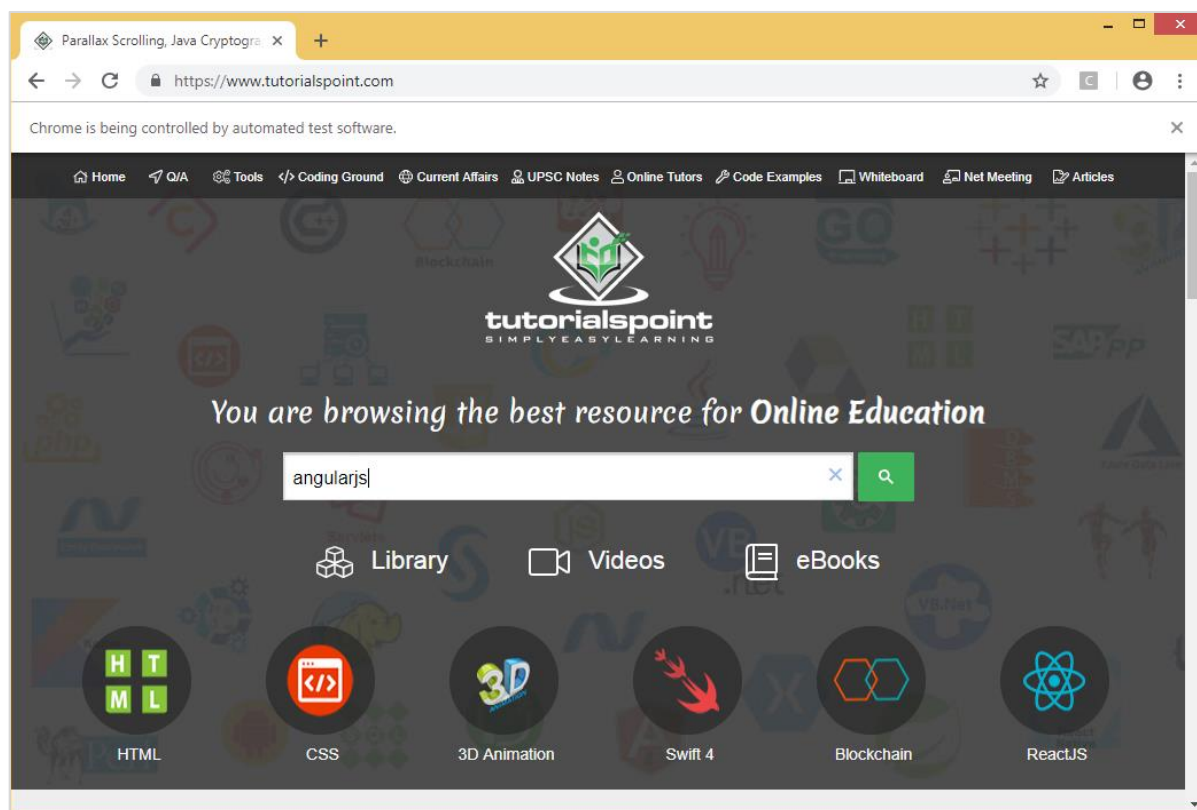
- **name** attribute of the textbox, it has to be *name:Nameofthetextbox* or *name=Nameofthetextbox*
- **id** of the textbox, it will be *id:idoftextbox* or *id=idoftextbox*
- **class** of the textbox, it will be *class:classfortextbox* or *class=classfortextbox*

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

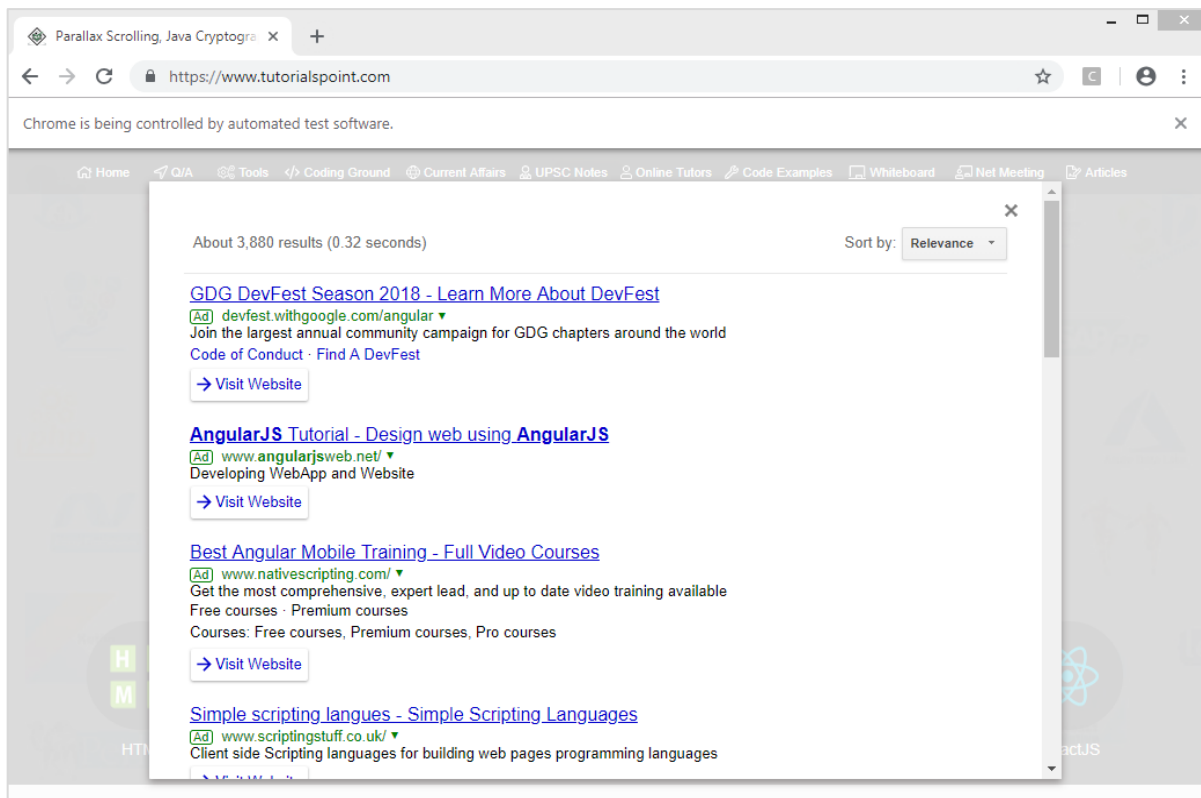
1	Open Browser	https://www.tutorialspoint.com/	chrome		
2	Input Text	name:search	angularjs		
3	Click Button	class:gsc-search-button-v2			
4					
5					

- **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:



Let us now see the reports and the log details:

Report

Textbox Test Report

Generated 20181008 18:54:16 GMT+05:30
52 seconds ago

Summary Information

Status: All tests passed
Start Time: 20181008 18:54:08.143
End Time: 20181008 18:54:16.970
Elapsed Time: 00:00:08.827
Log File: [log.html](#)

Test Statistics

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

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	

Test Details

Totals **Tags** **Suites** **Search**

Type:
☐ Critical Tests
☒ All Tests

Log

Textbox Test Log

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	
All Tests	1	1	0	00:00:09	

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	

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

- 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.

10. Robot Framework — Working With Radio Button

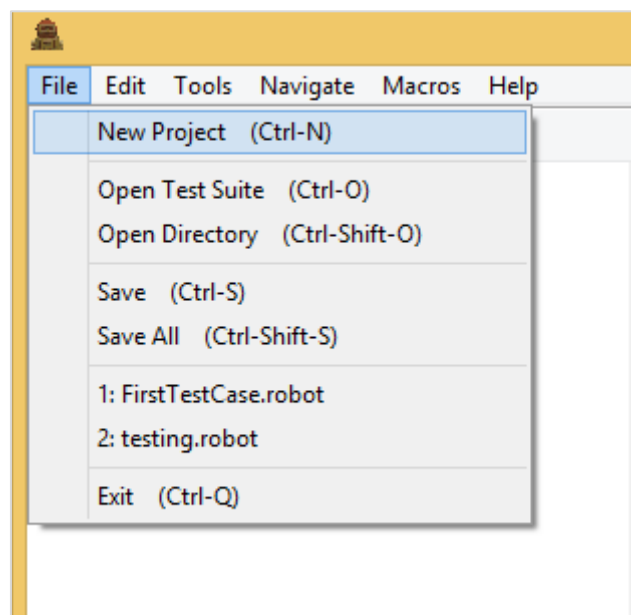
For testing, it becomes important to understand how to interact with the browser and locate the html elements. It is very easy to work with input fields with robot framework. In this chapter, we will learn how to work with radio button using Selenium Library. To work with radio button, we need the locator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

We are going to discuss the following over here:

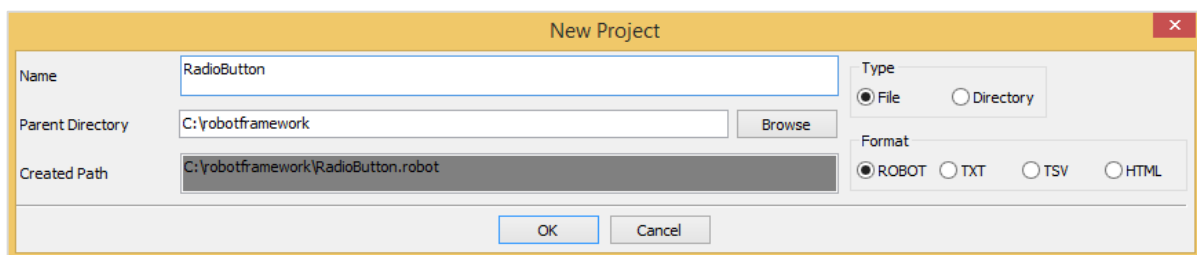
- Project Setup for Radio Button Testing
- Test case for Radio Button

Project Setup For Textbox Testing

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

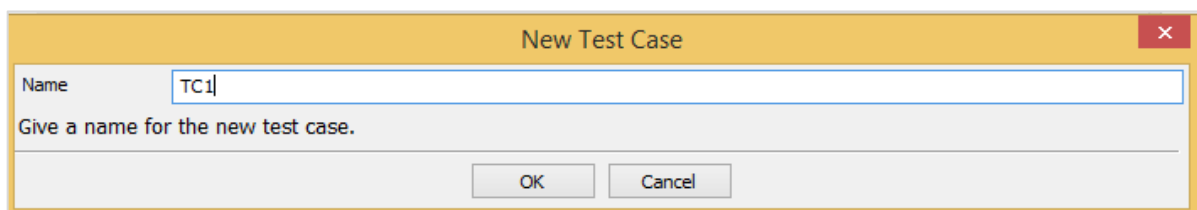
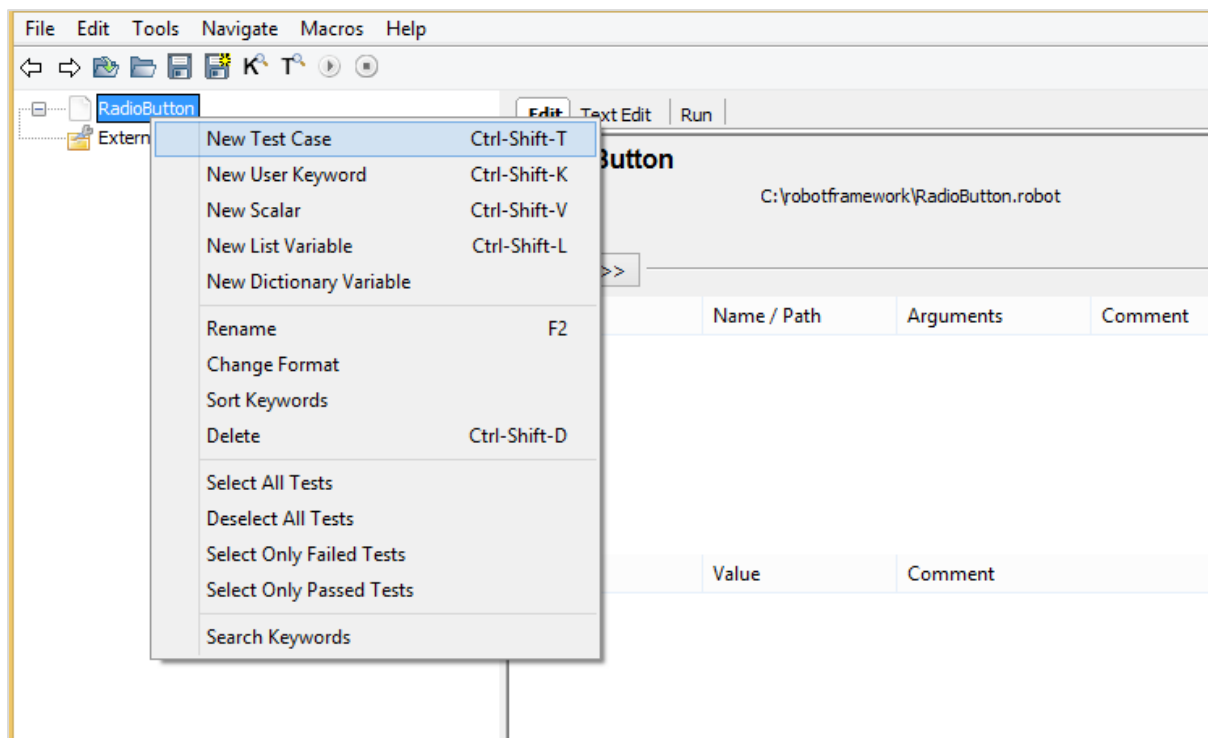


Click *New Project* and enter *Name* of your project as shown in the screenshot below.



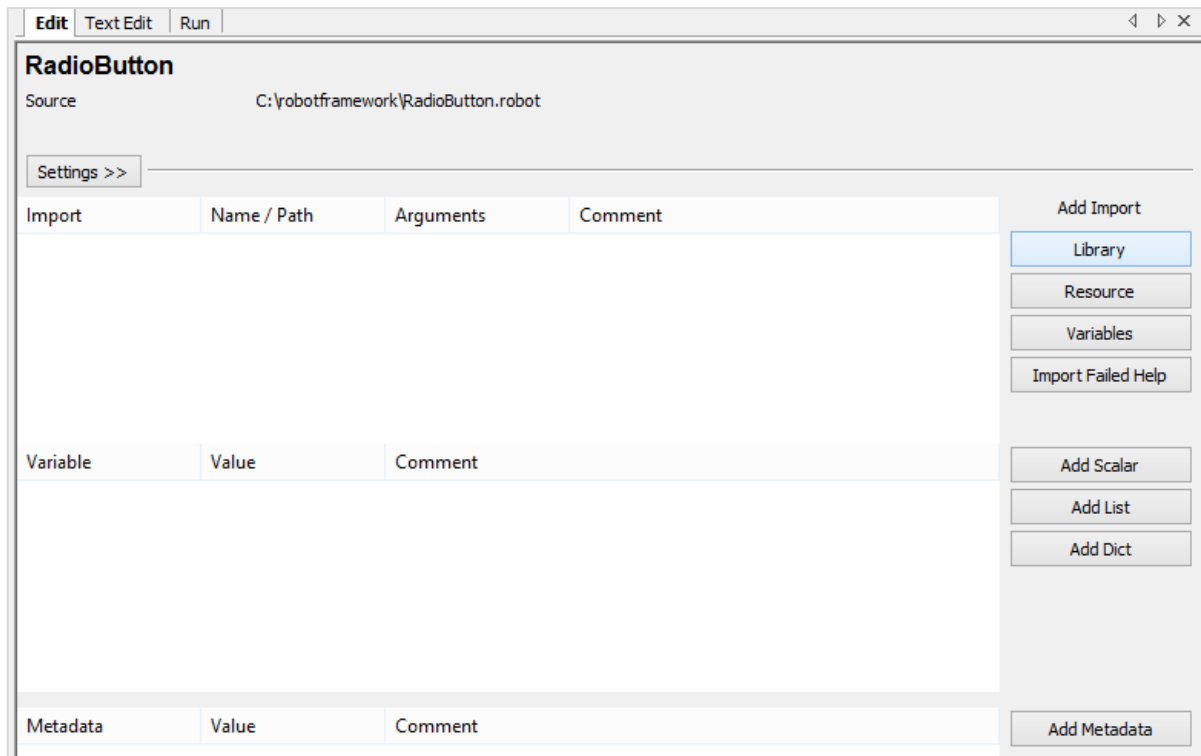
The name given is RadioButton. Click on *OK* button to save the project.

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

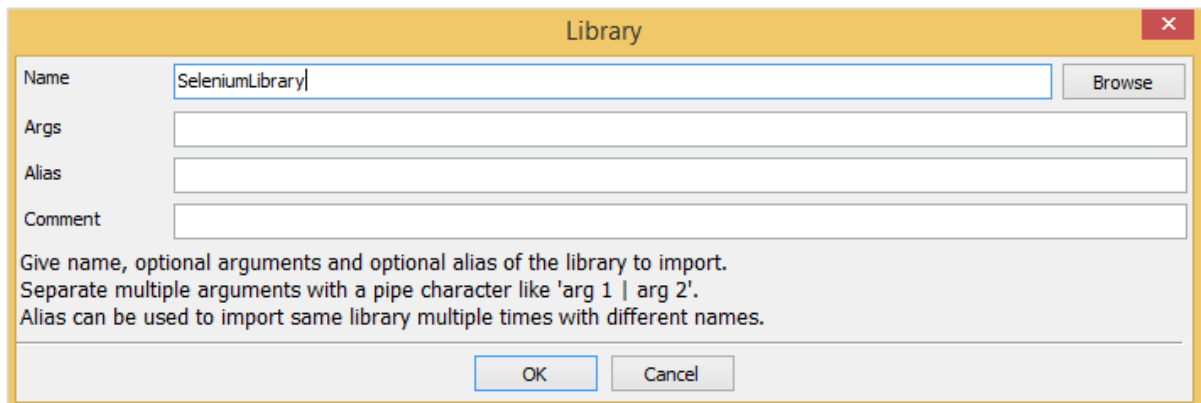


Give name to the test case and click OK to save it. We are done with the project setup and now will write test cases for the radio button. Since we need Selenium library, we need to import the same in our project.

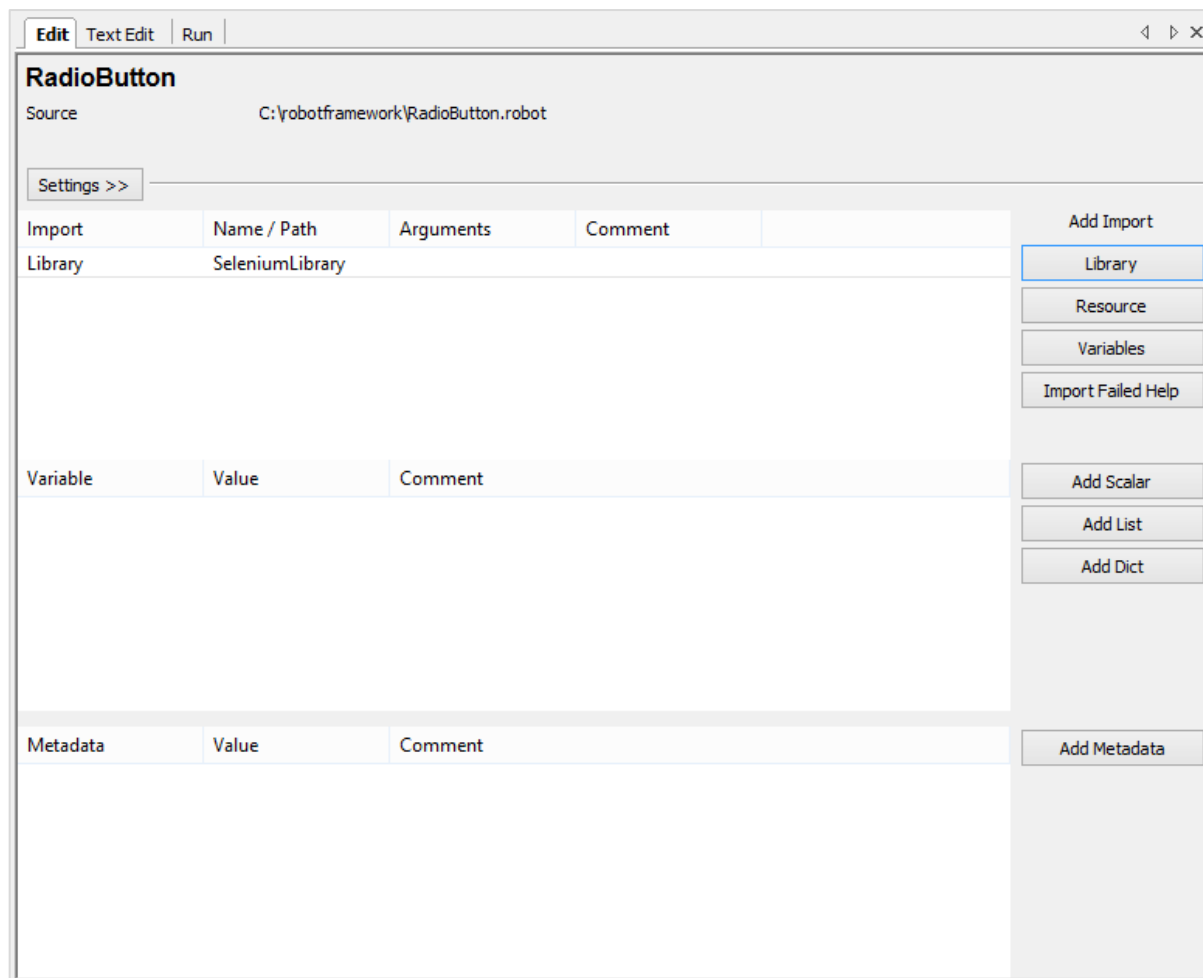
Click on your project on the left side and use *Library* from *Add Import*.



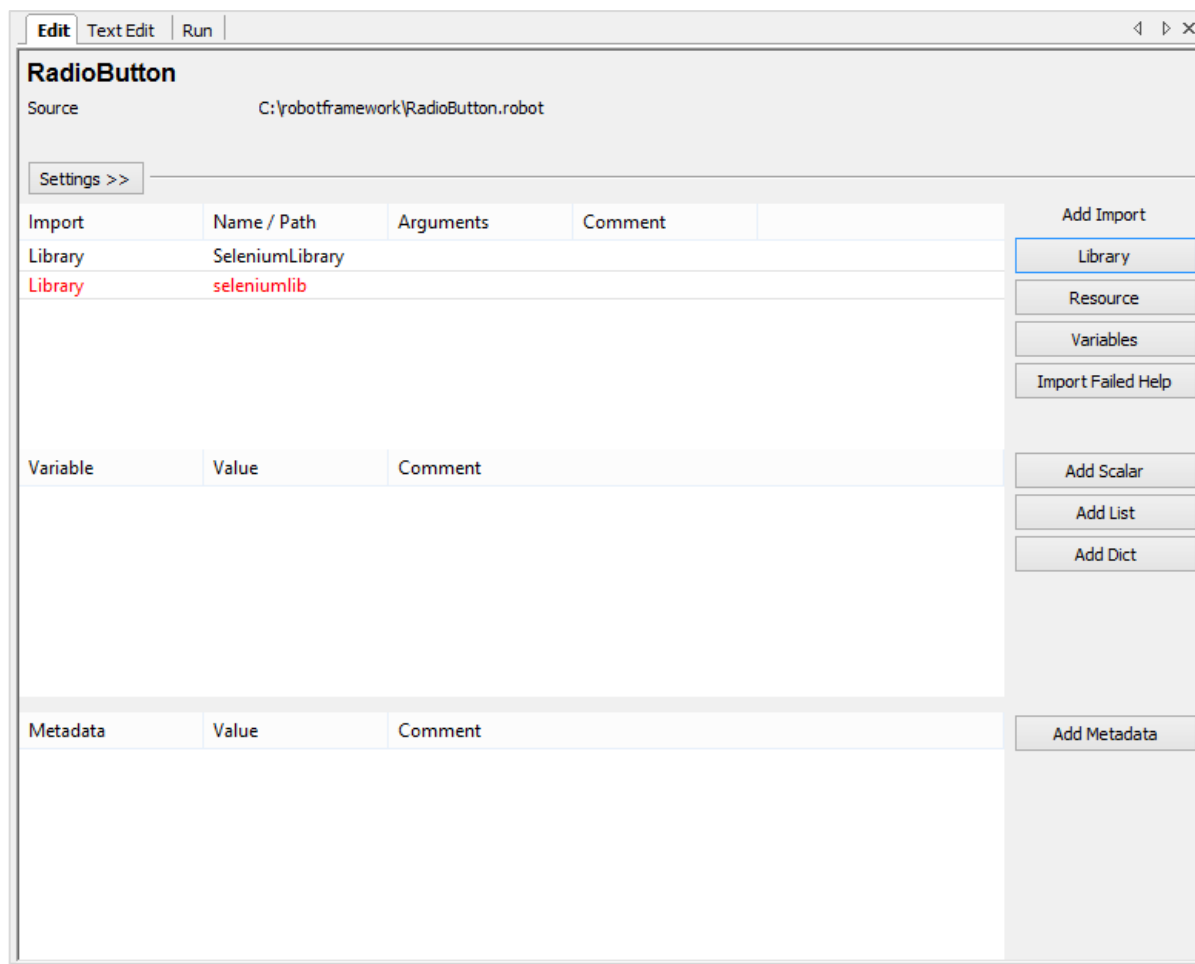
Upon clicking Library, a screen will appear where you need to enter the library name:



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



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:

Select Radio Button ✕

Name: Select Radio Button

Source: SeleniumLibrary <test library>

Arguments: [group_name | value]

Sets radio button group group_name to value.

The radio button to be selected is located by two arguments:

- group_name is the name of the radio button group.
- value is the id or value attribute of the actual radio button.

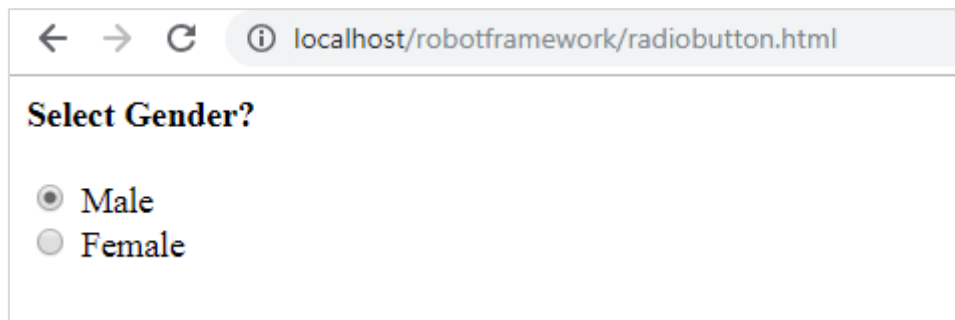
Examples:

`Select Radio Button`	size	XL
`Select Radio Button`	contact	email

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/rz/chrome		
2	Select Radio Button	gender	female	
3				
4				
5				
6				
7				

Following is the Test Page for radio button:



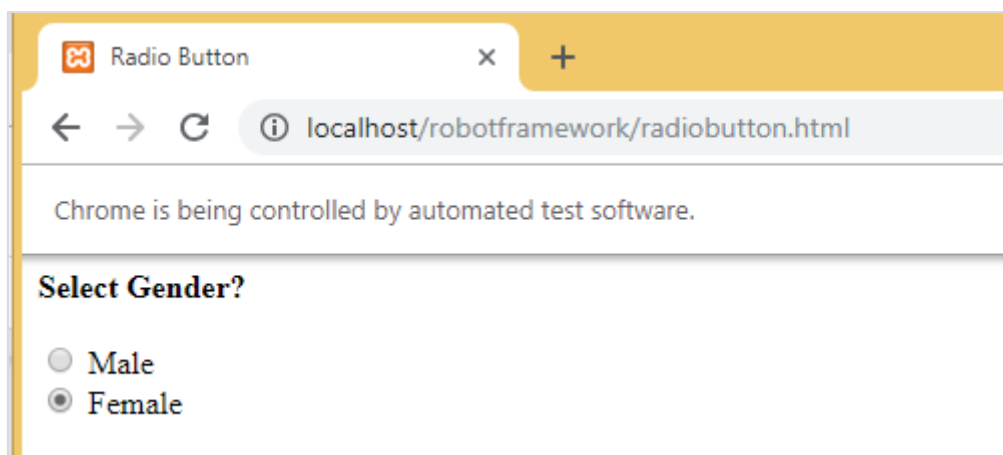
The screenshot shows a web browser window with the address bar displaying 'localhost/robotframework/radiobutton.html'. The page content is a form titled 'Select Gender?'. Below the title, there are two radio buttons. The first radio button is selected and is labeled 'Male'. The second radio button is not selected and is labeled 'Female'.

