Chapter No. 1
"Getting Started with the Robot Framework"
In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.1 "Getting Started with the Robot Framework"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Sumit Bisht currently works as a Senior Software Engineer at a software service outsourcing firm. He has experience in both project and product-based companies and has done Masters in Computer Applications in addition to self-taught learning as part of his learning process towards making computers work. He has keen interests in leveraging various open source technologies for developing better software, considers himself a polyglot programmer and has experience working with different programming. While not working, he devotes his time in learning new researches and technologies, reviewing technical books on a wide variety of topics, sharing interesting information, and evangelizing open source philosophy through his blog, sumitbisht.blogspot.in.

I’d like to thank Vikram Mohan Sujuanani for introducing me to the Robot Framework, while being my patient mentor, and Jayanti Kumar Srivastava for his support and expertise while unravelling elements of the framework. The entire Packt Publication team has worked tirelessly and persistently towards making this book a reality, and deserve a pat on the back. I am also grateful to Syed Mohd Mohsin Akhtar for reviewing this book. Last but not least, a huge thanks to Pekka Klärck and other members of the open source community for making available this versatile software and painstakingly extending help to various users.

For More Information:
Robot Framework Test Automation

Testing is essential for any activity to succeed; this applies to software as well. It not only covers verification but also the validation of the software. Commonly used testing practices include unit and functional testing that validates the needed functionality. However, manual efforts by testers or stakeholders are required to verify that the software being tested works as desired, as part of the software acceptance process.

As software has grown both in size and amount over the years, the acceptance testing process has become automated, leaving behind vast information in the form of audit trails of tests that need to be made available to the user at a glance and without which the purpose of acceptance test automation is defeated.

The Robot framework provides this missing link that integrates with the testing tool used for performing acceptance testing over the software or the "system under test" and structures this execution into discrete structures of tests arranged in a proper hierarchical manner; this not only results in execution in the desired manner, but also generates orderly reports.

Being an open source framework, this tool does not limit you to simply performing acceptance testing in a hierarchical manner and generating automated reports, but also allows you to customize it and create your own tool or even use it with other tools as part of a custom test environment setup required for your team.

What This Book Covers

Chapter 1, Getting Started with the Robot Framework, explains the need for acceptance test driven development, offers a high-level overview, and gives different methods of installation of the Robot Framework tool. It also discusses the various commands available for the user to interact with the framework and demonstrates simple test project creation and execution to the user.

Chapter 2, Configuring a Test Project, explains the test hierarchies and covers the different files and tables used in the framework as well as different syntax options to write the test files. It explains the commonly used test practices and explains how information can be re-used within the tests.

Chapter 3, Further Test Customization, covers test re-use, connectivity to external data sources and test templates, and user-defined keywords. It also provides different test writing styles and offers comparison with some other famous behavior-driven test tools.

For More Information:
Chapter 4, *Extending the Framework*, demonstrates real-world testing challenges and offers practical ways of using various built-in and external libraries to solve these challenges. It covers both the Web-based and desktop testing through object-oriented as well as image-based tests. In the absence of a compatible testing library, it also demonstrates the creation and integration of a custom written library.

Chapter 5, *Generating Reports*, covers the generated output of the tool in detail and offers different ways to customize the reports as well as the logs generated through the built-in as well as custom mechanisms. It also encourages the user to leverage the source code to create a customized reporting mechanism in the existing framework.

*For More Information:*
Getting Started with the Robot Framework

In this chapter we will cover the following topics:

• Definition and need for acceptance testing
• Introduction and a brief history of the Robot Framework
• Components of the Robot Framework
• Various installation methods
• Different installation environments
• Creation and execution of a sample project
• Brief description of the generated files

This book introduces you to the Robot Framework, which is a Python-based, keyword-driven, and acceptance test automation framework. It is very flexible and easy to use and extend according to your intentions. Built to provide acceptance test regardless of the platform size and scope of the software to be tested, also known as System Under Test (SUT), it is an ideal software tool to structure and manage different tests as part of a comprehensive test suite. Since this tool utilizes and manages the tests as well as deals with the software under test, it is beneficial for both testers and developers alike. Today, with the rise of versatile software tools and greater emphasis on test-driven development, the line between tester and developer has become blurred.
To learn and use this tool effectively, you will have to put yourself in the shoes of both tester as well as a developer. The Python and/or ports into Java and .Net framework such as, Jython and Ironpython are also required.

