AutomationFramework: User Guide

Overview

AutomationFramework is intended to be as independent of its platform and commands as possible while still retaining the usability that is required of a multi-client framework. The main philosophy is to keep as few commands built-in while everything else can be added as plugins. The plugins themselves are easy to write and follow the same structure as will be illustrated in this document.

The AutomationFramework consists of two main components:

- An Automation Server
- The Automation agents running on various machines

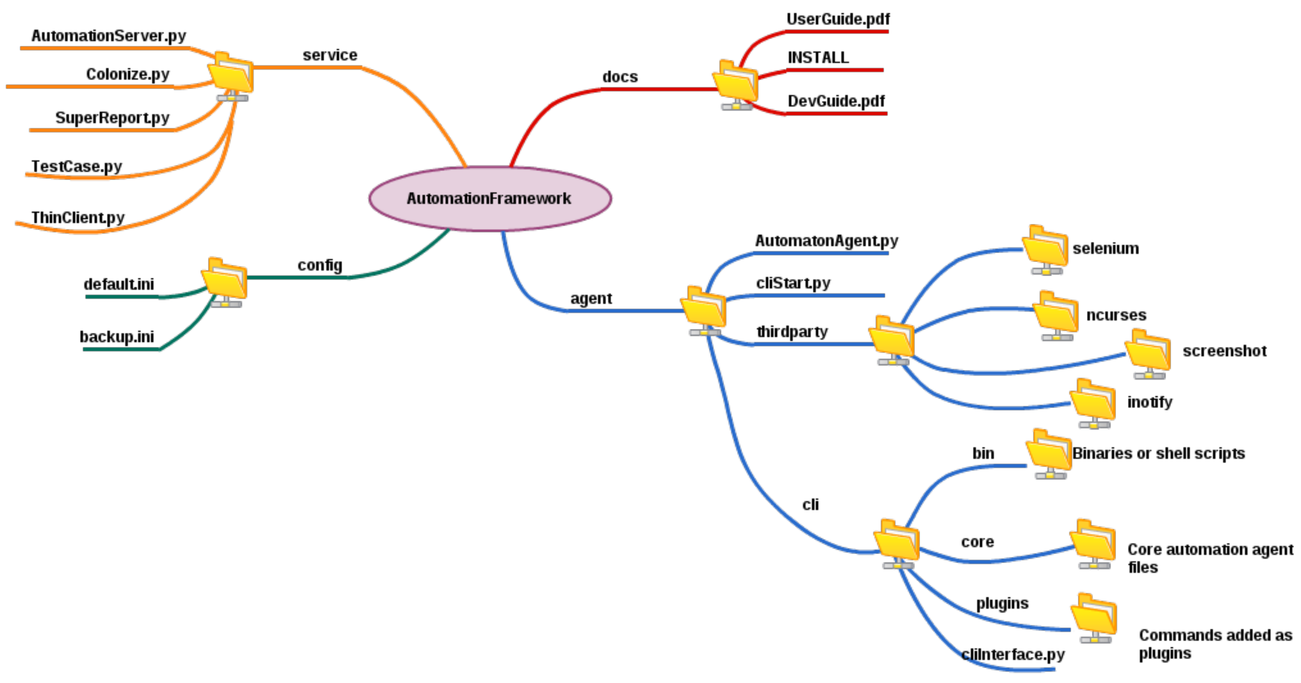
Purpose

The purpose of the AutomationFramework is to provide the test engineers with an easy extensible language, to formulate as well as run repeated tests on linux clients. By giving the power of desigining scenarios and test cases to each test engineer, the system aims to create bug free software. Also this can be seen as a tool to formalize the bug reports with scenarios expressed as a sequence of CLI commands.
Therefore it is possible to have purely blackbox or whitebox or an intermediate kind of testing performed while increasing the productivity as well as the knowledge of the tester.

Background

The current automation frameworks are namesake, with scripts that have much dependency on the developer to add features or code test cases. The number of bugs found from an automated system does not compare favourably with a manual tester and in that measure, they should be considered redundant.

Directory Layout



Copying

As laid out in the Mozilla Public licence https://www.mozilla.org/MPL/2.0/
Note: Thirdparty components are properties of their respective owners and may follow separate licensing scheme

Disclaimer of Warranty

Covered Software is provided under this License on an *as is* basis, without warranty of any kind, either expressed, implied, or statutory, including, without limitation, warranties that the Covered Software is free of defects, merchantable, fit for a particular purpose or non-infringing. The entire risk as to the quality and performance of the Covered Software is with You. Should any Covered Software prove defective in any respect, You (not any Contributor) assume the cost of any necessary servicing, repair, or correction. This disclaimer of warranty constitutes an essential part of this License. No use of any Covered Software is authorized under this License except under this disclaimer.

