twister

LUXOFT

Twister User Guide

SUBJECT:

Twister User Guide

 Issue: 0.6

DATE:

July 11, 2014

TWISTER USER GUIDE

Change History

Issue	Date	Description
0.1	30th August 2013	1st version
0.2	6th September 2013	2nd version
0.3	18th November 2013	3rd version
0.4	07th February 2014	Added binding information
0.5	03rd July 2014	Updated API function names after changes for pylint compliance
0.6	11th July 2014	Minor updates for quick start section

TWISTER USER GUIDE

TABLE OF CONTENTS

Contents

TWISTER USER GUIDE

1   Introduction

A test automation framework is a set of assumptions, concepts and tools that provide support for automated software testing.

Twister is a test automation framework that helps user in building functional, regression and load test suites.

2   Definitions, acronyms and abbreviations

APPLET - the GUI of the Twister Framework. It's written in Java and can be accessed in the browser. Its role is to create projects (group suites and files), edit tests, run and stop the test-cases execution, view logs and configure the framework (paths, e-mail, database, test-bed, global parameters, plug-ins).

CE - Central Engine. It's a collection of services that form the Twister server. A lot of functions are exposed via XML-RPC, or RPyc. Its role is to send files and libraries to User EEs, collect statistics and logs from each file executed, send e-mail and save to database after each Project execution and run plug-ins.

CLI - command line interface.

CLIENT MACHINE - this is the machine where the execution engines are installed.

EE/EP - Execution Engine. It's a script that waits the Start signal from CE and sends the logs to CE. When the Start is detected, the EE launches the Runner. If the Stop signal is detected, the EE instantly kills the Runner. One EE belongs to one User and has a unique name for that particular user. One EE cannot be shared between more users;

PROJECT FILE - XML file that contains a set of test files, grouped inside suites. OneProject can run on multiple test-beds, that have defined a number of EEs;

RA - Resource Allocator is a service included in CE. All functions are exposed via XML-RPC, or RPyc. Its role is to manage nodes that represent test-beds and real devices;

REPORTING - the Reporting Server, is used to view the results of the test executions. The reports can be fully customized;

WEB INTERFACE - the Central Engine web interface, is used mostly for debugging. Its role is to view statistics, logs and the connected users. A user can also start and stop the EEs. For more complex operations, the user must use the *applet*;

TWISTER USER GUIDE

RUNNER - Test-case Runner has the following roles:
- connects to CE to receive the libraries that must be executed in this project;
- reads the START/ STOP/ PAUSE/ RESUME status and if it's PAUSE, it waits for RESUME;
- checks for current file dependencies, if there are any, it waits for the dependency to be executed;
- skips the files that have status Skip;
- downloads the files that are not Runnable, without executing them;
- downloads and executes the files that must be executed;
- the files that must be executed can be in many formats, ex: Python, TCL, Perl and Java;
the Runner detects them by extension and starts the appropriate ScriptRunner;

SERVER MACHINE - this is the machine where the central engine is installed.

TWISTER_SERVER_PATH - when running the installer, the user can choose the path to the servers. This path can be anywhere on a Linux server machine. By default, the path is `**/opt/twister**`.

TWISTER_CLIENT_PATH - when running the installer, the client path is always the **$HOME_PATH** of the user running the installer + `**/twister/**`. This cannot be changed.

USER - one client that uses Twister. Each user can define a number of EEs, which are used to run test-cases, simultaneously. The test-cases are grouped into suites and can be saved for later use, into Projects;

USER MACHINE - this is the machine where the user uses a browser to access the framework GUI.

TWISTER USER GUIDE

3   What is Twister?

Twister is an open source test automation framework.

The code can be downloaded from: https://github.com/Luxoft/Twister.

Twister helps building functional, regression and load test suites.

Key features of Twister:
- multi-user architecture, each has groups and roles;
- web based GUI intuitive & user friendly interface;
- easy to manage projects/ suites/ tests;
- real time monitoring of execution;
- distributed execution - each user has its own processes;
- configure TestBeds, devices and group them inside SUTs; SUTs can be tested in parallel;
- flexible reporting mechanism - database schema is not fixed and there is no need to change the framework to fit with a new DB schema;
- automatic sending of e-mails with reports, after the execution is finished;
- support for different scripting languages and for record & play GUI tools;
- support for Continuous Integration, Source Revision Control, Bug tracking;
- support for plugins - specific functionality can be loaded dynamically as plugin;
- OpenFlow 1.0 and 1.3 module available for conformance testing;
- a default set of Libraries written in Python for SSH, Telnet, FTP, Threads and UnitTest;
- each test can have one of the following statuses: pending, pass, fail, skip, abort, not executed, timeout, waiting;
- advanced statistics available for each test, like: the User/ Process/ Suite/ File name, the date and elapsed time, the IP/ host/ OS of the Execution Process or the Server, Python revision, etc;
- multiple logs available, for different levels: log debug, log test, log running;
- pre/ post execution scripts (at the beginning or the end of a project);
- setup/ teardown files (at the beginning or the end of a suite);
- delay between tests; mandatory/ optional scripts;
- properties for each test, that can be accessed while running and can make the same code behave differently depending on the parameter;
- global variables, passed **TO** tests and **BETWEEN** tests;
- panic detect (check for a "panic" word in the CLI logs and kill everything if the "panic" word is found).

TWISTER USER GUIDE

4   Quick start with Twister

Here are some steps about how to start with Twister on an Ubuntu 12.04 linux machine.