For More Information:
As with any other physical creation, software is built with an aim of directly or indirectly changing our lives to solve a task or desire of someone. However, programming is an abstract science that is not dependent upon its end-user's desires, commonly referred to as garbage in – garbage out. A difference between the created software and its expected behavior by its user determines its implementation by the user. Thus, the end user must accept the software that is intended to be sold to him. However, the user often does not wants to be involved in the finer details and just needs to get the things done as he envisages with each iteration of the software. In order to perform this, the interactions that the end user does with the software is needed to be verified beforehand, which has led to creation of testing especially designed to perform this process of testing and verification. This process is known as acceptance testing. However, as the software grows, more and more, acceptance tests come up that give rise to a sense of chaos as upon failure of a test, proper context is not easily identified.

As the scale and complexity of software has grown up, so has the need for its quality assurance. Manual tests often are quite easy to set up, but they give diminishing returns of scale and are only feasible up to an extent, where a tester can manually work through different scenarios and identify bugs and errors in time without affecting the delivery schedule of the resultant product.

**The need for acceptance testing**

For tests that are large in size or complexity, a structured approach can help you to pinpoint the errors, which arise while testing for the system is carried out under test's acceptance. Increase in the development speed and efficiency as well as create accountability for various features of the software are also taken into consideration. These benefits can be summarized as follows:

- Pinpoint application failure
- Reduced error rate
- Provide automation and reusability
- Create a test audit trail

**Pinpoint application failure**

Through testing, it is possible for you to identify complete or partial failures as well as identify bottlenecks in performance that might have slipped during development or in other forms of testing.

---

For More Information:
Reducing the error rate
Through automation, the predetermined steps involved to run the program can be performed exactly as desired with no interference as well as no extra or erroneous user interactions. This is different from monkey testing as in acceptance testing; only the happy path scenario is to be dealt with.

Providing automation and re-use
Testers or any other human resources are expensive than computation cycles. So it is best to automate the repetitive tasks, which will also reduce time that is normally spent in typing, clicking, and digesting the user interface as well by the test user. Furthermore, test can be reused or iterated over, which reduces the amount of tests while making sure that the complete acceptance testing remains while you can focus on other problems.

Creating the a test audit trail
By keeping a record of various test results, you can gather interesting facts about acceptance testing such as how much of the system under test is covered under acceptance tests as well as how many failures were reported. This can be useful in changing management as well as re-engineering/modernization of the existing software.

What is the Robot Framework?
The Robot Framework is an open source, general purpose test automation framework used for acceptance testing and streamlines it into mainstream development, giving rise to the concept of acceptance test driven development (ATDD). While commercial and in-house automated testing tools have been used traditionally to provide this kind of test automation. It suffers from the problem of reinventing the wheel and vendor lock-in as well as lack of flexibility to use tests with different software and under different circumstances. It stands out from other tools used for the same purpose by working on easy-to-use tabular test files that provide different approaches towards test creation. As different projects require acceptance testing in various ways, there is a need to make tests flexible, as the Robot Framework is flexible and extensible enough for handling these scenarios.

For More Information:
It is the extensible nature of the tool that makes it so versatile that it can be adjusted into different scenarios and used with different software backend. While it is most popularly used with selenium as a website automation tool, it can also be used with image-based testing software like sikuli and also with software that require remote access over multiple machines while only running the tests over a given machine. All of these can be made easily available through creation of custom libraries, which link up the Robot Framework configuration code keywords with tasks associated with whatever software the Robot Framework is using. On the other hand, the output that the framework produces can also be used in multiple ways, first of which is the HTML report and log file that not only produces a XUnit styled output, but also contains test operations in detail while signifying the execution order and test hierarchy of entire tests. This contrasts with the optional .xml generation that can be useful in further manipulation of the processes. One such example is creation of customized programs that use the information obtained by running tests to create a wide variety of results. Another example is the log files' subsequent use in continuous delivery environments that can allow a build to continue or fail based on all the individual acceptance tests which are in use.