Limitation of liability

Under no circumstances and under no legal theory, whether tort (including negligence), contract, or otherwise, shall any Contributor, or anyone who distributes Covered Software as permitted above, be liable to You for any direct, indirect, special, incidental, or consequential damages of any character including, without limitation, damages for lost profits, loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses, even if such party shall have been informed of the possibility of such damages. This limitation of liability shall not apply to liability for death or personal injury resulting from such partyâ€™s negligence to the extent applicable law prohibits such limitation. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so this exclusion and limitation may not apply to You.

Installation

Hardware requirements

The software should be able to run on any hardware that has a dependable python port and a reasonable linux distribution. There are no special requirements and the software has been so designed to have minimal dependencies and light weight.

Software Pre-requisites

- Python 2.6 or 2.7
- Dependencies are part of core linux installs and are detailed here

Quickstart

- Uncompress the server.tgz and start server as mentioned here.
- Edit the default.ini in config directory to include some basic pilot test like a Sleep.
- Make sure SSH is enabled on all the client machines, then install client on the machines using the server based installer.
- Check server console to see clients checking in and reports generated.

Congratulations. You have successfully set up the automation server!

Uninstall

To uninstall the AutomationFramework, delete the root of the package and the log files that are created in the /tmp directory. The AutomationFramework leaves no other legacy files or metadata.

Plugins install and update

Plugins come packaged with the AutomationFramework. But since, over time bugs may be uncovered mostly in plugins and to avoid versioning of the AutomationFramework itself, it was thought advisable to have a system wherein plugins can be updated at runtime using a simple HTTP/FTP public repository.
This is due to the fact that the AutomationFramework itself has minimal commands, and therefore the bugs and enhancements will more likely be included in plugins over time. This prevents release of the core engine where nothing is expected to change, instead testers can use existing setups to dynamically update.

To do this, copy the latest versions of all plugins to an HTTP or FTP repository that is *not* password protected. For example:
http://myintranet.repository.local/latest-plugins/
Use the Capability add command to download a plugin, put it as the first test case in all your configuration files

```
[Plugin update]
rank = 1
desc = Update plugins to latest
commands = ["Capability add http://myintranet.repository.local/latest-plugins Browser File IP"]
```

This will install the Browser, File and IP plugins.

Plugins uninstall

It might in some situations be advisable to uninstall all plugins when the test cases have finished execution. This can be achieved by:

```
[Plugin remove]
rank = 100
desc = Remove plugins
commands = ["Capability remove"]
```

Warning:Removes all plugins, there is no facility to remove individual plugins

Dependencies

Component / Library	Version	Comment
libc6	>=2.4	Standard C library
libpng12	>=1.2.13-4	PNG library
libX11-6	compatible version with X running on target system	X11 client library
libxtst-6	compatible version with X running on target system	Standard C library
xautomation	1.09	Xautomation kit X11

Configuration

Test Configuration file

AutomationFramework uses the ini files in the config directory when run as server-client to provision test cases to the client. A typical configuration entry consists of:

- Name - Is the test case name as will be used in the reports.
- Desc - Is the test case description, a small description for more verbosity in the reports.
- rank - This specifies the order of the execution of the test cases. 1 being the first.
- commands - This is a list of commands, that are executed to realize the test case. The syntax is in the form of a python list data structure.

An example test case would be:

```
[Dummy test]
desc = dummy test operations
rank = 10
commands = ["Process alive X", "Process kill X"]
```

Each test case can consist of multiple commands that execute sequentially. But a test case aborts execution if a command fails. So in the above example if Process alive X fails, the next commands that follow it do not execute and the execution moves to the next test case.

Sometimes, like in a server client scenario may sometimes require different clients to get different test cases, there is a facility given to be able to specify a different config file for either the subnet or a particular ip address.
As an example consider that you have 2 subnets 192.168.10.* and 192.168.20.* and you wish to provision 2 different copies of test cases to each. In this case create 2 configuration files:

- 192.168.10.*.ini
- 192.168.20.*.ini

The AutomationFramework will take care of the rest.
Further if you wish, you could create another file 192.168.10.12.ini for specifically a machine matching that ip address.
Exact ip address match has higher priority over the subnet and finally when no match is found, default.ini is used.