The following steps should be done as root.

1. Install SSH and start the service

2. Install python 2.7

3. Install pip

4. Install mysql server and start the service

5. Install Apache and start the service; make sure the document root is set to /var/www

6. Install Oracle Java 1.7

7. Install Java plugin for Firefox

8. Create user "user" on Linux machine with with home directory /home/user and set the shell to /bin/bash

9. Set user password to "password".

10. Add user "user" in the list of sudoers (needed for Twister install)

11. Create database "twister_demo" and table "results" using file twister/installer/packages/twister_demo.sql file:

mysql –u root –p < twister/installer/packages/twister_demo.sql

The following steps should be done as "user".

1. Download Twister from github on your Ubuntu machine.

2. Go in twister/installer directory

3. Install pre-requisites for Twister using "sudo python installer.py" command

4. Install Twister server using "sudo python installer.py" command

5. Install Twister client using "python installer.py" command

6. Make /var/www/twister directory and copy twister/binaries/applet/* into this directory

7. Start Twister server : sudo /opt/twister/bin/start_server

8. Start Twister client : /cd home/user/twister/bin; ./start_client start

9. Use browser to connect to your host http://localhost/twister

10. Login with user/password credentials

5 High level description

Concept
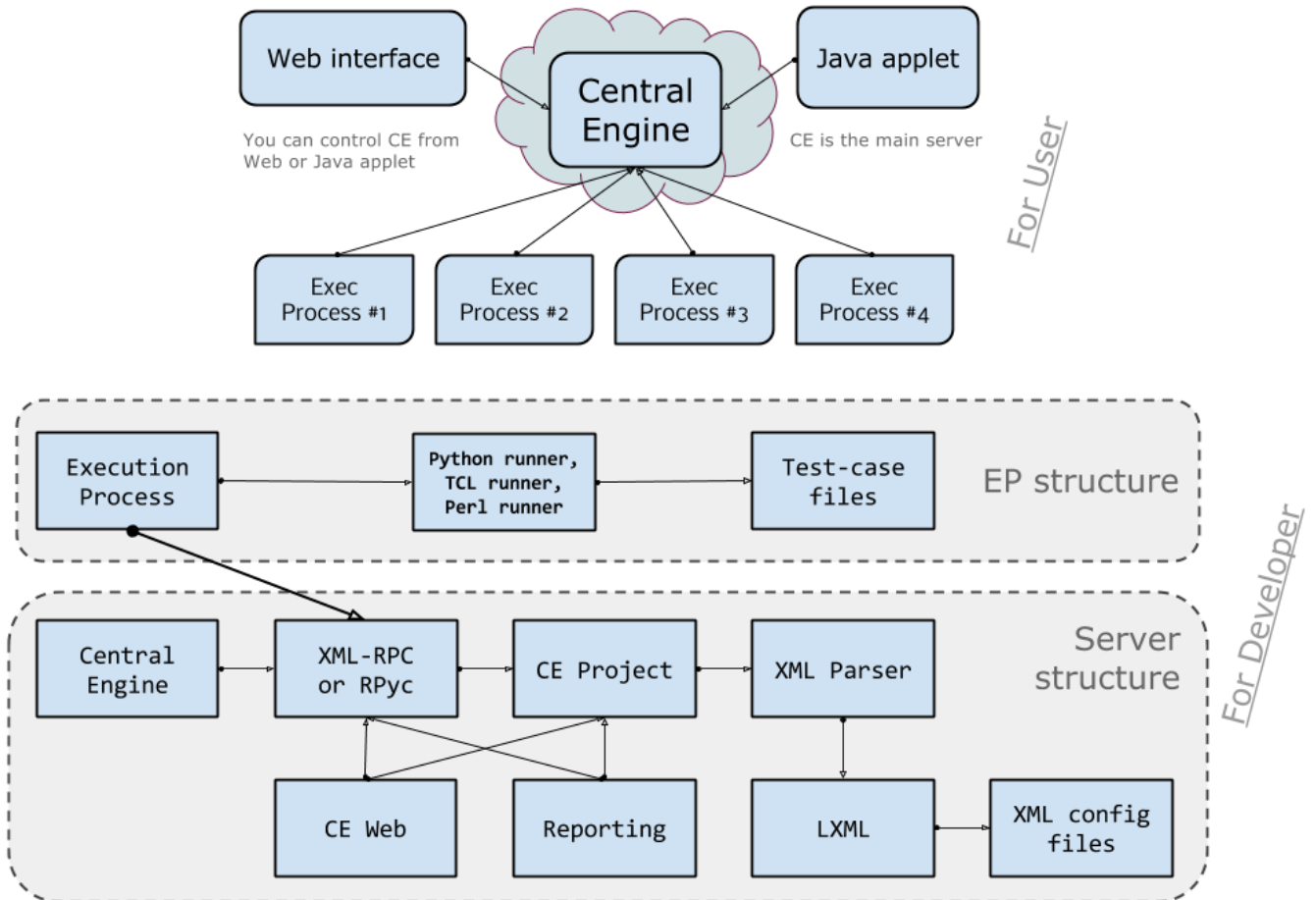
TWISTER USER GUIDE

Twister framework has a client/server architecture. In the same time, Twister has multi-tenant support that allows execution of test cases for multiple users in the same time without any interference between processes. Twister has a component referred as **Central Engine** (CE) that serves and manages the test-cases and libraries for many **users**, each user having one or multiple **EEs**. The users interact with the framework through a Java user interface, or command line.