It was created by Pekka Klärck as part of his master's thesis (http://eliga.fi/Thesis-Pekka-Laukkalan.pdf) and was developed within Nokia Siemens Networks in 2005. Its second version has been open sourced under Apache License, Version 2 since 2008 and has an active community of volunteers. It is available at http://code.google.com/p/robotframework.

The Robot Framework ecosystem

The following diagram presents a conceptual, high-level overview of the framework, and offers an insight into various components involved:
This can be explained broadly as follows:

- **Tests & Test Data**: This is the configuration of the tests, the closest to what most testers of the framework will be. It comprises test and data files and folders as well as the contents of those which dictate the test execution.

- **Test Results**: These are the end products of the tests, which are used to determine the results of tests as well as logs that can be used to assess various portions of the test.

- **Robot Framework**: This is the core framework that performs the actual heavy lifting to get things done.

- **Test tool driver**: This provides communication between the framework and the actual tools in place. It can be custom-tailored to meet specific requirements by the testing tool in place.

- **Testing Tool**: This is the actual software that is used to perform acceptance testing.

- **End Application (System under test)**: This is the actual software that is to be tested for usability for its acceptance by the client or the end user.

### Installing and setting up the Robot Framework

The current version of the Robot Framework requires Python 2.7 for setup.

As of now, only pybot script gets created from a Python environment. In case of other environments, only the corresponding execution script gets created. If you happen to have multiple environments, then installation of different scripts is also possible. This differs from the previous versions where on installation, both pybot and jybot scripts were installed.

For custom installation, you will require Python, Jython, or Ironpython pre-installed; and environment `PATH` variable set correctly as the Robot Framework will use the first Python binary available on `PATH` or the exact library if supplied with the python command. You can verify this from the command line:

```
sumit@darkstar066:~$ python --version
Python 2.7.4
sumit@darkstar066:~$ jython --version
Jython 2.5.2
sumit@darkstar066:~$
```

On the project downloads page, there are various options, listed here under the following self-explanatory headings.

---

**For More Information:**

Source installation

To use the source, you can either download the zip containing sources or clone the project using mercurial hg clone: https://code.google.com/p/robotframework.

This will clone the project in the current directory and then you can either straightaway install the project, or make some modifications for customizing the framework.

Now go to the folder where source is checked out/unzipped and perform the following commands based upon the environment present:

```
python setup.py install
```

The preceding command installs the Python based version with pybot and rebot runner scripts.

```
jython setup.py install
```

The preceding command installs the Jython based version, with jybot and jyrebot scripts.

```
ipy setup.py install
```

The preceding command installs the Ironpython based runtime with ipybot and ipyrebot scripts.

If you see this folder, along with standard `setup.py`, there is another file, `install.py`, that can be used to install (it is the similar as installation from `setup.py`), reinstall, or uninstall the framework that can be used as follows:

```
python install.py in   [Installation]
python install.py un   [Uninstallation]
python install.py re   [Re-Installation]
```

To install with Jython or IronPython, replace `python` with `jython` or `ipy` respectively in the command. You may have to use `sudo/run` console as administrator if you run into any authentication errors, depending upon the user privileges.
One-click graphical installer
If you happen to be running Windows XP (32-bit), than you will want to use the one-click installer that installs the Robot Framework as well as Python and optionally, Jython and sets the paths without requiring any intervention. Other graphical installers for windows also exist in 32 and 64 bit versions.

Java environment installation
You can use the standalone jar that contains a bundled Jython as well as the framework. You just need to have Java installed on your system to execute the runnable jar for its installation.

In this method, instead of a command, the jar file is executed:

```
java -jar robotframework.jar run [options] data_sources
```
The Python package installation

The pip install mechanism only requires, you to have the Python and package managers such as easy_install or pip installed on your computer. To install this, you just have to type pip install robotframework or easy_install robotframework from the command prompt and the Python based Robot Framework gets installed.

Note that, for Python to run correctly, you'll need elementtree module as the default one is already broken.

The user can install more than one environment simultaneously on a computer and use the specified commands separately without affecting either of the installations.