Configuration pitfalls and notes

- Listed here are a few things to consider in case your deployment does not behave as expected:
    - The ini file is read everytime a new client checks in. This allows the facility to edit the file and reflect changes without having to restart the automation server
    - The file *default.ini* should always be present. Deleting this file can have undefined consequences.
    - The file *backup.ini* is provided as an example illustration of how different commands can be invoked.
    - Since IP address matching scheme relies on the client ip, the AutomationFramework currently will not work in cases such as NAT
    - More than one test case can have the same rank, in this case the order in which they execute is not defined with respect to each other, though they will still execute after other test cases that have higher rank
    - The test case command data is transfered json encoded. Though there are cases, like iso8859-1 default encoding of MS-Windows text files as opposed to UTF-8 Linux text files that can cause problems. Especially when using different locales and charactersets involving special characters like umlaut.
    - The server is very sensitive to INI file errors and prone to crashing on bad input here.
    - Https is not supported
    - It is best advised to have linux line endings to the config files, this can be achieved using:

      ```
      $ dos2unix config/*.ini
      ```

Default values

- When run as a client of the automation service, the AutomationFramework has a few configurable values, the defaults of which are given below:
    - Default INI file to load test cases for a client is *default.ini*
    - Timeout for a command to finish execution = 1800 seconds
    - Rank of a Test case when not specified = 10
    - Default port that the server listens on = 8080

Note: These values and features are applicable only when running as client. In standalone mode, if a command blocks, it will block forever

Deployment Guide

This section describes the best practices and some useful features that make this a powerful tool to automate day to day tasks. It is recommended to have the plugins on a public server and updated on runtime as explained later to avoid release cycles for the AutomationFramework itself.

How to deploy and use the AutomationFramework?

There is a utility provided to check system compatibility and dependencies. It can be invoked using:

```
$ python -B agent/cli/bin/sanity.py
```

The server can be started using:

```
$ python service/AutomationServer.py
```

To start the client:

```
$ python agent/AutomationAgent.py server <httpurl> [ --debug ]
httpurl: The url where the server is listening
--debug: Enable extra debug logs to be uploaded to the server
```

Alternately, You can use the client installer to start the client from the server using SSH.

How to add a command?

1. Use the simple Echo command as template and make a copy.
2. Rename the copy as you want the command name to be. For example, if the command is named Foobar, then the file is renamed Foobar.py
3. Change the name of the do_Echo function to do_Foobar
4. Use the superb tutorial provided http://docopt.org/ to make your own CLI command.

How to set and use variables?

In general, Variables can be set using the Echo command:

```
[Variable setting example]
desc = Illustrates how to set a
variable
rank = 1
commands = ["${MY_VAR}=Echo foobar"]
```

This example sets the value of MY_VAR to foobar.
Alternately if you have written a plugin that returns an output that you want to save to a variable and use later, then

```
[Variable setting example]
desc = Illustrates how to set a  variable using a custom plugin

rank = 1
commands = ["${MY_VAR}=MyCommand <Myarguments>"]
```

Variables can be used in a command in place of real arguments or appended with strings or numbers.

```
[Variable using example]
desc = Illustrates how to use a  variable

rank = 10
commands = ["Ping ${SERVER_IP}"]
```

What are built-in variables?

```
${SERVER} - Is the address of the Automation server
${IPADDRESS} - Is the ipadress where the agent is running
${HOSTNAME} - Is the hostname of the machine where the agent is running
${CID} - Is the client ID assigned to the machine where the agent is running
${LOOP} - Is the repetition number. Useful to append to a generated file name like a screenshot per iteration
```

How to synchronize the order of test cases on different machines?

The Depends plugin takes care of synchronizing the order of test cases that have dependencies on other machines. For example, before starting a client on Machine A, you may have to wait for server to start on Machine B.
To realize this case, you will create 2 INI files one each for server and client.
The server (192.168.10.1) configuration file (192.168.10.1.ini) will look like this:

```
[Server starter]
desc = Start the foo server  that processes requests
rank = 8
commands = ["Server foo start"]
```

The client configuration file (192.168.10.*.ini) will look like this:

```
[Client starter]
desc = Start the bar client that sends requests to foo server

rank = 1
commands = ["Depends 192.168.10.1 8", "Client bar start"]
```

Which tells the client that the command depends on 192.168.10.1 machine to finish executing a test case that has rank 8.

How to repeat test cases?

The Repeat command can be used to repeat test cases within rank range, N number of times.

```
Usage:
    Repeat <fromrank> <torank> <times>
```

How to skip test cases?