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>
```

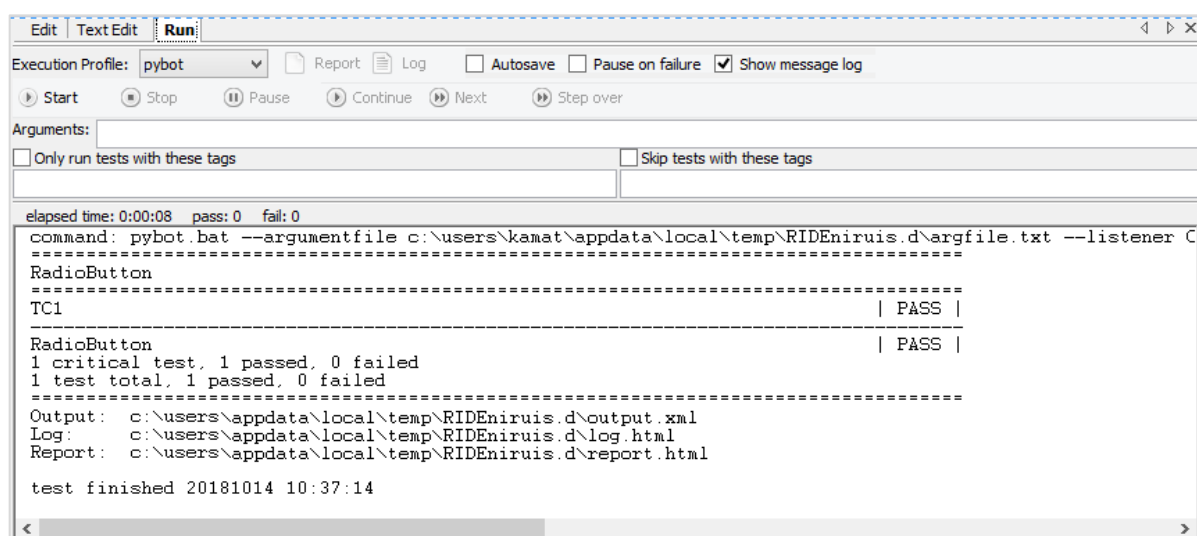
In the above form, we are planning to select *female*, which is a radio button. The name and value are given in the test case. Now, we will execute the test case and check the selection of the checkbox for the above form.

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:



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

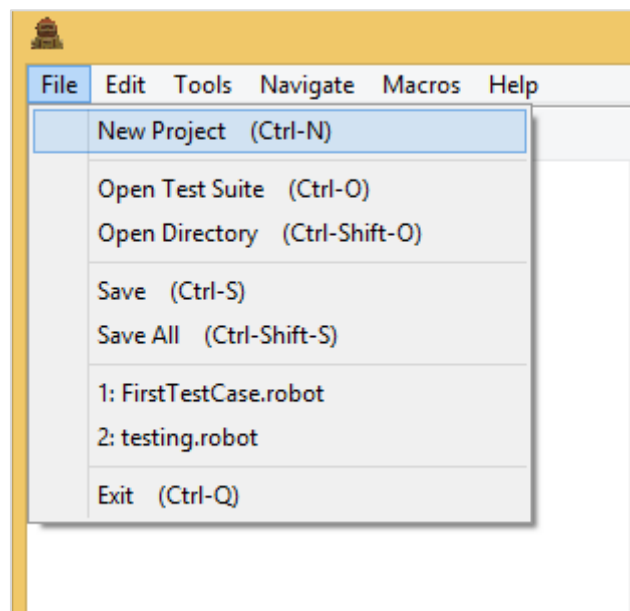
We have seen how to select value of radio button by giving the group name of the radio button to the test case. Using the keywords available with robot framework and the library imported, we can locate the radio button and select the value of the radio button. We do get the details of the test-case executed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

11. Robot Framework — Working With Checkbox

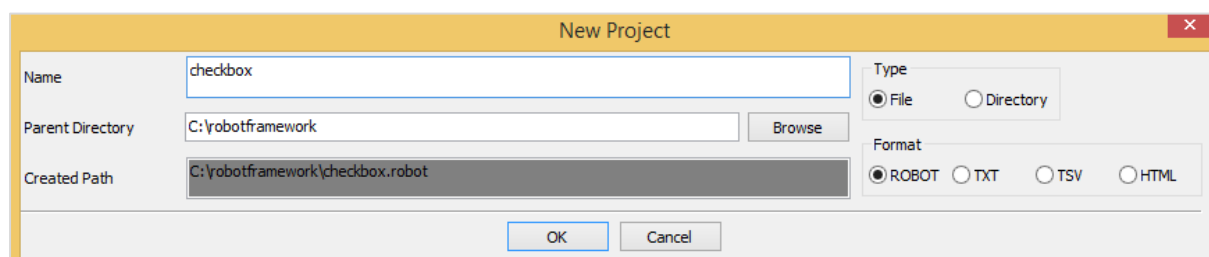
For testing, it becomes important to understand how to interact with the browser and locate the html elements. It is very easy to work with input fields with robot framework. In this chapter, we will learn how to work with checkbox using Selenium Library. To work with checkbox, we need the locator, which 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.

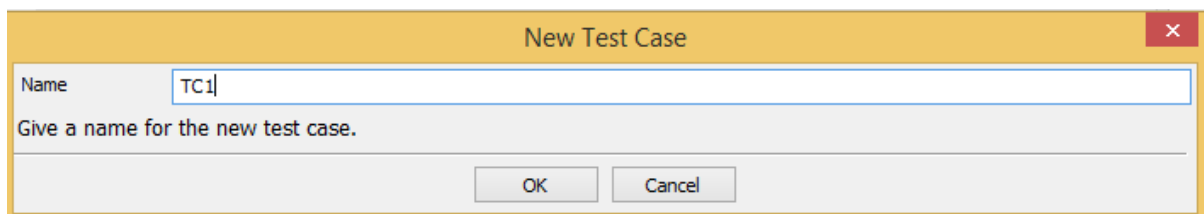
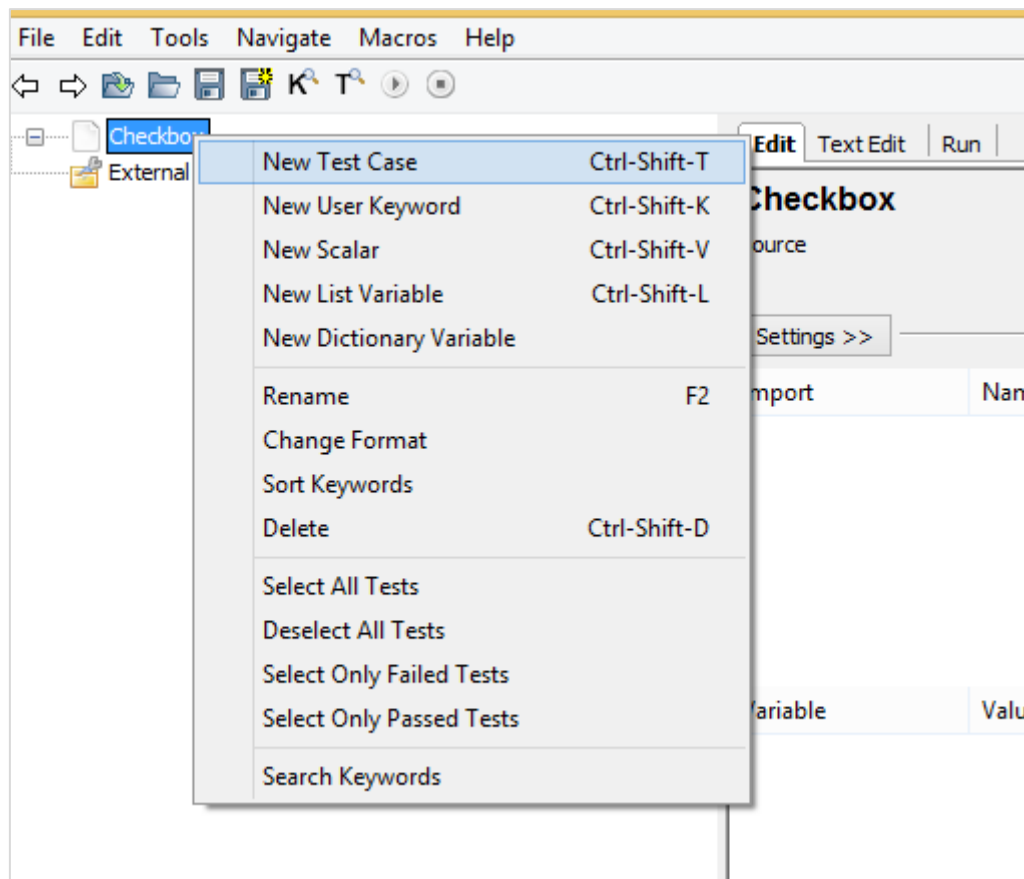


Click on *New Project* and enter *Name* of your project as shown in the screenshot below.



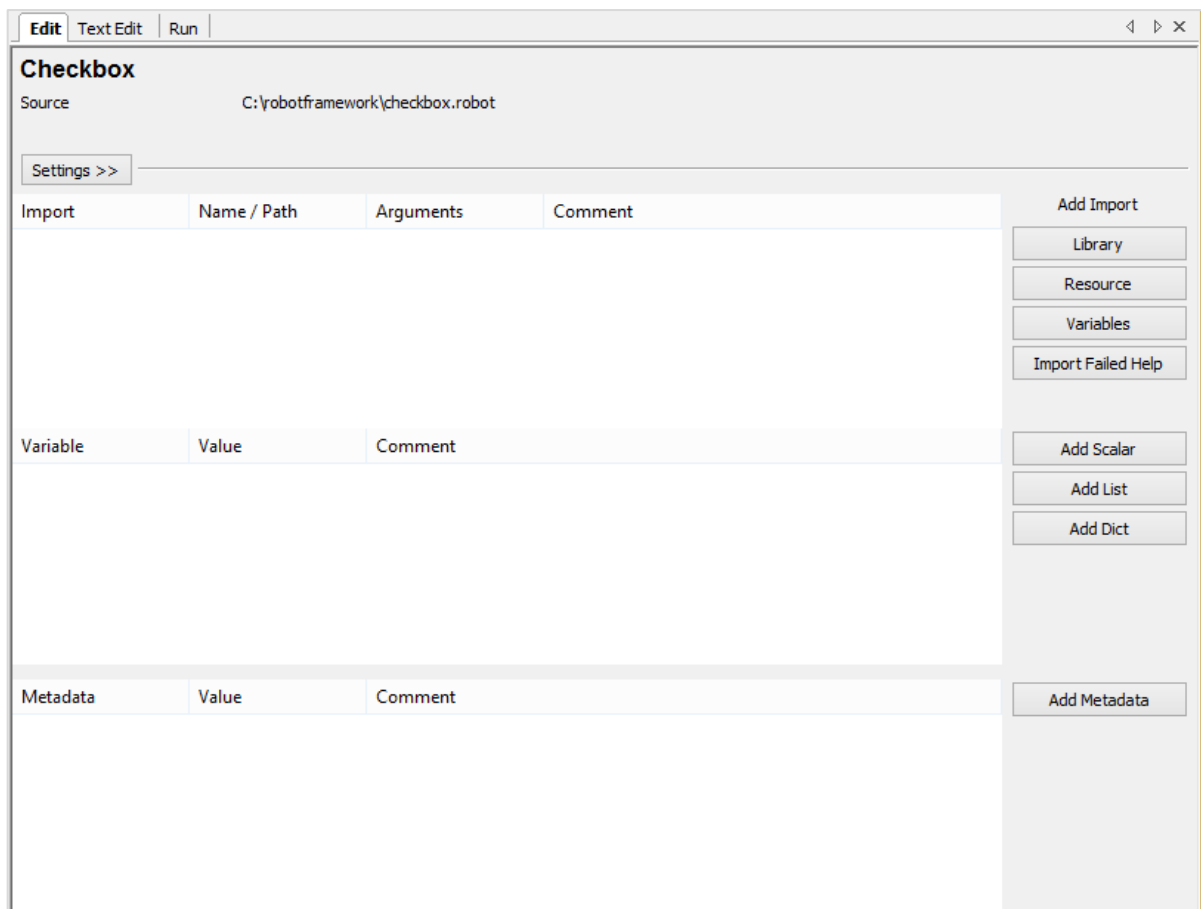
The name given for the project is *Checkbox*. Click OK to save the project.

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

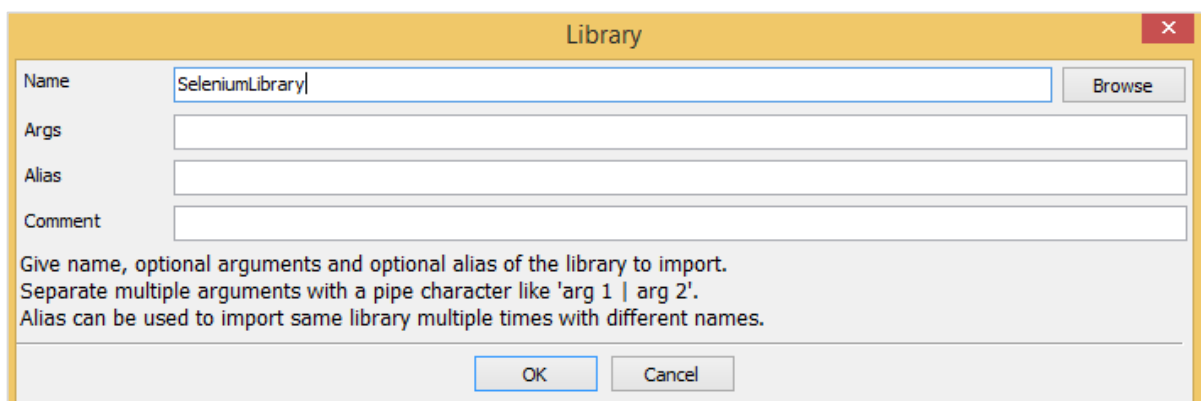


Give name to the test case and click OK. We are done with the project setup. Now we will write test cases for checkbox. Since we need Selenium library, we need to import the same in our project.

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



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.

Checkbox

Source C:\robotframework\checkbox.robot

Settings >>

Import	Name / Path	Arguments	Comment
Library	SeleniumLibrary		

Variable	Value	Comment
----------	-------	---------

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

Add Import

Library

Resource

Variables

Import Failed Help

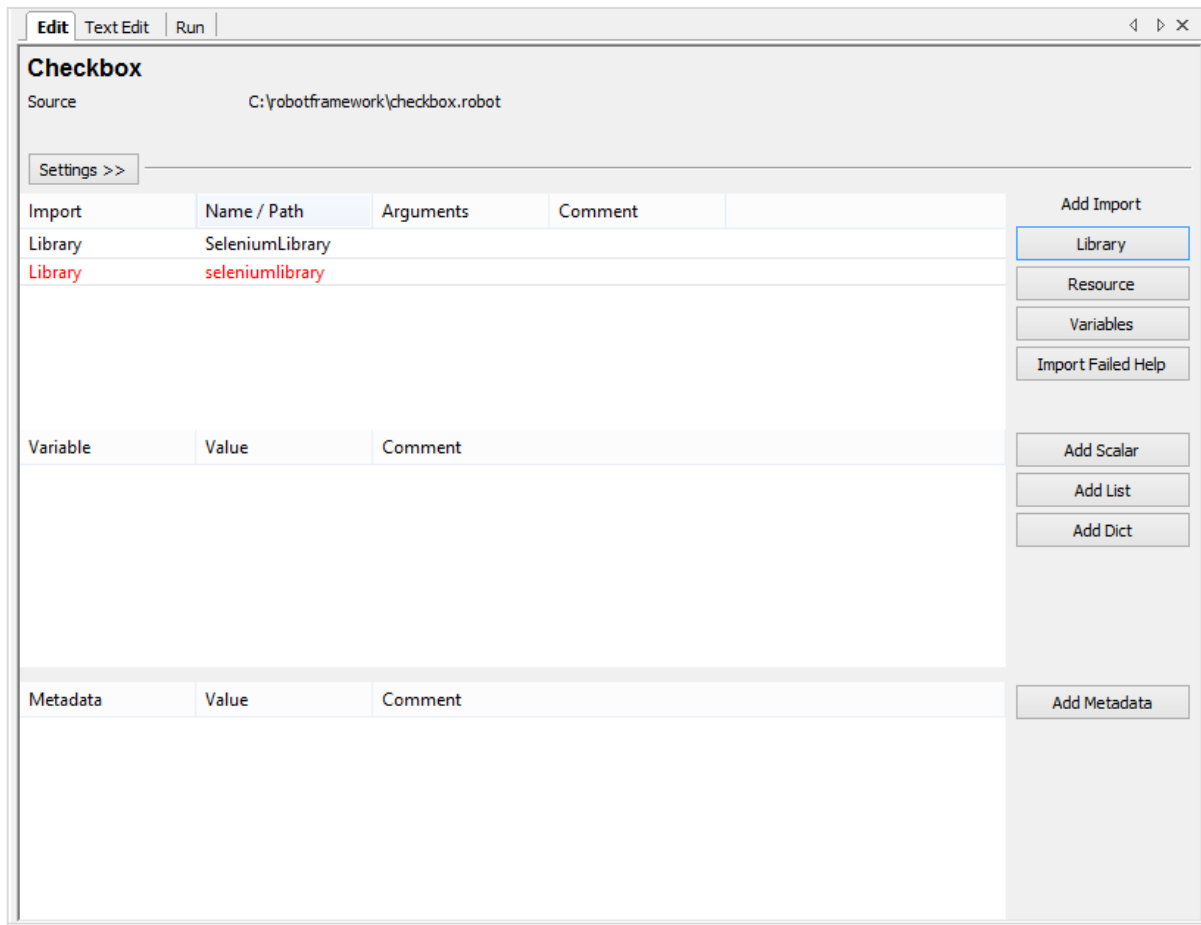
Add Scalar

Add List

Add Dict

Add Metadata

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:



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:

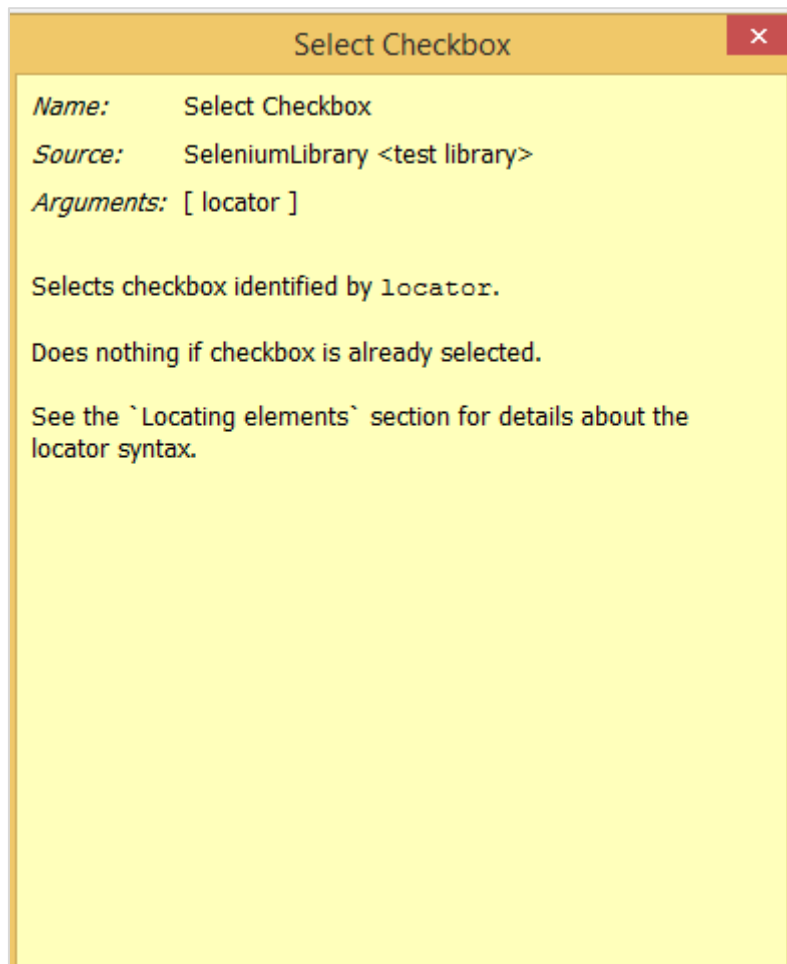
- Open browser: URL – <http://localhost/robotframework/checkbox.html> in Chrome
- Enter details of checkbox.
- Execute the test case.

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

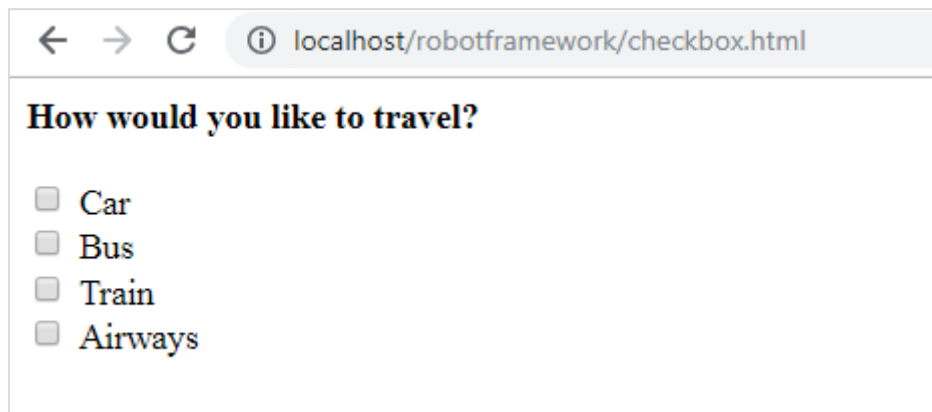
The command details from ride is as follows:



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				

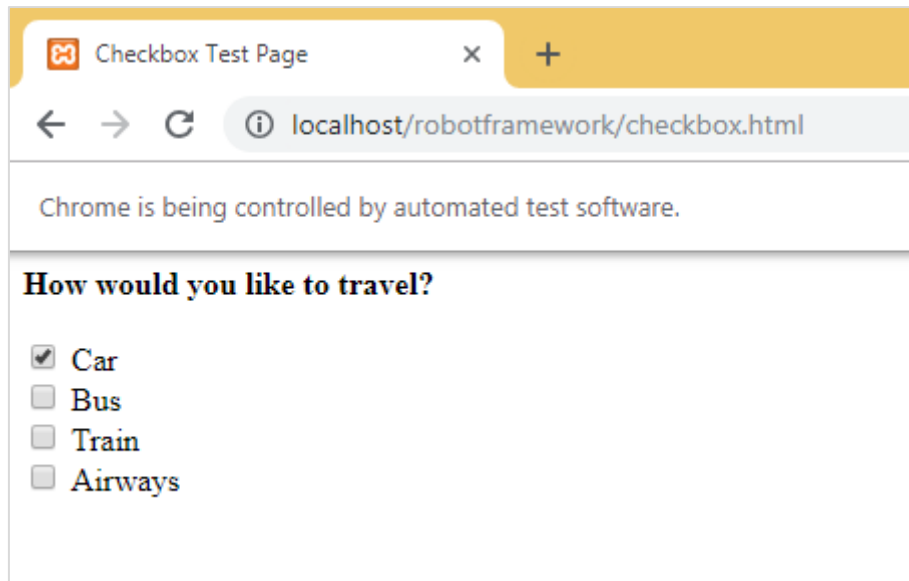
This is how the URL is:



checkbox.html

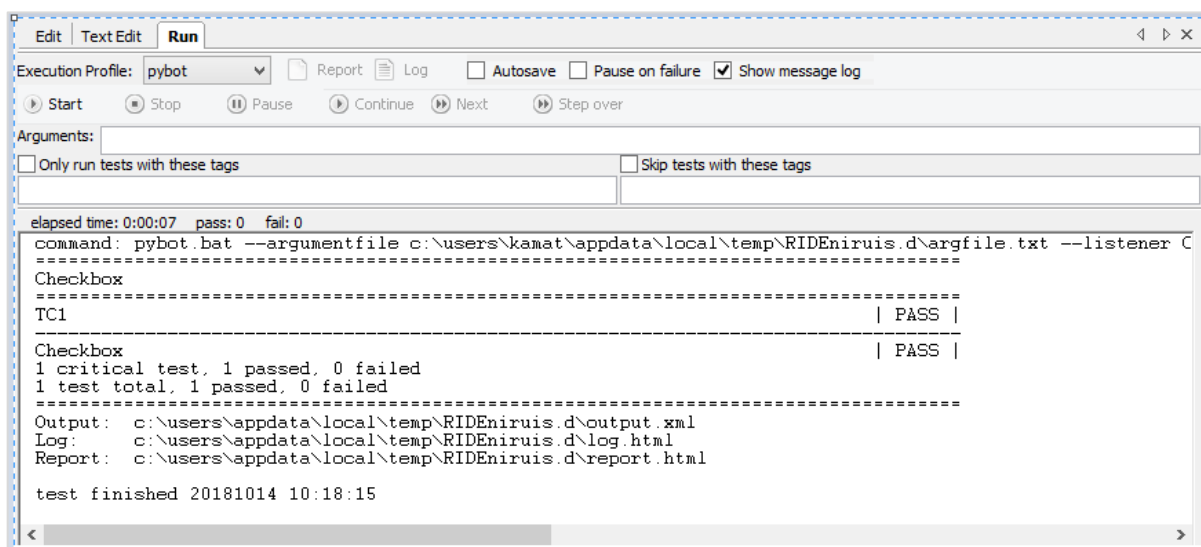
```
<html>
<head>
<title>Checkbox Test Page</title>
</head>
<body>
<form name="myform" method="POST">
<b>How would you like to travel?</b>
<div><br>
<input type="checkbox" name="option1" value="Car"> Car<br>
<input type="checkbox" name="option2" value="Bus"> Bus<br>
<input type="checkbox" name="option3"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



When the test case is executed, it opens the URL <http://localhost/robotframework/checkbox.html> and selects the name Car given in the test case.

Here are the execution details:



Details of Report

Checkbox Test Report

Generated
20181014 10:18:14 GMT+05:30
3 minutes 49 seconds ago

Summary Information

Status: All tests passed
Start Time: 20181014 10:18:08.740
End Time: 20181014 10:18:14.588
Elapsed Time: 00:00:05.848
Log File: [log.html](#)

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:05	
All Tests	1	1	0	00:00:05	

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

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Checkbox	1	1	0	00:00:06	

Test Details

Totals | **Tags** | **Suites** | **Search**

Type:
☐ Critical Tests
☐ All Tests

Checkbox Test Log

Generated
20181014 10:18:14 GMT+05:30
4 minutes 18 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:05	
All Tests	1	1	0	00:00:05	

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

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Checkbox	1	1	0	00:00:06	

Test Execution Log

SUITE

Checkbox

Full Name: Checkbox
Source: [C:\robotframework\checkbox.robot](#)
Start / End / Elapsed: 20181014 10:18:08.740 / 20181014 10:18:14.588 / 00:00:05.848
Status: 1 critical test, 1 passed, 0 failed
 1 test total, 1 passed, 0 failed

TEST

TC1

Details of Log

SUITE

Checkbox

Full Name:

Checkbox

Source:

C:\robotframework\checkbox.robot

Start / End / Elapsed:

20181014 10:18:08.740 / 20181014 10:18:14.588 / 00:00:05.848

Status:

1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

TEST

TC1

Full Name:

Checkbox.TC1

Start / End / Elapsed:

20181014 10:18:09.501 / 20181014 10:18:14.584 / 00:00:05.083

Status:

PASS (critical)

KEYWORD

SeleniumLibrary.Open Browser

http://localhost/robotframework/checkbox.html, chrome

Documentation:

Opens a new browser instance to the given url.

Start / End / Elapsed:

20181014 10:18:09.517 / 20181014 10:18:14.130 / 00:00:04.613

10:18:09.517

INFO

Opening browser 'chrome' to base url 'http://localhost/robotframework/checkbox.html'.

KEYWORD

SeleniumLibrary.Select Checkbox

name:option1

Documentation:

Selects checkbox identified by locator.

Start / End / Elapsed:

20181014 10:18:14.132 / 20181014 10:18:14.582 / 00:00:00.450

10:18:14.134

INFO

Selecting checkbox 'name:option1'.

Conclusion

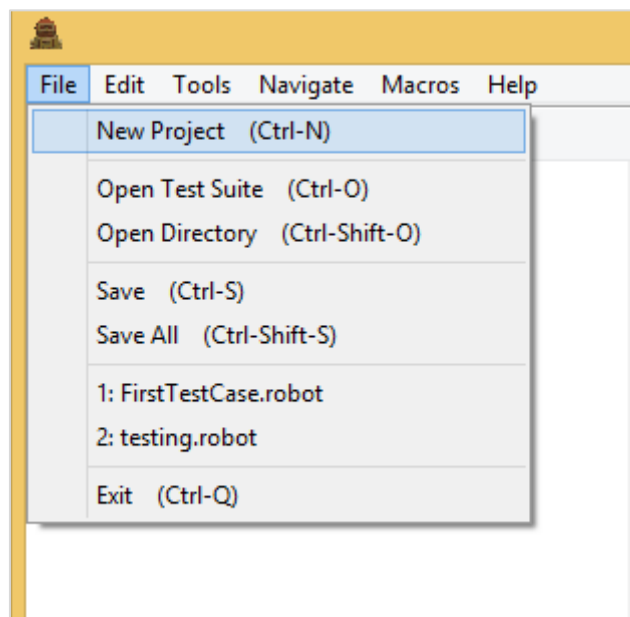
In this chapter, we learnt how we can select a checkbox by giving the locator of the checkbox. The log and Reports give the details of the execution of the test case along with the time spent for each test case.

12. Robot Framework — Working With Dropdown

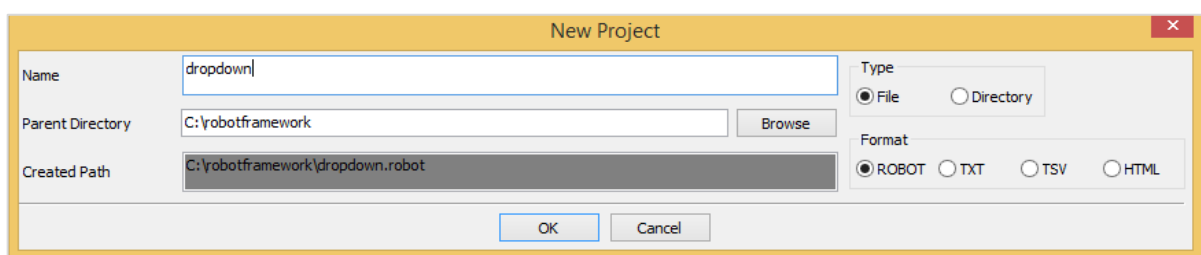
In this chapter, we will learn how to work with dropdown using Selenium Library.

Project Setup for Dropdown Testing

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

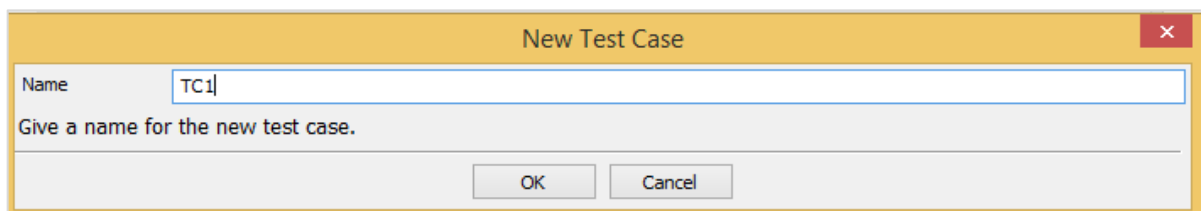
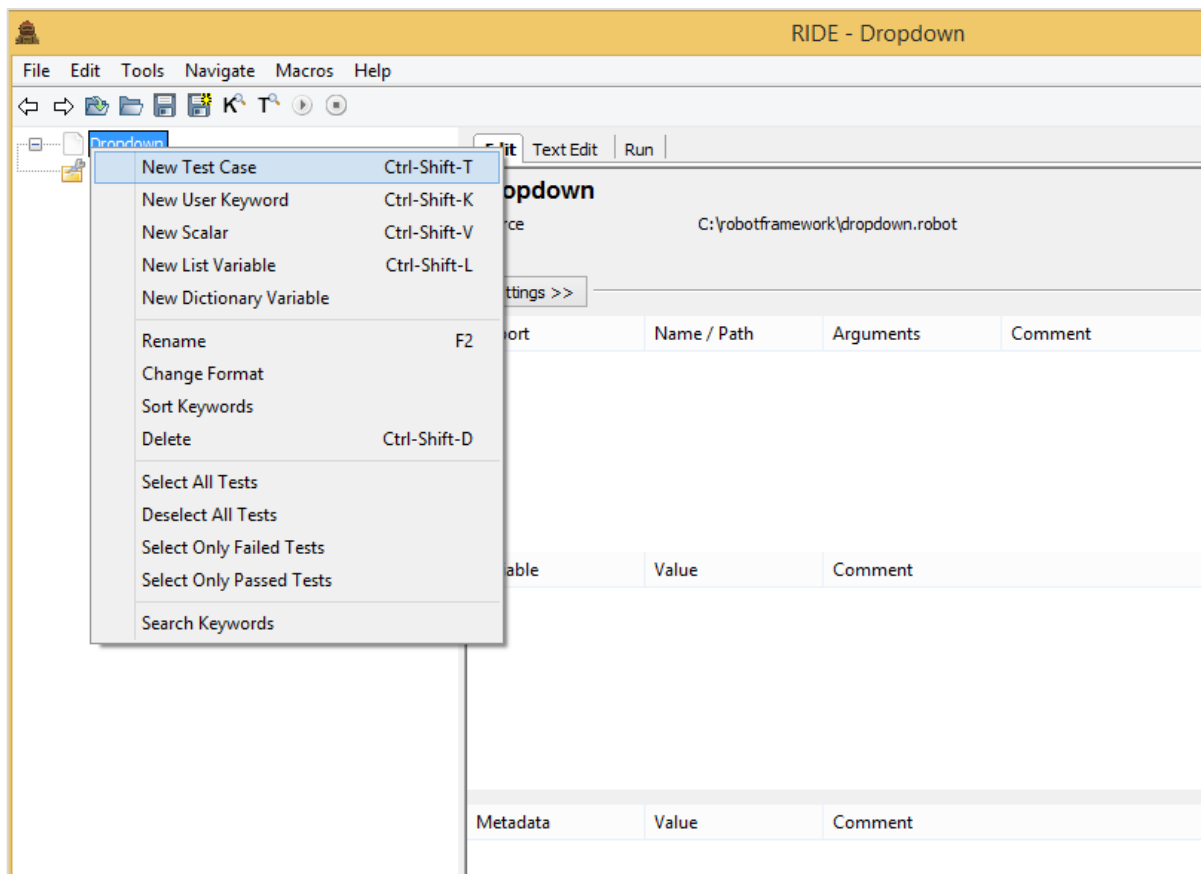


Click *New Project* and give a name to your project.



The name given is dropdown. Click OK to save the project.

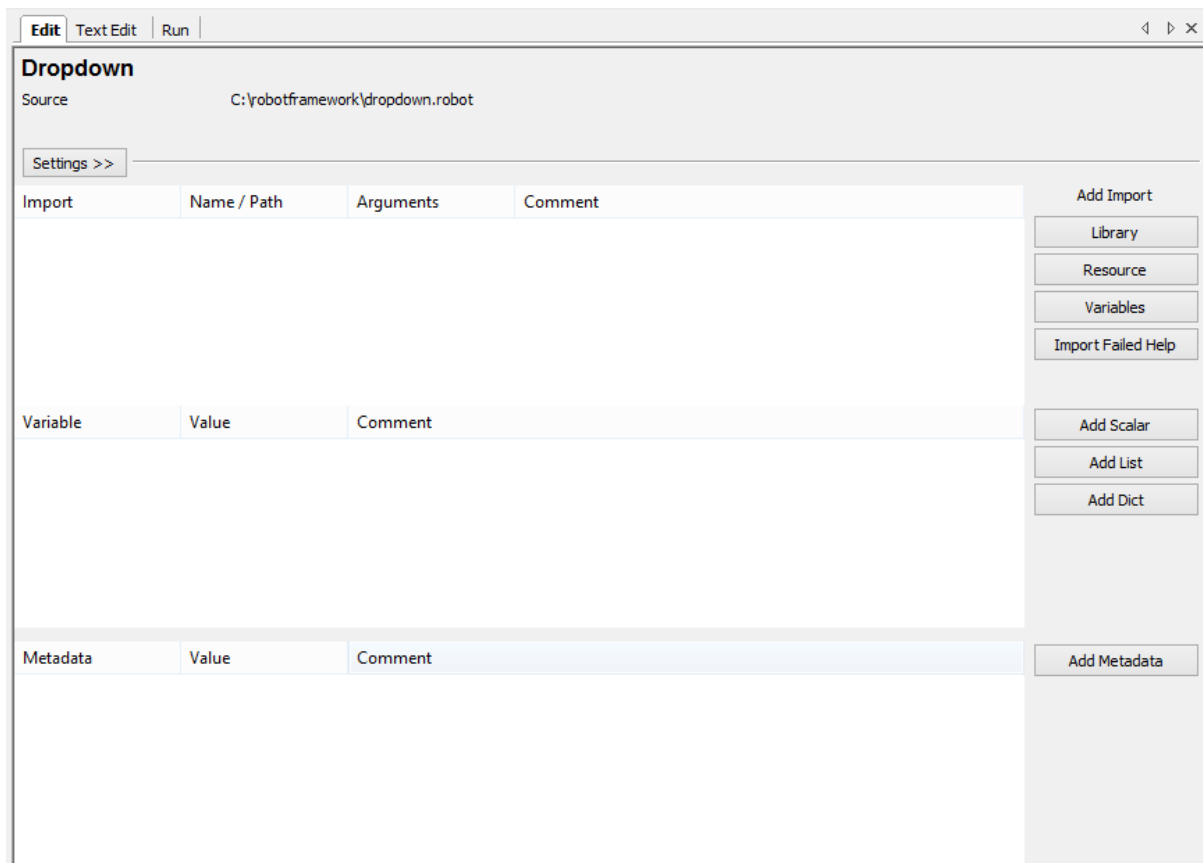
Right-click on the name of the project created and click *New Test Case*:



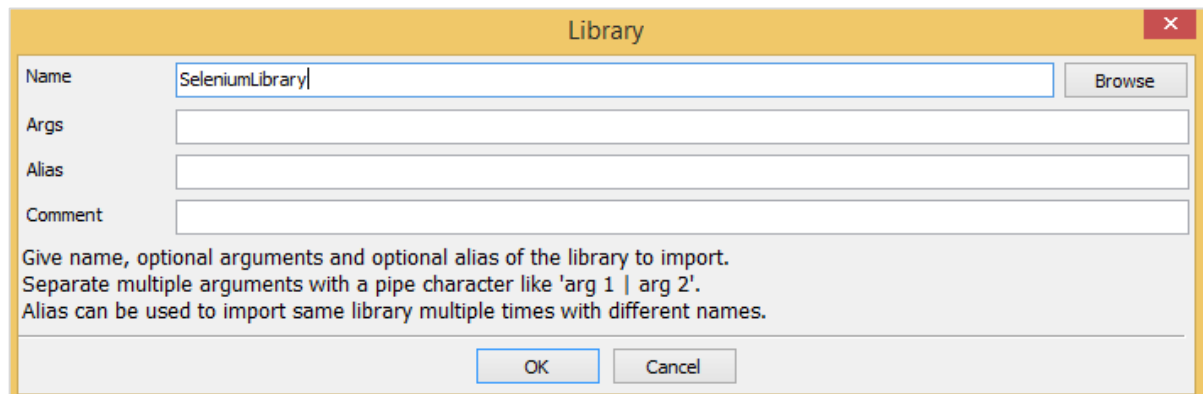
Give name to the test case and click OK to save it.

We are done with the project setup. Now, we will write test cases for the dropdown. Since we need Selenium library, we need to import the same in our project.

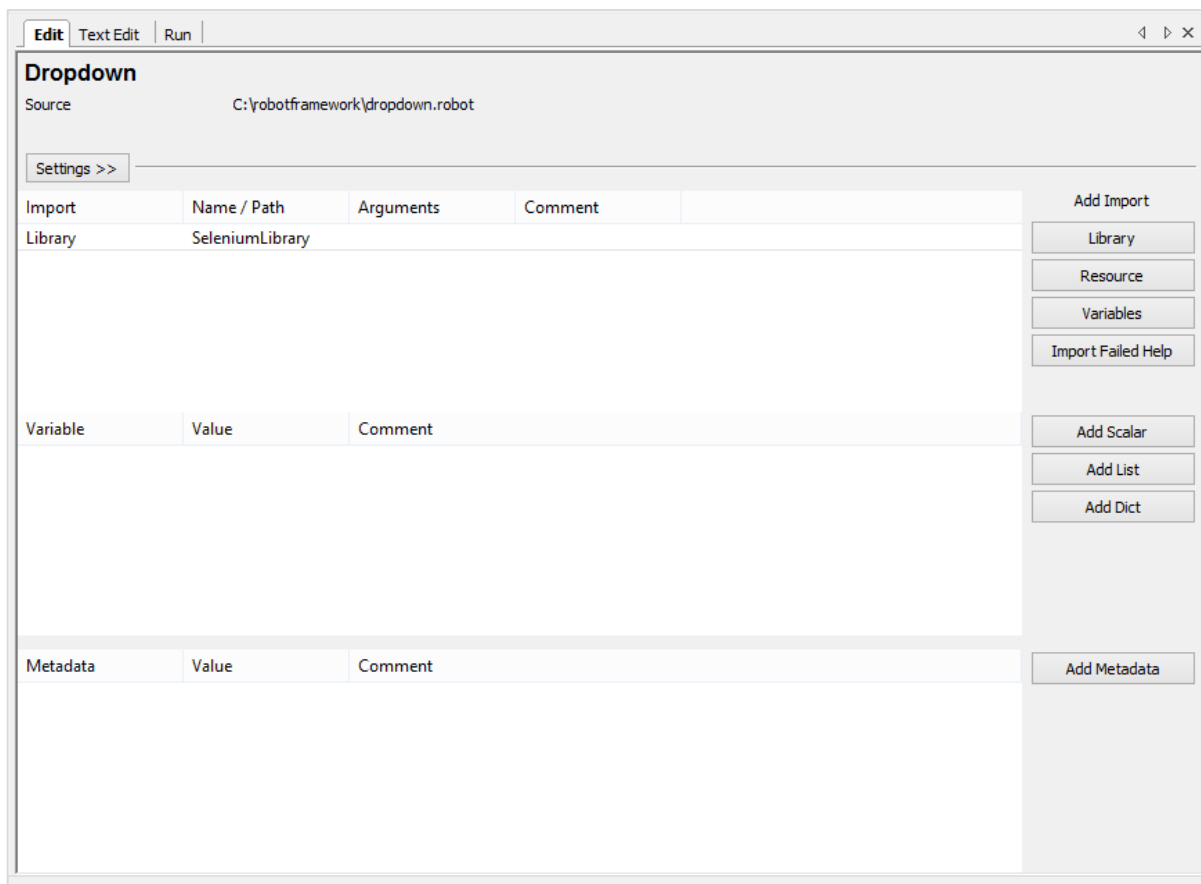
Click on your project on the left side and use *Library* from *Add Import* as shown below:



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

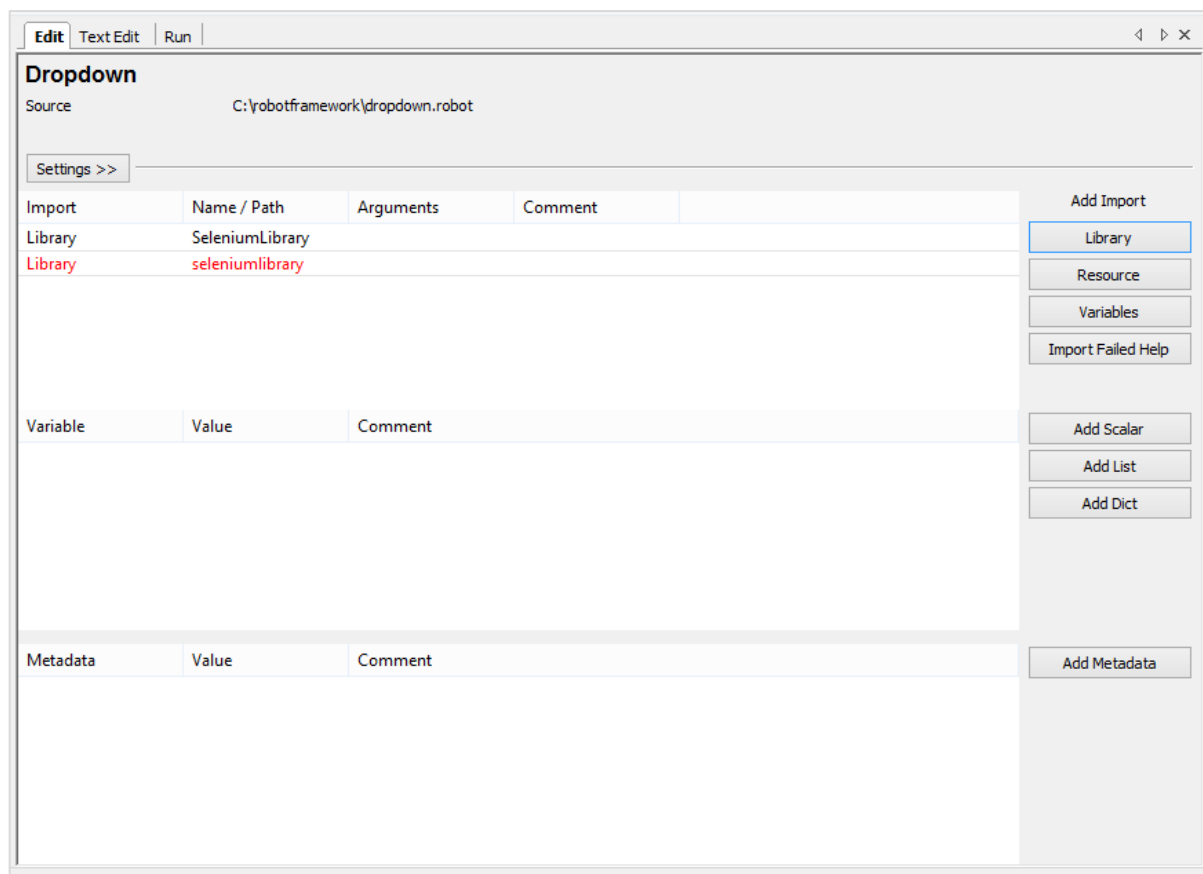


Click *OK* and the library will be displayed in the settings.



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

In case the name does not match, the library name will show in red:



Library import in red is as good as the library does not exist inside python. So now we are done with selenium library import.

Test Case for Dropdown

The test case for dropdown will select the value from the dropdown. To go about working with this, we need the locator (identifier) for that dropdown.

Consider the following html display for dropdown:

```
<select name="carbrand">
<option value="">Select car brand..</option>
<option value="audi">AUDI</option>
<option value="bmw">BMW</option>
<option value="chevrolet">CHEVROLET</option>
<option value="datsun">DATSUN</option>
</select>
```

For dropdown, *name* is the *locator*. In the above example, the *name* is *carbrand*. We also need the value so that we can select the same. The values in the above example are – *audi*, *bmw*, *chevrolet* and *datsum*.

Now, we will create a test page with dropdown, open the same in the browser and select the value from the dropdown.

The test case details will be as follows:

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

While writing the keyword for test cases in RIDE, press Ctrl + Spacebar. This gives all the details of the command.

For dropdown, we have three ways of doing it:

- Select From List By Index
- Select From List By Label
- Select From List By Value

We will work on an example to show working for all the cases mentioned above.

In our test page, we will create 3 dropdowns and will use above test cases to select the dropdown by index, label and value.

dropdown.html

```
<html>
<head>
<title>Dropdown</title>
</head>
<body>
<form name="myform" method="POST">
<div>
Dropdown By Index:
<select name="months">
<option value="">Select Months.</option>
<option value="Jan">January</option>
<option value="Feb">February</option>
<option value="Mar">March</option>
<option value="Apr">April</option>
<option value="May">May</option>
<option value="Jun">June</option>
<option value="Jul">July</option>
```

```

<option value="Aug">August</option>
<option value="Sept">September</option>
<option value="Oct">October</option>
<option value="Nov">November</option>
<option value="Dec">December</option>
</select>
</div>
<br/>
<br/>
<div>
Dropdown By Label:
<select name="days">
<option value="">Select Day..</option>
<option value="01">01</option>
<option value="02">02</option>
<option value="03">03</option>
<option value="04">04</option>
<option value="05">05</option>
<option value="06">06</option>
<option value="07">07</option>
<option value="08">08</option>
<option value="09">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23">23</option>
<option value="24">24</option>

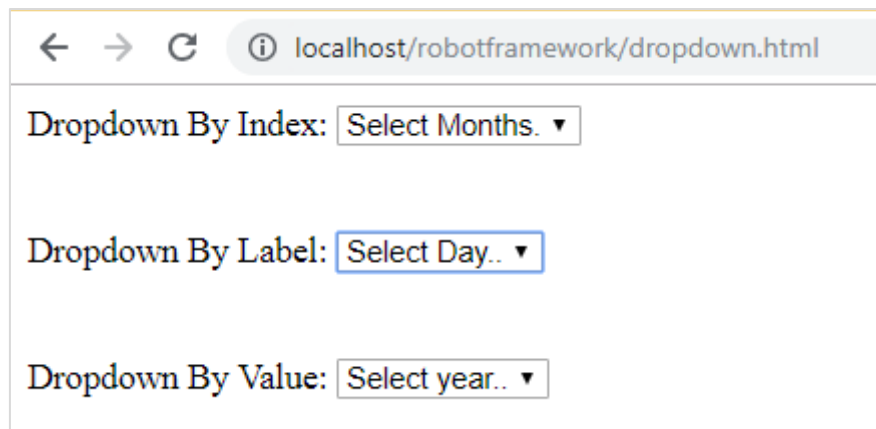
```

```

<option value="25">25</option>
<option value="26">26</option>
<option value="27">27</option>
<option value="28">28</option>
<option value="29">29</option>
<option value="30">30</option>
<option value="31">31</option>
</select>
</div>
<br/>
<br/>
<div>
Dropdown By Value:
<select name="year">
<option value="">Select year..</option>
<option value="0">2000</option>
<option value="1">2001</option>
<option value="2">2002</option>
<option value="3">2003</option>
<option value="4">2004</option>
<option value="5">2005</option>
<option value="6">2006</option>
<option value="7">2007</option>
<option value="8">2008</option>
<option value="9">2009</option>
<option value="10">2010</option>
<option value="11">2011</option>
<option value="12">2012</option>
<option value="13">2013</option>
<option value="14">2014</option>
<option value="15">2015</option>
<option value="16">2016</option>
<option value="17">2017</option>
<option value="18">2018</option>
</select>
</div>
</form>

```

```
</body>
</html>
```



We will add test cases for all 3 dropdown selection in Ride.