Aside from the Java applet interface, the Twister framework is developed in Python programming language.

All servers run on top of CherryPy library, which means that the exposed functions can be accessed from a browser, via XML-RPC, or via RPyc (Transparent, Symmetric Distributed Computing).

The picture depicts the high level architecture of the framework and it's interactions with other parties.
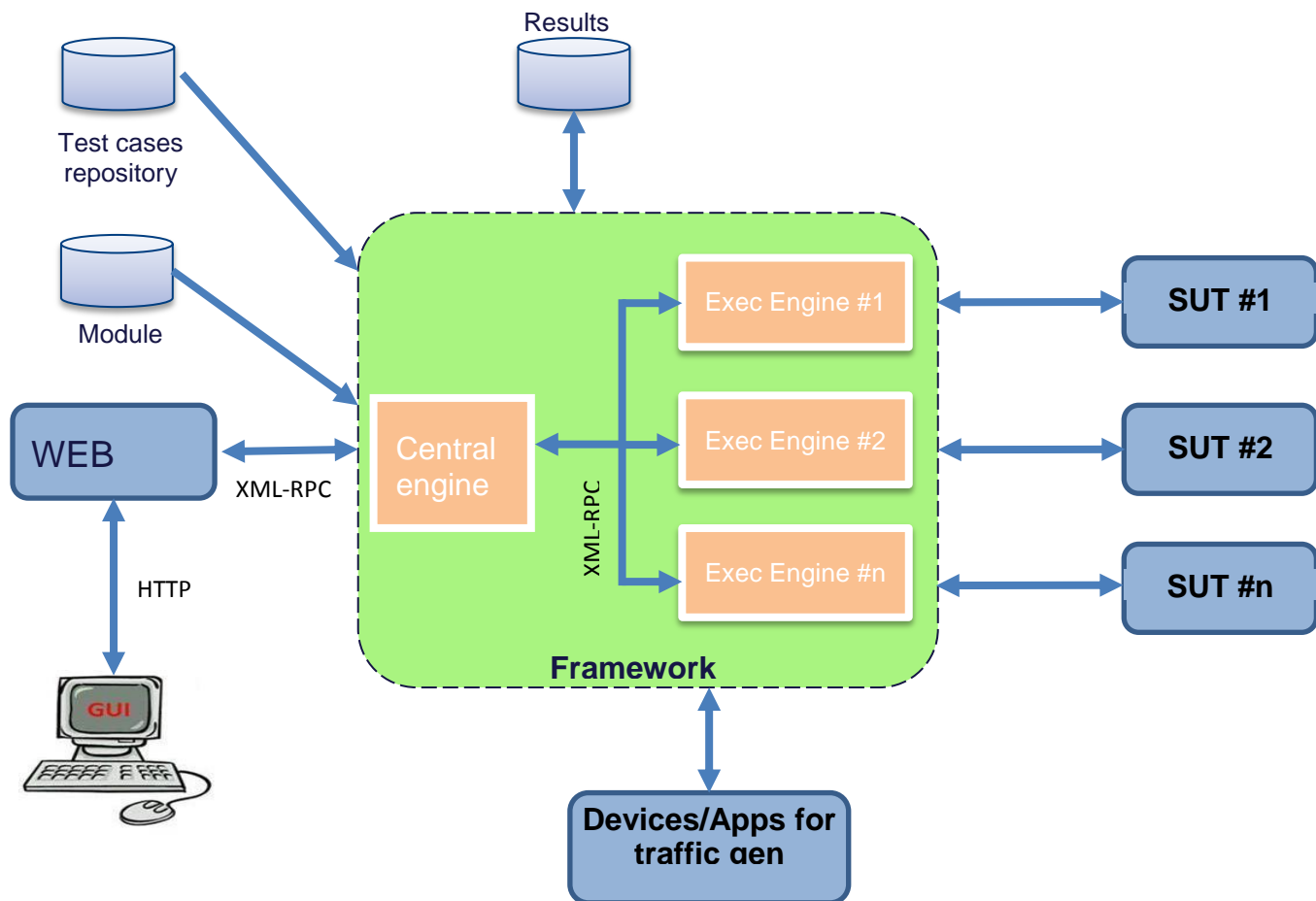

Figure 1 High Level Architecture

The main components of the framework are the graphical user Interface (GUI), the central engine (CE) and the execution engines (EE).

TWISTER USER GUIDE

The GUI represents the interface of the user to the framework. The user uses the interface to configure the framework, to define the details of the systems under test, to define the suite(s) of test cases that are executed against the systems under test, to run the test cases and to view the reports.

The Central Engine represents the core of the framework. It assures the link with the GUI, the execution engines and other parties (e.g. database server). It manages all the information defined in the configuration files, manages the suite of test case, delivers the right test cases to the right execution engines and saves the execution results into the database. The Central Engine ensures the interaction with different modules integrated into framework and makes the link between different plugins and the framework.

The execution engine(s) represents the interface of system(s) under test with the framework. It executes the test cases that it receives from the central engine, collects the output from system under test to send it to the central engine and reports back to the Central Engine the status of every test case that is executed.

TWISTER USER GUIDE

6   How to install/upgrade the framework

Twister hardware requirements are not high and depending on the hardware/server you are installing on the performance and the numbers of the EPs will be different.