The Skip command can be used to skip test cases within a rank range

```
Usage:
    Skip <fromrank> <torank>
```

How to achieve loops and if-then-else type logic?

As noted here, test cases abort command execution when a command fails and the execution moves to the next test case. This property, in conjunction with repeat - skip commands can be used to achieve if-then-else and for type loops. Consider the following example:

```
[Trick example]
desc = Illustrate if-then-else
rank = 1
commands = ["Process alive foobar", "Skip 2 2"]

[I will be skipped if foobar is dead]
desc = Dummy test to illustrate skip else case
rank = 2
commands = ["Sleep 5", "Skip 3 3", "Repeat 1 1 1"]

[I will execute only if foobar is alive]
desc = Dummy test to illustrate skip if case and loop 1 - 3 forever unless foobar dies

rank = 3
commands = ["Sleep 5", "Skip 3 3", "Repeat 1 3 1"]
```

In the above case, test cases 1 - 3 execute skipping 2, for as long as process *foobar* is alive. Once killed, the loop breaks and test case 2 executes that will explictly skip case 3 thereby breaking the loop.
This feature allows for flexible composition of test cases with no prior programming knowlege required on the part of the testers.

Future enhancements under consideration

1. Just as good programming languages have a try - catch - finally block, where the finally block is used as a cleanup or sanity block to return the program to a usable state, in a similar way it might prove to be useful to have a list of cleanup commands that follow the commands list of a test case to be run irrespective of whether the test case passed or failed.
   These commands, delete any files or registries that this test case modified or kill any spawned processes, thus resetting the system to its vanilla state so that one test case should not overlap or pollute the results of its following test cases. By resetting the entropy of the system to initial, the results are guaranteed to not be independent of the tests that ran before.

The things to be considered are as listed below:

- Should the finally list of commands be run irrespective of the test results?
- Should the finally commands run only if the tests fail?
- Or should there be 2 lists one for successful completion of test case and one in case of a failed run?
- Will the tester appreciate added configuration responsibility or will they prefer rebooting the system as opposed to the overhead of keeping track of entropy?

For more information about Finite State Machine or Automata, click here.

For a more Sci-fi take on this subject click here.

1. X server event record and playback using xnee.

Since GUI test cases building can be tedious, there is an alternate way provided by xnee that uses the same Xtst library as that of xautomation tools and gives the facility of record and playback of X events. Though the project seems to have stopped development, but depending on the demand or need it might seem there is a case for including a genuine record and playback interface to the framework.

- Xnee provides many tools both command line and graphical that allows recording and playback:
  - cnee command line program
  - gnee gui program
  - pnee Gnome panel applet

Interfaces for access

This section describes the Command Line Interface, the stats page and the server client communication.

Text interface

The shell can be started by using:

```
$ python agent/cliStart.py shell
```

There we can use the index number of a command to launch a specific command shell.
"?" in each shell gives the usage.

Commands Shipped

Depends command

```
    Usage:
        Depends <ipaddr> <rank>
        Depends (-h | --help )
```

Mcast command

```
    Usage:
        Mcast server <filepath>
        Mcast client
        Mcast (-h | --help )
```

Ftp command

```
    Usage:
        Ftp upload <server-ip> <user> <password> <localfile> <remotefile>
        Ftp download <server-ip> <user> <password> <localfile> <remotefile>
        Ftp (-h | --help )
```

Echo command

```
    Usage:
        Echo ARGS...
        Echo (-h | --help )
```

Ping command

```
    Usage:
        Ping <host>
        Ping (-h | --help )
```

Process command

```
    Usage:
        Process monitor <name> <duration>
        Process spawn PATH...
        Process exec PATH...
        Process kill <name>
        Process alive <name>
        Process dead <name>
        Process pid <name>
        Process (-h | --help )
```

Browser command

```
    Usage:
        Browser clearHistory
        Browser citrixapp <username> <password> <url> APPNAME...
        Browser citrixdesktop <username> <password> <url> APPNAME...
        Browser kiosk <url>
        Browser (-h | --help )
```

Sleep command

```
    Usage:
        Sleep <sec>
        Sleep (-h | --help )
```

Network command

```
    Usage:
        Network on
        Network off
        Network restart
        Network (-h | --help )
```

IP command

```
    Usage:
        IP equals <interface> <address>
        IP contains <interface> <address>
        IP (-h | --help )
```