For index, we need to pass the locator of that dropdown – *name* or *id* and the *index* of the element that needs to be selected.

Select List by Index – Example

```
<select name="months">
<option value="">Select Months.</option> // index 0
<option value="Jan">January</option> //index 1
<option value="Feb">February</option> // index 2
<option value="Mar">March</option> // index 3
<option value="Apr">April</option> // index 4
<option value="May">May</option> // index 5
<option value="Jun">June</option> // index 6
<option value="Jul">July</option> // index 7
<option value="Aug">August</option> // index 8
<option value="Sept">September</option> //index 9
<option value="Oct">October</option> //index 10
<option value="Nov">November</option> //index 11
<option value="Dec">December</option> // index 12
</select>
```

Now, we want to select month as May so the index to be given in the test case is 5.

Select List by Label – Example

Label is seen when you open the dropdown on screen.

Dropdown By Index:

Dropdown By Label:

Dropdown By Value:

Select Day.. ▲
 01
 02
 03
 04
 05
 06
 07
 08
 09
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19

If you want to select a day, you can choose one from the dropdown.

Select From List by Value

Here is the list of the year. The list has values from 0 to 18.

```
<select name="year">
<option value="">Select year..</option>
<option value="0">2000</option>
<option value="1">2001</option>
<option value="2">2002</option>
<option value="3">2003</option>
<option value="4">2004</option>
<option value="5">2005</option>
<option value="6">2006</option>
<option value="7">2007</option>
<option value="8">2008</option>
<option value="9">2009</option>
<option value="10">2010</option>
<option value="11">2011</option>
```

```

<option value="12">2012</option>
<option value="13">2013</option>
<option value="14">2014</option>
<option value="15">2015</option>
<option value="16">2016</option>
<option value="17">2017</option>
<option value="18">2018</option>
</select>

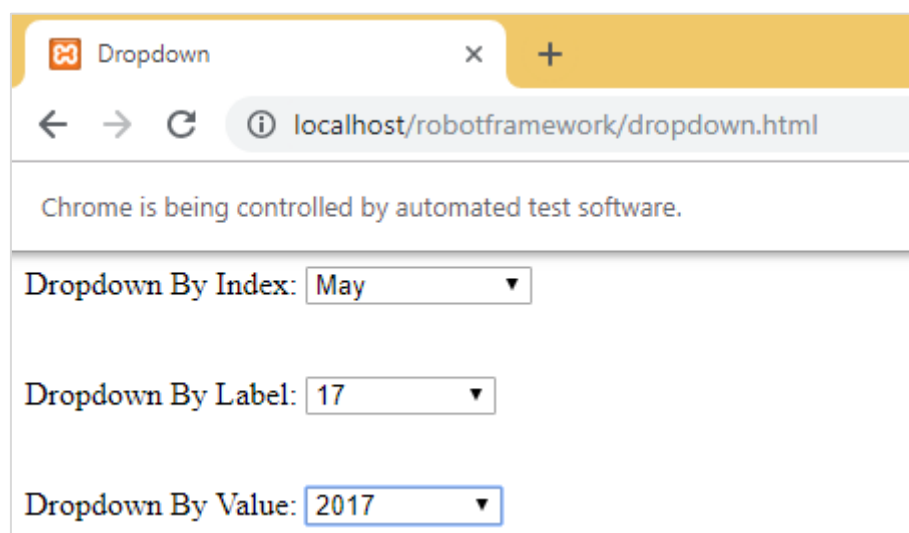
```

If you want to select any year, take the value corresponding to the year and add the same in test case. For example, if you want to select year 2017 the value is 17.

Here is the final list of test cases:

1	Open Browser	http://localhost/robotframework/d	chrome		
2	Select From List By Index	name:months	5		
3	Select From List By Label	name:days	17		
4	Select From List By Value	name:year	17		
5					
6					
7					

After execution, here is the selection done for dropdowns based on the test case:



Execution Details

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\RIDE5y_mcs.d\argfile.txt --listener C
=====
Dropdown
=====
TC1 | PASS |
=====
Dropdown | PASS |
=====
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
=====
Output: c:\users\appdata\local\temp\RIDE5y_mcs.d\output.xml
Log: c:\users\appdata\local\temp\RIDE5y_mcs.d\log.html
Report: c:\users\appdata\local\temp\RIDE5y_mcs.d\report.html
test finished 20181014 15:15:13
  
```

Report Details

Dropdown Test Report

Generated 20181014 15:15:13 GMT+05:30
2 minutes 20 seconds ago

Summary Information

Status: All tests passed
Start Time: 20181014 15:15:06.621
End Time: 20181014 15:15:13.239
Elapsed Time: 00:00:06.618
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
Dropdown	1	1	0	00:00:07	

Test Details

Totals | Tags | Suites | Search

Type: ☐ Critical Tests ☐ All Tests

Log Details

Dropdown Test Log

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
Dropdown	1	1	0	00:00:07	

Generated
 20181014 15:15:13 GMT+05:30
 2 minutes 50 seconds ago

Test Execution Log

SUITE

 Dropdown

Full Name: Dropdown
 Source: C:\robotframework\dropdown.robot
 Start / End / Elapsed: 20181014 15:15:06.621 / 20181014 15:15:13.239 / 00:00:06.618
 Status: 1 critical test, 1 passed, 0 failed
 1 test total, 1 passed, 0 failed

TEST

 TC1

Test case details from log

TEST

 TC1

Full Name: Dropdown.TC1
 Start / End / Elapsed: 20181014 15:15:07.440 / 20181014 15:15:13.234 / 00:00:05.794
 Status: PASS (critical)

KEYWORD

 SeleniumLibrary.Open Browser http://localhost/robotframework/dropdown.html, chrome

Documentation: Opens a new browser instance to the given url.
 Start / End / Elapsed: 20181014 15:15:07.440 / 20181014 15:15:12.029 / 00:00:04.589
 15:15:07.440 INFO Opening browser 'chrome' to base url 'http://localhost/robotframework/dropdown.html'.

KEYWORD

 SeleniumLibrary.Select From List By Index name:months, 5

Documentation: Selects options from selection list locator by indexes.
 Start / End / Elapsed: 20181014 15:15:12.032 / 20181014 15:15:12.583 / 00:00:00.551
 15:15:12.034 INFO Selecting options from selection list 'name:months' by index 5.

KEYWORD

 SeleniumLibrary.Select From List By Label name:days, 17

Documentation: Selects options from selection list locator by labels.
 Start / End / Elapsed: 20181014 15:15:12.585 / 20181014 15:15:12.907 / 00:00:00.322
 15:15:12.588 INFO Selecting options from selection list 'name:days' by label 17.

KEYWORD

 SeleniumLibrary.Select From List By Value name:year, 17

Documentation: Selects options from selection list locator by values.
 Start / End / Elapsed: 20181014 15:15:12.909 / 20181014 15:15:13.232 / 00:00:00.323
 15:15:12.911 INFO Selecting options from selection list 'name:year' by value 17.

Conclusion

We have seen how to work with dropdown by value, index and label. We can refer to logs and reports to get the details of the test case executed.

13. Robot Framework — Working With Keywords

In Robot Framework, test cases are constructed in test case tables using keywords. In this chapter, we will cover the details on keywords used in Robot Framework. There are 2 types of keywords used in Robot:

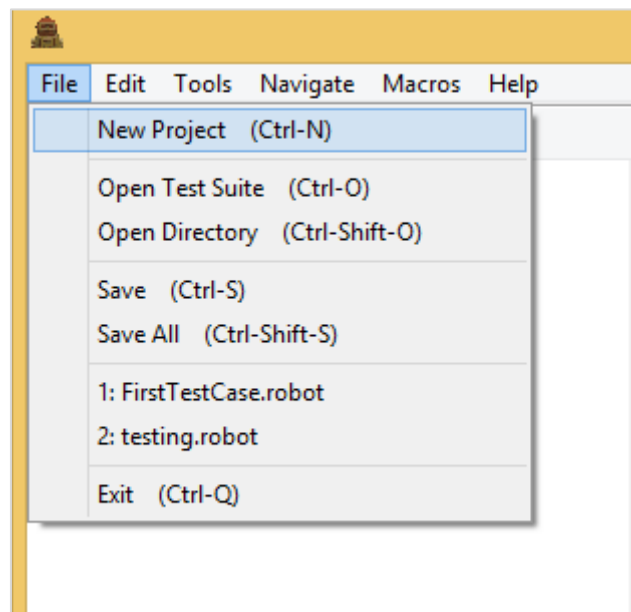
- Library Keywords
- User Defined Keywords

Library Keywords

Library Keywords are keywords that come from the library we import in Robot Framework. We will now take a look at the Selenium library, which helps us interact with the browser. We will discuss some of the important keywords associated with selenium library.

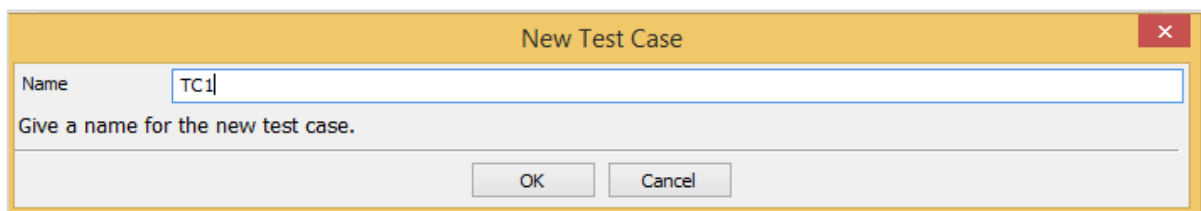
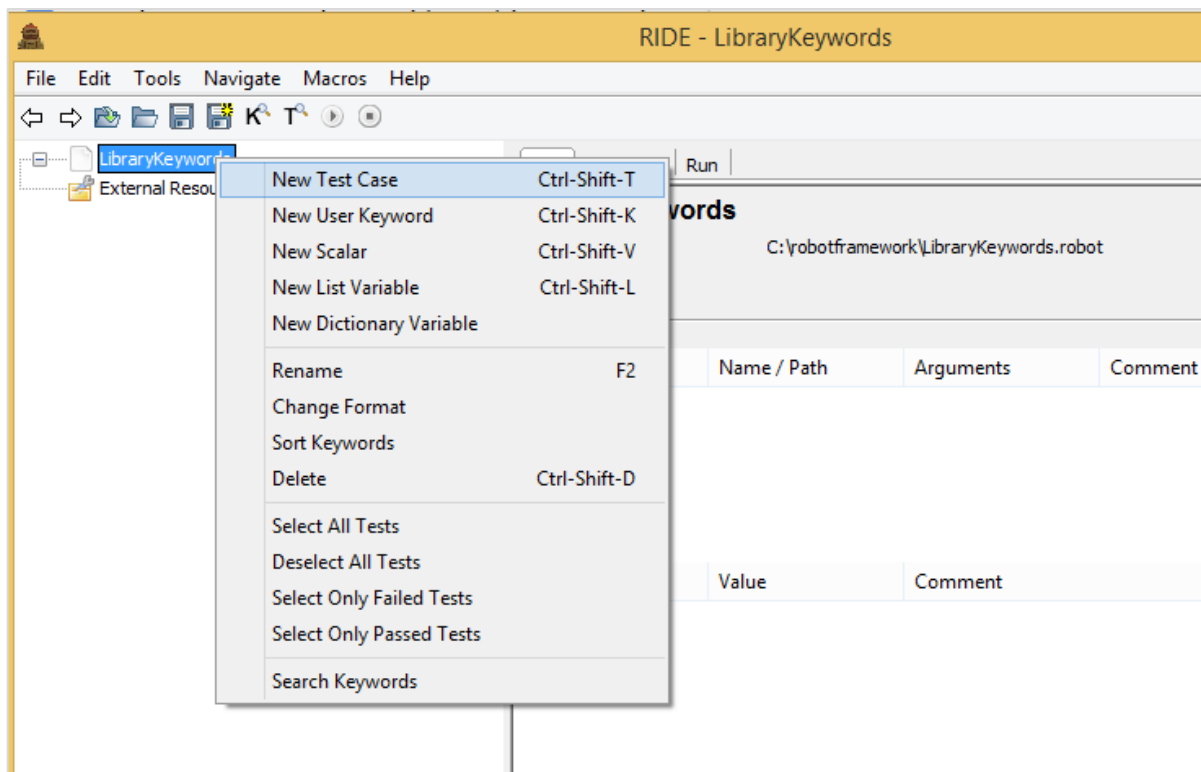
Follow the steps shown below to import Selenium library:

The details relating to the installation of Selenium library is discussed in chapter “**Working with Browsers using Selenium Library**”. Open ride using **ride.py** from the command line.



Click on New Project and give name to your project. The name given to the project is **LibraryKeywords**.

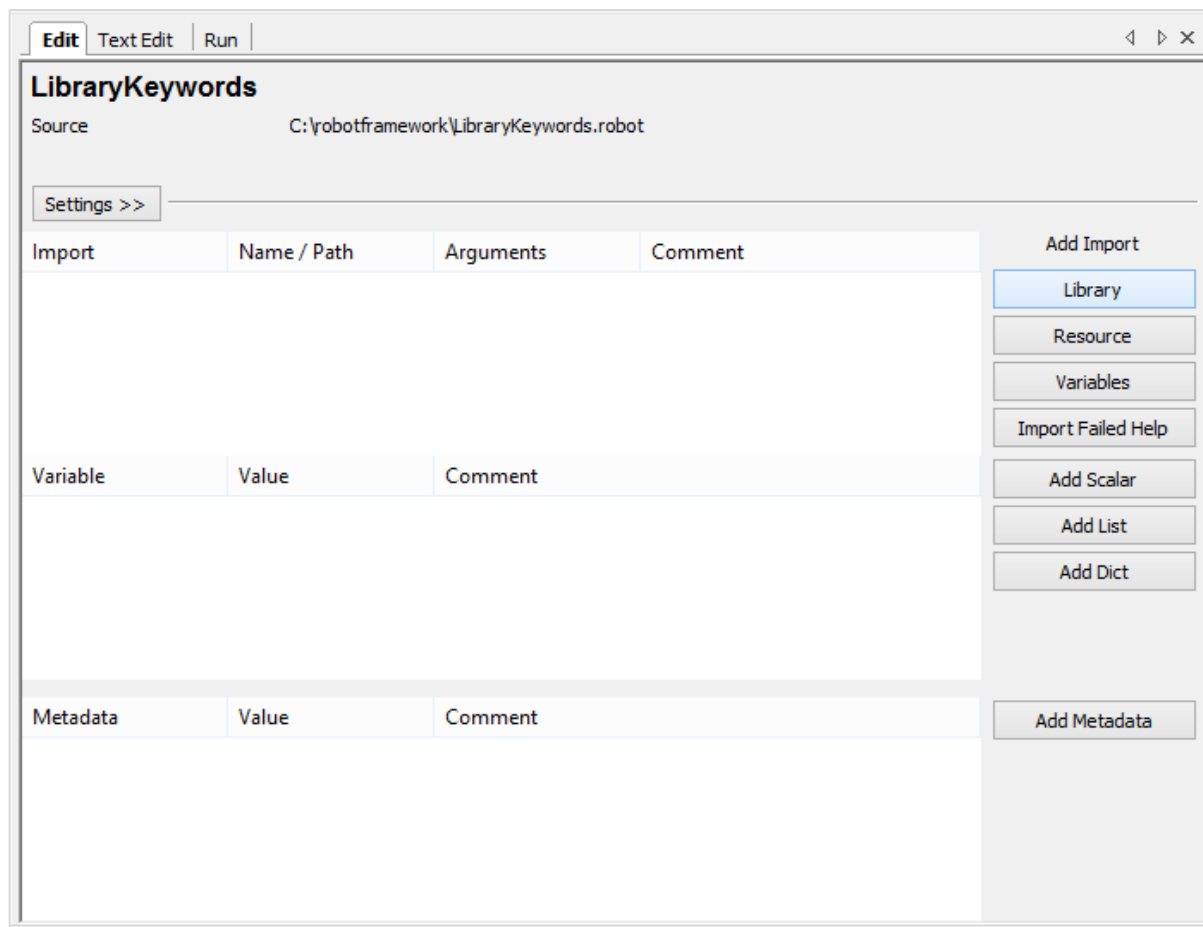
Right-click on the name of the project created and click on *New Test Case*:



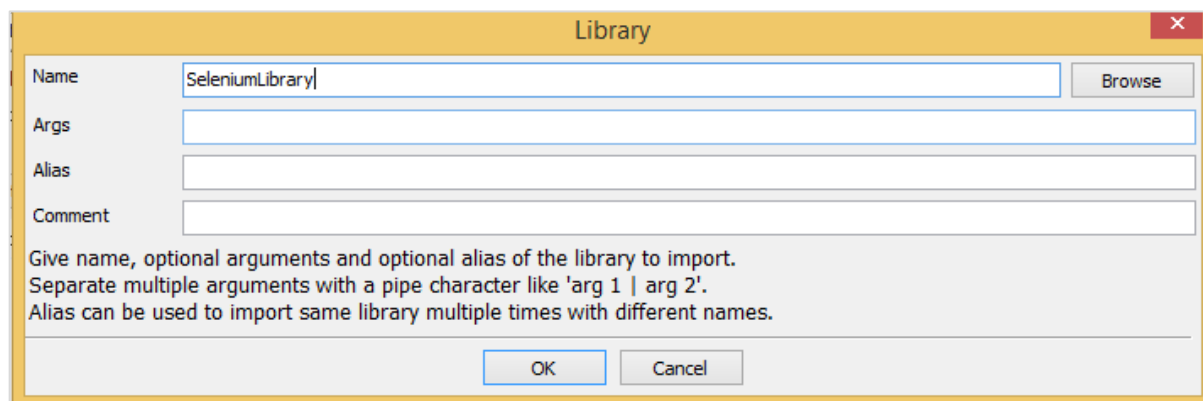
Give a name to the test case and click OK.

We are done with the project setup. Now, we will write test cases to show the working of library keywords. Since we need Selenium library, we need to import the same in our project.

Click on your project on the left side and click Library.



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.

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

Now will create test case in the project created and use a few important keywords.

Click on your test case created TC1 and in the tabular form enter the keywords to open the browser and enter data inside the form opened.

Here is a simple test case using Library Keywords:

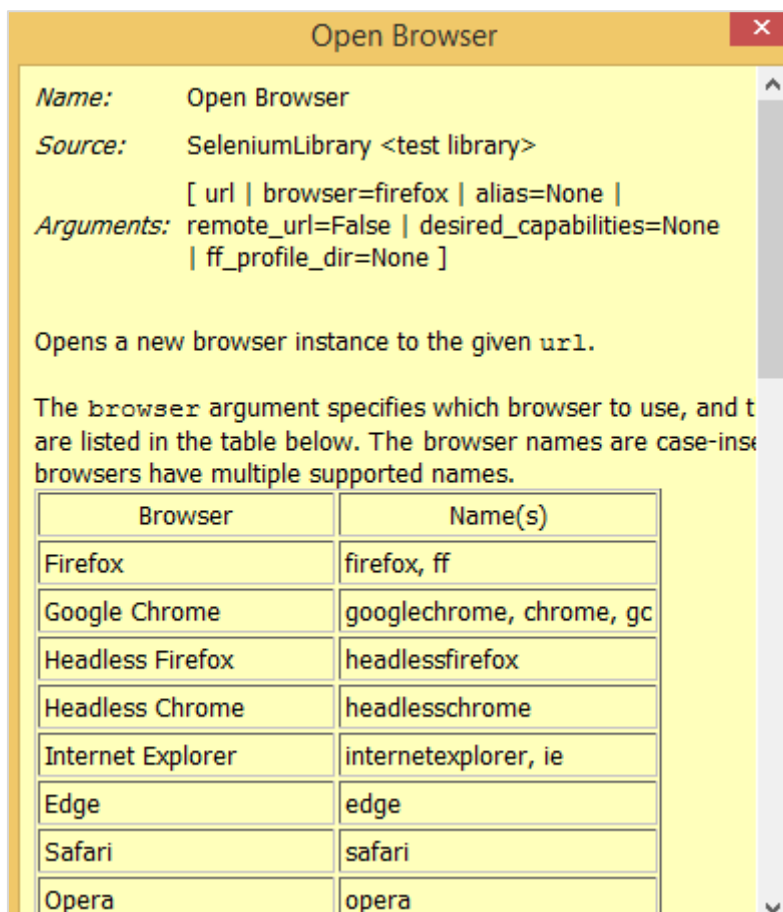
1	Open Browser	https://www.tutorialspoint.com	chrome
2	Maximize Browser Window		
3	Input Text	name=search	This is entered from robotframework testcase

Open Browser

To get more details of this keyword, while typing the keyword press ctrl + spacebar. It will show the details of the library keyword entered.

Here is an example for Open Browser, and if any help required for that keyword you can make use of ctrl + spacebar while typing the keyword.

Open Browser Keyword Details



Open Browser

Name: Open Browser

Source: SeleniumLibrary <test library>

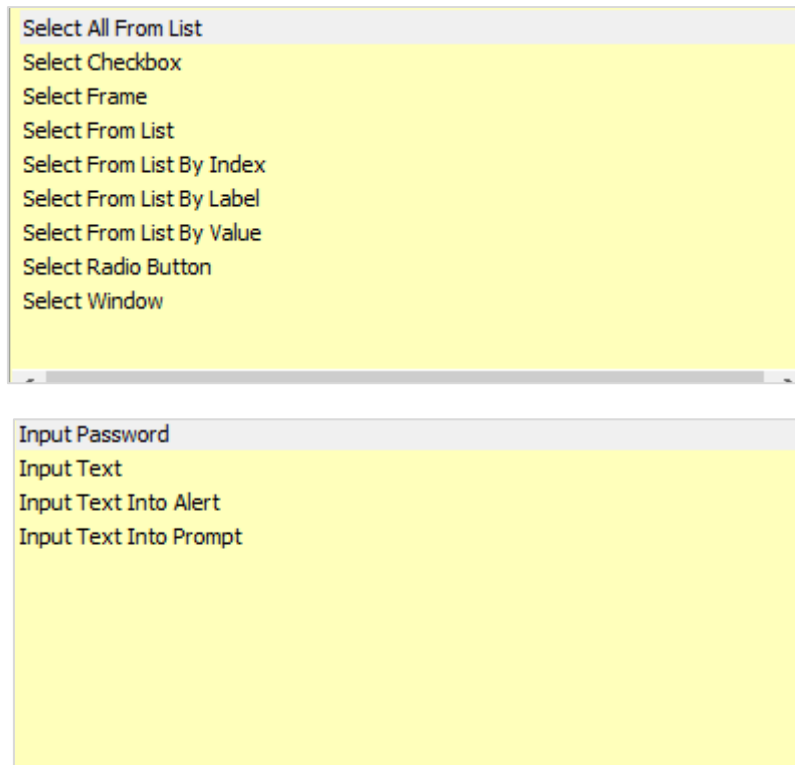
Arguments: [url | browser=firefox | alias=None | remote_url=False | desired_capabilities=None | ff_profile_dir=None]

Opens a new browser instance to the given url.

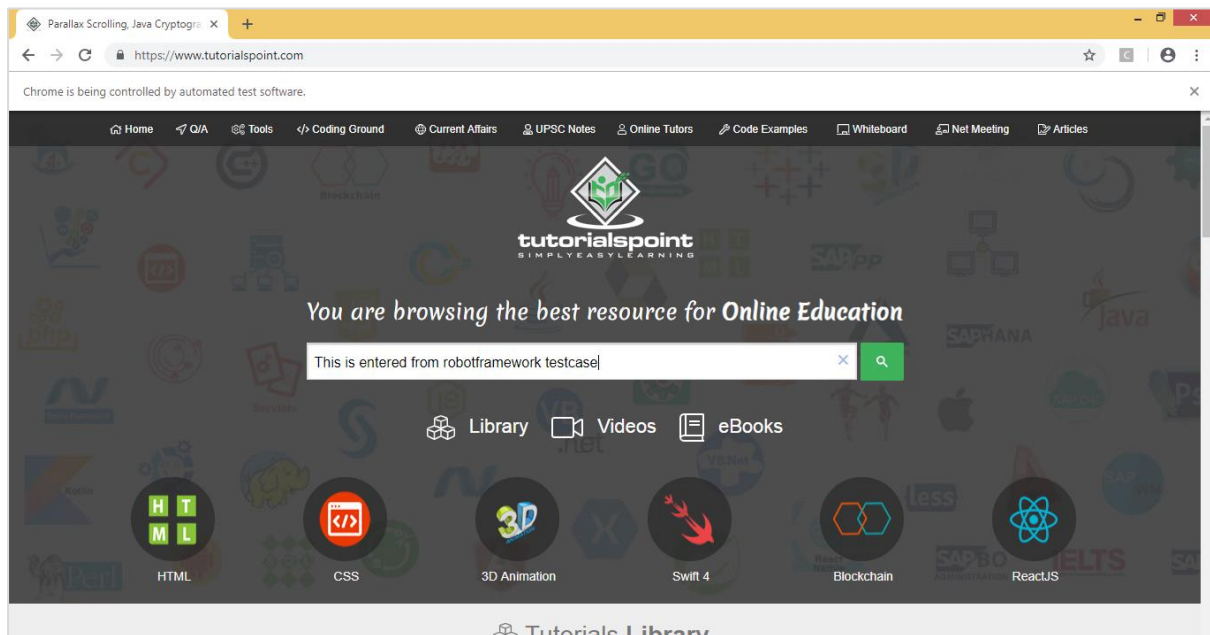
The **browser** argument specifies which browser to use, and the names are listed in the table below. The browser names are case-insensitive. Some browsers have multiple supported names.

Browser	Name(s)
Firefox	firefox, ff
Google Chrome	googlechrome, chrome, gc
Headless Firefox	headlessfirefox
Headless Chrome	headlesschrome
Internet Explorer	internetexplorer, ie
Edge	edge
Safari	safari
Opera	opera

Similarly, we have Library keywords to work with Input, Radio, Text, etc



We will execute the test case we entered to open the browser with URL: <https://www.tutorialspoint.com> and enter details in the input text.



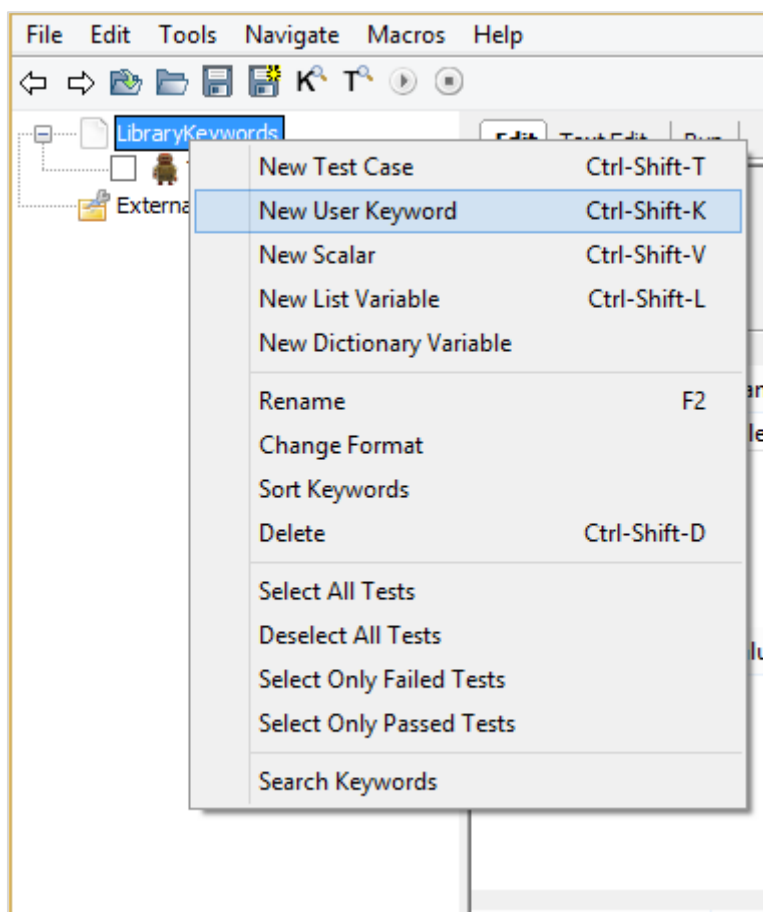
We have executed the test case. You can see the textbox has all the details we gave in the test case.

User-defined Keywords

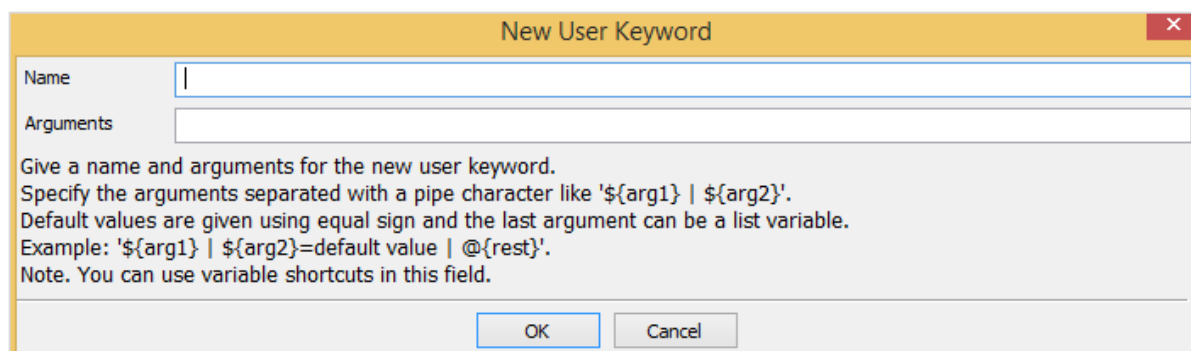
User-defined keywords can be created to perform a particular action in the test case or it can also be created using the library keywords and built-in keywords in robot framework. We will work on an example and see how we can create keywords for our test case.

We will use the same project that we created above and create user-defined keywords in that and use in the test case.

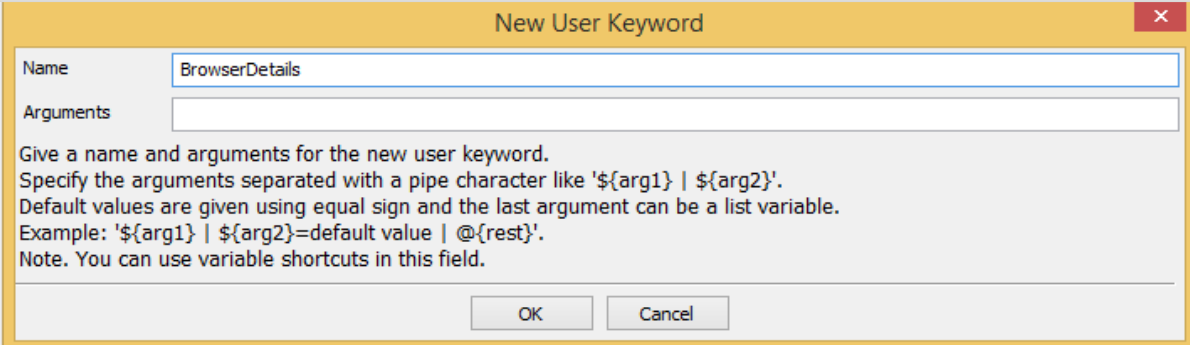
To create keyword in Ride, right-click on your project and click on *New User Keyword* as shown below:



Upon clicking New User Keyword, a screen appears as shown below:



Enter the Name of the keyword and click OK. The screen also shows Arguments. We will discuss what arguments have to do with Keywords in a subsequent section.



New User Keyword

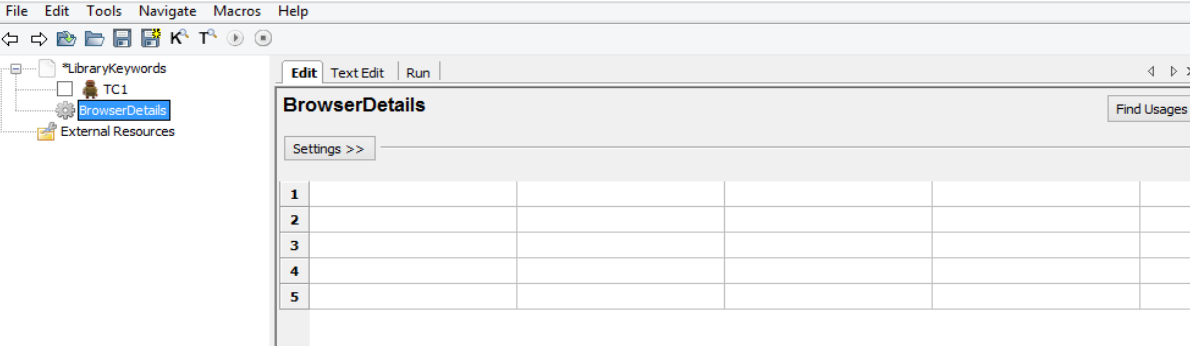
Name:

Arguments:

Give a name and arguments for the new user keyword.
Specify the arguments separated with a pipe character like '\${arg1} | \${arg2}'.
Default values are given using equal sign and the last argument can be a list variable.
Example: '\${arg1} | \${arg2}=default value | @rest'.
Note. You can use variable shortcuts in this field.

OK Cancel

We have given the name BrowserDetails to the keyword. Click OK to save it. The keyword BrowserDetails is created.



File Edit Tools Navigate Macros Help

LibraryKeywords TC1 BrowserDetails External Resources

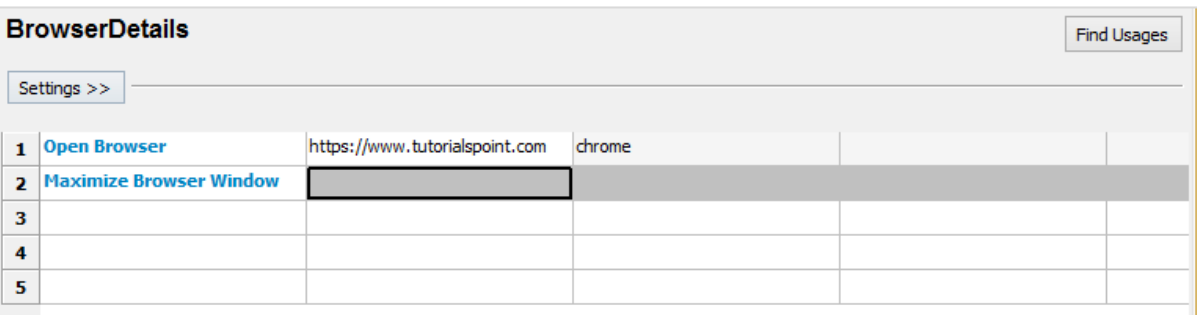
BrowserDetails Find Usages

Settings >>

1				
2				
3				
4				
5				

To test the URL in the browser, we repeatedly have to enter open browser, **maximize browser** keywords.

Now, we will create a user-defined keyword that will have *open browser* and *maximize browser* details. The keyword created will be used in our test case.



BrowserDetails Find Usages

Settings >>

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

Our BrowserDetails keyword is a combination of other keywords used repeatedly.

Now, we will use the keyword created in the test case as shown below.

Test case

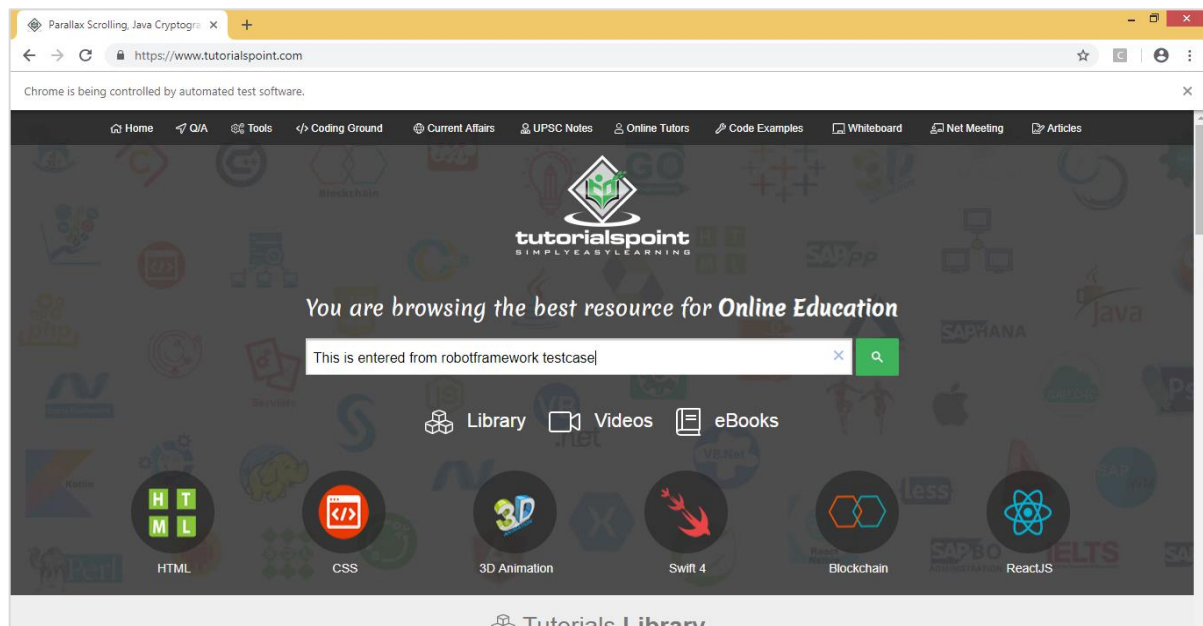
1	Open Browser	https://www.tutorialspoint.com	chrome
2	Maximize Browser Window		
3	Input Text	name=search	This is entered from robotframework testcase

Considering the above test case, we are going to use the user-defined keyword *BrowserDetails*.

We will now replace 1 and 2 keywords with the user-defined keyword:

1	BrowserDetails		
2	Input Text	name=search	This is entered from robotframework testcase
3			

Let us now run the test case to see the output:



The execution of the test case works perfectly fine.

Now, we will see the use-case of arguments in keywords.

Here is the keyword that we created:

BrowserDetails					Find Usages
Settings >>					
1	Open Browser	https://www.tutorialspoint.com	chrome		
2	Maximize Browser Window				
3					
4					
5					

The name of the keyword is *BrowserDetails*. We can use this keyword in other test cases created under the project. The keyword contains the URL of the browser hardcoded. If we want to use the keyword in another test case with a different URL, it will not be possible.

We can use arguments to help us with the hardcoded parameters. We will go back to the keyword created and make use of arguments.

Edit

Text Edit

Run

Find Usages

BrowserDetails

Settings <<

Documentation

Tags

<Add New>

Arguments

Teardown

Return Value

Timeout

Edit

Clear

Edit

Clear

Edit

Clear

Edit

Clear

Edit

Clear

1

Open Browser

https://www.tutorialspoint.com

chrome

2

Maximize Browser Window

3

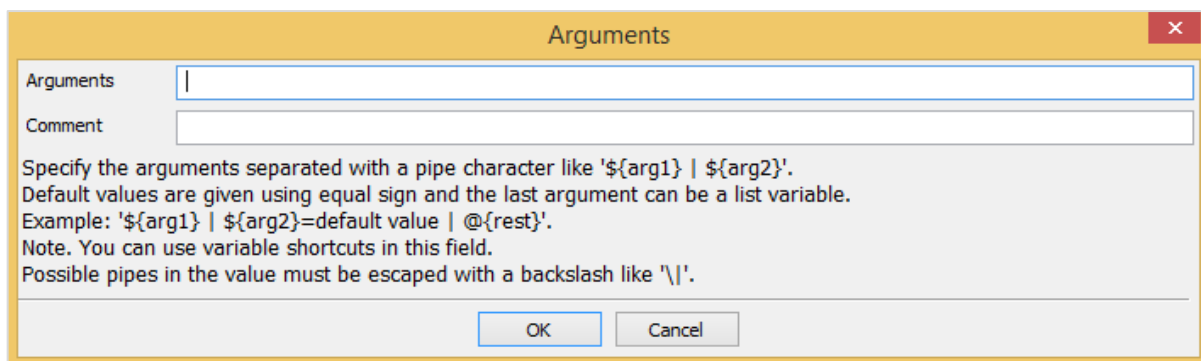
4

5

6

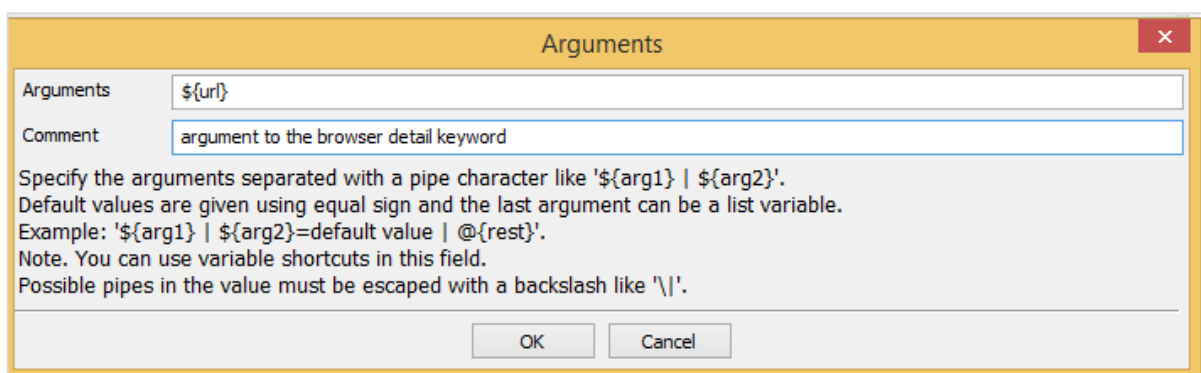
7

Click on Edit against the Arguments.



The dialog box is titled "Arguments" and has a close button (X) in the top right corner. It contains two input fields: "Arguments" and "Comment". The "Arguments" field is currently empty, and the "Comment" field is also empty. Below the input fields, there is a text area with the following text: "Specify the arguments separated with a pipe character like '\${arg1} | \${arg2}'. Default values are given using equal sign and the last argument can be a list variable. Example: '\${arg1} | \${arg2}=default value | @rest'. Note. You can use variable shortcuts in this field. Possible pipes in the value must be escaped with a backslash like '\\'. At the bottom of the dialog box, there are two buttons: "OK" and "Cancel".

Enter the argument to be used with the keyword.

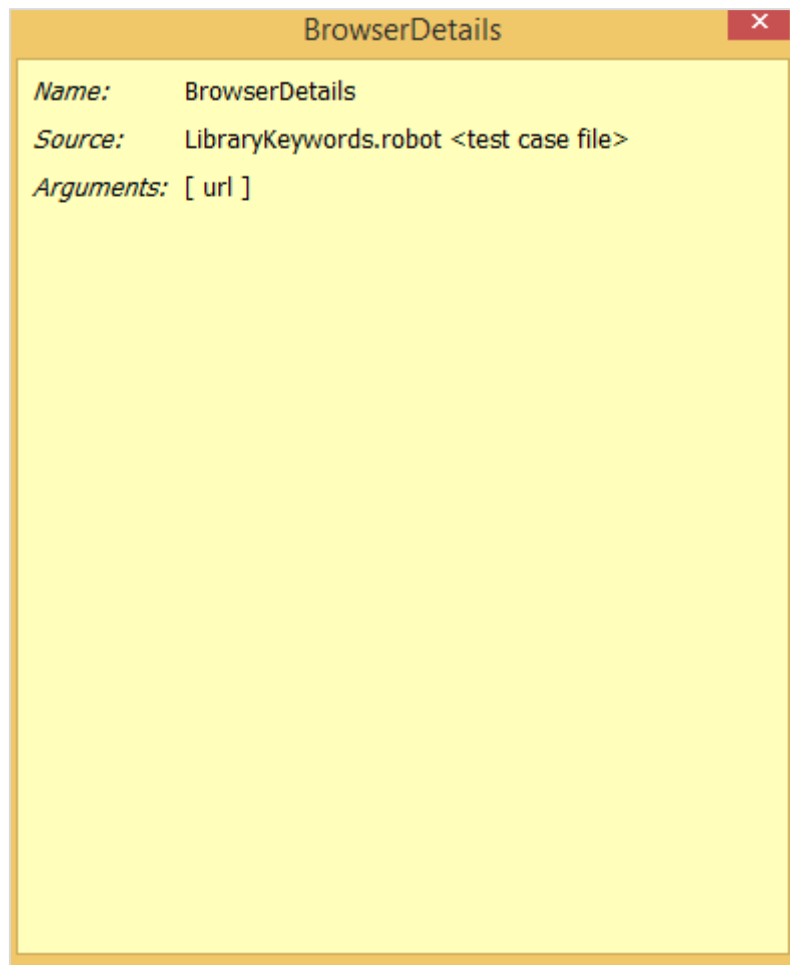


The dialog box is titled "Arguments" and has a close button (X) in the top right corner. It contains two input fields: "Arguments" and "Comment". The "Arguments" field now contains the text "\${url}", and the "Comment" field contains the text "argument to the browser detail keyword". Below the input fields, there is a text area with the following text: "Specify the arguments separated with a pipe character like '\${arg1} | \${arg2}'. Default values are given using equal sign and the last argument can be a list variable. Example: '\${arg1} | \${arg2}=default value | @rest'. Note. You can use variable shortcuts in this field. Possible pipes in the value must be escaped with a backslash like '\\'. At the bottom of the dialog box, there are two buttons: "OK" and "Cancel".

If there is more than 1 argument, you can separate them using pipe (|). We will now use the argument in the Keyword specified as follows:

In the test case, when you type the user-defined keyword and press Ctrl + Spacebar, it gives the details of the keyword along with the arguments.

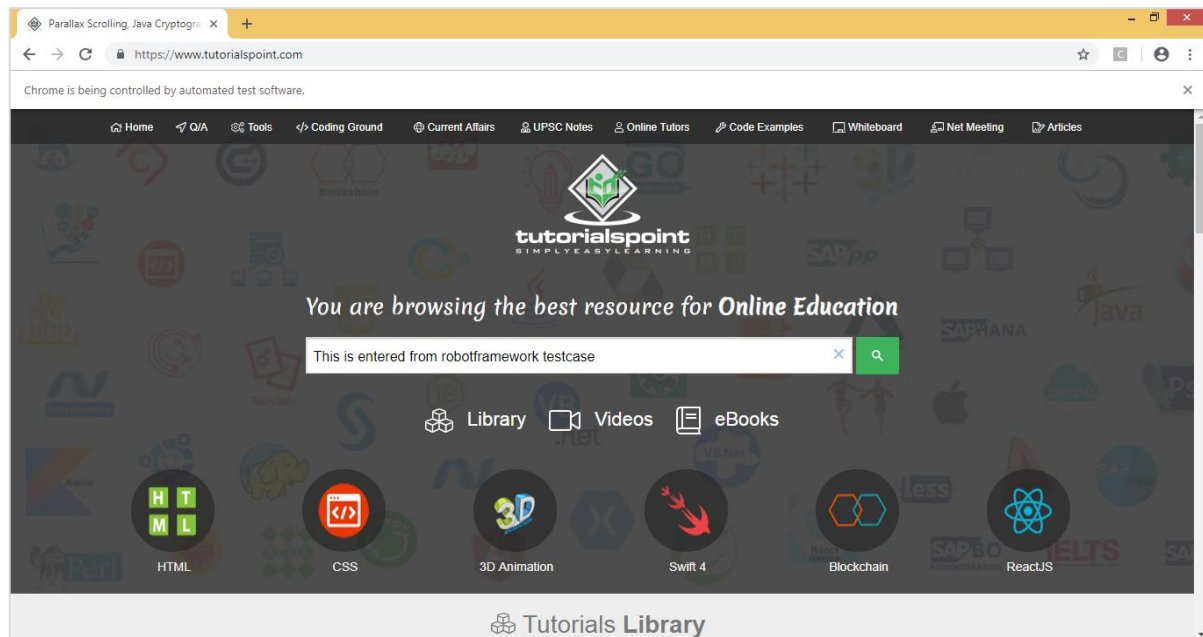
Following are the details for keyword BrowserDetails:



The test case now will have the URL to be passed as argument.

1	BrowserDetails	https://www.tutorialspoint.com		
2	Input Text	name=search	This is entered from robotframework testcase	
3				
4				

Let us now run the test case to see the output:



The Keyword and the arguments passed to the user-defined keyword are working fine.

Let us now change the URL; we will use <https://www.google.com>.

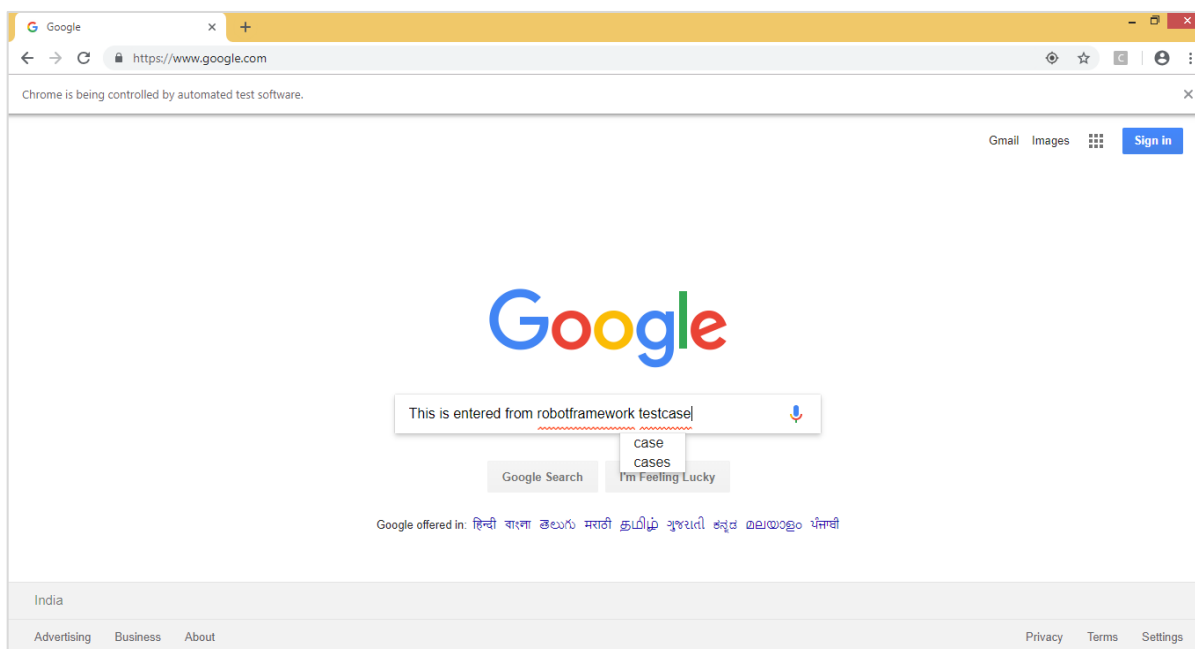
1	BrowserDetails	https://www.google.com	
2	Input Text	id=lst-ib	This is entered from robotframework testcase
3			
4			
5			

The URL for keyword BrowserDetails is changed to <https://www.google.com>.

We have changed the argument to Input Text to the id available from google site. To get the id or name or class of the input field, you can inspect and check in the browser.

Let us run the above test case and see the output.

Upon successful execution, the above test case generates the following output:



Conclusion

In this chapter, we have seen how to get help for built-in keywords. We have also seen how to create user-defined keywords, which can be a combination of library keywords and built-in keywords.

14. Robot Framework — Working With Variables

In this chapter, we will discuss how to create and use variables in Robot Framework. Variables are used to hold a value, which can be used in test cases, user-defined keywords, etc.

We are going to discuss following variables available in Robot Framework

- Scalar Variable
- List Variable
- Dictionary Variable

We will understand the working of each of this variable with the help of test cases in RIDE.

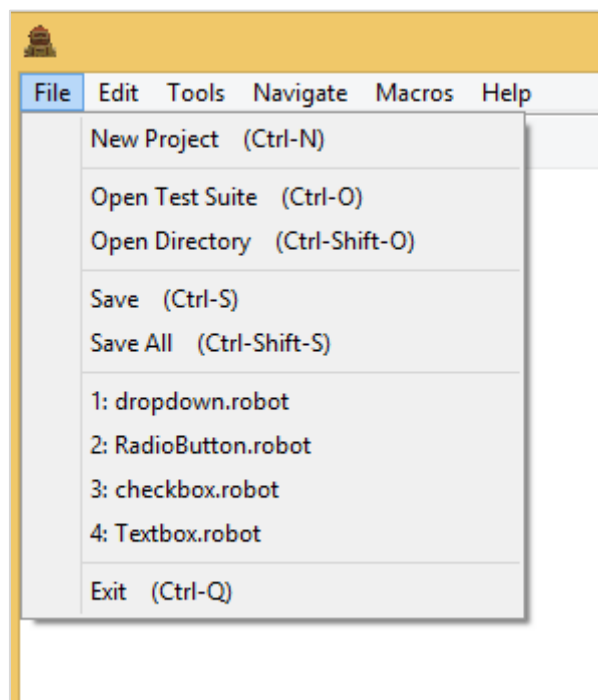
Scalar Variable

Scalar variables will be replaced with the value they are assigned. The syntax for scalar variable is as follows:

```
${variablename}
```

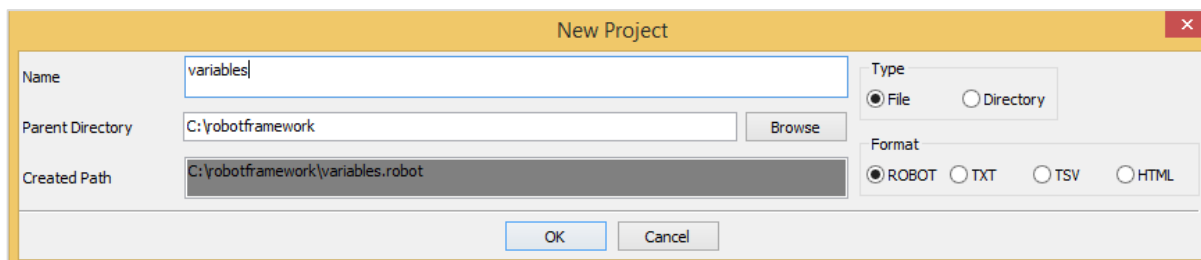
We can use scalar variable to store strings, objects, lists, etc. We will first create a simple test case and make use of scalar variable in it.

Open RIDE using **ride.py** in the command line and create a new project.



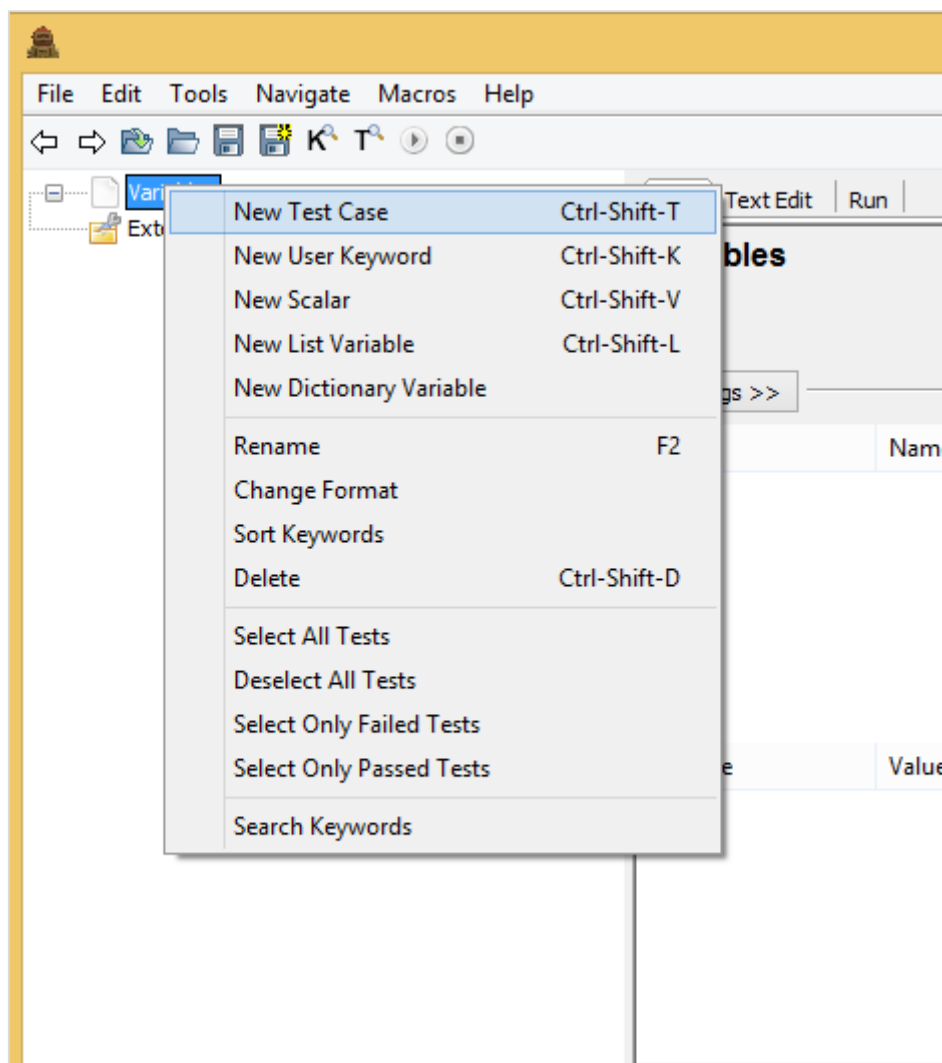
Click **New Project**.

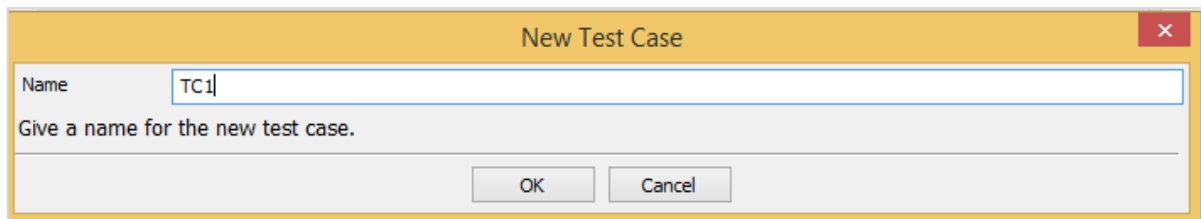
Now, give a name to your project.



The name given is *variables*. Click OK to save the project.

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



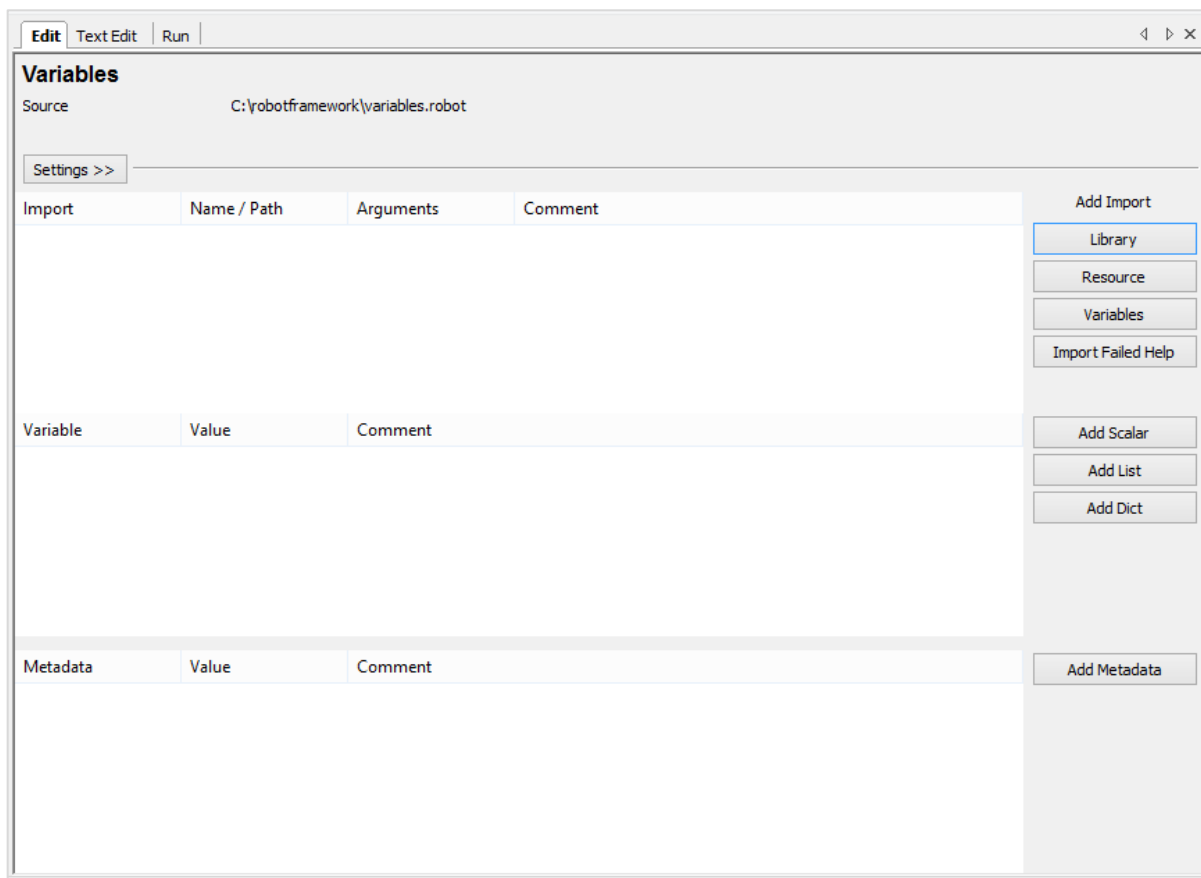


A dialog box titled "New Test Case" with a close button (X) in the top right corner. It contains a text input field labeled "Name" with the text "TC1" entered. Below the input field is a label "Give a name for the new test case." At the bottom are two buttons: "OK" and "Cancel".

Give a name to the test case and click OK.

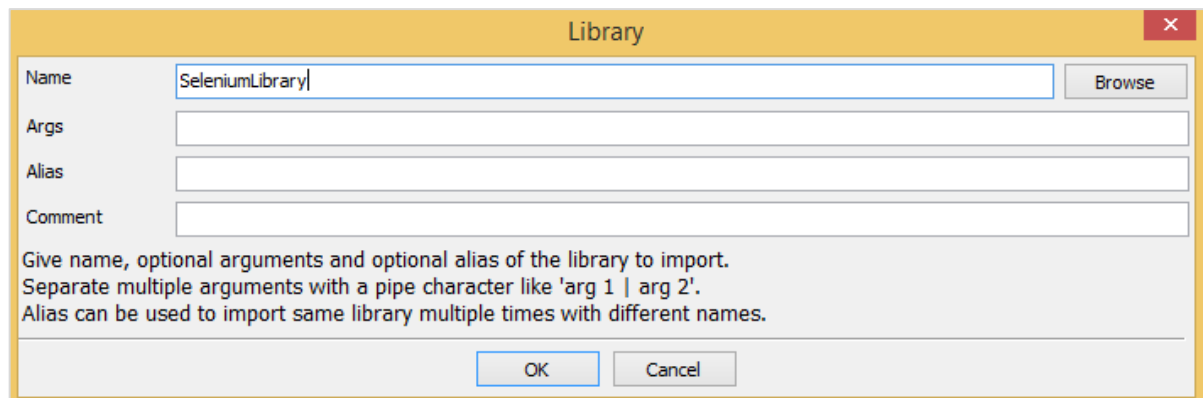
We are done with the project setup and now will write test cases for the scalar variables to be used in our test case. Since we need Selenium library, we need to import the same in our project.

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



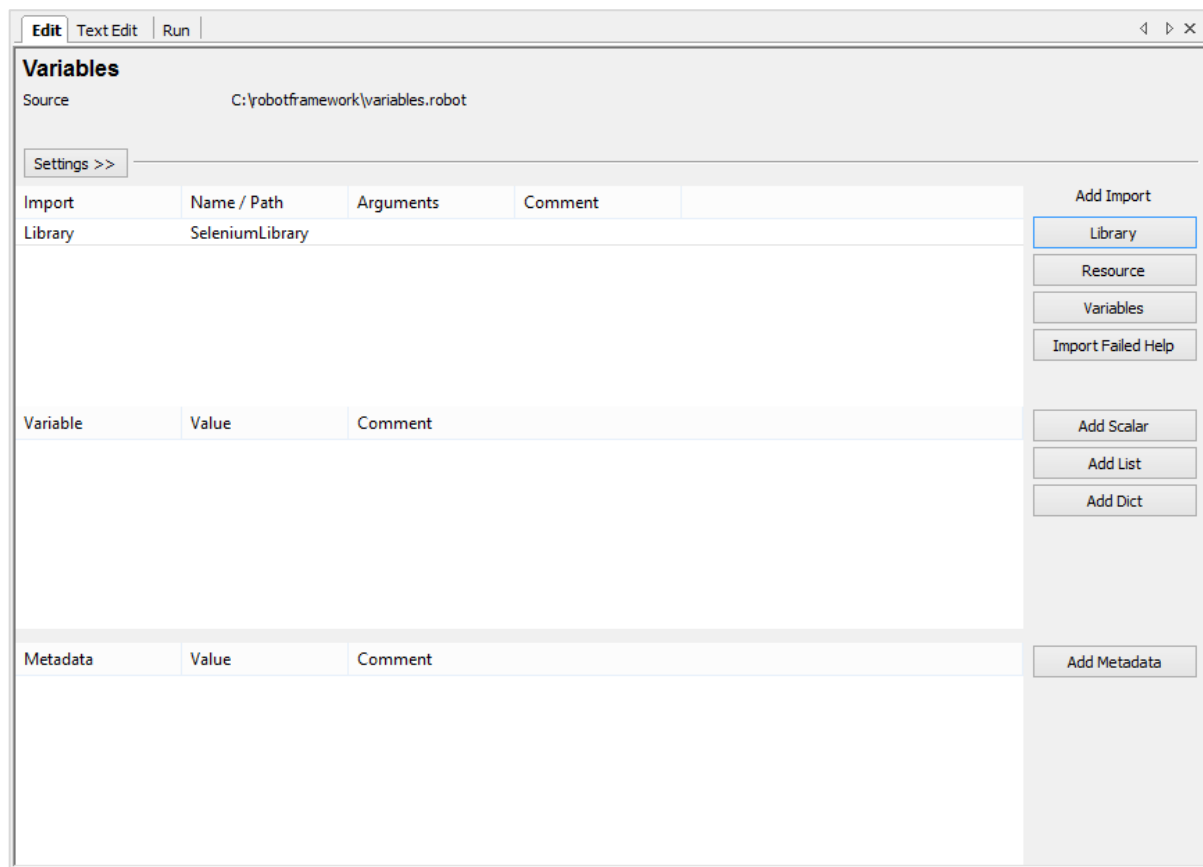
The "Variables" window in the Robot Framework IDE. It shows the source file "C:\robotframework\variables.robot". Below the source path is a "Settings >>" button. The main area contains three tables: "Import", "Variable", and "Metadata". The "Import" table has columns "Import", "Name / Path", "Arguments", and "Comment". The "Variable" table has columns "Variable", "Value", and "Comment". The "Metadata" table has columns "Metadata", "Value", and "Comment". On the right side, there is an "Add Import" section with buttons for "Library", "Resource", "Variables", and "Import Failed Help". Below that is an "Add Scalar" section with buttons for "Add Scalar", "Add List", and "Add Dict". At the bottom right is an "Add Metadata" button.

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



A dialog box titled "Library" with a yellow header bar. It contains four input fields: "Name" (with "SeleniumLibrary" entered), "Args", "Alias", and "Comment". A "Browse" button is next to the "Name" field. Below the fields is a text area with instructions: "Give name, optional arguments and optional alias of the library to import. Separate multiple arguments with a pipe character like 'arg 1 | arg 2'. Alias can be used to import same library multiple times with different names." At the bottom are "OK" and "Cancel" buttons.

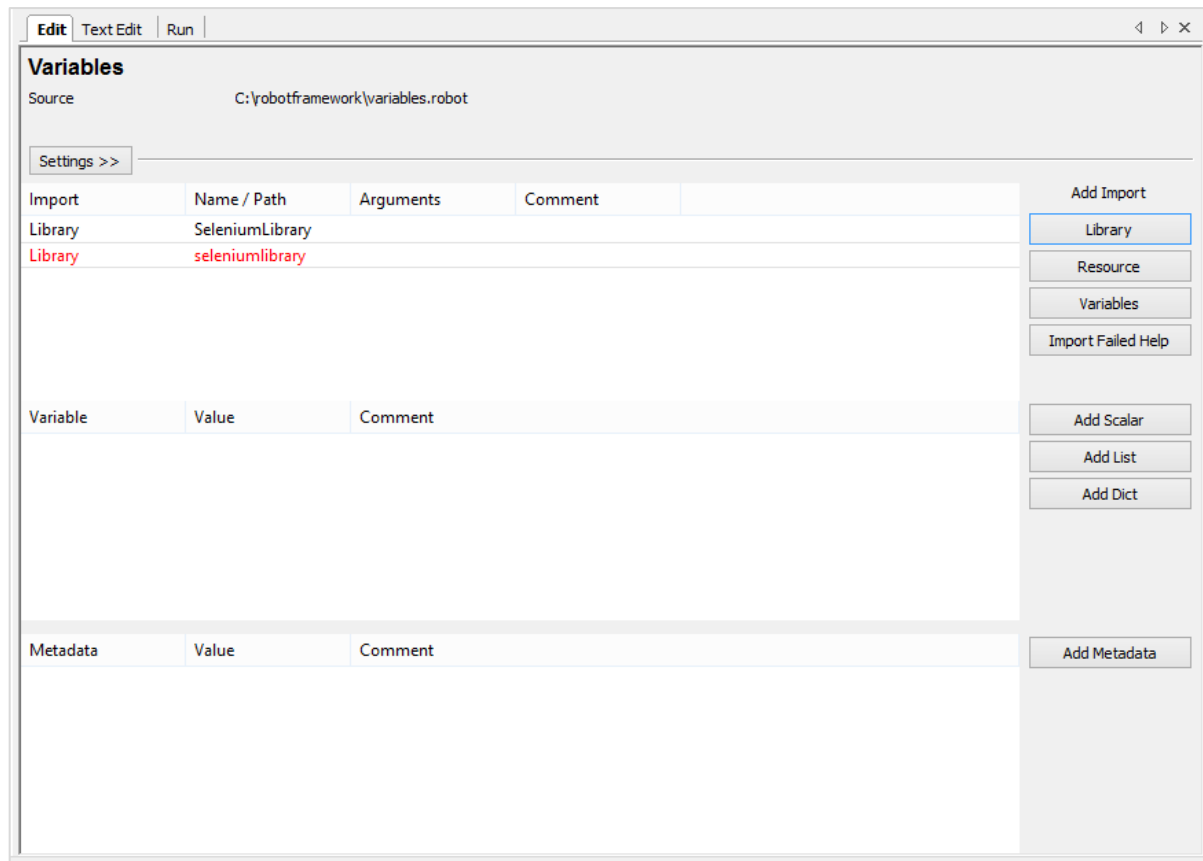
Click OK and the library will get displayed in the settings.



The Robot Framework Variables editor window. It has tabs for "Edit", "Text Edit", and "Run". The "Variables" section shows the source file "C:\robotframework\variables.robot". Below is a "Settings >>" button and a table with columns "Import", "Name / Path", "Arguments", and "Comment". The table contains one entry: "Library" with "SeleniumLibrary" in the "Name / Path" column. To the right of the table are buttons for "Add Import" (Library, Resource, Variables, Import Failed Help), "Add Scalar", "Add List", "Add Dict", and "Add Metadata".

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

If the name does not match, the library name will be shown in red:

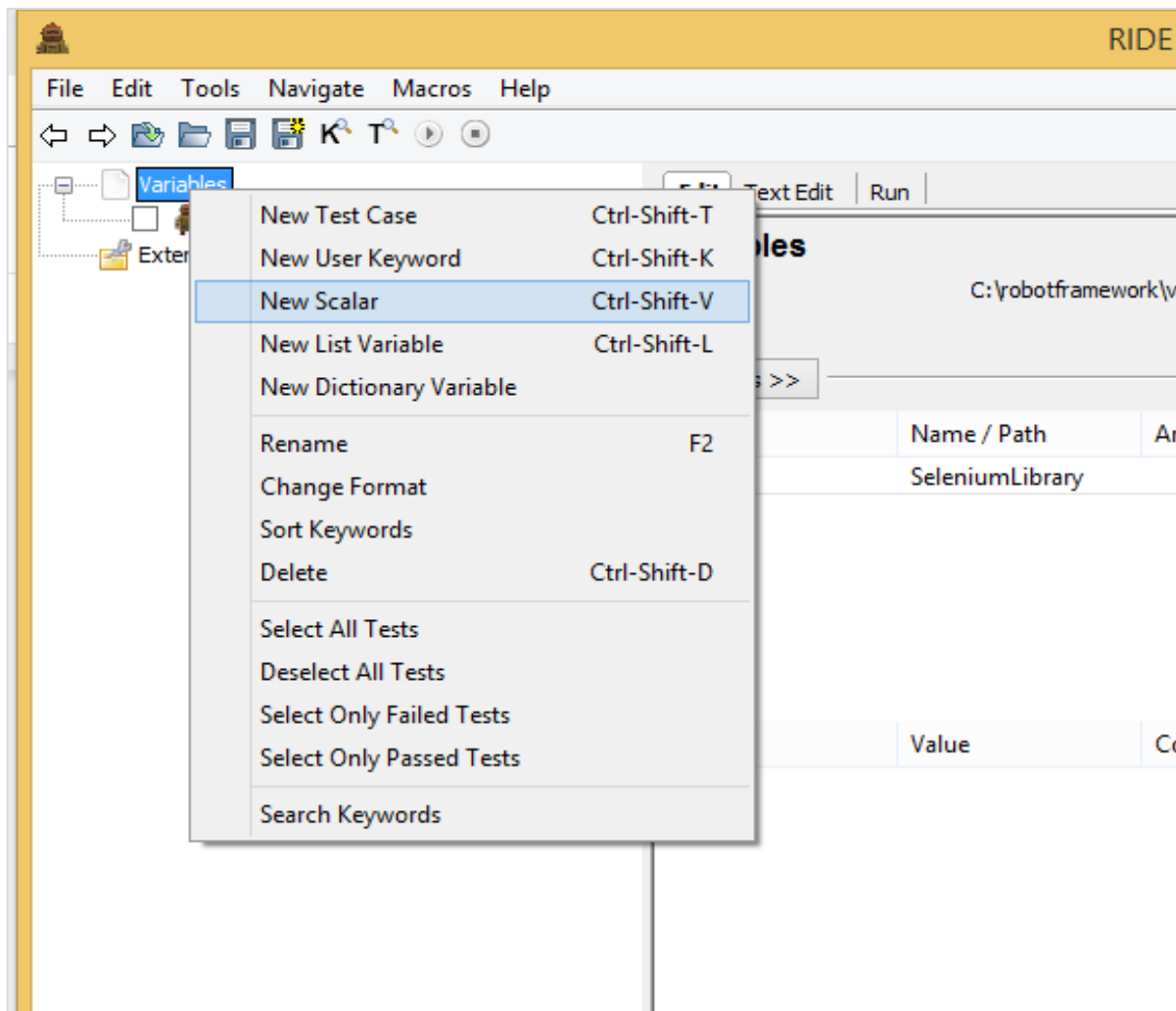


Test Case for Scalar Variable

In the above test cases we hardcoded the values like the URL, email, password, which we are giving to the test case. The values used can be stored in a variable and instead of hardcoding, we can use the variable in those places.

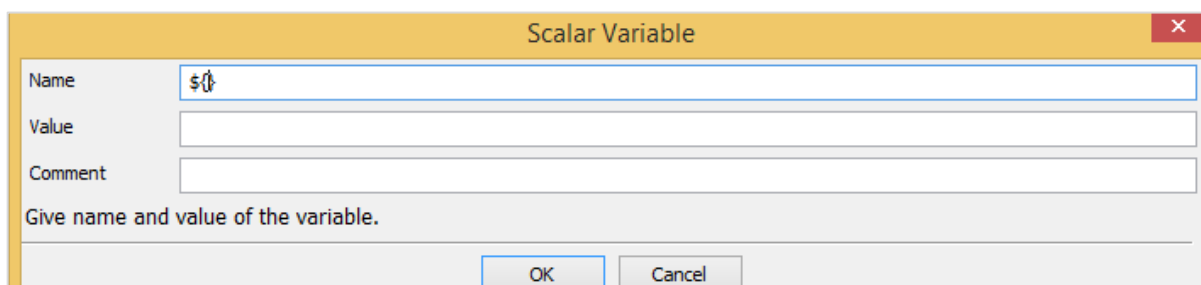
1	Open Browser	http://localhost/robotframework/lo	chrome		
2	Input Text	id:email	admin@gmail.com		
3	Input Password	id:passwd	admin		
4	Click Button	id:btnsubmit			
5					

To create scalar variable, right-click on your project and click on *New Scalar* as shown below:



Clicking on *New Scalar* will open the following screen to create the variable and the value we need to replace with when the variable is used inside test cases.

We get `${}` for the *Name* field.



Here we need to enter the name of the variable inside the curly braces as shown in the screen below:

Scalar Variable

Name:

Value:

Comment:

Give name and value of the variable.

The name of the variable is `${url}`. The value is – <http://localhost/robotframework/login.html>.

We added the comment as shown above. Click OK to save the scalar variable. The details of the variable are added as shown below:

Variables

Source: C:\robotframework\variables.robot

Settings >>

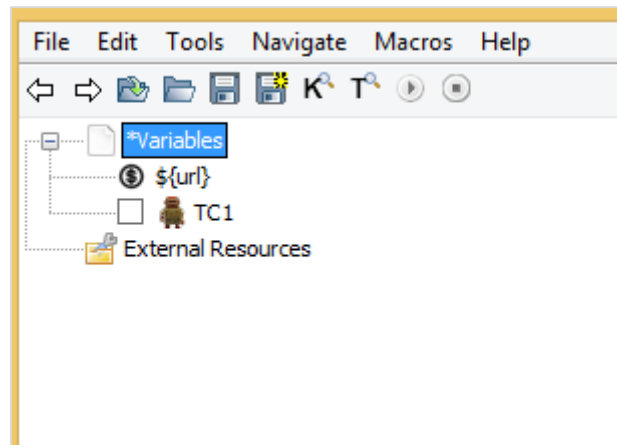
Import	Name / Path	Arguments	Comment
Library	SeleniumLibrary		

Variable	Value	Comment
<code>\${url}</code>	http://localhost/robotframework/login.html	# this variable is used to store the value of the url.

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

Buttons on the right: Add Import, Library, Resource, Variables, Import Failed Help, Add Scalar, Add List, Add Dict, Add Metadata.

The variable name is shown under the project created as follows:



Let us now use the scalar variable created inside our test case.

Test case with URL hardcoded

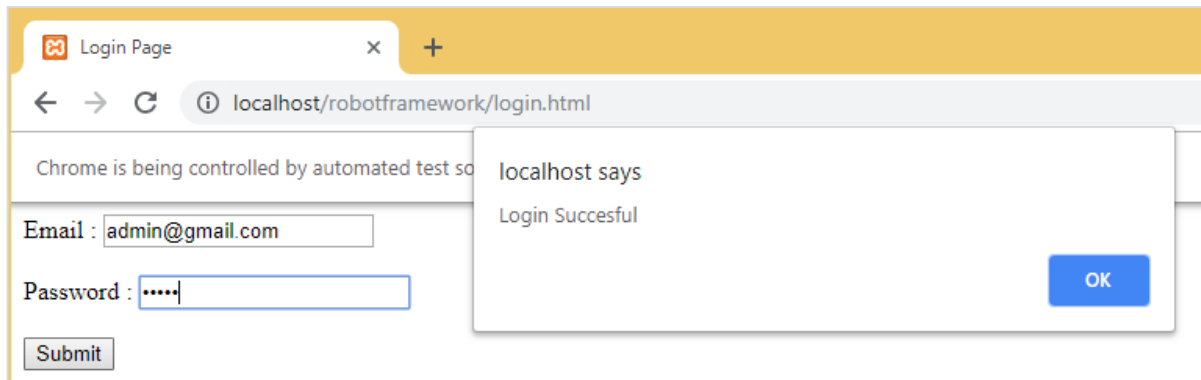
1	Open Browser	http://localhost/robotframework/lo	chrome		
2	Input Text	id:email	admin@gmail.com		
3	Input Password	id:passwd	admin		
4	Click Button	id:btnsubmit			
5					

In the above test case, we have to replace the URL with the variable we just created above.

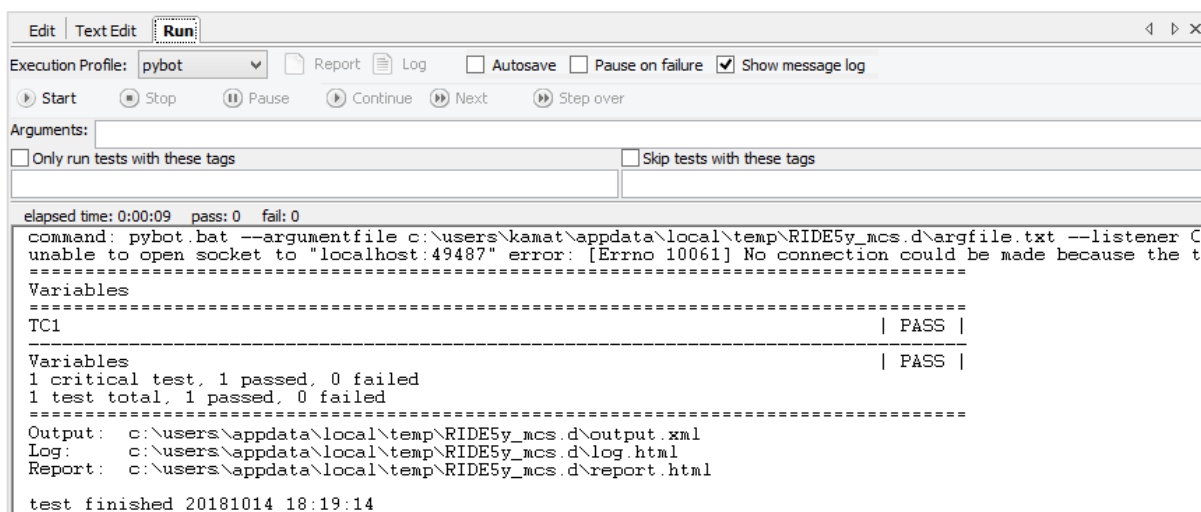
Test Case with Scalar Variable for URL

1	Open Browser	<code>\$url</code>	chrome		
2	Input Text	id:email	admin@gmail.com		
3	Input Password	id:passwd	admin		
4	Click Button	id:btnsubmit			
5					

Now, we will run the test case to see if it is taking the URL from the variable. Below is the output that we get when we run it. The URL <http://localhost/robotframework/login.html> is picked up from the scalar variable we created.



Execution Details



The advantage of using variables is that you can change the value for that variable and it will be reflected in all test cases. You can use the variables in many test cases that you create under that project. Hardcoding of values can be a serious problem when you want to change something, you will have to go to individual test case and change the values for it. Having variables in one place gives us the flexibility to test the way we want with different values to the variables.

Now, we will look into the next type of variable called the List variable.

List Variable

List variable will have an array of values. To get the value, the list item is passed as the argument to the list variable.

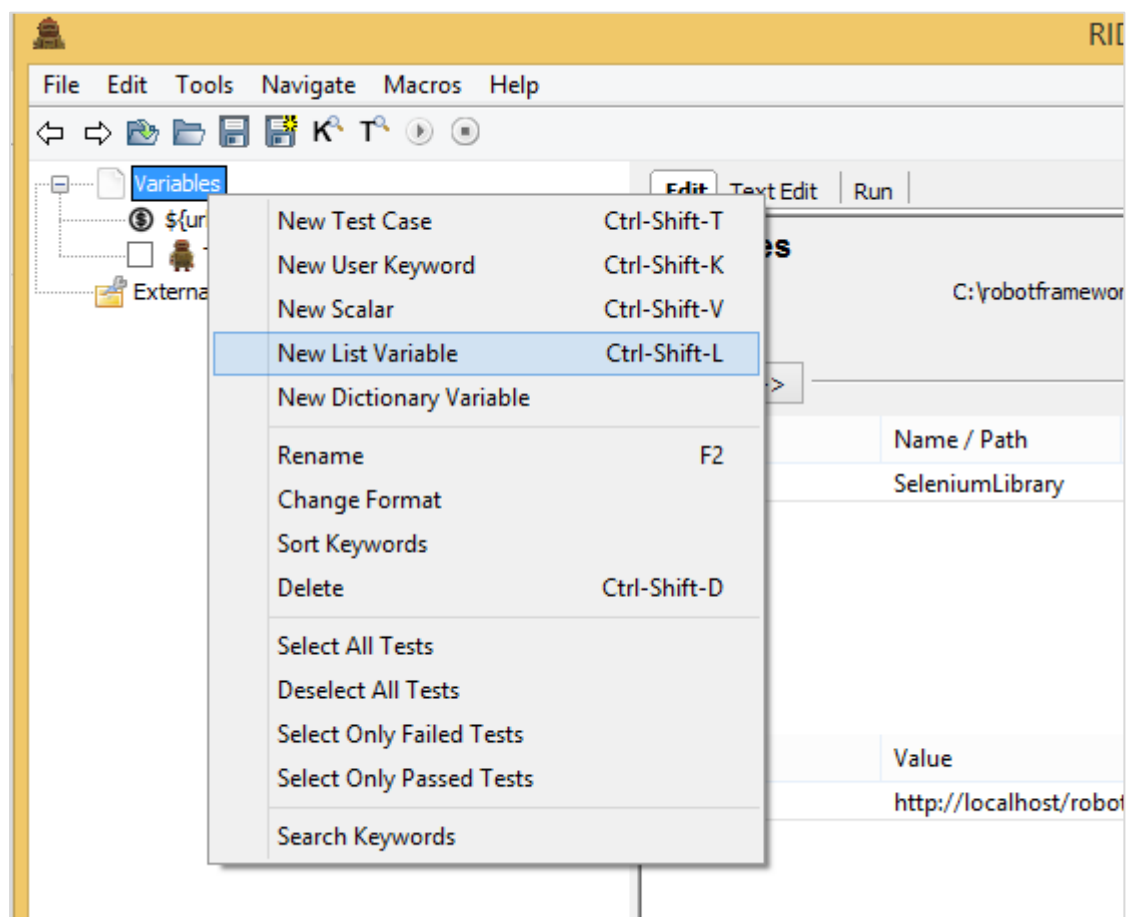
Syntax