Hardware requirements:
- Minimum requirements (1 CE and 1 user with 5 EPs):
    - Single Core CPU
    - 512MB RAM
    - 1GB HDD (at least)
- Recommended requirements (1 CE and 10 users with 50 EPs in total):
    - Dual Core CPU
    - 2GB RAM
    - 10GB HDD (at least to store all of the log reports)

Installing Twister:
In order to install the Twister Framework, a few requirements must be met:

- A **Linux** machine. All the services must run on **Linux** (tested on *Ubuntu* and *OpenSuse*);
- SSH service installed and started on Linux machine.
- Python **2.7**. Python is installed by default, on most Linux systems; the framework is written and tested in Python 2.7;
- Python Tkinter. Required only if you need to run TCL tests. This is included by default in Python, but some Linux distributions don't have the `python-tk` lib, so it has to be installed with:`sudo apt-get install python-tk`;
- TCL Expect libraries. Required only if you need to run TCL tests with Expect. To test the functionality, open a Python 2.7 interpreter, then type:

```
from Tkinter import Tcl
t = Tcl()
t.eval('package require Expect')
# If this fails, you must install Expect from your package manager, or compile it from sources
# The sources are at: sf.net/projects/expect; download, extract, ./configure, sudo make install
exit()
```

- Perl Inline Python. This is required only if you need to run Perl scripts.

The Twister repository is located at: https://github.com/luxoft/twister.

The installer is located in the folder `installer` and it's also written in Python. It works only in **Linux**.

It has 3 options that user can select:
- install dependencies
- install Twister server (central engine)
- install Twister client

TWISTER USER GUIDE

When installing the dependencies, the script must be executed as **ROOT** or as a user with root privileges. In this stage, all the dependencies (see chapter 3 for a list) are installed on the machine. At this stage, it is STRONGLY RECOMMENDED to have an internet connection to allow the setup of all the dependencies; otherwise, you have to install the dependencies manually.

You might need to configure the **proxy** to access the internet. In this case, edit the file `installer.py`, locate the line with **HTTP_PROXY** and type: **HTTP_PROXY** = 'http://UserName:PassWord@http-proxy:3128'. If the username and password are not required for your proxy, you can omit them.

When installing the server, it is recommended to run as **ROOT**, but is not mandatory.

In order to serve the Java applet, you will also need *Apache*, or *Lighttpd* server and *Open-SSH* server.

The recommended command for starting the installer:

```
sudo -E python2.7 installer.py
```

Some tests and libraries will require additional dependencies! Example of dependencies: `paramiko`, `pExpect`, `RpcLib`, `Suds`, `Requests`, or `Gevent`. If you need to run these tests, or libraries, you can install `pip` (tool for installing and managing Python packages - www.pip-installer.org) and use `sudo pip install <package>`.

The installer will guide you through all the steps:

1. Select what you wish to install (*dependencies*, *client*, or *server*);
2. If the `twister` folder is already present, you are asked to back up your data in order to continue, because everything is DELETED, except for the `config` folder. The backup has to be done manually.

Twister Client will be installed in the home of your user, in the folder `twister`; this folder **cannot be changed**. The server will be installed by default in `/opt/twister`, but it can be changed.

NOTE: The Twister framework cannot be installed on Windows OS, however, one or more EPs can run on Windows, with full functionality, in `portable mode`, without installing the framework.
For more details, check the `**EPs on Windows**` section.

Following is an example of a fresh install.

Installing the server:

```
root@Ubuntu13:/home/atoma/twister_rel2/installer# python2.7 installer.py


Please select what you wish to install:
[1] the Twister dependencies (must be ROOT)
[2] the Twister clients
[3] the Twister servers
[q] e[x]it, don't install anything
Your choice: 3
```

```
Will install servers.

Welcome to the Twister Installer !

Please type where you wish to install the servers.
Don't forget to add `twister` at the end of the path!
Leave EMPTY to install in default path `/opt/twister`:
Path :
Warning! Cannot delete Twister dir `/opt/twister/` !
Created folder `/opt/twister/`.

Created folder structure `/opt/twister/bin`.
Copied file `/home/atoma/twister_rel2/bin/cli.py` to `/opt/twister/bin`.
Copied file `/home/atoma/twister_rel2/bin/start_server` to `/opt/twister/bin`.
Created folder structure `/opt/twister/doc`.
Copied dir `/home/atoma/twister_rel2/doc/` to `/opt/twister/doc`.
Created folder structure `/opt/twister/server`.
Copied dir `/home/atoma/twister_rel2/server/` to `/opt/twister/server`.
Created folder structure `/opt/twister/common`.
Copied dir `/home/atoma/twister_rel2/common/` to `/opt/twister/common`.
Created folder structure `/opt/twister/lib`.
Copied dir `/home/atoma/twister_rel2/lib/` to `/opt/twister/lib`.
Created folder structure `/opt/twister/config`.
Copied file `/home/atoma/twister_rel2/config/resources.json` to `/opt/twister/config`.
Copied file `/home/atoma/twister_rel2/config/services.ini` to `/opt/twister/config`.
Copied file `/home/atoma/twister_rel2/config/users_and_groups.ini` to `/opt/twister/config`.
Copied file `/home/atoma/twister_rel2/config/server_init.ini` to `/opt/twister/config`.
Created folder structure `/opt/twister/plugins`.
Copied dir `/home/atoma/twister_rel2/plugins/` to `/opt/twister/plugins`.
Created folder structure `/opt/twister/services`.
Copied dir `/home/atoma/twister_rel2/services/` to `/opt/twister/services`.

Twister installation done!

root@Ubuntu13:/home/atoma/twister_rel2/installer#
```