File command

```
    Usage:
        File contains <path> <expectation>
        File exists <path>
        File delete <path>
        File create <path>
        File purge <path>
        File wait create <path>
        File wait delete <path>
        File wait write  <path>
        File wait open <path>
        File wait move <path>
        File wait access <path>
```

Dot1x command

```
    Usage:
        Dot1x tls <cacert> <clientcert> <privkey> <pkeypass> <authmode>
        Dot1x peap <cacert> <authmode>
        Dot1x reset
        Dot1x (-h | --help )
```

Desktop command

```
    Usage:
        Desktop logout
        Desktop resolution <resolution>
        Desktop screenshot <filename>
        Desktop windowid <process>
        Desktop windowfocus <windowid>
        Desktop windowclose <windowid>
        Desktop windowimg <filename> <windowid>
        Desktop keydown <key>
        Desktop keyup <key>
        Desktop keypress <key>
        Desktop type KEYS...
        Desktop mouseclick <windowid> <windowimg> PATTERN...
        Desktop (-h | --help )
```

Skip command

```
    Usage:
        Skip <fromrank> <torank>
        Skip (-h | --help )
```

Repeat command

```
    Usage:
        Repeat <fromrank> <torank> <times>
        Repeat (-h | --help )
```

Capability command

```
    Usage:
        Capability add <url> PLUGINS...
        Capability remove
```

Timeout command

```
    Usage:
        Timeout set <nsecs>
```

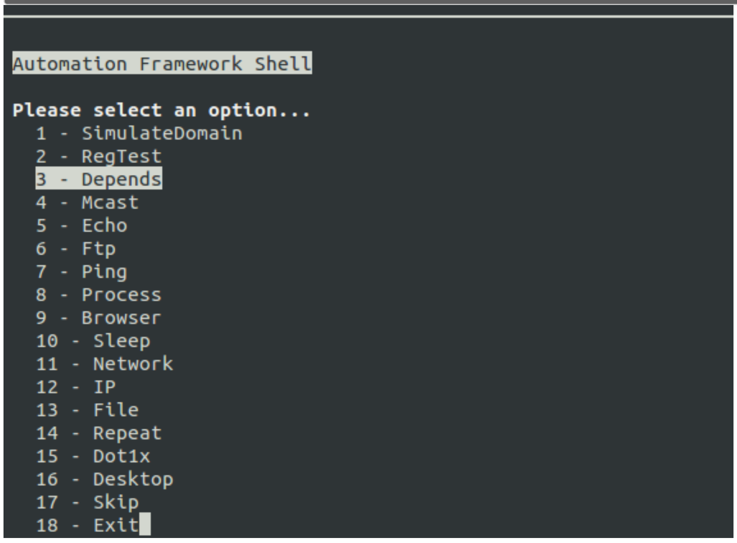
Selenium command

```
    Usage:
        Selenium start     <which>
        Selenium stop
        Selenium clickname <which>
        Selenium clickcss  <which>
        Selenium clickid   <which>
        Selenium open      <which>
        Selenium fillname  <which> WHAT...
        Selenium fillcss   <which> WHAT...
        Selenium fillid    <which> WHAT...
        Selenium (-h | --help | --version)
```

NCurses interface

The ncurses interface can be launched using:

```
$ python agent/cliStart.py ncurses
```



```
Automation Framework Shell

Please select an option...
  1 - SimulateDomain
  2 - RegTest
  3 - Depends
  4 - Mcast
  5 - Echo
  6 - Ftp
  7 - Ping
  8 - Process
  9 - Browser
  10 - Sleep
  11 - Network
  12 - IP
  13 - File
  14 - Repeat
  15 - Dot1x
  16 - Desktop
  17 - Skip
  18 - Exit
```

Use the arrow keys to select and launch a specific command shell.
Warning: The ncurses interface is dependent on ncurses libraries installed on the system and is only provided as an add-on with the same functionality as the agent shell, except for more convenient selection using arrow keys than numbers. In case this does not work properly, revert to the plain shell.

Miscellaneous

Commands run as server-client mode, are saved in /tmp/automation-history.txt It is possible to run the commands from the client machine again, using a utility feature provided:

```
$ python agent/cliStart.py file /tmp/automation-history.log
```

When run in standalone (not as a slave to a server), no logs are generated in the logfile and no history is saved to the history file

Server based client installer

From the very inception, the AutomationFramework has been designed to enable automation everywhere possible, including installation on new machines. When the server is started, it creates a FIFO called /tmp/colonize

The tester can therefore, at any time add a client information that would allow the server to use SSH session to logon to the machine and install the automation agent and launch it with appropriate parameters.
This could also be leveraged via a cron job or automated builds.
This requires some preparation, the client tarball should be uploaded on a FTP or HTTP where it can be downloaded from.