```
@{variablename}
```

Suppose we have values A, B. To refer the values, we need to pass the list item as follows:

```
@{variablename}[0] // A
@{variablename}[1] // B
```

To add list variable, right-click on the project and click **New List Variable**.



Upon clicking *New List Variable*, a screen appears where we can enter the values:

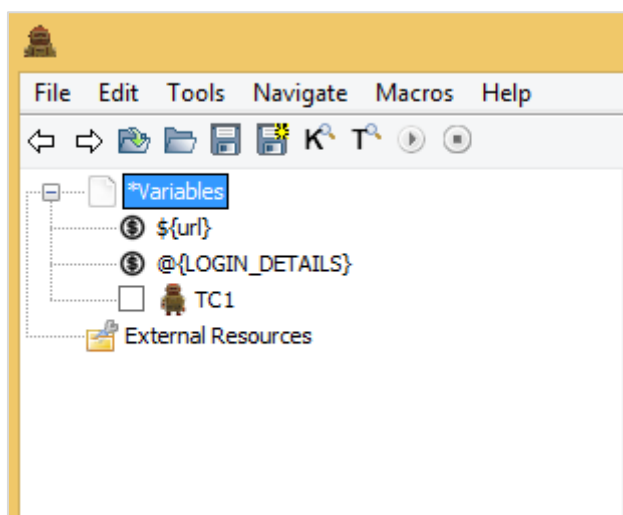
The screenshot shows the 'List Variable' dialog box. The 'Name' field is set to '@{}'. The 'Value' field is empty. The 'Columns' dropdown is set to '4'. The 'Comment' field is empty. The dialog has 'OK' and 'Cancel' buttons at the bottom.

The Name is given as **@{}** followed by Value. It also has 4 Columns selected. Right now, we will use just Column 1 and create the list variable, which will have values, email id and password as follows:

The screenshot shows the 'List Variable' dialog box. The 'Name' field is set to '@{LOGIN_DETAILS}'. The 'Value' field contains 'admin@gmail.com' and 'admin'. The 'Columns' dropdown is set to '1'. The 'Comment' field contains 'emailid and password'. The dialog has 'OK' and 'Cancel' buttons at the bottom.

The name of the list variable is **@{LOGIN_DETAILS}** and the values given are **admin@gmail.com** and **admin**, which has email id and password for the login page.

Click OK to save the list variable. The variable is listed below the project as shown here:



The details of variables used are listed in the settings tab:

Variables

Source: C:\robotframework\variables.robot

Settings >>

Import	Name / Path	Arguments	Comment
Library	SeleniumLibrary		

Variable	Value	Comment
<code>\$_url</code>	http://localhost/robotframework/login.html	# this variable is used to store the value of the url.
<code>@{LOGIN_DETAILS}</code>	admin@gmail.com admin	# emailid and password

Now, we will add the list variable inside the test cases as shown below.

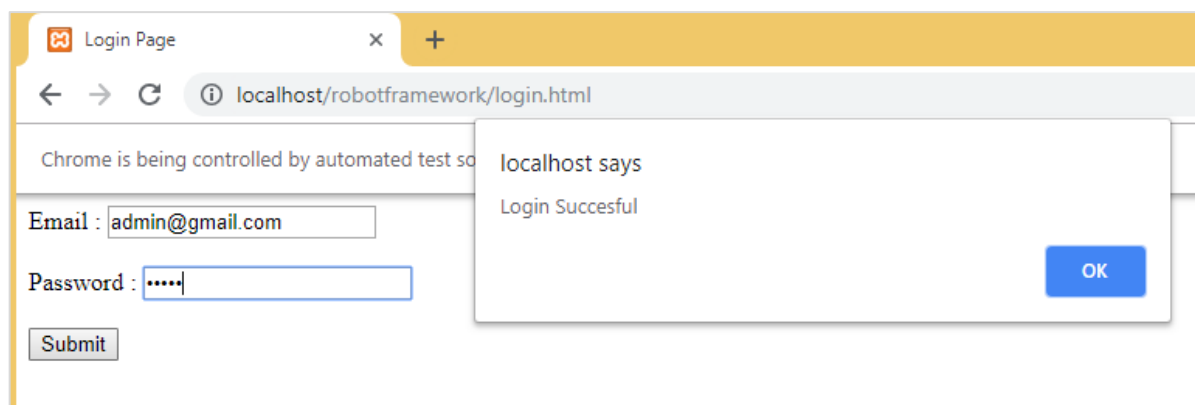
Here, we have hardcoded values for the Input Text and Password. Now, we will change it to use the list variable.

1	Open Browser	\$_url	chrome		
2	Input Text	id:email	admin@gmail.com		
3	Input Password	id:passwd	admin		
4	Click Button	id:btnsubmit			
5					

Using List Variable

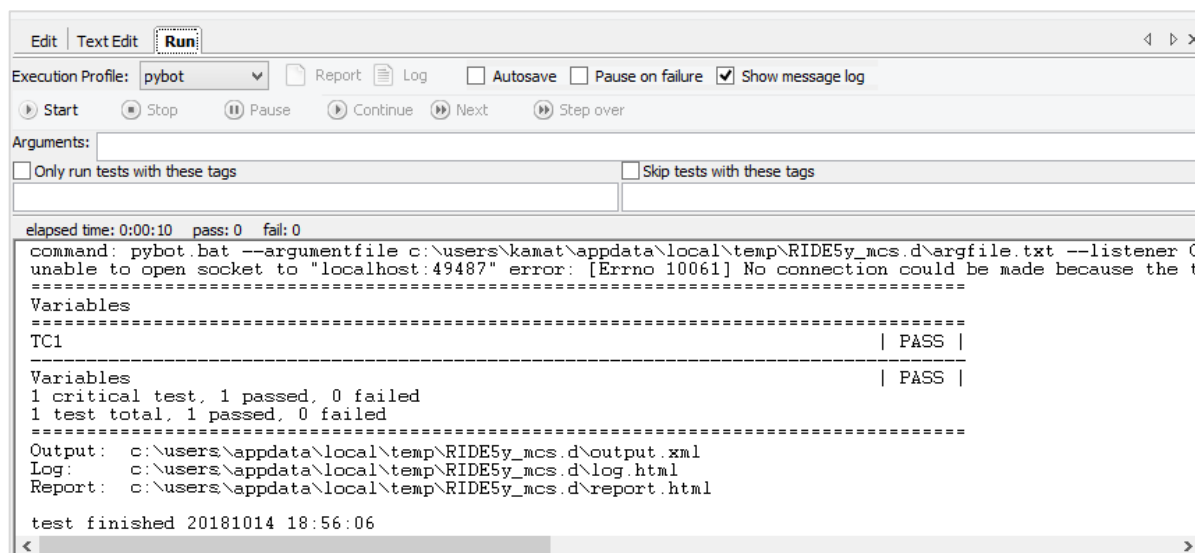
1	Open Browser	\$_url	chrome		
2	Input Text	id:email	@{LOGIN_DETAILS}[0]		
3	Input Password	id:passwd	@{LOGIN_DETAILS}[1]		
4	Click Button	id:btnsubmit			
5					

Now, we will execute the test case to see if it is taking the values from the list variable:



It has taken the email id and password from the list variable as shown above in the test screen.

The following screenshot shows the execution details for the same:



In our next section, we will learn about the Dictionary Variable.

Dictionary Variable

Dictionary Variable is similar to list variable wherein we pass the index as an argument; however, in case of dictionary variable, we can store the details – key value form. It becomes easier to refer when used in the test case instead of using the index as 0, 1, etc.

Syntax

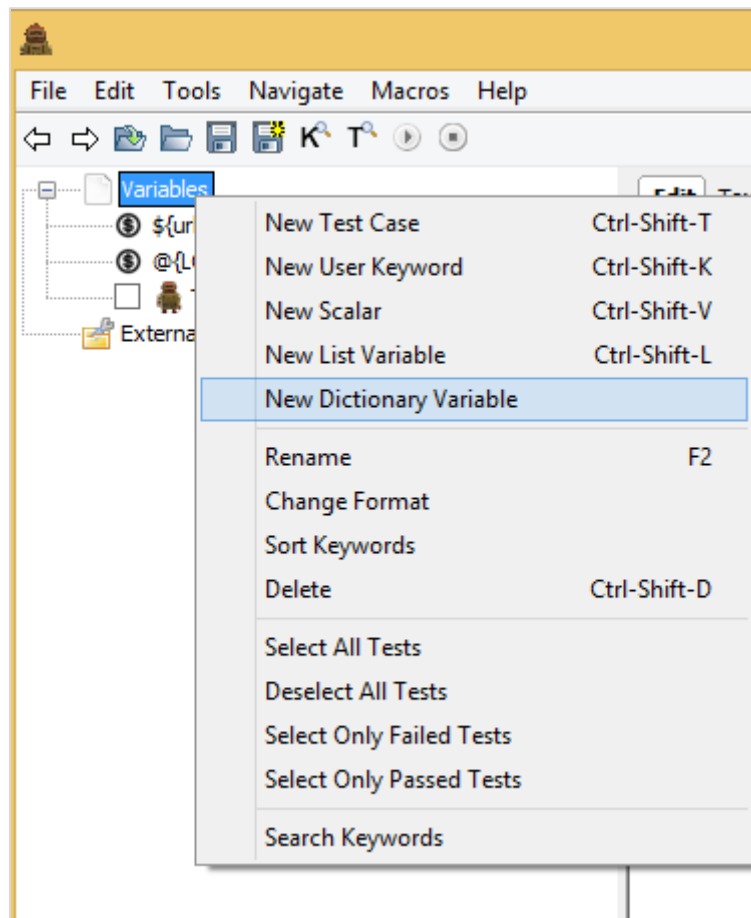
```
&{Variablename}
```

Suppose we are storing the values as key1=A, key2=B. It will be referred in the test case as :

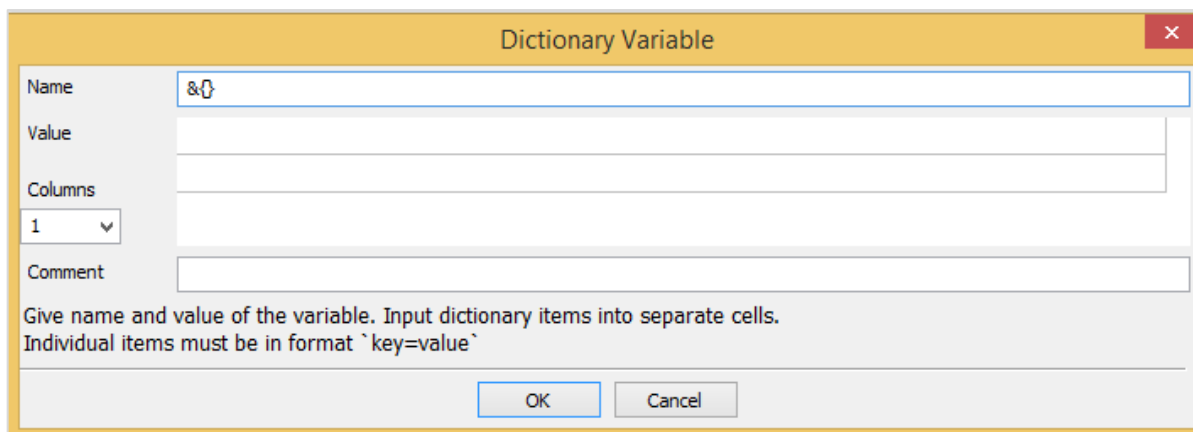
```
&{Variablename}[key1] // A  
&{Variablename}[key2] // B
```

Let us create dictionary variable in Ride.

Right-click on Project and click on *New Dictionary Variable*.



Upon clicking **New Dictionary Variable**, a screen will appear as shown below:



Dictionary Variable

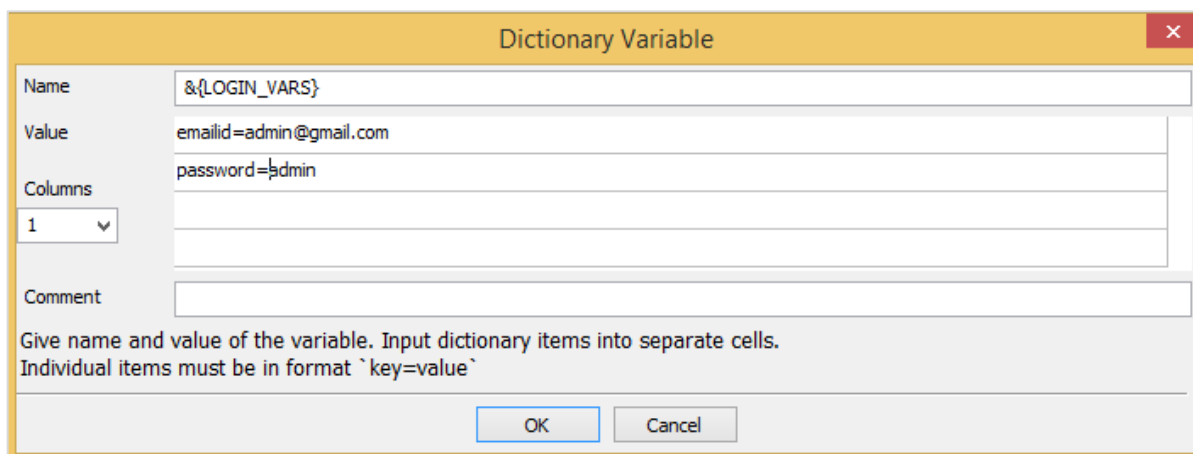
Name: &{}
Value:
Columns: 1
Comment:

Give name and value of the variable. Input dictionary items into separate cells.
Individual items must be in format `key=value`

OK Cancel

The Name by default in the screen is **&{ }** and it has Value and Columns option.

We will enter the Name and the Values to be used in the test case.



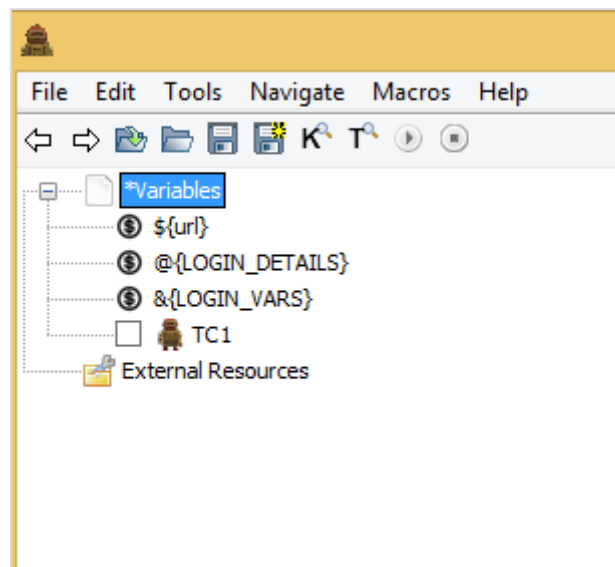
Dictionary Variable

Name: &{LOGIN_VARS}
Value: emailid=admin@gmail.com
password=admin
Columns: 1
Comment:

Give name and value of the variable. Input dictionary items into separate cells.
Individual items must be in format `key=value`

OK Cancel

Click OK to save the variable. The variable will be listed under the project and also in the settings as follows:



Variables
Source: C:\robotframework\variables.robot

Settings >>

Import	Name / Path	Arguments	Comment
Library	SeleniumLibrary		

Variable	Value	Comment
\${url}	http://localhost/robotframework/login.html	# this variable is used to store the value of the url.
@LOGIN_DETAILS	admin@gmail.com admin	# emailid and password
&LOGIN_VARS	emailid=admin@gmail.com password=admin	

We will change the test case to take the dictionary values.

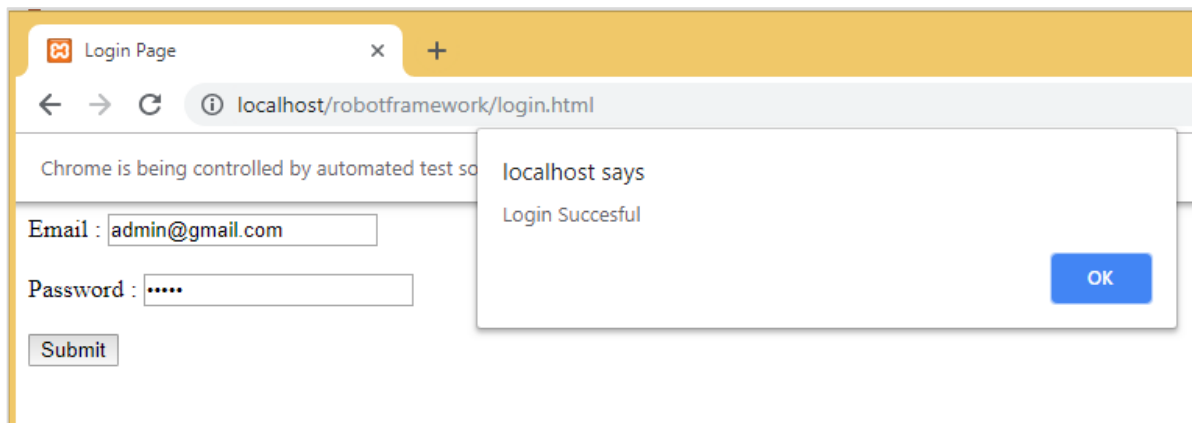
1	Open Browser	\${url}	chrome		
2	Input Text	id:email	@LOGIN_DETAILS[0]		
3	Input Password	id:passwd	@LOGIN_DETAILS[1]		
4	Click Button	id:btnsubmit			
5					

We will change to dictionary variable as shown below.

Using Dictionary Variable

1	Open Browser	\${url}	chrome		
2	Input Text	id:email	&{LOGIN_VARS}[emailid]		
3	Input Password	id:passwd	&{LOGIN_VARS}[password]		
4	Click Button	id:btnsubmit			
5					

Upon clicking run, we get the following:



The execution details are as follows:

```

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: 0:00:09 pass: 0 fail: 0
command: pybot.bat --argumentfile c:\users\kamat\appdata\local\temp\RIDE5y_mcs.d\argfile.txt --listener C
unable to open socket to "localhost:49487" error: [Errno 10061] No connection could be made because the t
=====
Variables
=====
TC1 | PASS |
=====
Variables
1 critical test, 1 passed, 0 failed | PASS |
1 test total, 1 passed, 0 failed
=====
Output: c:\users\appdata\local\temp\RIDE5y_mcs.d\output.xml
Log: c:\users\appdata\local\temp\RIDE5y_mcs.d\log.html
Report: c:\users\appdata\local\temp\RIDE5y_mcs.d\report.html
test finished 20181014 19:16:44

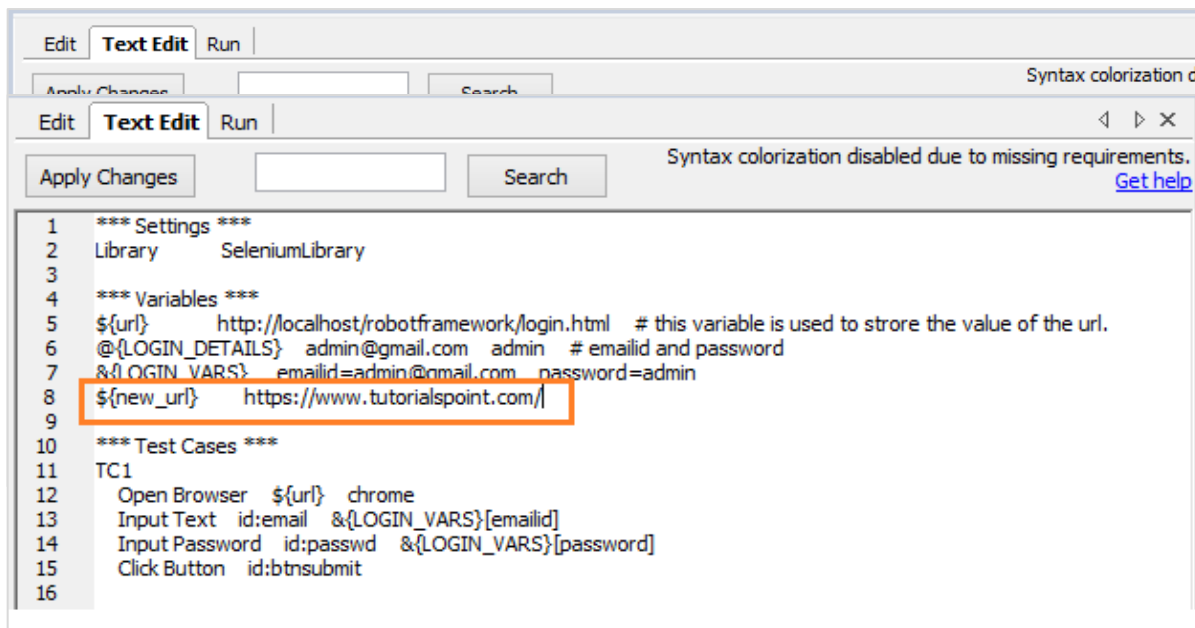
```

We have seen the Edit and Run Tab so far. In case of TextEdit, we have the details of the test case written. We can also add variables required in TextEdit.

Test case

1	Open Browser	\${url}	chrome		
2	Input Text	id:email	&{LOGIN_VARS}[emailid]		
3	Input Password	id:passwd	&{LOGIN_VARS}[password]		
4	Click Button	id:btnsubmit			

We have used scalar variable and dictionary variable in the above test case. Here is the code so far in TextEdit; this is based on the test case written:



```

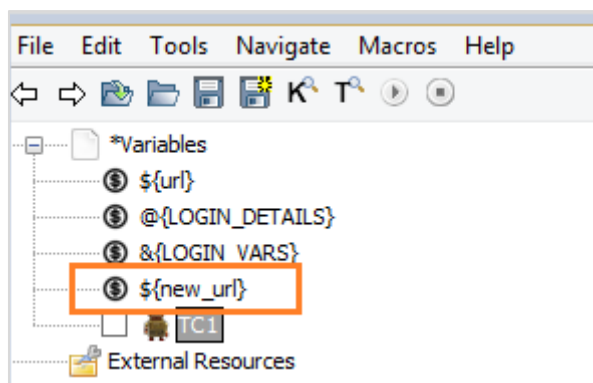
1  *** Settings ***
2  Library           SeleniumLibrary
3
4  *** Variables ***
5  ${url}            http://localhost/robotframework/login.html  # this variable is used to store the value of the url.
6  @<{LOGIN_DETAILS}  admin@gmail.com  admin  # emailid and password
7  &{LOGIN_VARS}      emailid=admin@gmail.com  password=admin
8  ${new_url}         https://www.tutorialspoint.com/
9
10 *** Test Cases ***
11 TC1
12   Open Browser  ${url}  chrome
13   Input Text    id:email  &{LOGIN_VARS}[emailid]
14   Input Password id:passwd &{LOGIN_VARS}[password]
15   Click Button  id:btnsubmit
16

```

The variables used are highlighted in Red. We can also create variables we want directly in TextEdit as shown below:

We have added a scalar variable called **`${new_url}`** and the value given is <https://www.tutorialspoint.com/>.

Click **Apply Changes** button on the top left corner and the variable will be seen under the project as shown below:



Similarly, other variables – list and dictionary variables can be created directly inside TextEdit tab whenever required.

Conclusion

We have seen how to create and use variables. There are three types of variables supported in robot framework – scalar, list and dictionary. We discussed in detail the working of all these variables.

15. Robot Framework – Working With Command Line

In this chapter, we will learn how to make use of the command line to run test cases.

To begin with, let us open the command prompt and go to the folder where your test cases are saved. We have created test cases and saved in the folder **robotframework** in C Drive.



```
Command Prompt

C:\robotframework>dir
Volume in drive C has no label.
Volume Serial Number is 16D7-97E3

Directory of C:\robotframework

18-10-2018  15:31    <DIR>          .
18-10-2018  15:31    <DIR>          ..
08-10-2018  15:36             274 BrowserTestCases.robot
14-10-2018  10:08             194 checkbox.robot
14-10-2018  15:15             308 dropdown.robot
08-10-2018  11:16             183 FirstTestCase.robot
14-10-2018  10:00    <DIR>          html
18-10-2018  15:31             377 LibraryKeywords.robot
14-10-2018  10:37             205 RadioButton.robot
07-10-2018  09:55              25 testing.robot
08-10-2018  18:39             235 Textbox.robot
14-10-2018  19:28             592 variables.robot
          9 File(s)                2,393 bytes
          3 Dir(s)  108,533,227,520 bytes free

C:\robotframework>
```

Test cases created so far are available in the folder **C:\robotframework**.

If you have saved your project as a file, the command is:

```
robot -T nameoftestcase.robot
```

If you have saved your project as a directory, the command is:

```
robot -T projectname testsuite
```

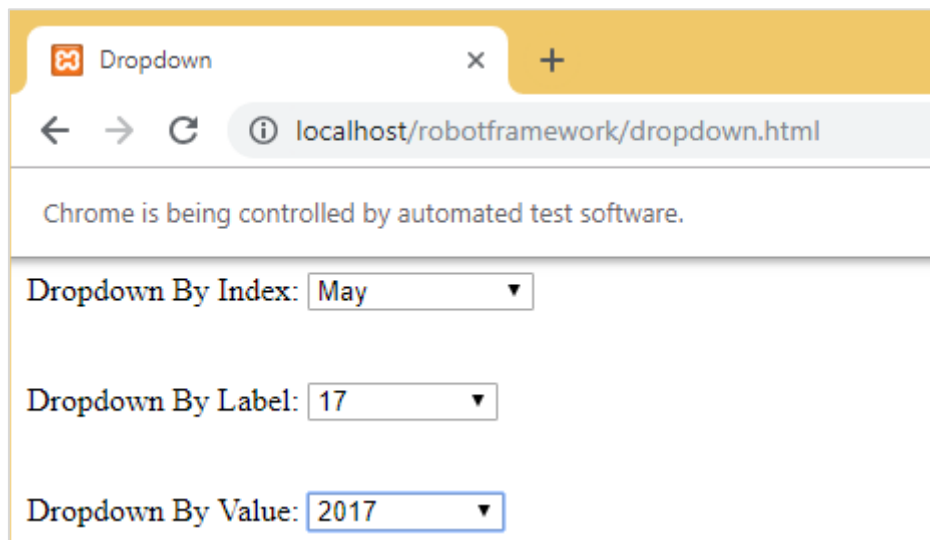
We will run one of the test created from the folder as shown below:

```
C:\robotframework>robot -T dropdown.robot
=====
Dropdown
=====
TC1
1018/155453.247:ERROR:install_util.cc(629)l Failed to read HKLM\SOFTWARE\Policies\Google\Chrome\MachineLevelUserCloudPolicyEnrollmentToken: The system cannot find the file specified. (0x2)