Installing the client:

```
atoma@Ubuntu13:~/twister_rel2/installer$ python2.7 installer.py

Please select what you wish to install:
[1] the Twister dependencies (must be ROOT)
[2] the Twister clients
[3] the Twister servers
[q] e[x]it, don't install anything
Your choice: 2
Will install clients.

Hello `atoma` !
```

TWISTER USER GUIDE

```
Created folder structure `/home/atoma/twister/bin`.
Copied file `/home/atoma/twister_rel2/bin/cli.py` to `/home/atoma/twister/bin`.
Copied file `/home/atoma/twister_rel2/bin/start_client` to `/home/atoma/twister/bin`.
Copied file `/home/atoma/twister_rel2/bin/start_client.py` to `/home/atoma/twister/bin`.
Copied        file         `/home/atoma/twister_rel2/bin/start_packet_sniffer.py`        to
`/home/atoma/twister/bin`.
Created folder structure `/home/atoma/twister/doc`.
Copied dir `/home/atoma/twister_rel2/doc/` to `/home/atoma/twister/doc`.
Created folder structure `/home/atoma/twister/demo`.
Copied dir `/home/atoma/twister_rel2/demo/` to `/home/atoma/twister/demo`.
Created folder structure `/home/atoma/twister/config`.
Copied dir `/home/atoma/twister_rel2/config/` to `/home/atoma/twister/config`.
Created folder structure `/home/atoma/twister/client`.
Copied dir `/home/atoma/twister_rel2/client/` to `/home/atoma/twister/client`.
Created folder structure `/home/atoma/twister/services/PacketSniffer`.
Copied        dir         `/home/atoma/twister_rel2/services/PacketSniffer/`        to
`/home/atoma/twister/services/PacketSniffer`.
Copied file `/home/atoma/twister_rel2/services/__init__.py` to `/home/atoma/twister/services`.
Created folder structure `/home/atoma/twister/common`.
Copied file `/home/atoma/twister_rel2/common/__init__.py` to `/home/atoma/twister/common`.
Copied file `/home/atoma/twister_rel2/common/constants.py` to `/home/atoma/twister/common`.
Copied file `/home/atoma/twister_rel2/common/suitesmanager.py` to `/home/atoma/twister/common`.
Copied file `/home/atoma/twister_rel2/common/configobj.py` to `/home/atoma/twister/common`.
Created folder structure `/home/atoma/twister/common/jython`.
Copied dir `/home/atoma/twister_rel2/common/jython/` to `/home/atoma/twister/common/jython`.

Twister installation done!

atoma@Ubuntu13:~/twister_rel2/installer$
```

Upgrading Twister

Before upgrading Twister you must backup your tests (on the client side) in case your tests are located in your home Twister folder and on the server side you must backup your libraries and your plugins (in case you implemented custom libraries or plugins).

The config folders are saved automatically and preserved both on the client and server side.
Note that sometimes because of the new features added some of your old config files might be incompatible with the new version so you will need to remake your configuration.

Example of upgrading

Following is an example of upgrading the server and client.

TWISTER USER GUIDE

For server:

```
root@Ubuntu13:/home/atoma/twister_rel2/installer# python2.7 installer.py

Please select what you wish to install:
[1] the Twister dependencies (must be ROOT)
[2] the Twister clients
[3] the Twister servers
[q] e[x]it, don't install anything
Your choice: 3
Will install servers.

Welcome to the Twister Installer !

Please type where you wish to install the servers.
Don't forget to add `twister` at the end of the path!
Leave EMPTY to install in default path `/opt/twister`:
Path :

WARNING! Another version of Twister is installed at `/opt/twister/`!
If you continue, all files from that folder will be PERMANENTLY DELETED!!
If you created custom libs (in lib/ folder) and plugins (in plugin/ folder),
you should make a back-up, then restart the installer.
Are you sure you want to continue? (yes/no): yes

Back-up `config` folder (from `/opt/twister/config` to `/tmp/twister_server_config`)...
Removed folder `/opt/twister/`.
Created folder `/opt/twister/`.

Created folder structure `/opt/twister/bin`.
Copied file `/home/atoma/twister_rel2/bin/cli.py` to `/opt/twister/bin`.
Copied file `/home/atoma/twister_rel2/bin/start_server` to `/opt/twister/bin`.
Created folder structure `/opt/twister/doc`.
Copied dir `/home/atoma/twister_rel2/doc/` to `/opt/twister/doc`.
Created folder structure `/opt/twister/server`.
Copied dir `/home/atoma/twister_rel2/server/` to `/opt/twister/server`.
Created folder structure `/opt/twister/common`.
Copied dir `/home/atoma/twister_rel2/common/` to `/opt/twister/common`.
Created folder structure `/opt/twister/lib`.
Copied dir `/home/atoma/twister_rel2/lib/` to `/opt/twister/lib`.
Created folder structure `/opt/twister/config`.
Copied file `/home/atoma/twister_rel2/config/resources.json` to `/opt/twister/config`.
Copied file `/home/atoma/twister_rel2/config/services.ini` to `/opt/twister/config`.
Copied file `/home/atoma/twister_rel2/config/users_and_groups.ini` to `/opt/twister/config`.
Copied file `/home/atoma/twister_rel2/config/server_init.ini` to `/opt/twister/config`.
Created folder structure `/opt/twister/plugins`.
Copied dir `/home/atoma/twister_rel2/plugins/` to `/opt/twister/plugins`.
Created folder structure `/opt/twister/services`.
Copied dir `/home/atoma/twister_rel2/services/` to `/opt/twister/services`.

Moving `config` folder back (from `/tmp/twister_server_config` to `/opt/twister/config`)...
```