Example: http://myintranet.repository.local/client.tgz

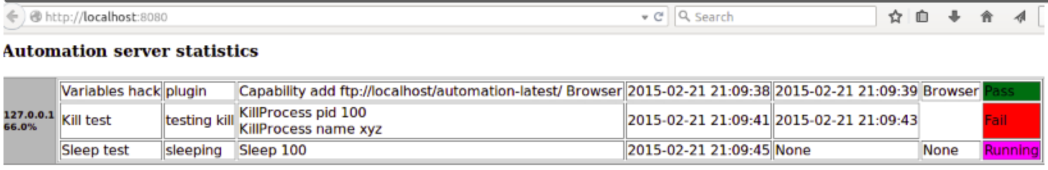
```
    $ echo 192.168.10.51 admin password /opt  url > /tmp/colonize
    192.168.10.51 : New host where the agent will be installed via SSH
    admin: Username for ssh session
    password: Password for the username above
    /opt: The destination directory where the package should be installed.
    url: The url, as explained above, to download the client tarball.
    Note: It has to be in tar format, not zipped


Warning: The SSH installer cannot check for errors, and so install can fail without being reported.
```

Web based statistics

While the server is running it is possible to check a summary of statistics listed per client. From any browser navigate to:

```
    http://<server-ip or hostname>:8080
```



http://localhost:8080

Automation server statistics

127.0.0.1 66.0%	Variables hack	plugin	Capability add ftp://localhost/automation-latest/ Browser	2015-02-21 21:09:38	2015-02-21 21:09:39	Browser	Pass
	Kill test	testing kill	KillProcess pid 100 KillProcess name xyz	2015-02-21 21:09:41	2015-02-21 21:09:43		Fail
	Sleep test	sleeping	Sleep 100	2015-02-21 21:09:45	None	None	Running

Test Results

The tests run by different agents are accumulated by the server at the end of each run and stored in the form of two artifacts, namely:

- Log files that record the run of tests
- HTML report that collects test results and a snapshot of the system