DevTools listening on ws://127.0.0.1:51575/devtools/browser/039467e9-e629-4ed5-8993-cc4e3cf76a70
TC1
=====
Dropdown
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
=====
Output: C:\robotframework\output-20181018-155451.xml
Log: C:\robotframework\log-20181018-155458.html
Report: C:\robotframework\report-20181018-155458.html
```

The output, log and report paths are displayed at the end as shown above.

The following screenshot shows the execution details:



Report

Dropdown Test Report

Generated
20181018 15:56:36 GMT+05:30
1 minute 15 seconds ago

Summary Information

Status: All tests passed
Start Time: 20181018 15:56:29.677
End Time: 20181018 15:56:36.505
Elapsed Time: 00:00:06.828
Log File: [log-20181018-155636.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
Dropdown	1	1	0	00:00:07	

Test Details

[Totals](#)
[Tags](#)
[Suites](#)
[Search](#)

Type:
☐ Critical Tests
☒ All Tests

Log

Dropdown Test Log

Generated
20181018 15:56:36 GMT+05:30
1 minute 42 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
Dropdown	1	1	0	00:00:07	

Test Execution Log

SUITE

Dropdown

Full Name: Dropdown
Source: [C:\robotframework\dropdown.robot](#)
Start / End / Elapsed: 20181018 15:56:29.677 / 20181018 15:56:36.505 / 00:00:06.828
Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

TEST

TC1

Full Name: Dropdown.TC1
Start / End / Elapsed: 20181018 15:56:30.507 / 20181018 15:56:36.495 / 00:00:05.988
Status: PASS (critical)

+

KEYWORD

SeleniumLibrary.Open Browser

[http://localhost/robotframework/dropdown.html](#), chrome

+

KEYWORD

SeleniumLibrary.Select From List By Index

name:months, 5

+

KEYWORD

SeleniumLibrary.Select From List By Label

name:days, 17

+

KEYWORD

SeleniumLibrary.Select From List By Value

name:year, 17

Conclusion

We can use command line to execute robot test cases. The details of the test case pass or fail are displayed in the command line along with log and report URLs.

16. Robot Framework — Working With Setup And Teardown

In this chapter, we will understand two important concepts of testing world — setup and teardown.

Setup

This is a set of keywords or instruction to be executed before the start of test suite or test case execution.

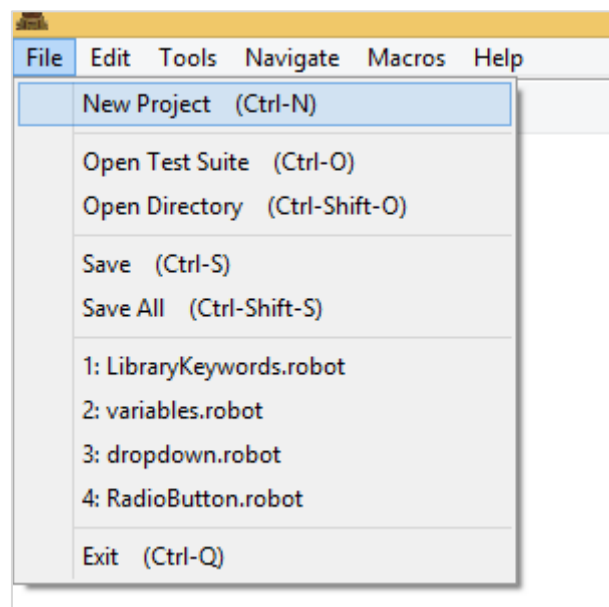
Teardown

This is a set of keywords or instruction to be executed after the start of test suite or test case execution.

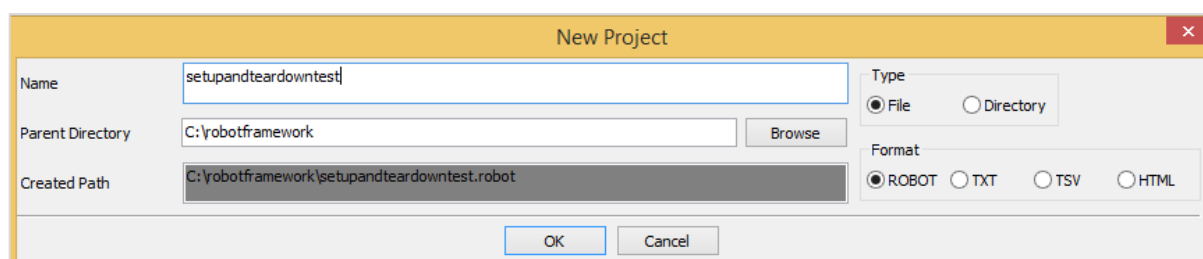
We will work on a project setup, where will use both setup and teardown. The opening and closing of browser are the common steps in test cases.

Now, we will add keyword **open browser** in the setup and close browser in teardown.

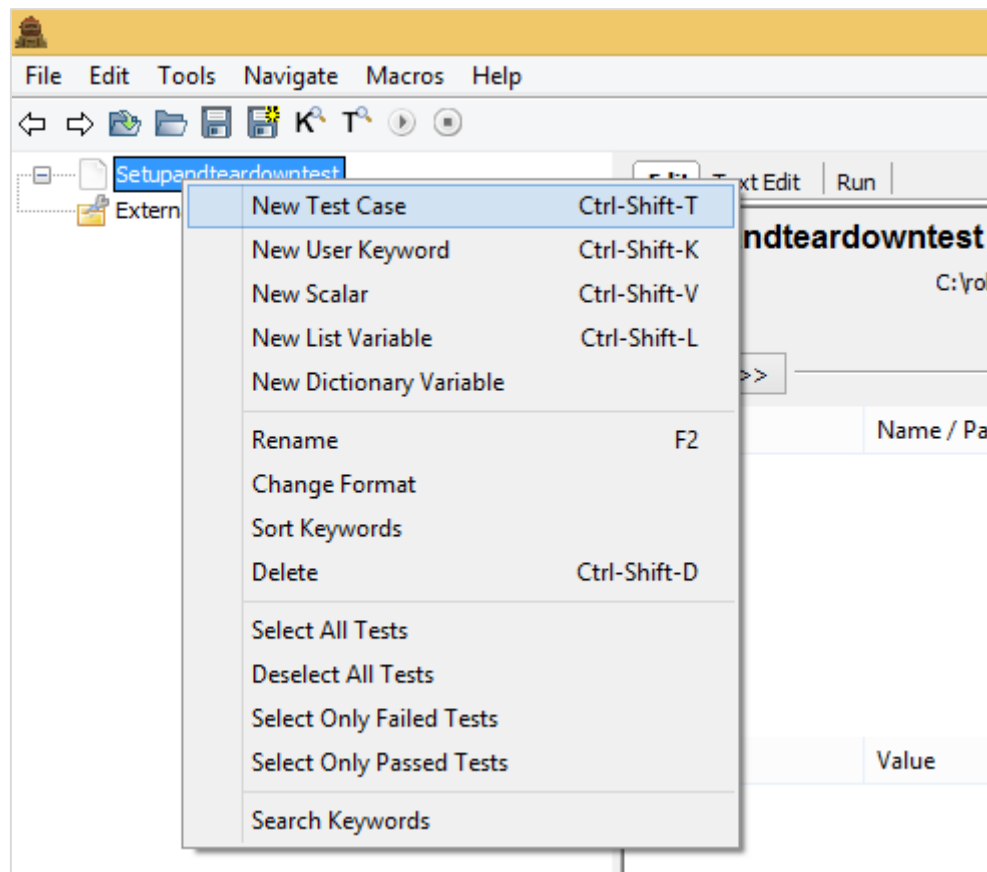
Open Ride using **ride.py** command from command line and create a new project.



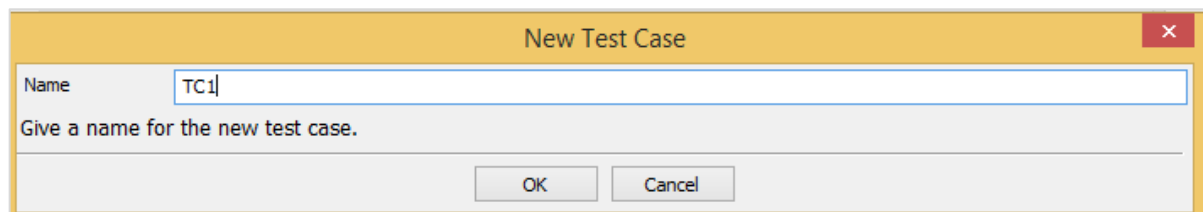
Click *New Project* to create project.



Click OK to save the project.



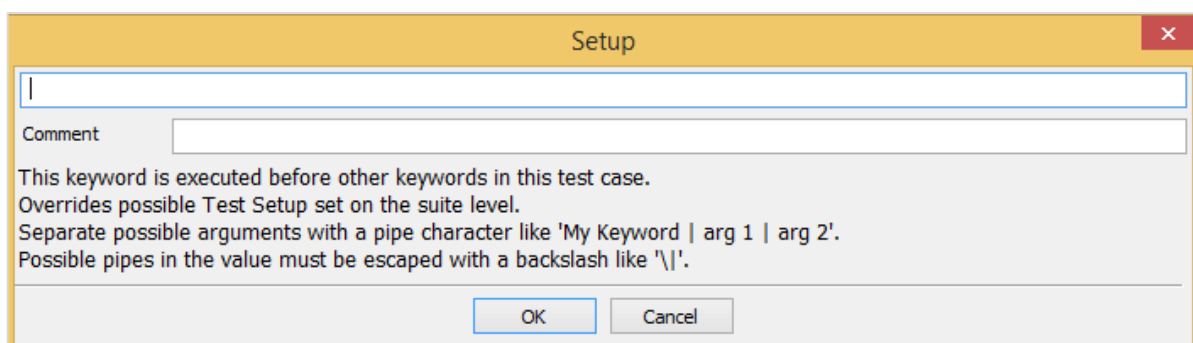
Click **New Test Case** to create one.



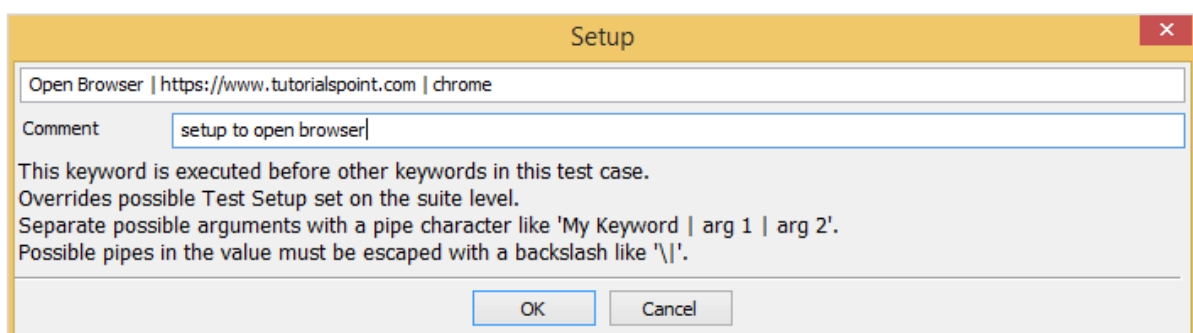
Click OK to save the test case.

Now we need to import the SeleniumLibrary to use the keywords related to browser and interact with the pages.

Now, enter the Keyword:



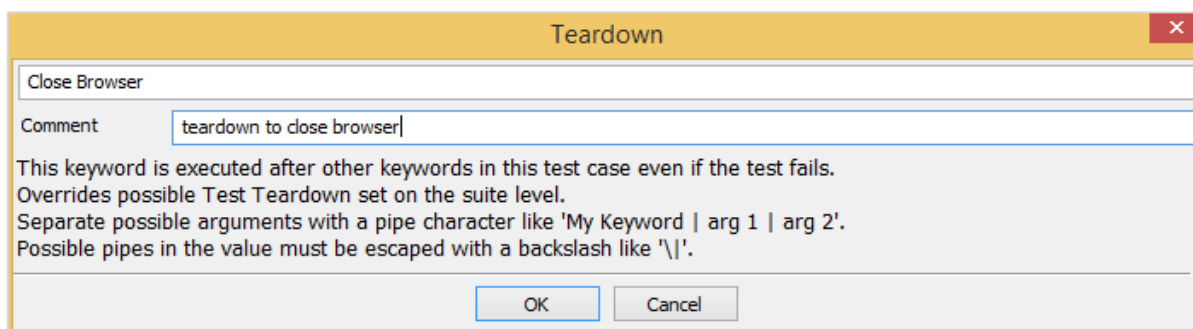
Arguments have to be separated with the pipe character (|).



Click OK to save the Setup. We have entered the Keyword **Open browser** and also added the arguments as shown above.

Now, we will enter the teardown case.

Click Edit for Teardown and enter the keyword.



Click OK to save the teardown.

Now, we will enter the keywords for test case.

TC1

Settings <<

Documentation

Setup: Open Browser | https://www.tutorialspoint.com | chrome | # setup to open browser

Teardown: Close Browser | # teardown to close browser

Tags: <Add New>

Timeout:

Template:

1	Input Text	name=search	This is coming from setup/teardown testcase	
2				
3				
4				
5				
6				

We have only Input Text in the test case. The opening and closing of the browser is done from Setup and Teardown Settings.

Test Execution Details

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: 0:00:29 pass: 1 fail: 0

command: pybot.bat --argumentfile c:\users\kamat\appdata\local\temp\RIDEid4bhi.d\argfile.txt --listener C:\Python27\lib\site-packa

=====
 Setupandteardowntest
 TC1 | PASS |
 Setupandteardowntest | PASS |
 1 critical test, 1 passed, 0 failed
 1 test total, 1 passed, 0 failed
 =====
 Output: c:\users\appdata\local\temp\RIDEid4bhi.d\output.xml
 Log: c:\users\appdata\local\temp\RIDEid4bhi.d\log.html
 Report: c:\users\appdata\local\temp\RIDEid4bhi.d\report.html
 test finished 20181028 11:54:02

Starting test: Setupandteardowntest.TC1
 20181028 11:53:35.016 : INFO : Opening browser 'chrome' to base url 'https://www.tutorialspoint.com'.
 20181028 11:53:57.071 : INFO : Typing text 'This is coming from setup/teardown testcase' into text field 'name=search'.
 Ending test: Setupandteardowntest.TC1

Conclusion

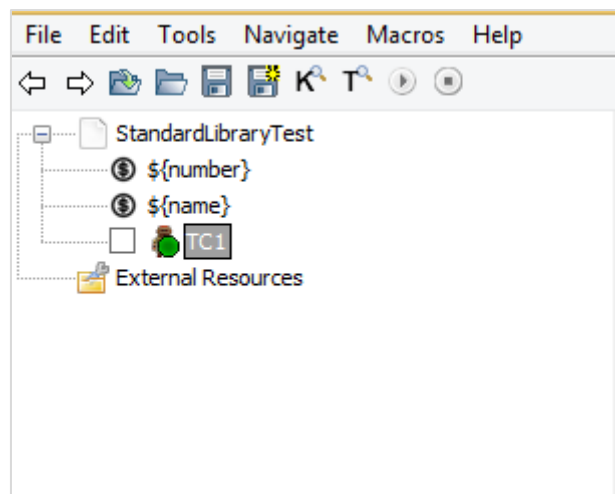
Setup and teardown play a crucial role in the testing world. We have seen how to use setup and teardown in our test cases and also how they are executed.

17. Robot Framework — Working with Built-In Library

In this chapter, we will cover some of the important built-in keywords, which come with the Robot Framework. We can use these keywords along with External libraries for writing test case. We also have the built-in library available with Robot framework by default. It is mostly used for verifications (for example – Should Be Equal, Should Contain), conversions (convert to integer, log to console, etc.).

We will work on a simple test case and will make use of built-in library in that.

We have created project in Ride and Test case as shown below:



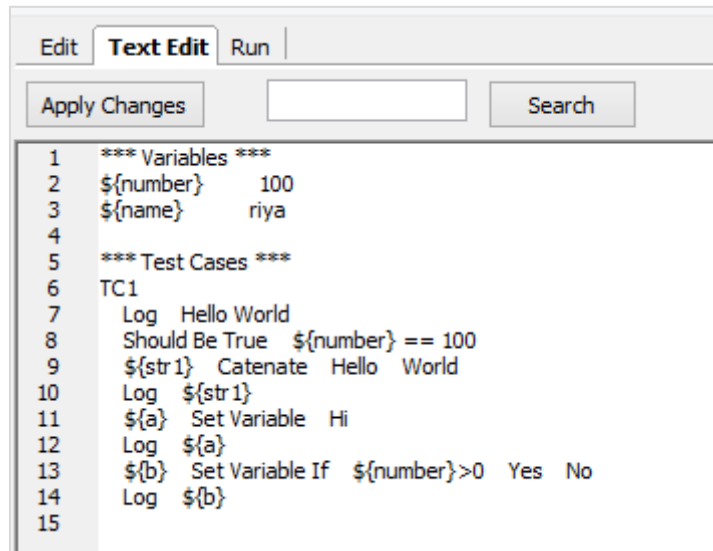
We have created 2 scalar variables – number and name as shown below:

Variable	Value	Comment
<code>\${number}</code>	100	
<code>\${name}</code>	riya	

Here are the test cases used for comparing number, string, concatenate, etc. We have used simple keywords in the test cases below. The keywords are shown in tabular format here:

1	Log	Hello World			
2	Should Be True	<code>\${number} == 100</code>			
3	<code>\${str1}</code>	Catenate	Hello	World	
4	Log	<code>\${str1}</code>			
5	<code>\${a}</code>	Set Variable	Hi		
6	Log	<code>\${a}</code>			
7	<code>\${b}</code>	Set Variable If	<code>\${number}>0</code>	Yes	No
8	Log	<code>\${b}</code>			

Following is the test code for above test cases from text edit:

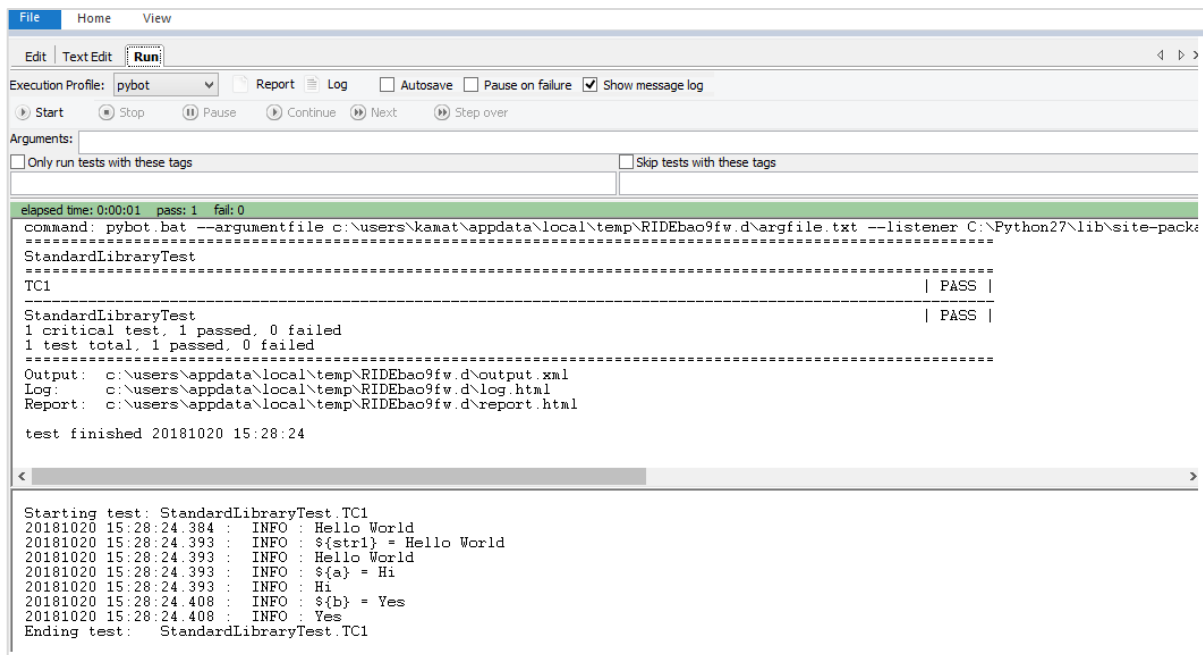


```

1  *** Variables ***
2  ${number}    100
3  ${name}      riya
4
5  *** Test Cases ***
6  TC1
7      Log      Hello World
8      Should Be True    ${number} == 100
9      ${str1} Catenate    Hello World
10     Log      ${str1}
11     ${a} Set Variable    Hi
12     Log      ${a}
13     ${b} Set Variable If    ${number}>0    Yes    No
14     Log      ${b}
15

```

Now, we will execute the test case to see the results:



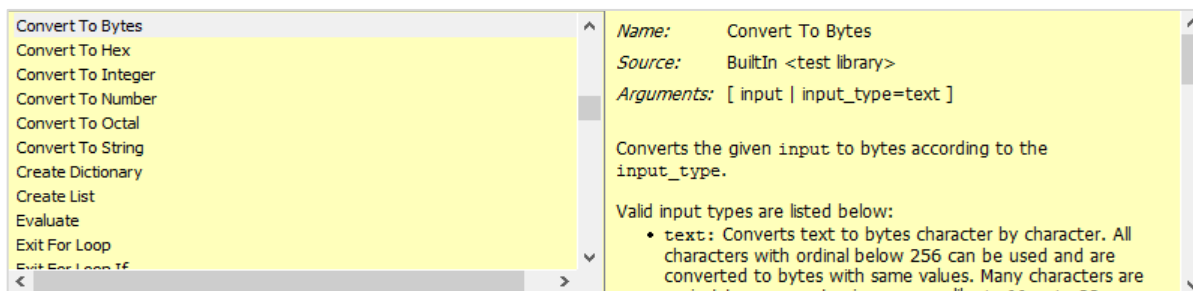
```

File Home View
Edit Text Edit Run
Apply Changes Search
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: 0:00:01 pass: 1 fail: 0
command: pybot.bat --argumentfile c:\users\kamat\appdata\local\temp\RIDEbao9fw.d\argfile.txt --listener C:\Python27\lib\site-packa
StandardLibraryTest
TC1 | PASS |
StandardLibraryTest | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
Output: c:\users\appdata\local\temp\RIDEbao9fw.d\output.xml
Log: c:\users\appdata\local\temp\RIDEbao9fw.d\log.html
Report: c:\users\appdata\local\temp\RIDEbao9fw.d\report.html
test finished 20181020 15:28:24

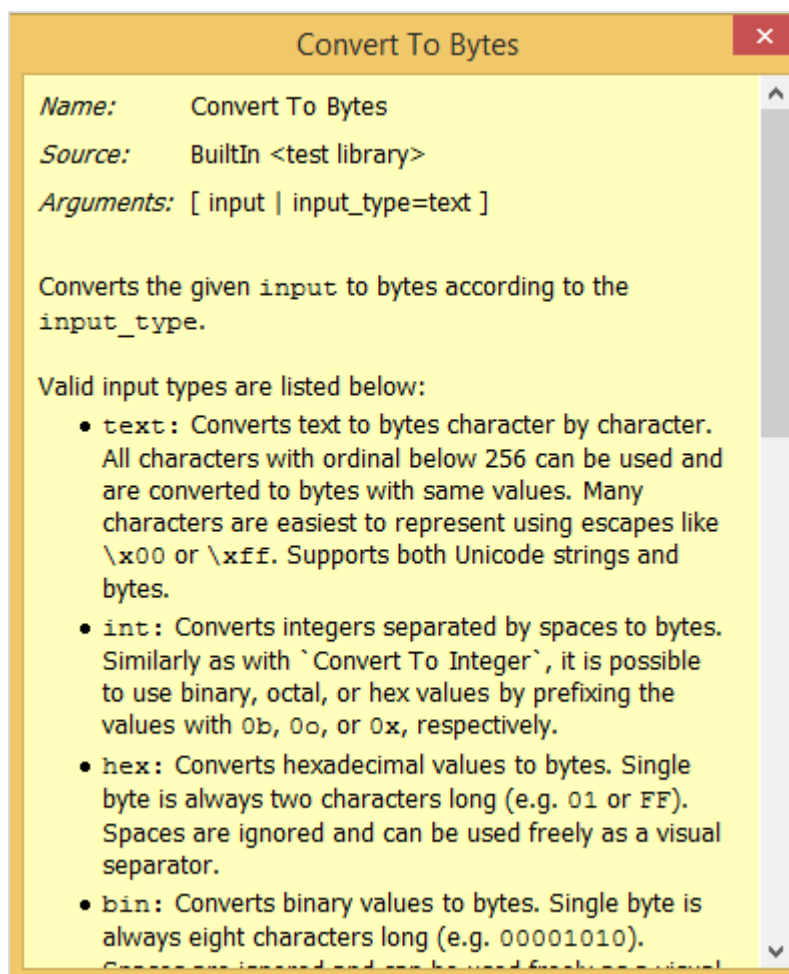
Starting test: StandardLibraryTest.TC1
20181020 15:28:24.384 : INFO : Hello World
20181020 15:28:24.393 : INFO : ${str1} = Hello World
20181020 15:28:24.393 : INFO : Hello World
20181020 15:28:24.393 : INFO : ${a} = Hi
20181020 15:28:24.393 : INFO : Hi
20181020 15:28:24.408 : INFO : ${b} = Yes
20181020 15:28:24.408 : INFO : Yes
Ending test: StandardLibraryTest.TC1

```

When you write your keywords in tabular format, press ctrl + spacebar. It gives the list of built-in keywords available with Robot Framework.



It gives the details of each keyword with example in the corresponding window. When we click on the corresponding window, it will open separately as shown below:



Conclusion

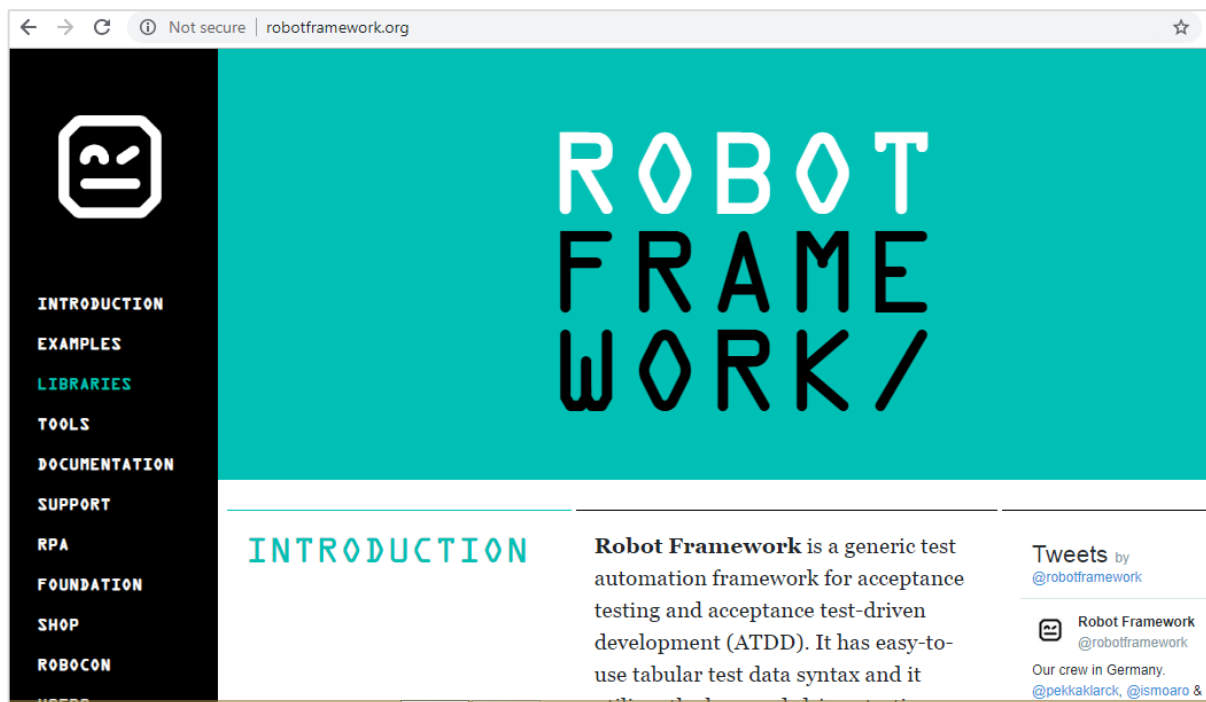
We have seen keywords related to string, numbers, log messages, which are available with robot framework by default. The same can be used along with external library and also can be used to create user-defined keyword to work with test-cases.

18. Robot Framework — Working With External Database libraries

We have seen how we can work with Selenium Library. The detailed installation/importing of Selenium Library is discussed in chapter “*Working with Browsers using Selenium Library*”.

In this chapter, we will discuss database library and how to connect and test database using Robot Framework.

Go to Robot framework site <http://robotframework.org/> and click **Libraries** as shown below:



Upon clicking Libraries, you will be redirected to a screen as shown below:

LIBRARIES

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.

STANDARD	EXTERNAL	OTHER
Builtin Provides a set of often needed generic keywords. Always automatically available without imports.	Dialogs Provides means for pausing the test execution and getting input from users.	Collections Provides a set of keywords for handling Python lists and dictionaries.
OperatingSystem Enables various operating system related tasks to be performed in the system where Robot Framework is running.	Remote Special library acting as a proxy between Robot Framework and test libraries elsewhere. Actual test libraries can be running on different machines and be implemented using	Screenshot Provides keywords to capture screenshots of the desktop.

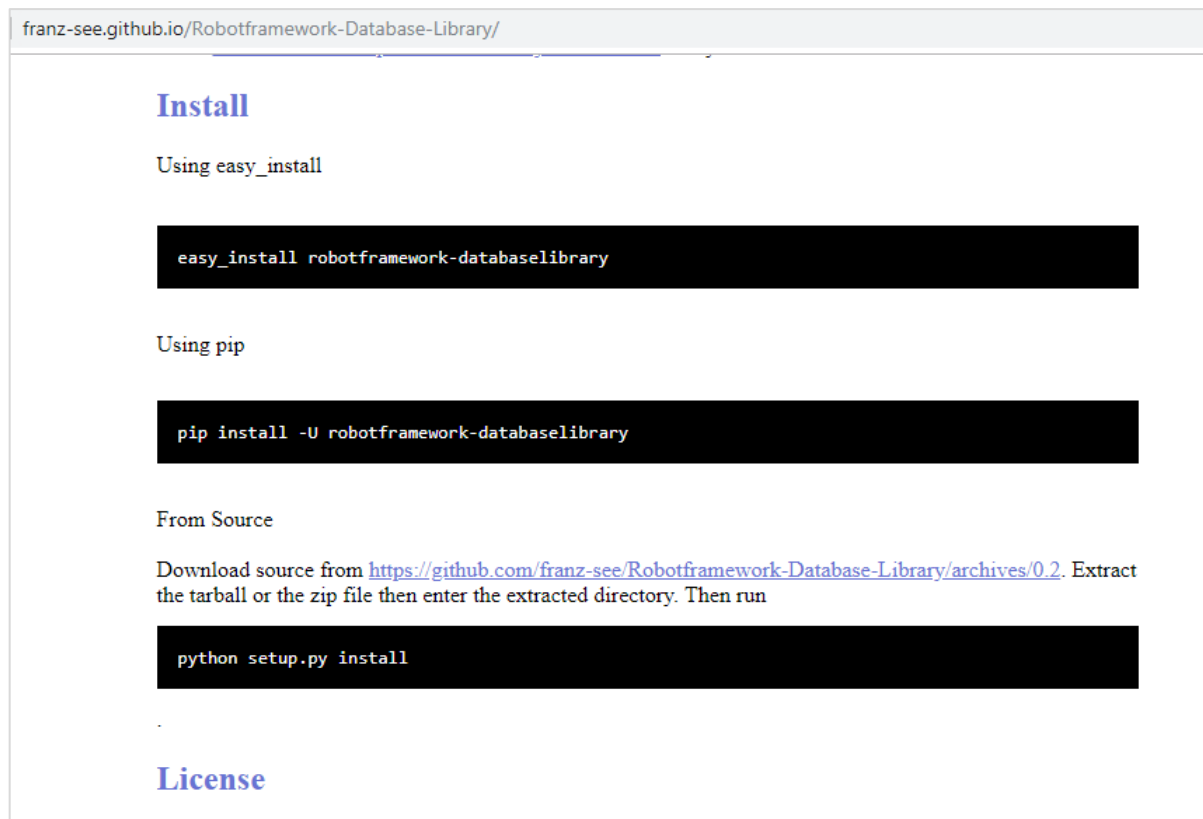
The Libraries are categorized as Standard, External and Other.

We will now take a look at the external library in this chapter. Upon clicking External, the following screen appears:

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.
Database Library (Java) Java-based library for database testing. Usable with Jython. Available also at Maven central .	Database Library (Python) Python based library for database testing. Works with any Python interpreter, including Jython.	Debug Library A debug library for RobotFramework, which can be used as an interactive shell(REPL) also.
Diff Library Library to diff two files together.	Django Library Library for Django , a Python web framework.	Eclipse Library Library for testing Eclipse RCP applications using SWT widgets.
robotframework-faker Library for Faker , a fake test data generator.	FTP library Library for testing and using FTP server with Robot Framework.	HTTP library (livetest) Library for HTTP level testing using livetest tool internally.

It shows the list of external libraries supported by Robot Framework. Here, we will focus more on the Database Library (Python). The same has been highlighted in the screenshot above.

Upon clicking the Database Library (Python), you will be redirected to the screen where the instruction for installation are listed as shown in the following screenshot:

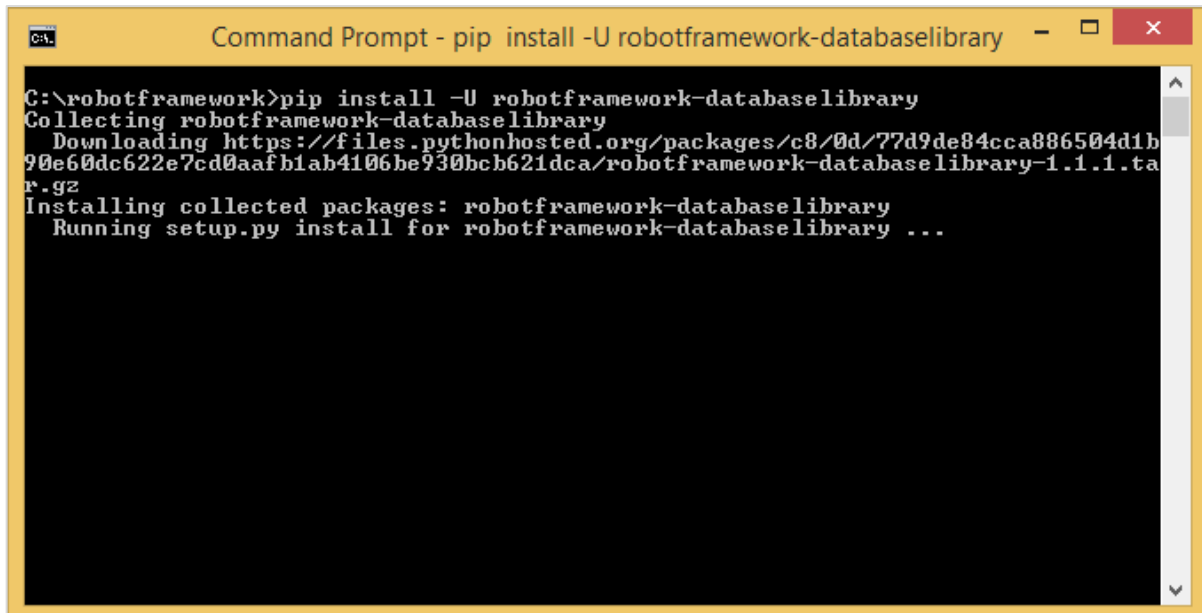


The screenshot shows a web browser window with the address bar displaying `franz-see.github.io/Robotframework-Database-Library/`. The page content includes a blue heading **Install**, followed by the text "Using easy_install" and a black terminal block containing the command `easy_install robotframework-databaselibrary`. Below this is the text "Using pip" and another black terminal block with the command `pip install -U robotframework-databaselibrary`. Further down, the text "From Source" is followed by instructions to download the source from <https://github.com/franz-see/Robotframework-Database-Library/archives/0.2>, extract the tarball or zip file, and then run a command. A black terminal block shows the command `python setup.py install`. At the bottom of the visible content is a blue heading **License**.

We can install the database library using pip and the command is:

```
pip install -U robotframework-databaselibrary
```

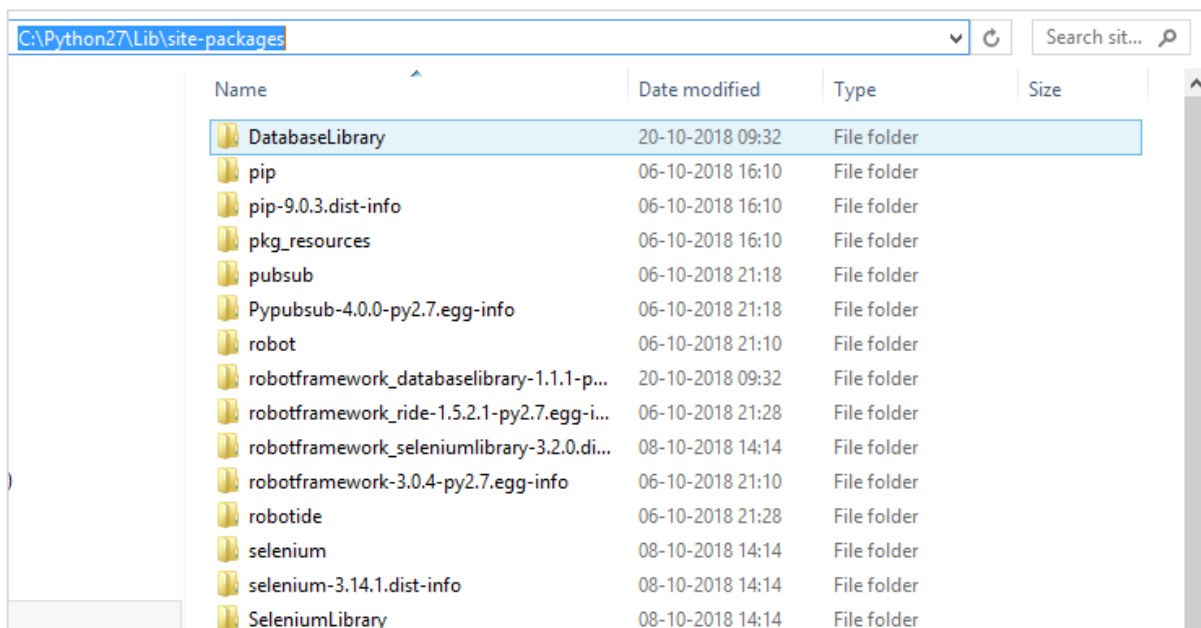
Run the above command in the command line as shown below:



```

C:\robotframework>pip install -U robotframework-databaselibrary
Collecting robotframework-databaselibrary
  Downloading https://files.pythonhosted.org/packages/c8/0d/77d9de84cca886504d1b90e60dc622e7cd0aafb1ab4106be930bcb621dca/robotframework-databaselibrary-1.1.1.tar.gz
Installing collected packages: robotframework-databaselibrary
  Running setup.py install for robotframework-databaselibrary ...
  
```

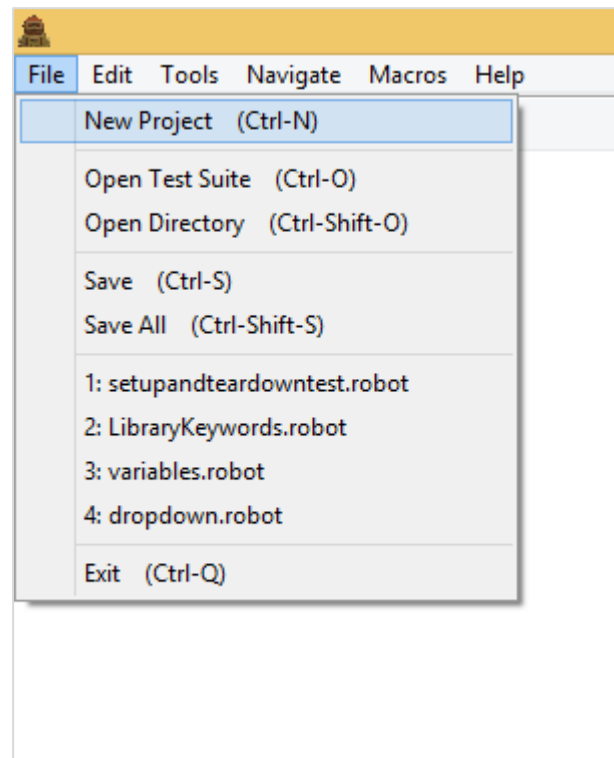
The Library is stored in python lib folder as shown below:



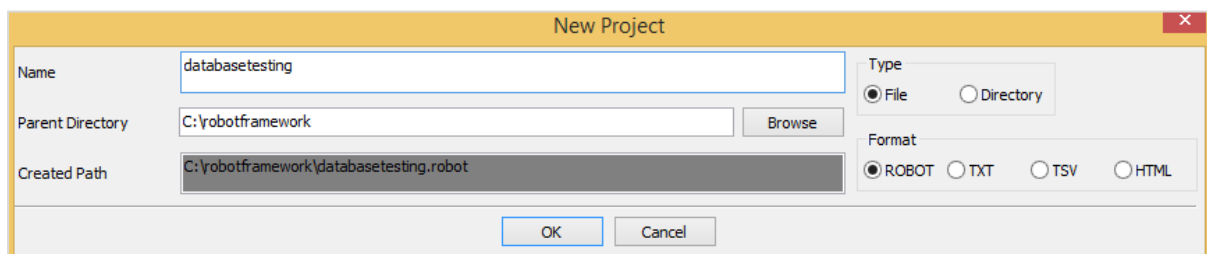
Once the installation is done, the next step is to import the library inside the project and use it with test cases.

Import Database Library

Open ride using **ride.py** from command line and create the project for testing database.



Click New Project and give a name to the project.



Click OK to save the project.

Click Library below Add Import.

Databasetesting

SourceC:\robotframework\databasetesting.robot

Settings >>

Import	Name / Path	Arguments	Comment

Variable	Value	Comment

Add Import

Library

Resource

Variables

Import Failed Help

Add Scalar

Add List

Add Dict

Library

Name

Browse

Args

Alias

Comment

Give name, optional arguments and optional alias of the library to import.
Separate multiple arguments with a pipe character like 'arg 1 | arg 2'.
Alias can be used to import same library multiple times with different names.

OK

Cancel

Enter the Name of the Library as DatabaseLibrary as shown below and click OK.

Library

Name

Browse

Args

Alias

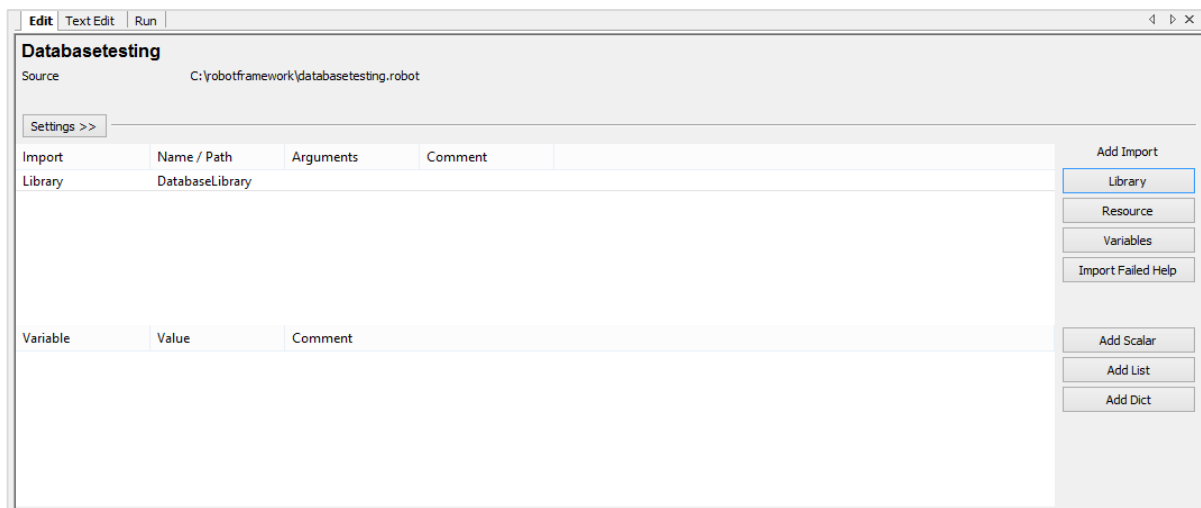
Comment

Give name, optional arguments and optional alias of the library to import.
Separate multiple arguments with a pipe character like 'arg 1 | arg 2'.
Alias can be used to import same library multiple times with different names.

OK

Cancel

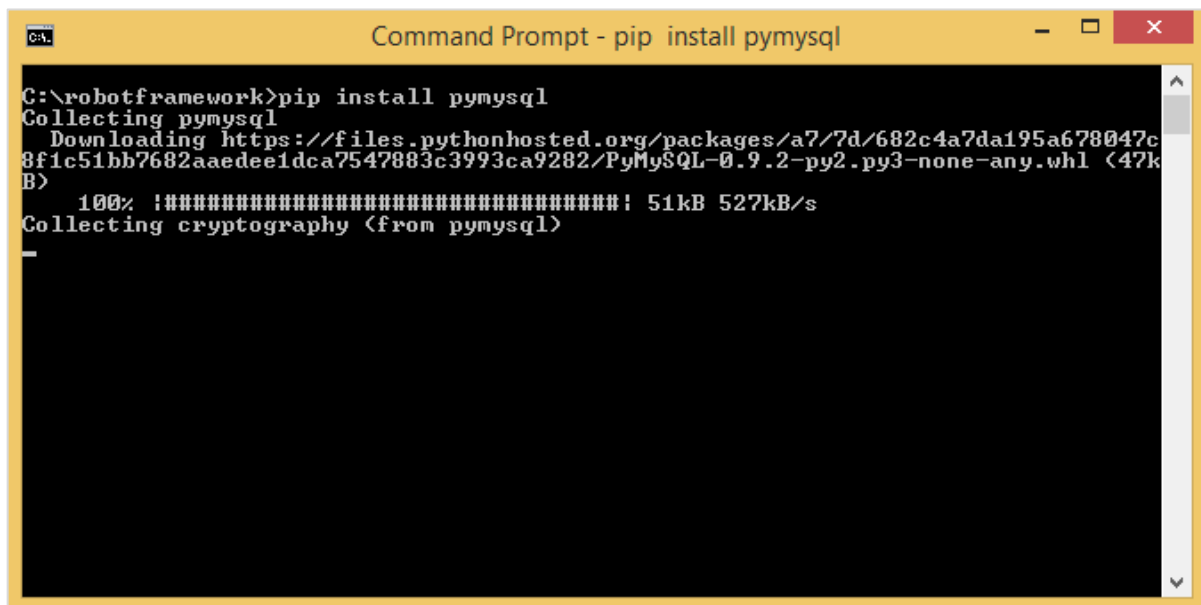
Once saved, the library is as shown below:



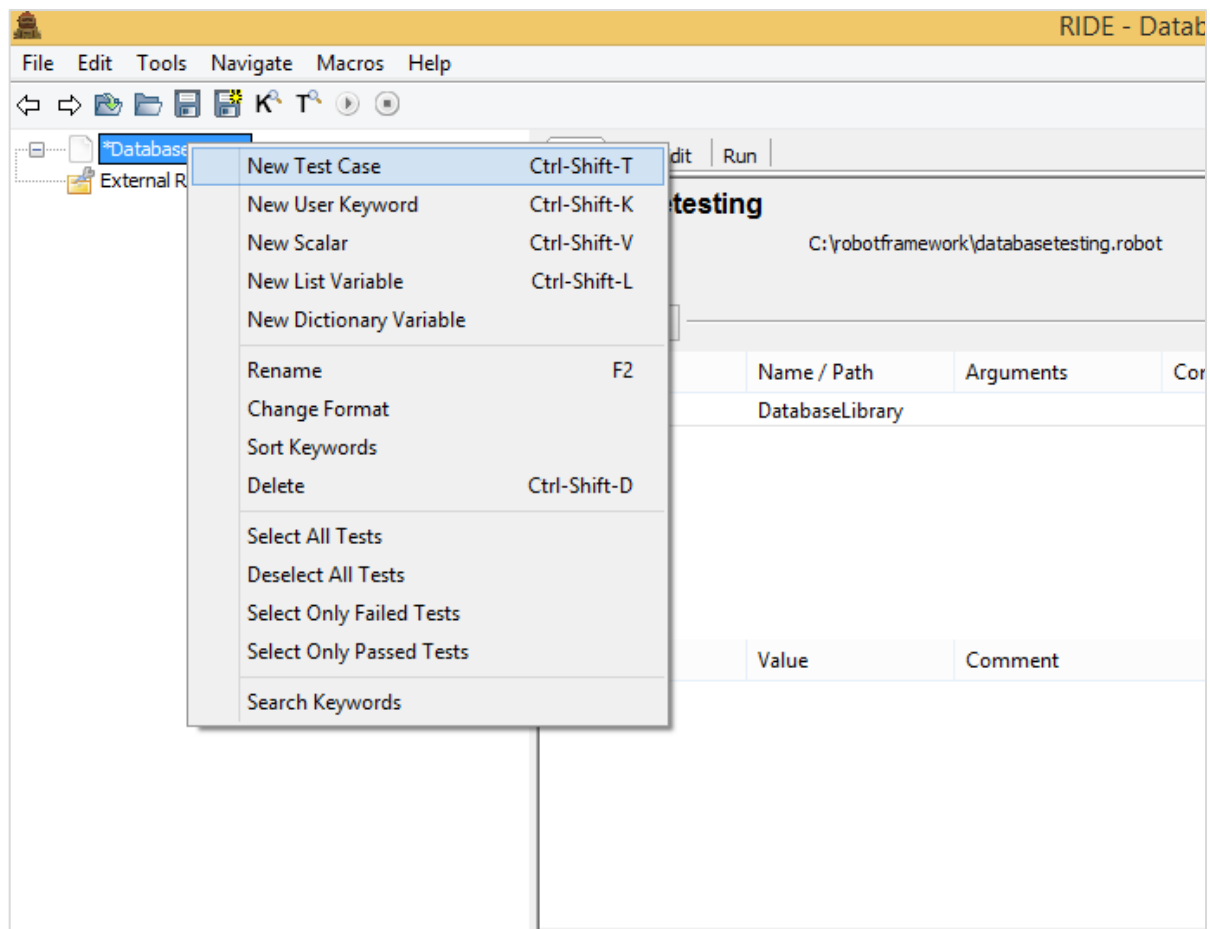
We are going to work with MySQL Database. To work with MySQL, we need to install the module.

Command

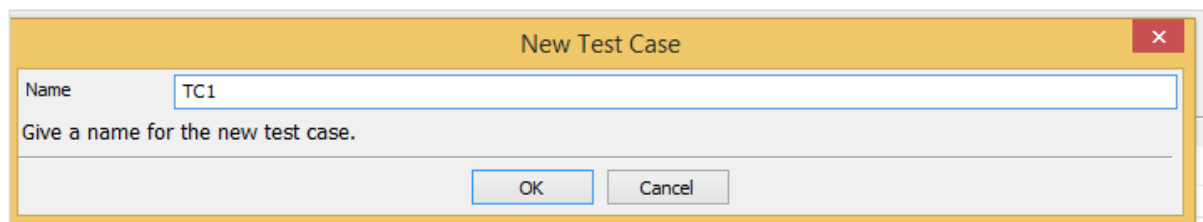
```
pip install pymysql
```



Now create test case under the project created.



Click New Test Case:



Enter the name of the test case and click OK.

We are going to use the existing database called *customers* available with us.

We will use phymyadmin to show the customer database:

Server: 127.0.0.1 » Database: customers » Table: customer

Browse Structure SQL Search Insert Export Import Privileges Operations More

Show query box

Showing rows 0 - 5 (6 total, Query took 0.0017 seconds.)

SELECT * FROM customer

[Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table

Sort by key: None

+ Options

	customer_id	customer_name	customer_email	status
☐ Edit Copy Delete	1	Jacob_William	jacob@gmail.com	1
☐ Edit Copy Delete	2	ssa	ss@gmail.com	1
☐ Edit Copy Delete	3	Aadarsh	aadarsh@gmail.com	1
☐ Edit Copy Delete	4	Aadhik Singh	aadhik@gmail.com	1
☐ Edit Copy Delete	5	Nidhi Singh	nidhi@gmail.com	0
☐ Edit Copy Delete	6	Siddi_Agarwal	siddi@gmail.com	0

We have a table called customer, which has data distributed in 6 rows. Now will write test-case which will connect to MySQL database customers and fetch the data from customer table.

Before we start, we will create scalar variables which will hold the data for dbname, dbuser, dbpasswd, dbhost, dbport and queryresult to store data, etc. Here are the variables created with values:

Variable	Value	Comment
\${dbname}	customers	
\${dbuser}	root	
\${dbpasswd}	admin	
\${dbhost}	localhost	
\${dbport}	3306	
@{queryResults}		

The command to connect to database is:

```
Connect To Database    pymysql    ${dbname}    ${dbuser}    ${dbpasswd}
${dbhost}    ${dbport}
```

1	Connect To Database	pymysql	\${dbname}	\${dbuser}	\${dbpasswd}	\${dbhost}	\${dbport}
2							
3							
4							
5							
6							

We will add some more test cases as shown below:

1	Connect To Database	pymysql	\${dbname}
2	Table Must Exist	customer	
3	Check If Exists In Database	SELECT * FROM customer	
4	@{queryResults}	Query	SELECT * FROM customer
5	Log	@{queryResults}[0]	
6			
7			
8			

Here are the details:

```

*** Settings ***
Library           DatabaseLibrary

*** Variables ***
${dbname}         customers
${dbuser}         root
${dbpasswd}       admin
${dbhost}         localhost
${dbport}         3306
@{queryResults}

*** Test Cases ***
TC1
    Connect To Database    pymysql    ${dbname}    ${dbuser}    ${dbpasswd}
    ${dbhost}    ${dbport}
    Table Must Exist      customer
    Check If Exists In Database    SELECT * FROM customer
    @{queryResults}    Query    SELECT * FROM customer
    Log    @{queryResults}[0]

```

We have connected to the database, checked if table customer exists in database, got the query executed and logged the details of the query.

We will execute the test case and see the output:

```

elapsed time: 0:00:02  pass: 1  fail: 0
command: pybot.bat --argumentfile c:\users\kamat\appdata\local\temp\RIDEtfvyrf.d\argfile.txt --listener C:\Python27\lib\site
=====
Databasetesting
=====
TC1
=====
Databasetesting
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
=====
Output:  c:\users\appdata\local\temp\RIDEtfvyrf.d\output.xml
Log:      c:\users\appdata\local\temp\RIDEtfvyrf.d\log.html
Report:   c:\users\appdata\local\temp\RIDEtfvyrf.d\report.html

test finished 20181020 11:09:38

Starting test: Databasetesting.TC1
20181020 11:09:38.011 : INFO : Connecting using : pymysql.connect(db=customers, user=root, passwd=admin, host=localhost, pc
20181020 11:09:38.042 : INFO : Executing : Table Must Exist | customer
20181020 11:09:38.042 : INFO : Executing : Row Count | SELECT * FROM information_schema.tables WHERE table_name='customer'
20181020 11:09:38.058 : INFO : Executing : Check If Exists In Database | SELECT * FROM customer
20181020 11:09:38.058 : INFO : Executing : Query | SELECT * FROM customer
20181020 11:09:38.058 : INFO : Executing : Query | SELECT * FROM customer
20181020 11:09:38.073 : INFO : @{queryResults} = [ (1, 'Jacob_William', 'jacob@gmail.com', 1) | (2, 'ssa', 'ss@gmail.com',
20181020 11:09:38.073 : INFO : (1, 'Jacob_William', 'jacob@gmail.com', 1)
Ending test: Databasetesting.TC1

```

The results from the table are shown for the queryResults.

Log Details

Databasetesting Test Log

Generated
20181020 11:09:38 GMT+05:30
1 minute 26 seconds ago

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
Databasetesting	1	1	0	00:00:00	

Test Execution Log

SUITE Databasetesting

Full Name: Databasetesting

Source: C:\robotframework\databasetesting.robot

Start / End / Elapsed: 20181020 11:09:37.617 / 20181020 11:09:38.073 / 00:00:00.456

Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

TEST TC1

Details of TC1

TEST	TC1	00:00:00.250
Full Name: Databasetesting.TC1		
Start / End / Elapsed: 20181020 11:09:37.823 / 20181020 11:09:38.073 / 00:00:00.250		
Status: PASS (critical)		
KEYWORD	DatabaseLibrary Connect To Database pymysql, \${dbname}, \${dbuser}, \${dbpasswd}, \${dbhost}, \${dbport}	00:00:00.203
Documentation: Loads the DB API 2.0 module given 'dbapiModuleName' then uses it to		
Start / End / Elapsed: 20181020 11:09:37.823 / 20181020 11:09:38.026 / 00:00:00.203		
11:09:38.011	INFO Connecting using : pymysql.connect(db=customers, user=root, passwd=admin, host=localhost, port=3306, charset=None)	
KEYWORD	DatabaseLibrary Table Must Exist customer	00:00:00.016
Documentation: Check if the table given exists in the database. Set optional input 'sansTran' to True to run command without an		
Start / End / Elapsed: 20181020 11:09:38.026 / 20181020 11:09:38.042 / 00:00:00.016		
11:09:38.042	INFO Executing : Table Must Exist customer	
11:09:38.042	INFO Executing : Row Count SELECT * FROM information_schema.tables WHERE table_name='customer'	
KEYWORD	DatabaseLibrary Check If Exists In Database SELECT * FROM customer	00:00:00.016
Documentation: Check if any row would be returned by given the input 'selectStatement'. If there are no results, then this will		
Start / End / Elapsed: 20181020 11:09:38.042 / 20181020 11:09:38.058 / 00:00:00.016		
11:09:38.058	INFO Executing : Check If Exists In Database SELECT * FROM customer	
11:09:38.058	INFO Executing : Query SELECT * FROM customer	
KEYWORD	@{queryResults} = DatabaseLibrary Query SELECT * FROM customer	00:00:00.015
Documentation: Uses the input 'selectStatement' to query for the values that will be returned as a list of tuples. Set optional		
Start / End / Elapsed: 20181020 11:09:38.058 / 20181020 11:09:38.073 / 00:00:00.015		
11:09:38.058	INFO Executing : Query SELECT * FROM customer	
11:09:38.073	INFO @{queryResults} = [(1, 'Jacob_William', 'jacob@gmail.com', 1) (2, 'ssa', 'ss@gmail.com', 1) (3, 'Aadarsh', 'aadarsh@gmail.com', 1) (4, 'Aadhik Singh', 'aadhi@gmail.com', 1) (5, 'Nidhi Singh', 'nidhi@gmail.com', 1)]	
KEYWORD	Builtin Log @{queryResults}[0]	00:00:00.000
Documentation: Logs the given message with the given level		
Start / End / Elapsed: 20181020 11:09:38.073 / 20181020 11:09:38.073 / 00:00:00.000		
11:09:38.073	INFO (1, 'Jacob_William', 'jacob@gmail.com', 1)	

Conclusion

We have seen how to import database library, and the installation of it. We now know how to connect to MySQL database in Robot Framework and test the tables.

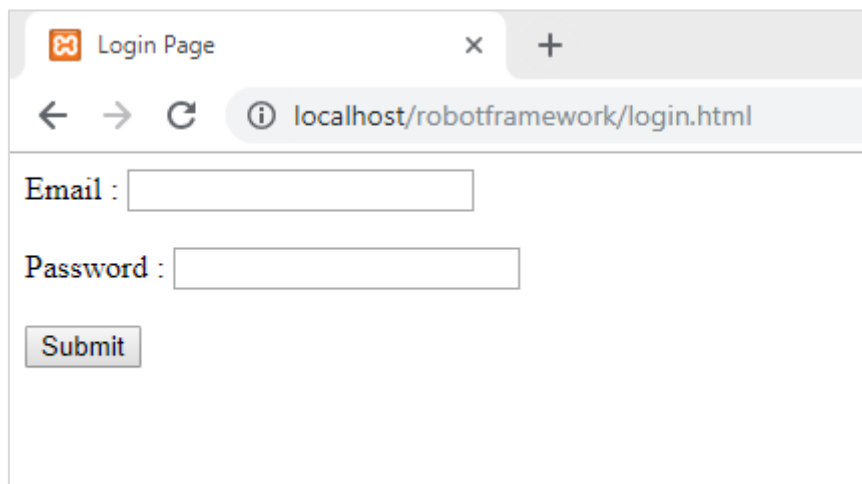
19. Robot Framework — Testing Login Page Using Robot Framework

With Robot Framework, we have learnt the following so far:

- Import Libraries
- Working with variables
- Create custom Keywords
- How to write test-cases
- How to create Setup and teardown
- How to execute test cases
- How to work with data driven test-cases

We will use all of the above features and use it to test login page in this chapter. We have a login page which takes in email-id and password. When you enter correct email id and password, you will be redirected to a welcome page. If the user enters invalid email id or password, the page will get redirected to error page.

The following screenshot shows a login page:



The screenshot shows a web browser window with a single tab titled "Login Page". The address bar displays "localhost/robotframework/login.html". The page content includes a form with two input fields: "Email :" and "Password :", each followed by a text input box. Below these fields is a "Submit" button.

HTML Code

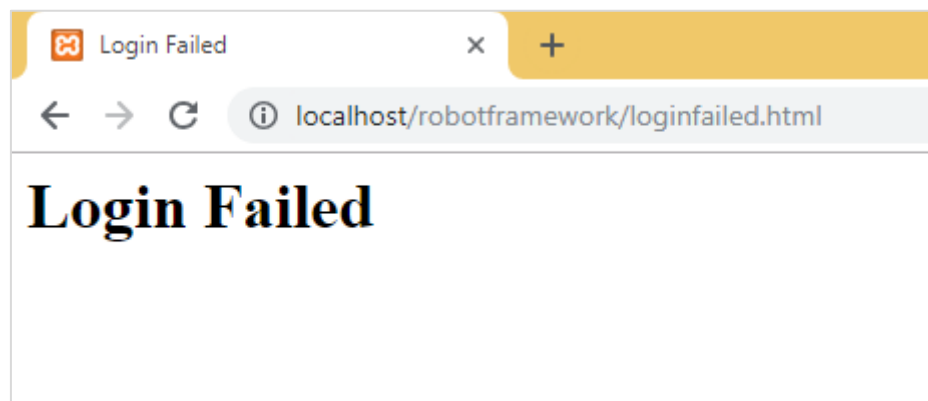
```
<html>
<head>
<title>Login Page</title>
</head>
<body>
<script type="text/javascript">
function wsSubmit() {
```

```

        if (document.getElementById("email").value == "admin@gmail.com" &&
document.getElementById("passwd").value == "admin") {
            location.href = "http://localhost/robotframework/success.html";
        } else {
            location.href = "http://localhost/robotframework/loginfailed.html";
        }
    }
</script>
<div id="formdet">
Email : <input type="text" id="email" value="" id="email" /><br/><br/>
Password : <input type="password" id="passwd" value="" /><br/><br/>
<input type="submit" id="btnsubmit" value="Submit" onClick="wsSubmit();" />
</div>
</body>
</html>

```

The following screen appears when either the email-id or the password is not valid:



HTML Code

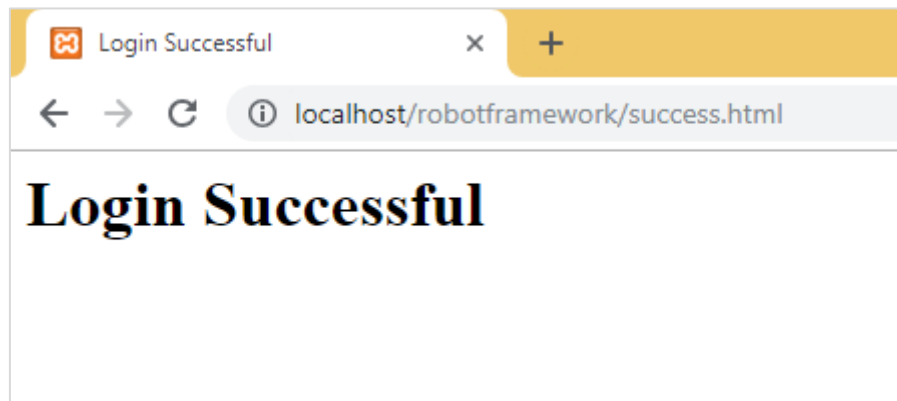
```

<html>
<head>
<title>Login Failed</title>
</head>
<body>
<div id="loginfailed">
<h1>Login Failed</h1>
</div>

```

```
</body>  
</html>
```

The following screen appears when both the email id and password are valid:



HTML Code

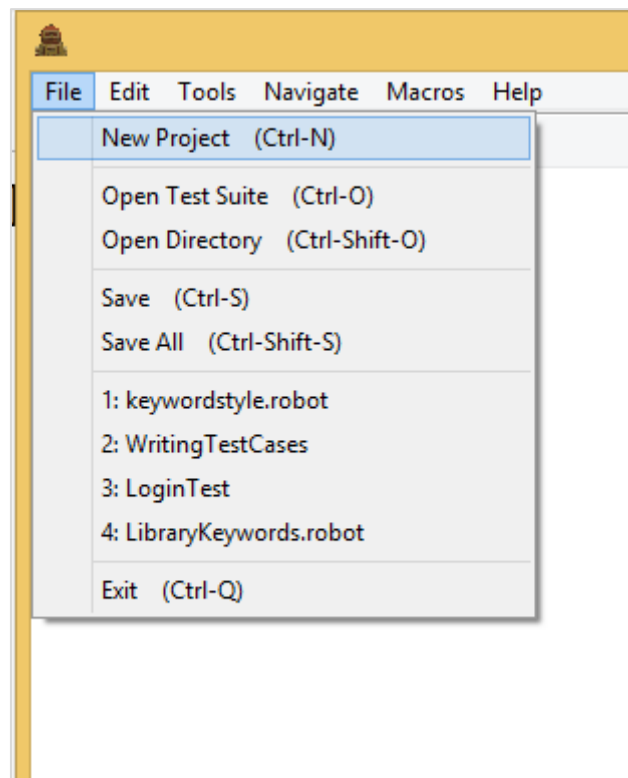
```
<html>  
<head>  
<title>Login Successful</title>  
</head>  
<body>  
<div id="loginfailed">  
<h1>Login Successful</h1>  
</div>  
</body>  
</html>
```

Now we are going to write test cases for the above test page. To start with it, we will first run the command to open Ride.

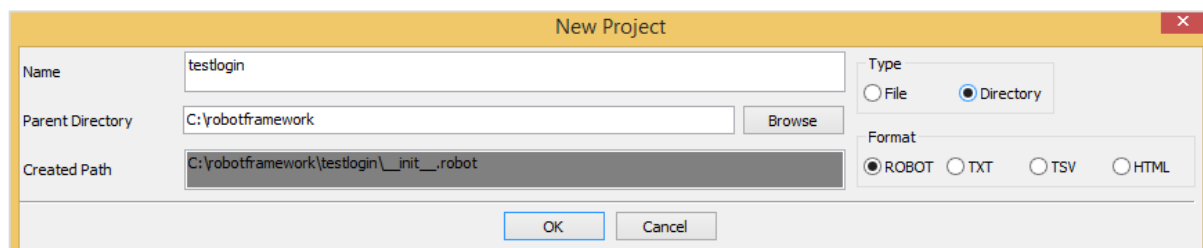
Command