TWISTER USER GUIDE

```
Twister installation done!

root@Ubuntu13:/home/atoma/twister_rel2/installer#
```

For client:

```
atoma@Ubuntu13:~/twister_rel2/installer$ python2.7 installer.py

Please select what you wish to install:
[1] the Twister dependencies (must be ROOT)
[2] the Twister clients
[3] the Twister servers
[q] e[x]it, don't install anything
Your choice: 2
Will install clients.

Hello `atoma` !

WARNING! Another version of Twister is installed at `/home/atoma/twister/`!
If you continue, all files from that folder will be PERMANENTLY DELETED,
Only the `config` folder will be saved!
Are you sure you want to continue? (yes/no): yes

Back-up `config` folder (from `/home/atoma/twister/config` to
`/home/atoma/twister_rel2/installer`)...

Created folder structure `/home/atoma/twister/bin`.
Copied file `/home/atoma/twister_rel2/bin/cli.py` to `/home/atoma/twister/bin`.
Copied file `/home/atoma/twister_rel2/bin/start_client` to `/home/atoma/twister/bin`.
Copied file `/home/atoma/twister_rel2/bin/start_client.py` to `/home/atoma/twister/bin`.
Copied file `/home/atoma/twister_rel2/bin/start_packet_sniffer.py` to
`/home/atoma/twister/bin`.
Created folder structure `/home/atoma/twister/doc`.
Copied dir `/home/atoma/twister_rel2/doc/` to `/home/atoma/twister/doc`.
Created folder structure `/home/atoma/twister/demo`.
Copied dir `/home/atoma/twister_rel2/demo/` to `/home/atoma/twister/demo`.
Created folder structure `/home/atoma/twister/config`.
Copied dir `/home/atoma/twister_rel2/config/` to `/home/atoma/twister/config`.
Created folder structure `/home/atoma/twister/client`.
Copied dir `/home/atoma/twister_rel2/client/` to `/home/atoma/twister/client`.
Created folder structure `/home/atoma/twister/services/PacketSniffer`.
Copied          dir          `/home/atoma/twister_rel2/services/PacketSniffer/`          to
`/home/atoma/twister/services/PacketSniffer`.
Copied file `/home/atoma/twister_rel2/services/__init__.py` to `/home/atoma/twister/services`.
Created folder structure `/home/atoma/twister/common`.
Copied file `/home/atoma/twister_rel2/common/__init__.py` to `/home/atoma/twister/common`.
Copied file `/home/atoma/twister_rel2/common/constants.py` to `/home/atoma/twister/common`.
Copied file `/home/atoma/twister_rel2/common/suitesmanager.py` to `/home/atoma/twister/common`.
Copied file `/home/atoma/twister_rel2/common/configobj.py` to `/home/atoma/twister/common`.
```

```
Created folder structure `/home/atoma/twister/common/jython`.
Copied dir `/home/atoma/twister_rel2/common/jython/` to `/home/atoma/twister/common/jython`.

Moving `config` folder back (from `/home/atoma/twister_rel2/installer/config` to
`/home/atoma/twister/config`)...

Twister installation done!

atoma@Ubuntu13:~/twister_rel2/installer$
```

6.1   Dependencies list

The dependencies will be installed automatically when you first install Twister. If an internet connection is not available, the installer will use the packages from `installer/packages` folder.

- **LXML**: (*www.lxml.de/*)
    – XML and HTML documents parser;
    – LXML is <u>included in Ubuntu</u> by default. The other Linux distributions must install it;
    – Dependencies: python-dev, libxslt-dev and libxml2-dev;

- **MySQL-python**: (*mysql-python.sourceforge.net/*)
    – Connects to MySQL databases. It is only used by the Central Engine;
    – MySQL-python requires the *python2.7-dev* headers in order to compile;
    – Dependencies: mysql server, python-dev, libmysqlclient-dev;

- **CherryPy**: (*www.cherrypy.org/*)
    – High performance, minimalist Python web framework;
    – CherryPy is used to serve the Central Engine, Resource Allocator and Reports;

- **RPyc**: (*rpyc.readthedocs.org/*)
    – Remote procedure calls, clustering and distributed-computing;
    – RPyc is used for communication between Central Engine and the EPs;

- **Mako**: (*www.makotemplates.org/*)
    – Hyperfast and lightweight templating for the Python platform;
    – Mako is used for templating the Central Engine REST and Report pages;

- **PyCrypto**: (*dlitz.net/software/pycrypto/*)
    – The Python Cryptography Toolkit;
    – PyCrypto is used to encrypt and decrypt sensitive data, for example user passwords;

- **Paramiko**: (*github.com/paramiko/paramiko/*)
    – Native Python SSHv2 protocol library;
    – Paramiko is used by the Central Engine to check the user and by the Twister SSH Lib to connect to remote machines;

- **Plumbum**: (*pypi.python.org/pypi/plumbum*)
  - shell combinators library
  - provides local and remote command execution (over SSH), local and remote file-system paths, easy working-directory and environment manipulation, and a programmatic Command-Line Interface (CLI) application toolkit