Support for different runtimes

The Robot Framework not only works on the native python (CPython/pypy installations), but also supports Java and .NET based runtimes in the form of Jython and ironpython respectively. While there are some features that are exclusive to native Python-or Jython-based Robot Framework installations, most of the functionality is equivalent on all the runtimes. As people might have different OS and application software a stack setup according to their needs or desires, so they can integrate this tool in their existing runtime without requiring a separate runtime.

Based upon the installer environment used, the Robot Framework will create appropriate startup and post-processing scripts:

<table>
<thead>
<tr>
<th>Environment</th>
<th>Startup command</th>
<th>Post-processing command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>pybot</td>
<td>rebot</td>
</tr>
<tr>
<td>Jython</td>
<td>jybot</td>
<td>jyrebot</td>
</tr>
<tr>
<td>Iron Python</td>
<td>ipybot</td>
<td>ipyrebot</td>
</tr>
</tbody>
</table>

In addition to these commands used in starting the execution, the Robot Framework can directly be started through the robot.run module itself if the standard Robot Framework is installed. This can also be used instead of the standard commands as the commands also call the module internally. The module can be called if the Python command in use is the one that has the Robot Framework installed:

```
python -m robot.run
jython .../run.py
ipy -m robot.run
```
This is handy if the Robot Framework is called by some the Python script. Instead of executing the scripts separately, you can call the framework from inside the same program easily.

The post processing command is useful to recreate test executions in the long run. After the test has been executed, you can save the XML file generated as output without saving any other file. To recreate the report and log files again in future, the `rebot` command can be used which takes the XML file as an argument and results in the generation of the log and report files without recalling or running the actual tests again.

## Command details

The `Pybot` command provides the following major options:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-N --name &lt;name&gt;</td>
<td>Sets the name of topmost test suite in the test hierarchy — thereby effectively customizes those areas.</td>
</tr>
<tr>
<td>-D --doc &lt;documentation&gt;</td>
<td>Sets the documentation of the top level test suite.</td>
</tr>
<tr>
<td>-M --metadata [name:value]</td>
<td>Sets the metadata of the top level test suite.</td>
</tr>
<tr>
<td>-G --settag &lt;tagname&gt;</td>
<td>Sets the given tag to all executed test cases.</td>
</tr>
<tr>
<td>-t --test name</td>
<td>Selects the test cases available by performing a pattern match against them.</td>
</tr>
<tr>
<td>-s --suite name</td>
<td>Selects the specified test suite by its name and also allows for the test reports to have the desired name instead of the name picked up by file/folder name.</td>
</tr>
<tr>
<td>-i --include tag</td>
<td>Selects a test for execution on the basis of its tag name.</td>
</tr>
<tr>
<td>-e --exclude tag</td>
<td>Opposite of include tag.</td>
</tr>
<tr>
<td>-R --runfailed output</td>
<td>Selected failed tests of earlier test runs have another goal.</td>
</tr>
<tr>
<td>-c --critical tag</td>
<td>Tests having this tag are considered critical (default for all tests).</td>
</tr>
<tr>
<td>-n --noncritical tag</td>
<td>Tests having this tag are overridden to be of non-critical type.</td>
</tr>
<tr>
<td>-v --variable name:value</td>
<td>Set variables in tests, only scalar variables($[]]) are supported.</td>
</tr>
<tr>
<td>-V --variablefile path</td>
<td>Specify explicitly the file that contains variables.</td>
</tr>
<tr>
<td>-d --output dir</td>
<td>Specify the desired directory where resultant files are placed.</td>
</tr>
</tbody>
</table>

For More Information:

### Options Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-o --output file</code></td>
<td>The generated XML output file.</td>
</tr>
<tr>
<td><code>-l --log file</code></td>
<td>The generated HTML log file.</td>
</tr>
<tr>
<td><code>-r --report file</code></td>
<td>The generated HTML log file.</td>
</tr>
<tr>
<td><code>-x --xunit file</code></td>
<td>xUnit compatible result file (not created by default).</td>
</tr>
<tr>
<td><code>-b --debugging rake</code></td>
<td>Debug file written during the execution(not created by default).</td>
</tr>
<tr>
<td><code>-T --timestampoutputs</code></td>
<td>Adds timestamp and provides a custom title to all output files.</td>
</tr>
<tr>
<td><code>-L --Loglevel</code></td>
<td>Threshold level for logging and logging and test order customization.</td>
</tr>
<tr>
<td><code>-W --monitorwidth</code></td>
<td>Specify the width of monitor output.</td>
</tr>
<tr>
<td><code>-C --monitorcolors</code></td>
<td>Specify whether to use color on console or not.</td>
</tr>
<tr>
<td><code>-K --monitormarkers</code></td>
<td>Specify test success on console for each test that passes.</td>
</tr>
<tr>
<td><code>-P --pythonpath path</code></td>
<td>Additional locations to search test libraries from.</td>
</tr>
<tr>
<td><code>-E --escape what</code></td>
<td>Specify escape characters in console with common representation.</td>
</tr>
<tr>
<td><code>-A --argumentfile path</code></td>
<td>Specify a text file to read more arguments in tests.</td>
</tr>
<tr>
<td><code>-h -? --help</code></td>
<td>Prints detailed help for the command.</td>
</tr>
<tr>
<td><code>--version</code></td>
<td>Prints the version of the installed Robot Framework.</td>
</tr>
</tbody>
</table>

### A small exercise

To demonstrate the Robot Framework, we will create a simple dummy application as follows:

1. Create a folder named `simpleapp`, this will serve as the application root folder.
2. Inside this, create a folder named `testsuites`, this will contain all the test configuration files.
3. Within the `testsuites` folder create a file, `Test_Case_1.txt`, with the following contents:
   ```
   ***Test Cases***
   
   First Test Action  log  this is a basic test
   ```

For More Information:

Note that there are two spaces before and after the log keyword.

4. Now run the project by calling Pybot script and passing the testsuites folder as an argument. You should get something similar to the following screenshot:

```
Sumit@darkstar066:/simpleapp$ pybot testsuites/
Testsuites
Testsuites.Test Case 1
First Test Action | PASS |
Testsuites.Test Case 1 | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
Testsuites
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
Output: /home/sumit/simpleapp/output.xml
Log: /home/sumit/simpleapp/log.html
Report: /home/sumit/simpleapp/report.html
Sumit@darkstar066:/simpleapp$  
```

This confirms that the project has run successfully. You can view the results and log from generated HTML pages and perform future calculations over the data by using the XML file.
As we used a simple log statement, you get a logged message in the out log, as shown in the following screenshot:

The generated XML is also easy-to-read (as shown in the following screenshot), which can then be used by any other external tool as an input parameter—thereby reducing the human intervention between different stages of testing if this process is also automated.

```xml
<robot generated="20130721 22:12:22.2183" generator="Robot trunk 20130614 (Python 2.7.4 on linux2)"
  --suite source="/home/sumit/testfr testsuits" id="s1" name="Testsuits">
  --suite source="/home/sumit/testfr testsuits/1_first_test id="s1-s1" name="First Test"
  --suite source="/home/sumit/testfr testsuits/1_first_test/1__Test Case 1.txt" id="s1-s1-s1" name="Test Case 1">
    --test id="s1-s1-s1-1" name="Case present">
      --kw type="kw" name="BuiltIn. Should Contain">
      --doc>
        Fails if `item1` does not contain `item2` one or more times.
      --/doc>
      --arguments>
        <arg>Hello world</arg>
        <arg>world</arg>
      --/arguments>
    --/kw>
  --/suite>
--/suite>
</suite>
```

For More Information:

Summary

In this chapter, we studied the need for acceptance test-driven development and how can we use the Robot Framework to achieve it. Various methods of installation were discussed over the supported platforms namely Python and its ported equivalents in form of Jython and Ironpython. We then proceed with exploring various commands that are available to perform various tasks as well as their possibilities. Later we created and run a basic test over the Robot Framework that validated the correctness of our setup. Lastly we examined the output files generated to get a look and feel of the framework.

Now that the basics have been covered, it is time for us to learn about the syntax, the different available actions, and different files involved in the Robot Framework project; which will enable us to write tests with various formats and be aware of the various options that are allowable in tests.

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.