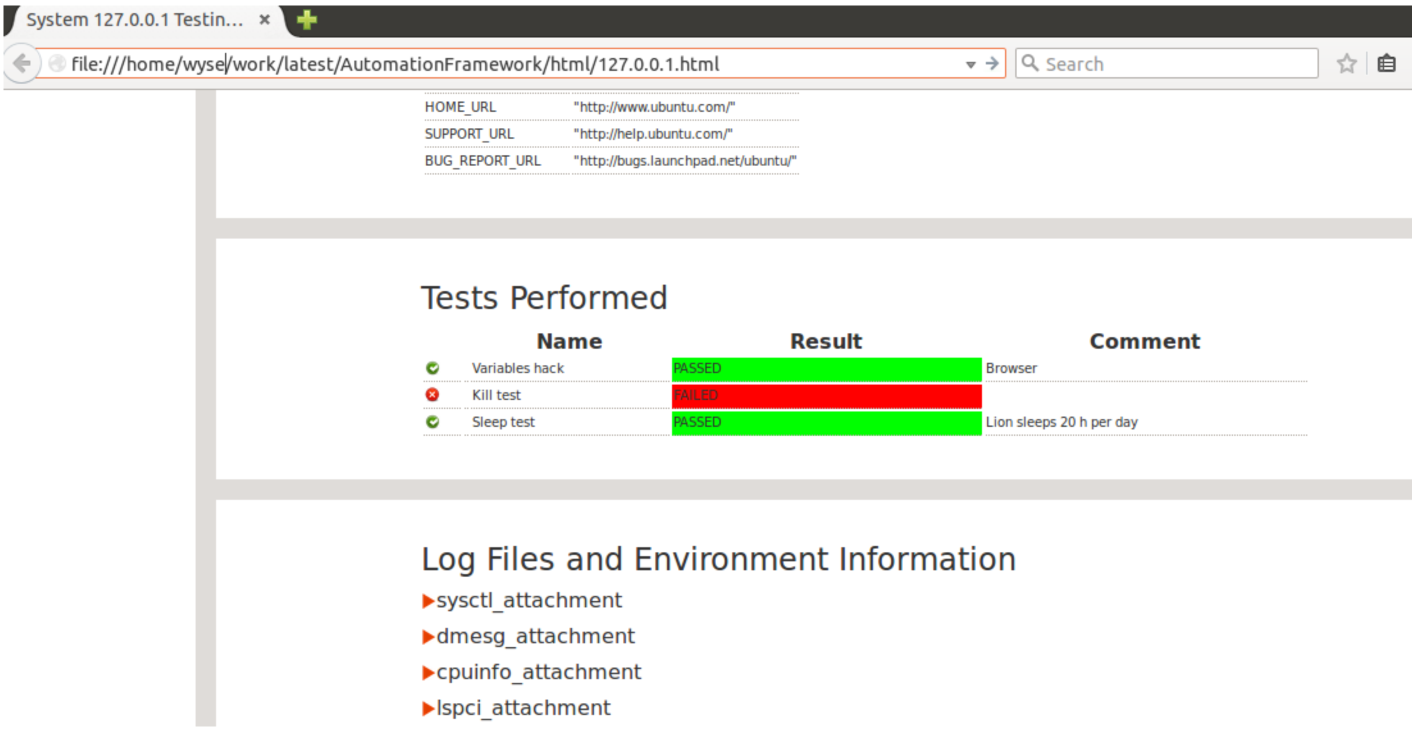
Log files

The logs are stored in the logs folder in the root of the server directory. They are named using the ip-address of the hosts on which agent was running and a timestamp to distinguish between different runs on the same host.

HTML test report

The html reports are stored in the html directory and follow the naming as above. When the agent is run with a --debug flag, the report contains the test results and additionally:

- Hardware Information
  - Desktop Management Interface information
  - sysfs-attributes
  - Processors
- Software Information
  - Packages Installed
- Log Files and Environment Information
  - Sysctl attachment
  - dmesg attachment
  - cpuinfo attachment
  - lspci attachment
  - modules attachment
  - env attachment
  - dmidecode attachment
  - modprobe attachment
  - lsmod attachment
  - lshw attachment
  - registry attachment


System 127.0.0.1 Testin...
file:///home/wyse/work/latest/AutomationFramework/html/127.0.0.1.html
Search

HOME_URL	"http://www.ubuntu.com/"
SUPPORT_URL	"http://help.ubuntu.com/"
BUG_REPORT_URL	"http://bugs.launchpad.net/ubuntu/"

Tests Performed

	Name	Result	Comment
✓	Variables hack	PASSED	Browser
✗	Kill test	FAILED	
✓	Sleep test	PASSED	Lion sleeps 20 h per day

Log Files and Environment Information

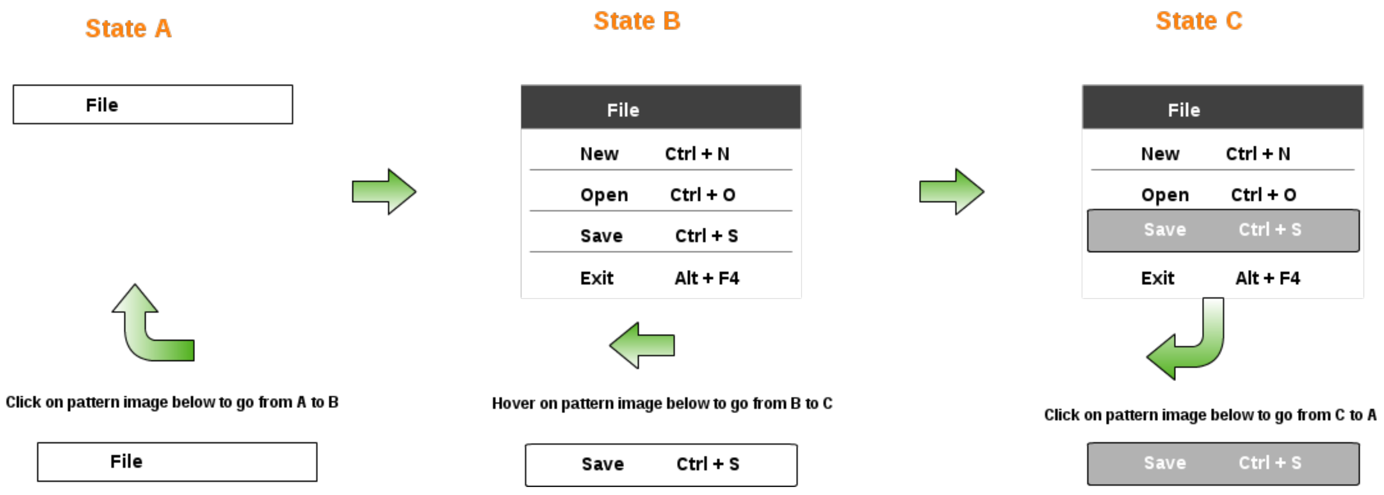
▶sysctl_attachment
▶dmesg_attachment
▶cpuinfo_attachment
▶lspci_attachment

Extras

In this section, we cover examples that may speed up your learning curve to be able to effectively write and execute test cases.