TWISTER USER GUIDE

~ Optional ~

- Scapy: (*pypi.python.org/pypi/scapy-real/*)
    - Interactive packet manipulation tool;
    - Scapy, is used by the Sniffer plug-in to capture packets and send them to the applet;

- *pExpect*: (*sourceforge.net/projects/pexpect/*)
    - Spawn child applications, control them, respond to expected patterns in their output;
    - pExpect is *optional*; it is used by some Python test cases to connect to FTP/ Telnet;

- *Requests*: (*http://docs.python-requests.org/*)
    - Elegant and simple HTTP library for Python, built for human beings;
    - Requests is *optional*; it is used by some Python test cases to connect to HTTP servers;

- *Gevent*: (*http://www.gevent.org/*)
    - Co-routine-based Python networking library that provides a high-level synchronous API;
    - Gevent is *optional*; it is used by some Python test cases to create sockets and threads;

- *RpcLib*: (*https://github.com/arskom/rpclib/*)
    - Create web services in Python (soap, rpc, rest servers);
    - RpcLib is *optional*; it is used by some Python test cases;

- *Suds*: (*https://fedorahosted.org/suds/*)
    - Lightweight SOAP python client for consuming Web Services;
    - Suds is *optional*; it is used by some Python test cases;

TWISTER USER GUIDE

7   Twister services

Twister framework has 2 services:

1. The **Central Engine** = central server for script and library files. It includes the Resource Allocator, Service Manager and Reporting Server. This must run as **ROOT** in order to have read/ write access to all user files.
2. The **Execution Process Manager** = client service that manages the EPs that have to be started for execution of test cases. This can be run as normal user, but for some functionality (packet sniffer) it has to be run as **ROOT.**

The executable script for central engine is located in `/opt/twister/bin/start_server`. The script doesn't require an input parameter, it has to be executed as is and it will launch the server in the background.

The executable script for execution process(s) manager is located in `/$USER_HOME/twister/bin/start_client`.

If your default Python executable is NOT Python2.7, you must edit the *start_server* and *start_client* files manually, locate the line: `PYTHON_PATH=/usr/bin/python` and replace the value with the path to Python2.7, for example `/usr/local/bin/python2.7`.

This script can take the following parameters:
-   start – to start the service;
-   start silent – start the service and no messages are printed
-   stop – to stop the service
-   restart – to restart the service
-   status – to display the status of the service and what EPs are running

The execution process manager service must be configured before run. You have to edit the file `epname.ini` from `twister/config/`folder; it contains the **name** of the available EPs, the **IP** and the **port** of the CE instance that it will run on. For every EP, there is an optional tag EP_HOST that can be set by the user to restrict the machine where that EP can be started. By default, if this tag is not set, the EP can be started. Otherwise, the comparison between the EP_HOST and the local machine is done and if there is a match, the EP is allowed to be started.

When the **start_client** script with *start* option is executed, a client service is started. This service is used to manage the start and stop of available execution processes on demand. The list of available EPs is obtained from the `epname.ini` file.

The start_client script reads this file and registers all the available EPs to the Central Engine, so the user is able to select EPs from GUI when execution is needed. When testing is started, the CE send the list of selected EPs to the EP manager and this one starts on demand all the EPs requested by the user. When execution of the test cases is completed, the EPs are stopped automatically by the EP manager.

The *start* option can be used in conjunction with *silent* to stop printing messages in terminal.

To stop the EP manager, the *stop* option must be used.

To restart the EP manager, the restart option must be used. Restart of the EP manager must occur when the list of available EPs is changed in the `epname.ini` file.

The status of the client and the list of started EPs are obtained using status option. The started EPs are listed only when the testing is in progress.

If needed, the EP manager can be started automatically at system boot, by adding it into the .rc files. Given the EP manager is user specific, it must be added for every user that has Twister client installed and it has to be started as that user.

7.1   Central engine web interface

While the **Central Engine** service is running, you can access a web interface that allows viewing some statistics, logs and users connected. You can also start and stop the processes.

Management interface Home;

Central Engine interface

IP and Port	127.0.0.1:8000
Machine	user-VirtualBox
System	Ubuntu 13.04 raring
Server Type	no_type
Server Version	2.024
Processor	
Memory	

Central Engine Logs;

Central Engine Log

Refresh ⟳

```
13-08-01 13:22:18   DEBUG    __init__: CE: Starting Twister Server version `2.016`...

13-08-01 13:22:18   DEBUG    __init__: CE: Running server type `no_type`.

13-08-01 13:22:18   DEBUG    __init__: CE: Initialization took 0.0000 seconds.

13-08-01 13:22:18   DEBUG    __init__: SM: Starting Service Manager...

13-08-01 13:22:18   DEBUG    __init__: SM: Found `5` services: `controller, controller_1, controller_2,

13-08-01 13:22:18   DEBUG    _load: RA: Resources loaded successfully.

13-08-01 13:22:18   INFO     [01/Aug/2013:13:22:18] ENGINE Listening for SIGHUP.

13-08-01 13:22:18   INFO     [01/Aug/2013:13:22:18] ENGINE Listening for SIGTERM.

13-08-01 13:22:18   INFO     [01/Aug/2013:13:22:18] ENGINE Listening for SIGUSR1.

13-08-01 13:22:18   INFO     [01/Aug/2013:13:22:18] ENGINE Bus STARTING

13-08-01 13:22:18   INFO     [01/Aug/2013:13:22:18] ENGINE Started monitor thread '_TimeoutMonitor'.

13-08-01 13:22:18   INFO     [01/Aug/2013:13:22:18] ENGINE Serving on 0.0.0.0:8000

13-08-01 13:22:18   INFO     [01/Aug/2013:13:22:18] ENGINE Bus STARTED

13-08-01 13:22:18   INFO     [01/Aug/2013:13:22:18] ENGINE Started monitor thread 'Session cleanup'.

13-08-01 13:22:18   DEBUG    createUser: Project: Created user `nisuser` ...
```