```
ride.py
```

Once done, we will get started with the project setup as shown below:

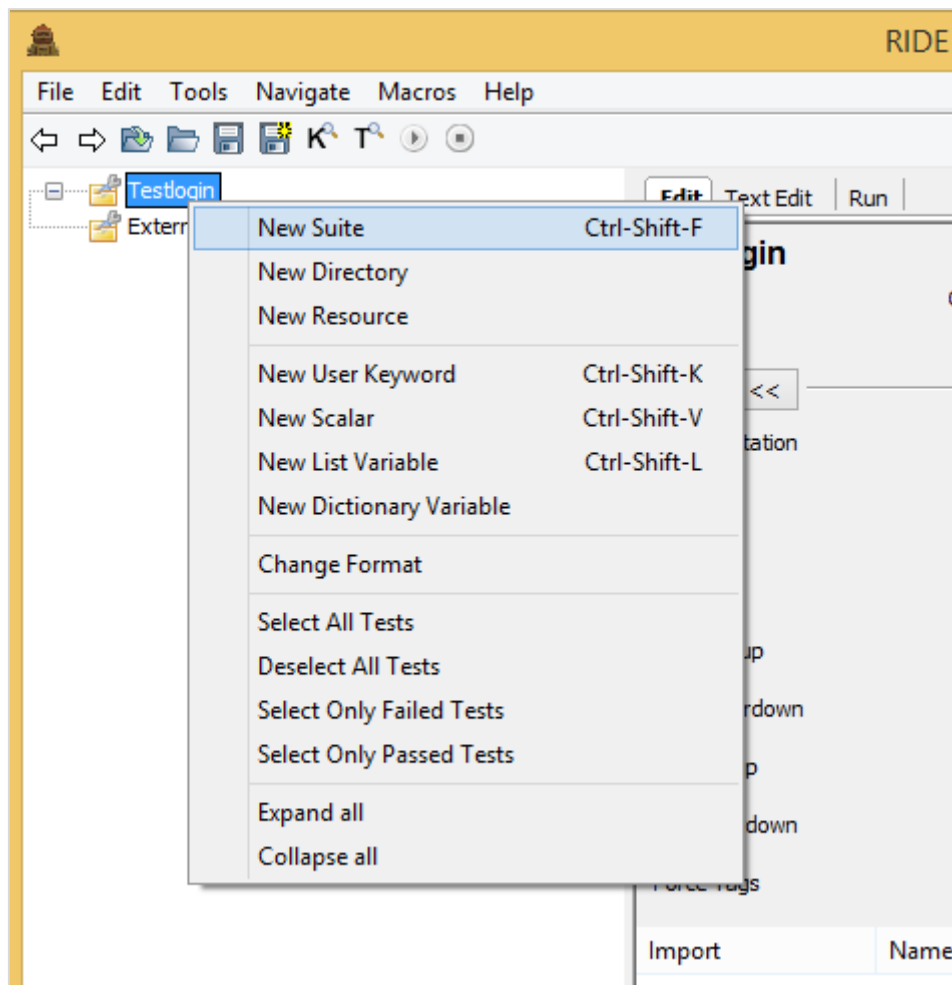


Click New Project and enter the name of the project.

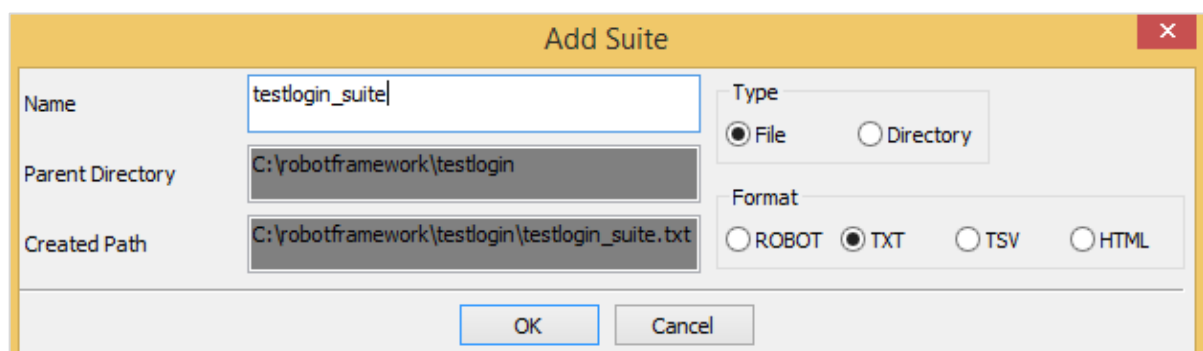


We will save the type of the project as Directory. The name given to the project is testlogin. Click OK to save the project.

Now, we will create test suite inside the project.

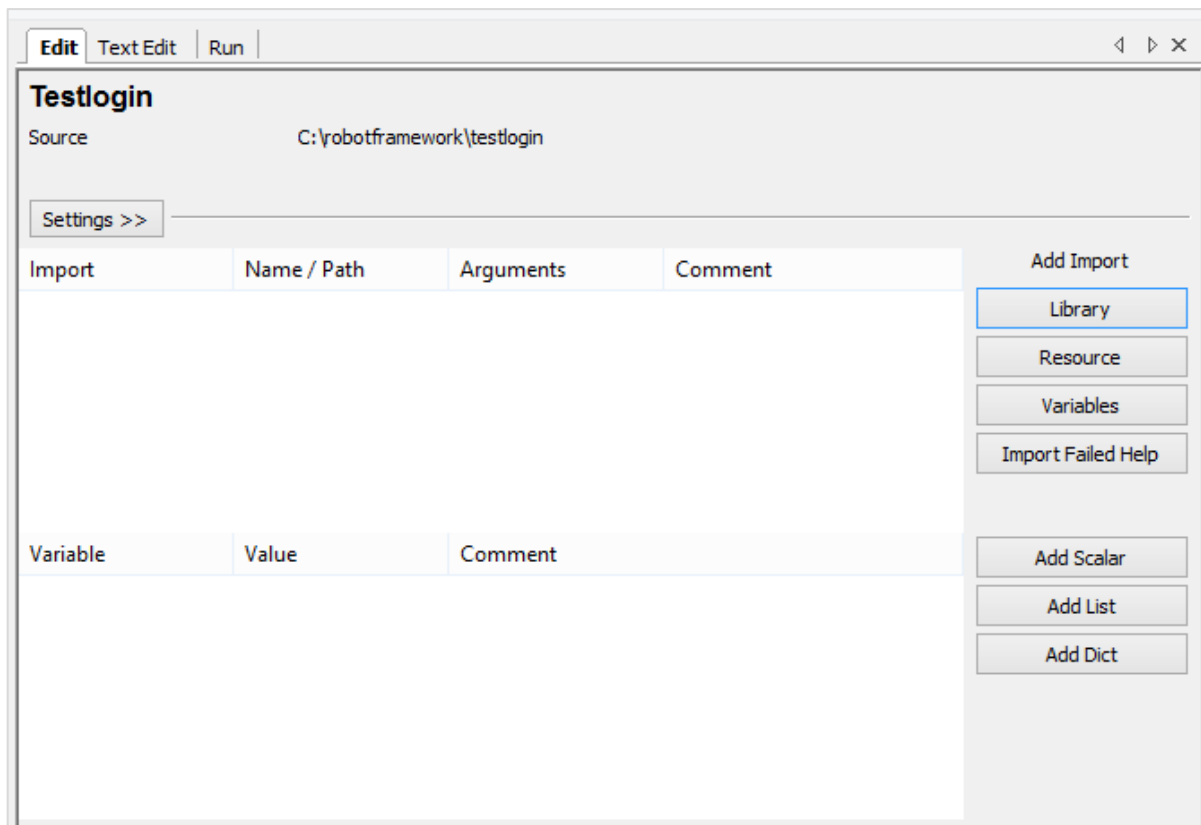


Click New Suite and it will display a screen as shown below:

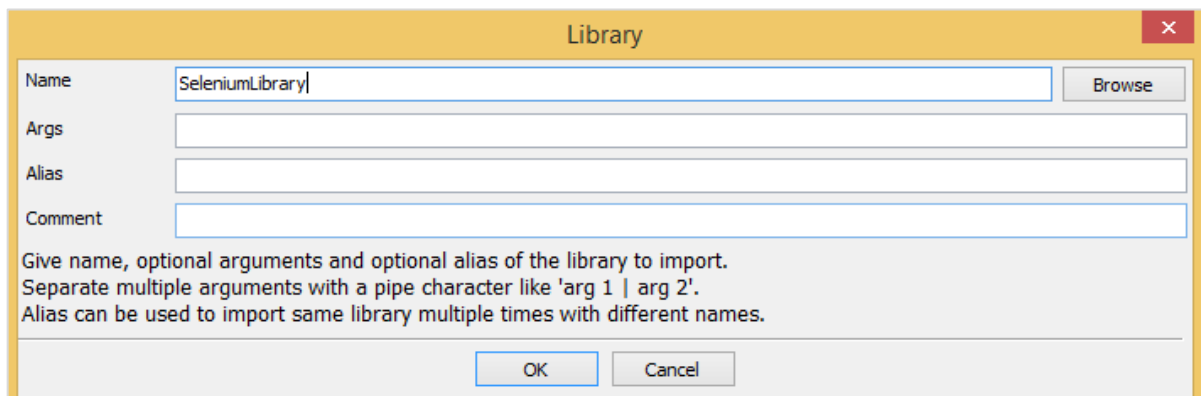


Click OK to save the test suite. We need to import the Selenium Library since we will be working with the browser.

Import Library in the main project and also to the test suite created.



Click Library as in the above screenshot. Upon clicking Library, the following screen will appear.



Click OK to save the library for the project.

Once the library is saved for the project, it will display the library in the settings:

Testlogin

Source C:\robotframework\testlogin

Settings >>

Import	Name / Path	Arguments	Comment
Library	SeleniumLibrary		

Variable	Value	Comment
----------	-------	---------

Add Import

Library

Resource

Variables

Import Failed Help

Add Scalar

Add List

Add Dict

Repeat the same step for the Test suite created.

Here is the library added for Test suite:

Testlogin Suite

Source C:\robotframework\testlogin\testlogin_suite.txt

Settings >>

Import	Name / Path	Arguments	Comment
Library	SeleniumLibrary		

Variable	Value	Comment
----------	-------	---------

Add Import

Library

Resource

Variables

Import Failed Help

Add Scalar

Add List

Add Dict

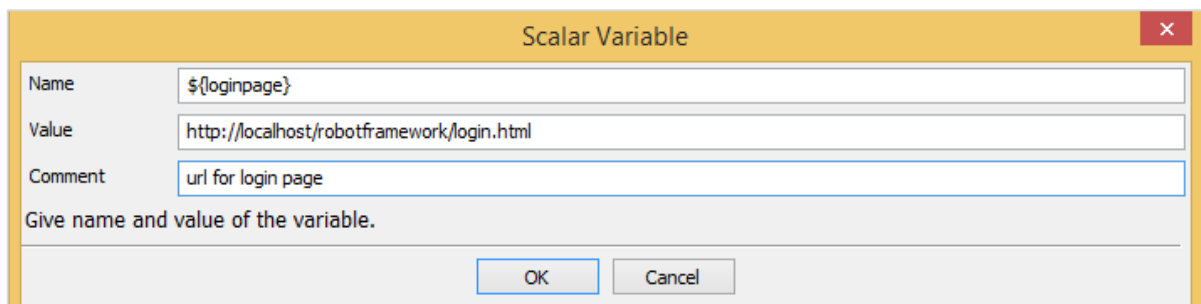
Now in the main Project, we will create a setup and teardown. We would like to open the login page in Chrome browser and maximize the window. In teardown, we will close the browser.

For setup, we will create a user-defined keyword called **Open Login Page**. This keyword will take 2 arguments, login page URL and browser name.

Now, we need 2 scalar variables that will help us store the values – url and the browser name.

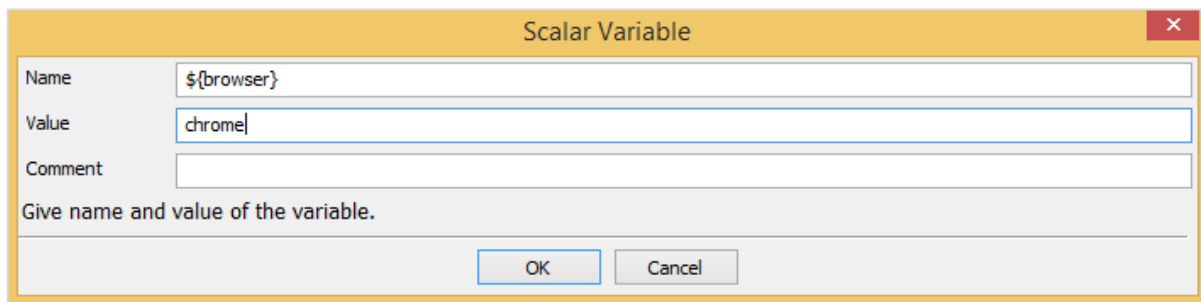
In ride, create 2 variables **`${loginpage}`** and **`${browser}`** as follows:

`${loginpage}`



The screenshot shows a 'Scalar Variable' dialog box with a yellow header and a red close button. It contains three input fields: 'Name' with the value `${loginpage}`, 'Value' with the value `http://localhost/robotframework/login.html`, and 'Comment' with the value `url for login page`. Below these fields is a text prompt 'Give name and value of the variable.' and two buttons, 'OK' and 'Cancel'.

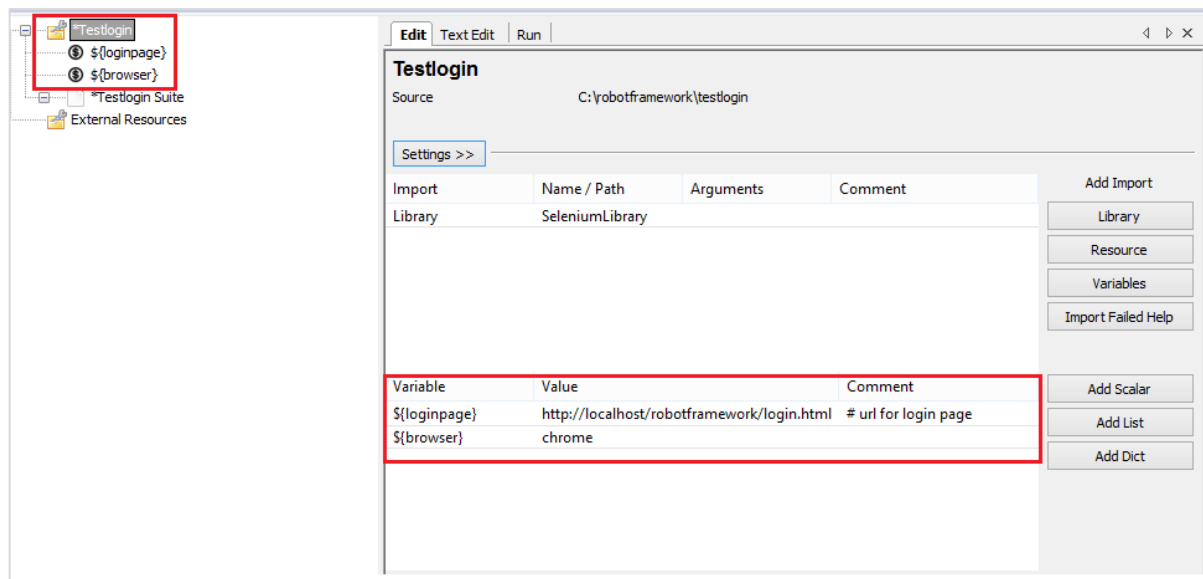
`${browser}`



The screenshot shows a 'Scalar Variable' dialog box with a yellow header and a red close button. It contains three input fields: 'Name' with the value `${browser}`, 'Value' with the value `chrome`, and 'Comment' which is empty. Below these fields is a text prompt 'Give name and value of the variable.' and two buttons, 'OK' and 'Cancel'.

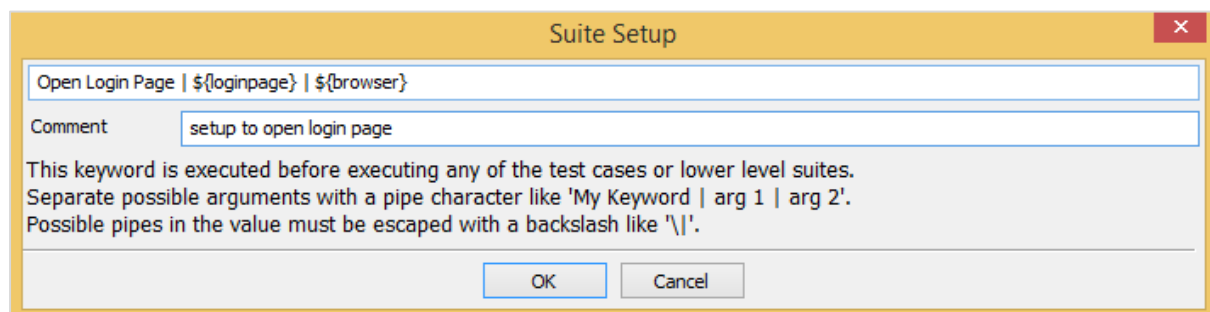
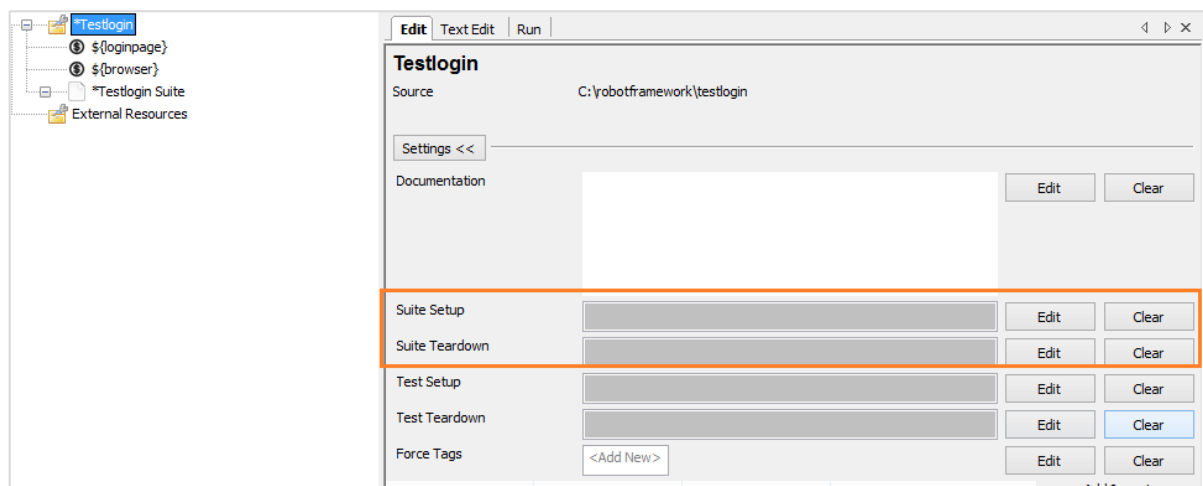
Save both variables.

The variables will be displayed under your project as follows:



Now, we will add the setup and teardown for the main project.

Click on the project on the left side. In settings, click Suite Setup.

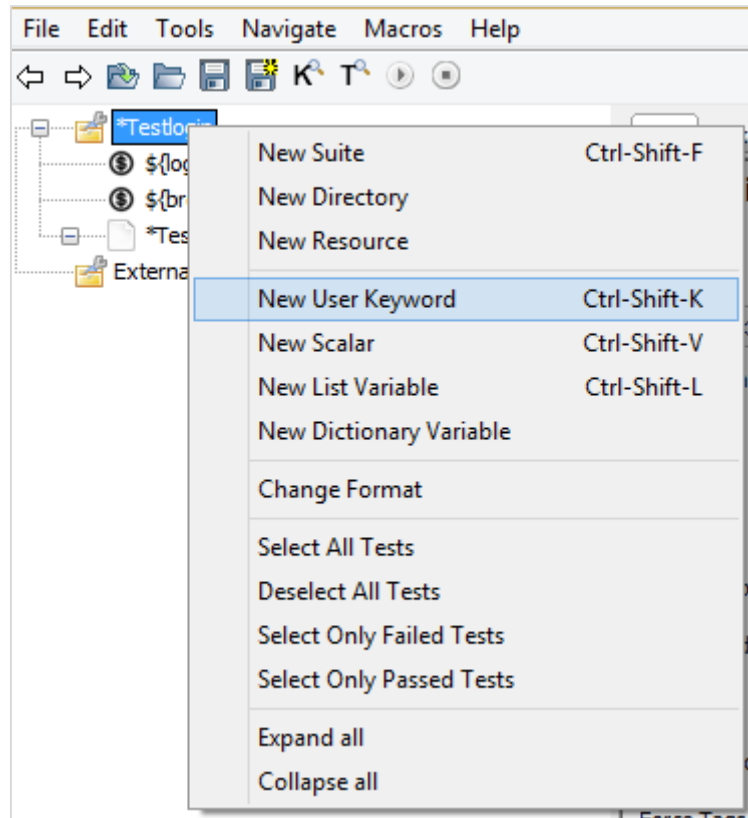


We have created setup that is using user keyword **Open Login Page** with arguments **`${loginpage}`** and **`${browser}`**.

Click OK to save the setup.

Now, we have to create the user-defined keyword **Open Login Page**, which is done as follows:

Right-click on the project and click **New User Keyword**:



Upon clicking **New User Keyword**, the following screen appears:

New User Keyword

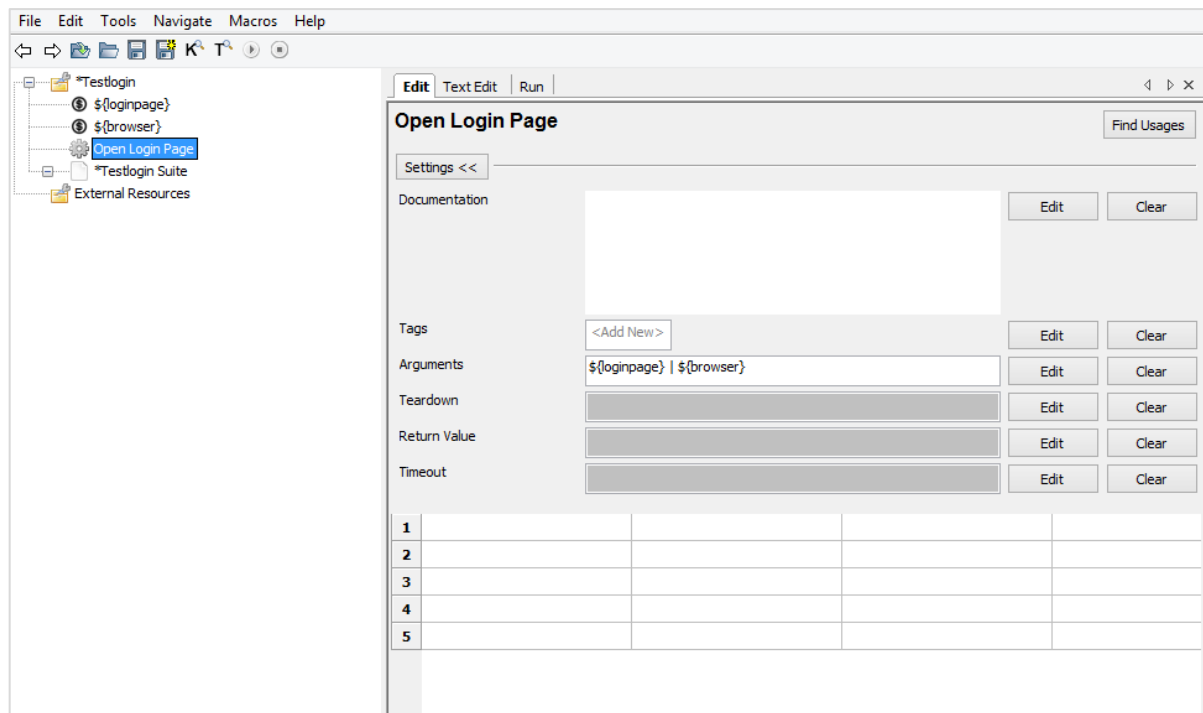
Name: Open Login Page

Arguments: `${loginpage} | ${browser}`

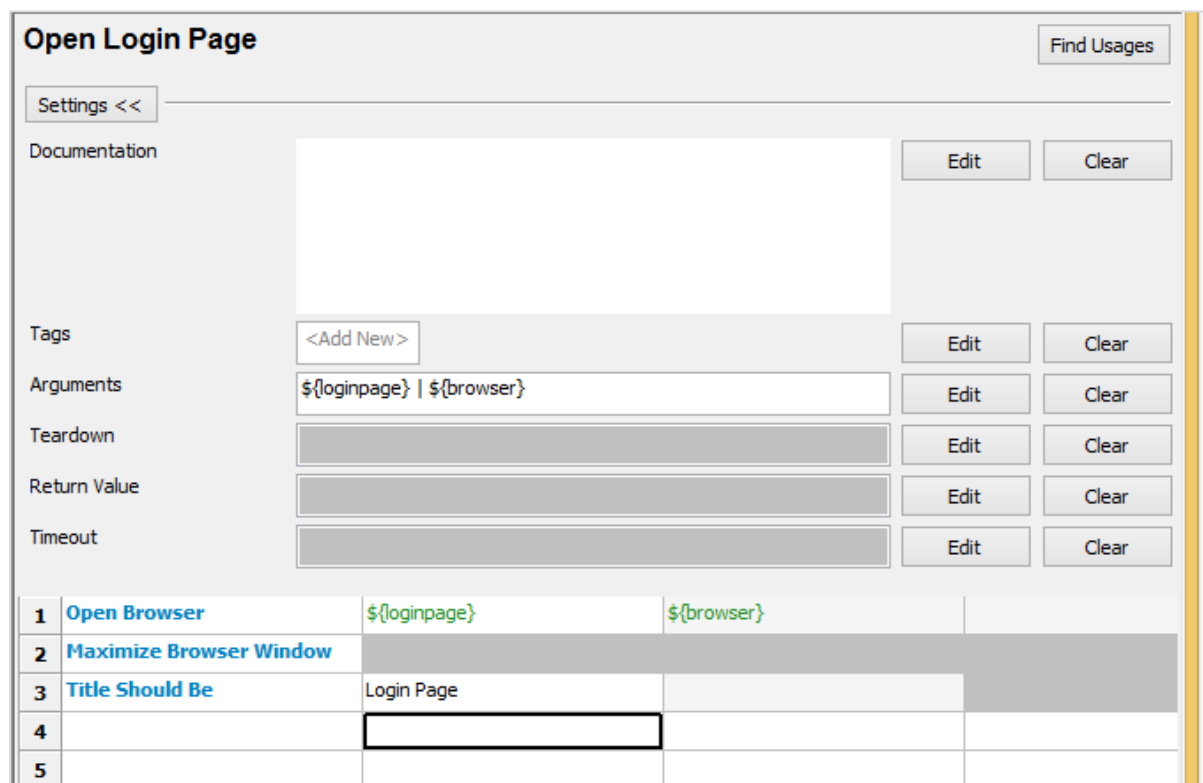
Give a name and arguments for the new user keyword.
 Specify the arguments separated with a pipe character like '`${arg1} | ${arg2}`'.
 Default values are given using equal sign and the last argument can be a list variable.
 Example: '`${arg1} | ${arg2}=default value | @rest`'.
 Note. You can use variable shortcuts in this field.

OK Cancel

Here the Keyword is given 2 arguments – **`${loginpage}`** and **`${browser}`**. Click OK to save the user keyword.



Now we need to enter the library keywords, which will open the URL.



Open Login Page user-defined keyword has the following details:

*** Keywords ***

Open Login Page

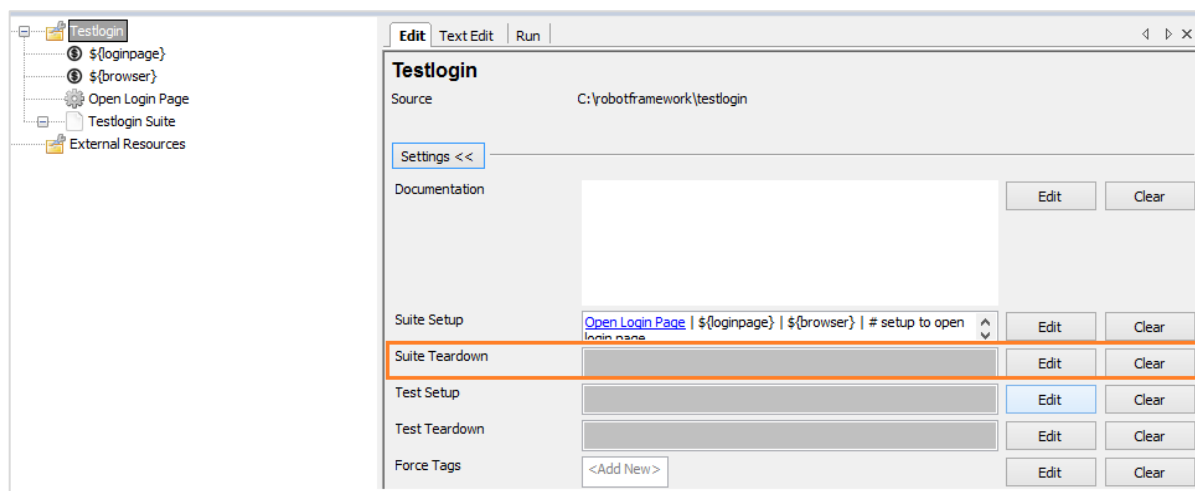
[Arguments] \${loginpage} \${browser}

Open Browser \${loginpage} \${browser}

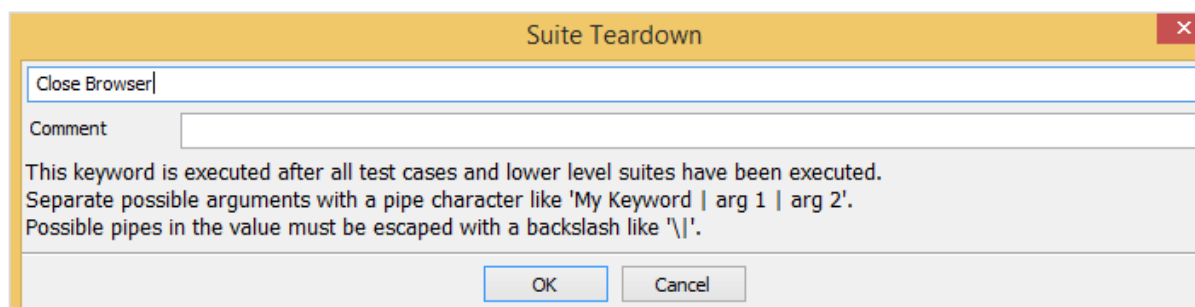
Maximize Browser Window

Title Should Be Login Page

Now, we will create **Suite Teardown** for the suite.

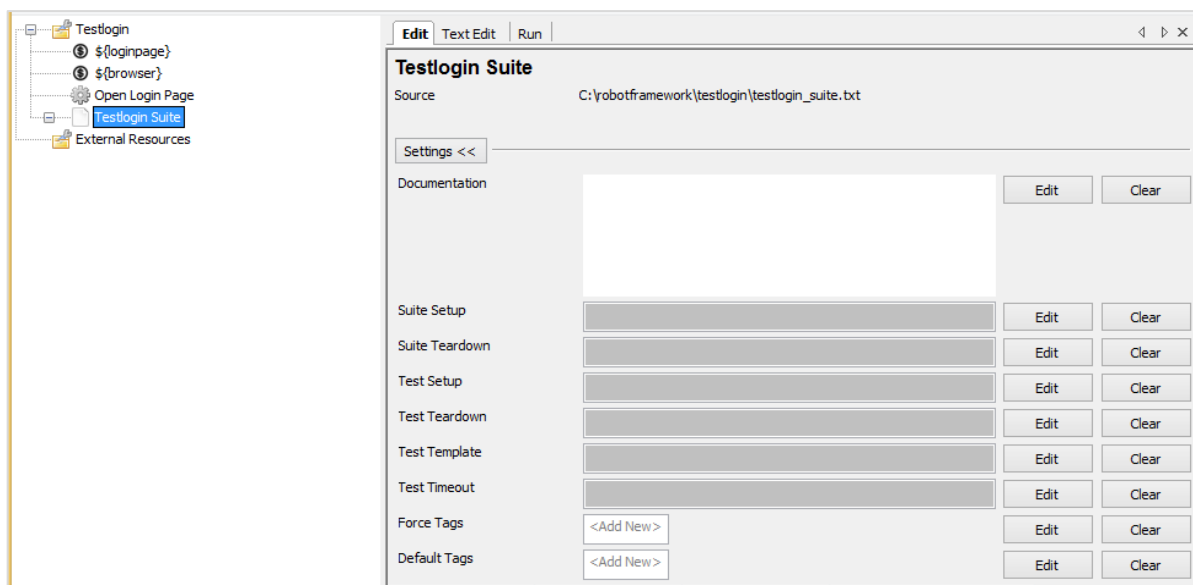


Click Edit for Suite Teardown and enter the details:



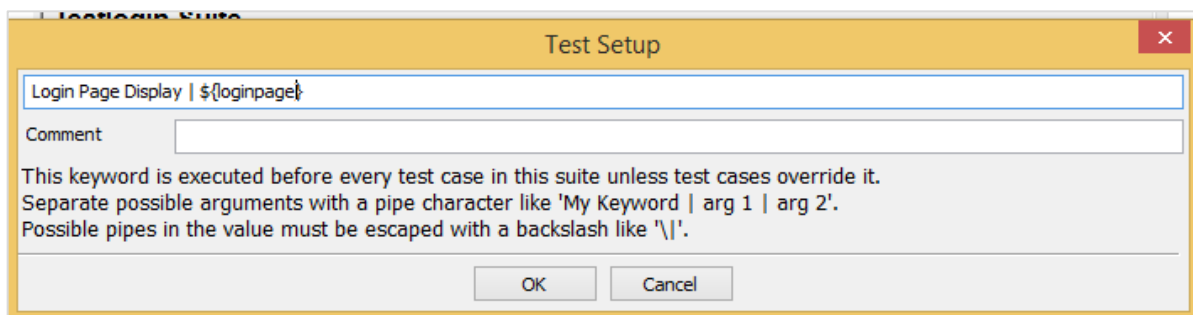
For Suite teardown, we are directly using library keyword, which will close the browser. Click OK to save the suite teardown.

Now, click the Testlogin Suite we have created.



Let us now create a setup for the test suite – Test Setup. This setup needs to get executed first.

Click Edit for Test Setup and enter the details.



For the Test Setup, we have created User defined Keyword called **Login Page Display**, which will take the argument as **\$ {loginpage}** as in the above screenshot.

Click OK to save the test setup.

Testlogin Suite

Source C:\robotframework\testlogin\testlogin_suite.txt

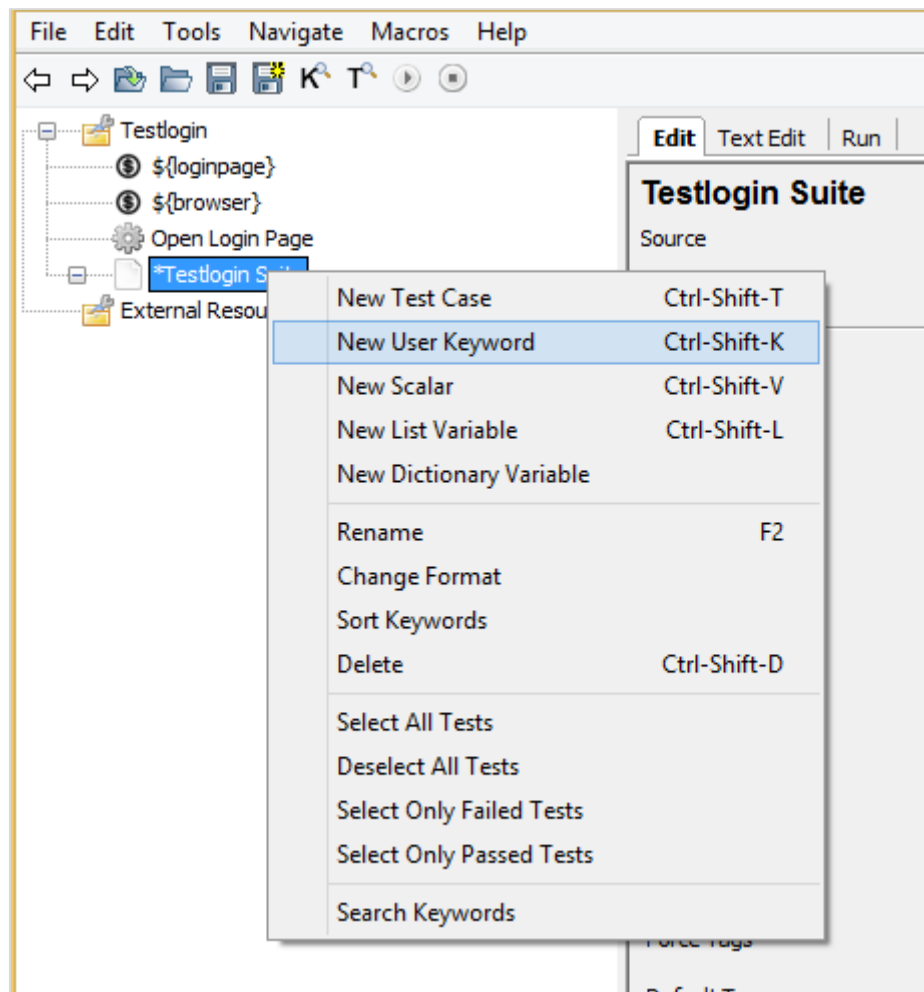
Settings <<

Documentation		Edit	Clear
Suite Setup		Edit	Clear
Suite Teardown		Edit	Clear
Test Setup	Login Page Display \${loginpage}	Edit	Clear
Test Teardown		Edit	Clear
Test Template		Edit	Clear
Test Timeout		Edit	Clear
Force Tags	<Add New>	Edit	Clear
Default Tags	<Add New>	Edit	Clear

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

Now, we need to create the user keyword **Login Page Display**.

Right-click on the test suite and click **New User Keyword** as shown below:



New User Keyword will display the screen as shown below:

×

New User Keyword

Name

Login Page Display

Arguments

`\${loginpage}`

Give a name and arguments for the new user keyword.
 Specify the arguments separated with a pipe character like `\${arg1} | \${arg2}`.
 Default values are given using equal sign and the last argument can be a list variable.
 Example: `\${arg1} | \${arg2}=default value | @rest`.
 Note. You can use variable shortcuts in this field.

OK

Cancel

Click OK to save the keyword.

Let us now enter the keyword we need for the user keyword **Login Page Display**.

Login Page Display

Find Usages

Settings <<

Documentation

Edit

Clear

Tags

<Add New>

Edit

Clear

Arguments

`${loginpage}`

Edit

Clear

Teardown

Edit

Clear

Return Value

Edit

Clear

Timeout

Edit

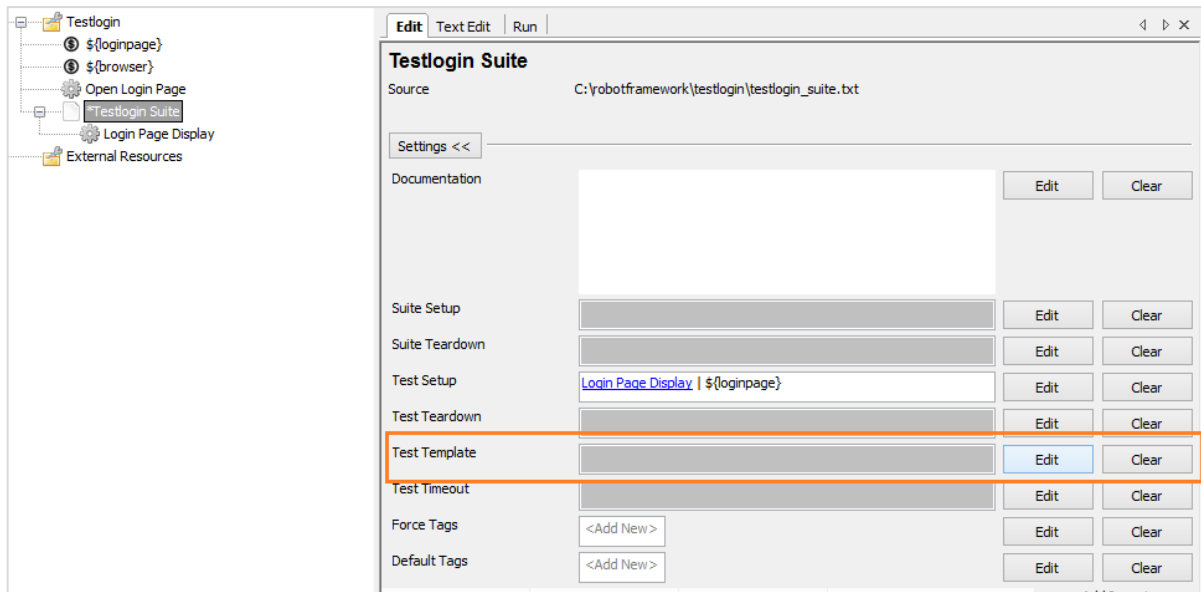
Clear

1	Go To	<code>\${loginpage}</code>		
2	Title Should Be	Login Page		
3				
4				
5				

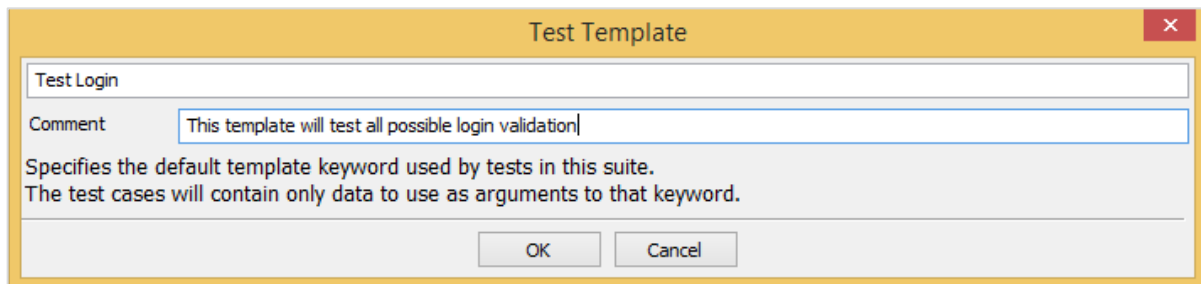
Here we want to go to the **loginpage** and check if the title of the page matches with the value given.

Now, we will add template to the test suite and create data driven test cases.

To create template, click on the suite and on right side click Edit for Test Template.



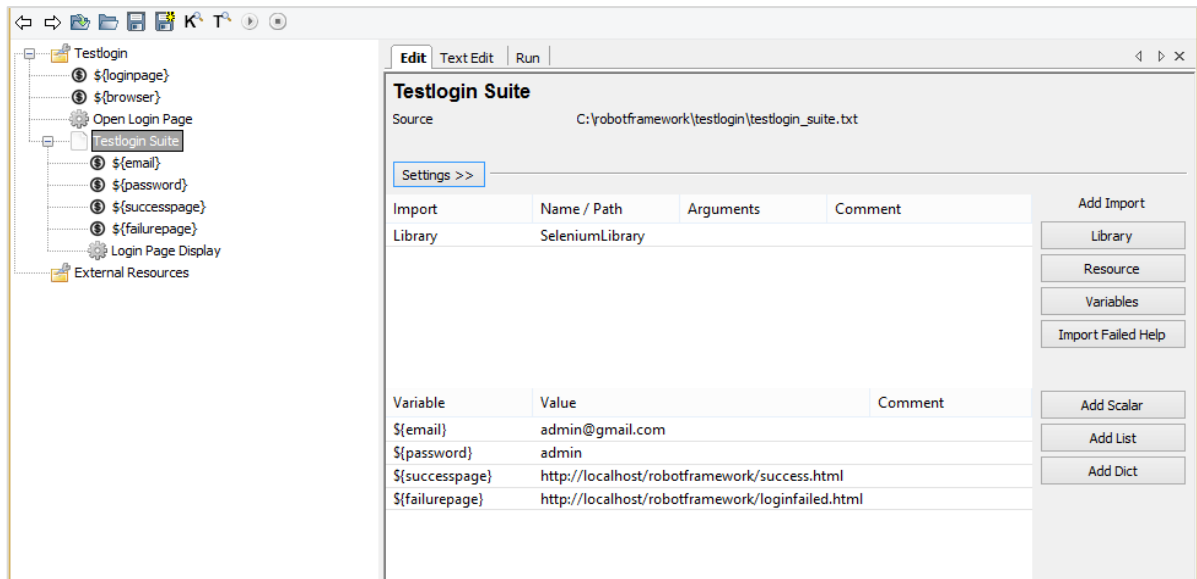
You will be directed to the following screen:



Test Login is again a user-defined keyword. Click OK to save the template.

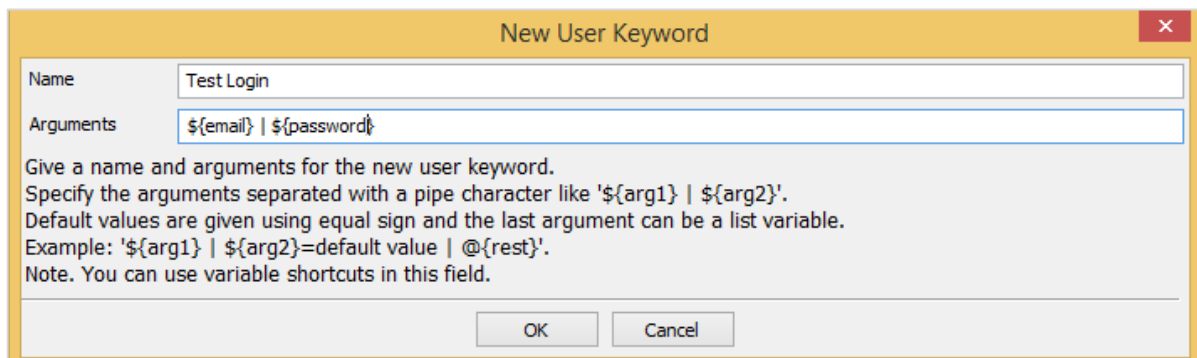
Before we create the Test Login keyword, we need some scalar variables. The scalar variables will have the details of the email-id, password, successpage, failurepage, etc.

We will create scalar variables for test suite as follows:



We have created email, password, successpage and failurepage scalar variables as shown in the above screenshot.

Now, we will create **Test Login** User defined Keyword. Right-click on the test suite and click on New User Keyword.



Click OK to save the keyword.

The following screenshot shows the keywords entered for Test Login:

Test Login

Find Usages

Settings <<

Documentation

Edit

Clear

Tags

<Add New>

Edit

Clear

Arguments

\$(email) | \$(password)

Edit

Clear

Teardown

Edit

Clear

Return Value

Edit

Clear

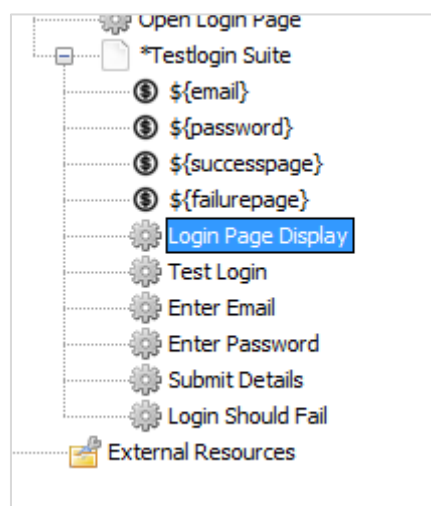
Timeout

Edit

Clear

1	Enter Email	\$(email)		
2	Enter Password	\$(password)		
3	Submit Details			
4	Login Should Fail			
5				

Enter Email, Enter Password, Submit Details and **Login Should Fail** are User Defined Keywords, which are defined as follows:



Enter Email

Enter Email

Find Usages

Settings <<

Documentation

Edit

Clear

Tags

<Add New>

Edit

Clear

Arguments

`\${email}`

Edit

Clear

Teardown

Edit

Clear

Return Value

Edit

Clear

Timeout

Edit

Clear

1	Input Text	email	`\${email}`	
2				
3				
4				
5				
6				

Enter Password

Enter Password

[Find Usages](#)

[Settings <<](#)

[Documentation](#)

[Edit](#)

[Clear](#)

Tags	<Add New>	Edit	Clear
Arguments	<code>\${password}</code>	Edit	Clear
Teardown		Edit	Clear
Return Value		Edit	Clear
Timeout		Edit	Clear

#	Input Text	passwd	\${password}	
1				
2				
3				
4				
5				
6				

Submit Details

Submit Details

Find Usages

Settings <<

Documentation

Edit

Clear

Tags

<Add New>

Edit

Clear

Arguments

Edit

Clear

Teardown

Edit

Clear

Return Value

Edit

Clear

Timeout

Edit

Clear

1	Click Button	btnsubmit		
2				
3				
4				
5				
6				

Login Should Fail

Login Should Fail

Find Usages

Settings <<

Documentation

Edit

Clear

Tags

<Add New>

Edit

Clear

Arguments

Edit

Clear

Teardown

Edit

Clear

Return Value

Edit

Clear

Timeout

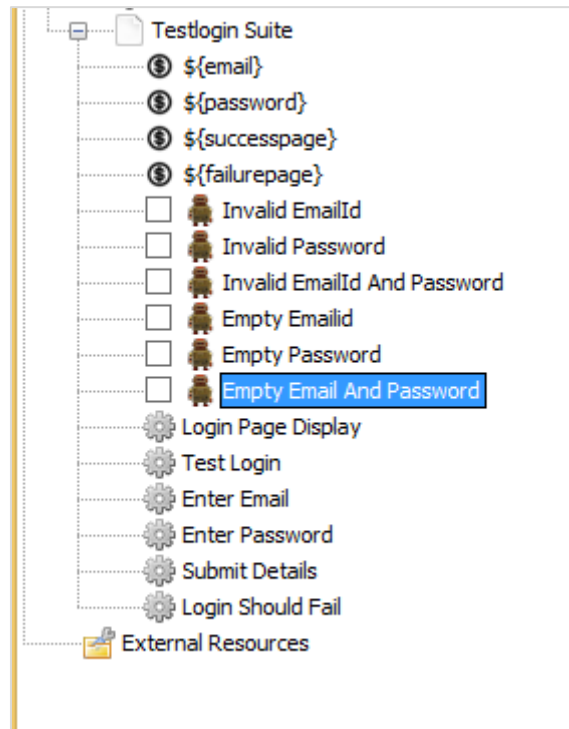
Edit

Clear

1	Location Should Be	`\${failurepage}`		
2	Title Should Be	Login Failed		
3				
4				
5				
6				
7				

Now, we will write test cases, which will take different email id and password details to the template created.

The following is a list of test cases:



Invalid email id Test case

	Email	Password		
1	abcd@gmail.com	\${password}		
2				
3				
4				
5				
6				

The email is passed with values abcd@gmail.com and \${password} is the password stored in the variable.

Invalid Password

	Email	Password		
1	\${email}	xyz		
2				
3				
4				
5				
6				

Invalid Email Id and Password

	Email	Password		
1	invalid	invalid		
2				
3				
4				
5				
6				

Empty Email Id

	Email	Password		
1	\${EMPTY}	\${password}		
2				
3				
4				
5				
6				

Empty Password

	Email	Password		
1	\${email}	\${EMPTY}		
2				
3				
4				
5				
6				

Empty Email and Password

	Email	Password		
1	<code>\${EMPTY}</code>	<code>\${EMPTY}</code>		
2				
3				
4				
5				
6				

Now, we are done with the test cases and can run the same.

Go to the Run tab and click Start to execute the test cases.

The screenshot shows the Robot Framework Run tab interface. The 'Start' button is highlighted. Below the 'Arguments' section, the test results are displayed. The test suite 'Testlogin' passed all 6 tests. The test cases are: 'Invalid EmailId', 'Invalid Password', 'Invalid EmailId And Password', 'Empty EmailId', 'Empty Password', and 'Empty Email And Password'. All tests passed.

```

elapsed time: 0:00:16  pass: 6  fail: 0
command: pybot.bat --argumentfile c:\users\kamat\appdata\local\temp\RIDEsz9smo.d\argfile.txt --listener C:\Python27\lib\site-packa
=====
Testlogin
Testlogin.Testlogin Suite
=====
Invalid EmailId | PASS |
Invalid Password | PASS |
Invalid EmailId And Password | PASS |
Empty EmailId | PASS |
Empty Password | PASS |
Empty Email And Password | PASS |
Testlogin.Testlogin Suite | PASS |
6 critical tests, 6 passed, 0 failed
6 tests total, 6 passed, 0 failed
=====
Testlogin | PASS |
6 critical tests, 6 passed, 0 failed
6 tests total, 6 passed, 0 failed
=====
Output: c:\users\appdata\local\temp\RIDEsz9smo.d\output.xml
Log: c:\users\appdata\local\temp\RIDEsz9smo.d\log.html
Report: c:\users\appdata\local\temp\RIDEsz9smo.d\report.html
test finished 20181027 18:06:17

```

Here are the log messages for the test cases:

```

20181027 18:11:40.353 : INFO : Opening browser 'chrome' to base url
'http://localhost/robotframework/login.html'.
20181027 18:11:45.960 : INFO : Page title is 'Login Page'.
Starting test: Testlogin.Testlogin Suite.Invalid EmailId
20181027 18:11:45.991 : INFO : Opening url
'http://localhost/robotframework/login.html'
20181027 18:11:46.169 : INFO : Page title is 'Login Page'.
20181027 18:11:46.180 : INFO : Typing text 'abcd@gmail.com' into text field
'email'.
20181027 18:11:46.706 : INFO : Typing text 'admin' into text field 'passwd'.

```

```
20181027 18:11:47.075 : INFO : Clicking button 'btnsubmit'.
20181027 18:11:47.565 : INFO : Current location is
'http://localhost/robotframework/loginfailed.html'.
20181027 18:11:47.584 : INFO : Page title is 'Login Failed'.
Ending test: Testlogin.Testlogin Suite.Invalid EmailId

Starting test: Testlogin.Testlogin Suite.Invalid Password
20181027 18:11:47.600 : INFO : Opening url
'http://localhost/robotframework/login.html'
20181027 18:11:47.767 : INFO : Page title is 'Login Page'.
20181027 18:11:47.783 : INFO : Typing text 'admin@gmail.com' into text field
'email'.
20181027 18:11:48.342 : INFO : Typing text 'invalid' into text field 'passwd'.
20181027 18:11:48.701 : INFO : Clicking button 'btnsubmit'.
20181027 18:11:49.035 : INFO : Current location is
'http://localhost/robotframework/loginfailed.html'.
20181027 18:11:49.051 : INFO : Page title is 'Login Failed'.
Ending test: Testlogin.Testlogin Suite.Invalid Password

Starting test: Testlogin.Testlogin Suite.Invalid EmailId And Password
20181027 18:11:49.054 : INFO : Opening url
'http://localhost/robotframework/login.html'
20181027 18:11:49.213 : INFO : Page title is 'Login Page'.
20181027 18:11:49.221 : INFO : Typing text 'invalid' into text field 'email'.
20181027 18:11:49.555 : INFO : Typing text 'invalid' into text field 'passwd'.
20181027 18:11:49.883 : INFO : Clicking button 'btnsubmit'.
20181027 18:11:50.162 : INFO : Current location is
'http://localhost/robotframework/loginfailed.html'.
20181027 18:11:50.176 : INFO : Page title is 'Login Failed'.
Ending test: Testlogin.Testlogin Suite.Invalid EmailId And Password

Starting test: Testlogin.Testlogin Suite.Empty Emailid
20181027 18:11:50.188 : INFO : Opening url
'http://localhost/robotframework/login.html'
20181027 18:11:50.302 : INFO : Page title is 'Login Page'.
20181027 18:11:50.306 : INFO : Typing text '' into text field 'email'.
20181027 18:11:50.486 : INFO : Typing text 'admin' into text field 'passwd'.
20181027 18:11:50.693 : INFO : Clicking button 'btnsubmit'.
```

```
20181027 18:11:50.935 : INFO : Current location is
'http://localhost/robotframework/loginfailed.html'.
20181027 18:11:50.958 : INFO : Page title is 'Login Failed'.
Ending test: Testlogin.Testlogin Suite.Empty Emailid

Starting test: Testlogin.Testlogin Suite.Empty Password
20181027 18:11:50.958 : INFO : Opening url
'http://localhost/robotframework/login.html'
20181027 18:11:51.063 : INFO : Page title is 'Login Page'.
20181027 18:11:51.071 : INFO : Typing text 'admin@gmail.com' into text field
'email'.
20181027 18:11:51.367 : INFO : Typing text '' into text field 'passwd'.
20181027 18:11:51.561 : INFO : Clicking button 'btnsubmit'.
20181027 18:11:51.796 : INFO : Current location is
'http://localhost/robotframework/loginfailed.html'.
20181027 18:11:51.808 : INFO : Page title is 'Login Failed'.
Ending test: Testlogin.Testlogin Suite.Empty Password

Starting test: Testlogin.Testlogin Suite.Empty Email And Password
20181027 18:11:51.811 : INFO : Opening url
'http://localhost/robotframework/login.html'
20181027 18:11:51.908 : INFO : Page title is 'Login Page'.
20181027 18:11:51.916 : INFO : Typing text '' into text field 'email'.
20181027 18:11:52.049 : INFO : Typing text '' into text field 'passwd'.
20181027 18:11:52.193 : INFO : Clicking button 'btnsubmit'.
20181027 18:11:52.419 : INFO : Current location is
'http://localhost/robotframework/loginfailed.html'.
20181027 18:11:52.432 : INFO : Page title is 'Login Failed'.
Ending test: Testlogin.Testlogin Suite.Empty Email And Password
```

Conclusion

We have seen here how to test a login page with different inputs, which will validate if the login is working fine or not. The details of how the execution takes place is given in the log section.