GUI automation

The GUI automation extensively uses X server tools and magic commands like visgrep to make GUI automation as simple as possible. It would be good for you to invest 10 minutes to know how it is done by hand with this video tutorial.
The automation framework uses the same semantics and therefore it is important to start thinking of a GUI application as a state machine.


State A
State B
State C
File
New Ctrl + N
Open Ctrl + O
Save Ctrl + S
Exit Alt + F4
Click on pattern image below to go from A to B
Hover on pattern image below to go from B to C
Click on pattern image below to go from C to A
File
Save Ctrl + S
Save Ctrl + S

- As illustrated in this picture, automation involves:
    - Letting the framework know what to click on by having an image(converted to a suitable pattern format) of the gui component to be manipulated.
    - Making sure the application you want to test is the current active window so that it receives the X events.
    - Clicking a screenshot (save it with a suitable name since it can be reused when the application goes back to the same state to avoid redundant screenshot clicking!) of the active window and using pattern image from first step to find the coordinate to click on.
    - Finding the coordinates of the gui element to click within the screenshot image(the state of the screen should not have changed in the meantime!)
    - Move mouse and click or simply hover or type text.

In the image above, for the sake of example, the picture shows images of 3 gui elements although you could accomplish the same effect with 2 images since a highlighted "Save" menu option in State C has the same coordinates as in State B.
An example test case would be as follows:

```
[Download pattern files via ftp]
rank = 1
desc = download pattern files to /tmp
commands = [
    "Ftp download 127.0.0.1 anonymous anonymous /tmp/FileMenuItem.pat FileMenuItem.pat",
    "Ftp download 127.0.0.1 anonymous anonymous /tmp/SaveMenuItem.pat SaveMenuItem.pat"
   ]

[State A screenshot]
rank = 2
desc = Screenshot of MyApplication in state A
commands = ["${WIN}=Desktop windowid MyApplication","Desktop windowfocus ${WIN}","Desktop windowimg /tmp/StateA.png ${WIN}"]


[Mouse click on File menu]
rank = 3
desc = mouse find and click file menu
commands = ["Desktop mouseclick ${WIN} /tmp/StateA.png /tmp/FileMenuItem.pat"]

[Screenshot window since changed to State B]
rank = 4
desc = Screenshot of MyApplication in state B
commands = ["Sleep 10", "Desktop windowimg /tmp/StateB.png ${WIN}"]

[Mouse click on Save menu]
rank = 5
desc = mouse find and click save menu
commands = ["Desktop mouseclick ${WIN} /tmp/StateB.png /tmp/SaveMenuItem.pat"]
```

As noted above, State C was redundant and was not done but provides a good example of how your GUI test cases can be structured to achieve results in minimum number of steps.

Selenium automation

This module uses the packaged selenium python library to give a command line interface that can be used to simulate and test a website or the browser itself. This module though should be considered a beta.
The commands allow sending either mouse or keyboard events either to elements selected via name or id or css selector.
Here is an example of Google search:

```
    [Selenium tests]
    desc = Selenium browser test
    rank = 1
    commands = ["Selenium start firefox",
        "Selenium open https://www.google.com",
        "Selenium fillname q chaos theory",
        "Selenium clickname btnG","Sleep 10",
        "Selenium stop"]
```

For other options of Desktop and Selenium commands refer to the Shipped commands section

Bibliography

Component / Library	**Author / Contact**	**Comment**
Ncurses Menu	http://blog.skeltonnetworks.com/2010/03/python-curses-custom-menu	Ncurses agent shell menu
PyInotify	Sebastien Martini seb@dbzteam.org	File monitor using inotify
MSS screenshot	Mickael Schoentgen contact@tiger-222.fr	Multi-monitor screenshot
Selenium	http://www.seleniumhq.org/	Browser automation
Docopt	http://www.docopt.org/	CLI argument parsing
Console	http://www.eskimo.com/~jet/python/examples/cmd/ James Thiele	CLI bootstrap
Code snippets	http://stackoverflow.com	Trouble shooting
Test report format	Zygmunt Krynicki from checkbox utility	Test report template
Python samples	http://python.org	Everything!