User interface;

User `user`

Status	stopped (▶ start | ■ stop | ⚠ reset!)
Master config	/home/user/twister/config/fwmconfig.xml
Project config	/home/user/twister/config/testsuites.xml
DB config path	/home/user/twister/config/db.xml
E-mail config path	/home/user/twister/config/email.xml
EPs file	/home/user/twister/config/epname.ini
Logs path	/home/user/twister/logs
Tests path	/home/user/twister/demo
User groups	user, admin
User roles	CHANGE_DB_CFG, CHANGE_EML_CFG, CHANGE_FWM_CFG, CHANGE_GLOBALS, CHANGE_PLUGINS, CHANGE_PROJECT, CHANGE_SERVICES, CHANGE_TESTBED, CREATE_PROJECT, DELETE_PROJECT, EDIT_TC, RUN_TESTS

Control the processes;

👤 Home   ☰ Processes   ☰ Logs

Processes for `tscguest`

✔ ■ EP-1001	Actions: ▶ | ❚❚ | ■ | ✎
🕐 ▶ EP-1002	
🕐 ■ EP-1003	⊟ 📁 Suite1
🕐 ■ EP-1004	📄 /home/tscguest/twister/demo/testsuite-python/init.py
	📄 /home/tscguest/twister/demo/testsuite-python/test002.py
🕐 ■ EP-1005	📄 /home/tscguest/twister/demo/testsuite-python/test001.py
	📄 /home/tscguest/twister/demo/testsuite-python/test003.py

Check all user logs;

Logs for `tscguest`

☰ logCli EP-1001	Py **debug**: For EP EP-1001, CE Server returned a new status: running.
☰ logCli EP-1002	EP **debug**: Received start signal from CE!
☰ logDebug	TC **debug**: TestCaseRunner started with User: tscguest ; EP: EP-1001.
☰ logRunning	TC **debug**: Connected to proxy, running tests!
☰ logSummary	Downloading library `/home/tscguest//twister/.twister_cache/EP-1001/ce_libs/ExposedLibraries.py` ...
☰ logTest	Downloading library `/home/tscguest//twister/.twister_cache/EP-1001/ce_libs/TscFtp.py` ...

```
                                Downloading library `/home/tscguest//twister/.twister_cache/EP-1001/ce_libs/TscTelnet.py` ...


                                ===== ===== ===== ===== =====
                                  Starting suite `100:Suite1`
                                ===== ===== ===== ===== =====


                                Downloading library `/home/tscguest//twister/.twister_cache/EP-1001/ce_libs/TscFtp.py` ...
                                Downloading library `/home/tscguest//twister/.twister_cache/EP-1001/ce_libs/TscTelnet.py` ...
```

This web service can be accessed in a browser, by going to: `http://central-engine-IP:PORT/ `,

for **example**: `http://localhost:8000/ `.

8   How to compile the Java GUI

The Java Graphical User interface **compiled version** can be found at `twister/**binaries**/applet`. You have to copy the `applet` folder in `/var/www` and if you have an *Apache* or *Lighttpd* Server, you will be able to access it in the browser.

If you have changed the sources, or you want to compile the JAR files yourself, the sources are located at `twister/client/userinterface/java`. Some binary JAR files are already included in folders `target` and `extlibs`, respectively.

After compilation, you have to move the JAR files, so that a server can serve them.

Steps **1-2** require *Oracle JDK **1.7*** (*Oracle Java Development Kit*).

Step **5** requires *Apache* or *Lighttpd* Server and your machine must have *OpenSSH **Server*** enabled on port 22.

Here are the steps:

1. Generate a key store, or import a certificate (*this is done only **the first time**!*);

```
PATH_TO_JDK/bin/keytool -genkey -keyalg rsa -validity 360000 -alias Twister -keypass password -storepass password
```

OR

```
PATH_TO_JDK/bin/keytool -import -alias Twister -file certificate_file.cer
```

2. Go in `client/userinterface/java`. Then, if you are on **Windows**, run `pack.bat`, on **Linux** run `./build.sh`. You might need to edit these files, to change the path to **JDK_PATH**;

3. Move all files from `target` and `extlibs` in `/var/www/twister` (path for Apache, or other web servers);

4. Copy `jquery.min.js` from `/opt/twister/server/static/js` also in `/var/www/twister`;

5. Open a browser that supports Java Applets and go to: `http://localhost/twister`.

9 Overview of the Java GUI

The **first tab (Suites)** is split in four panes:

- **Top left**, is where the test suites are defined. Any file from the right can be dragged in here. The files can be checked/ unchecked; the files that are not checked will not run;

- **Top right**, is where the test files are located. These files can be used in the suites;

- **Bottom left**, is where the project, suite and test information is added. The suite information is defined in the file `DB.xml`, section name `field_section` (*more about this in the configuration section*);

- **Bottom right**, you can see the title and description of the currently selected test file.

A configuration with Python scripts:
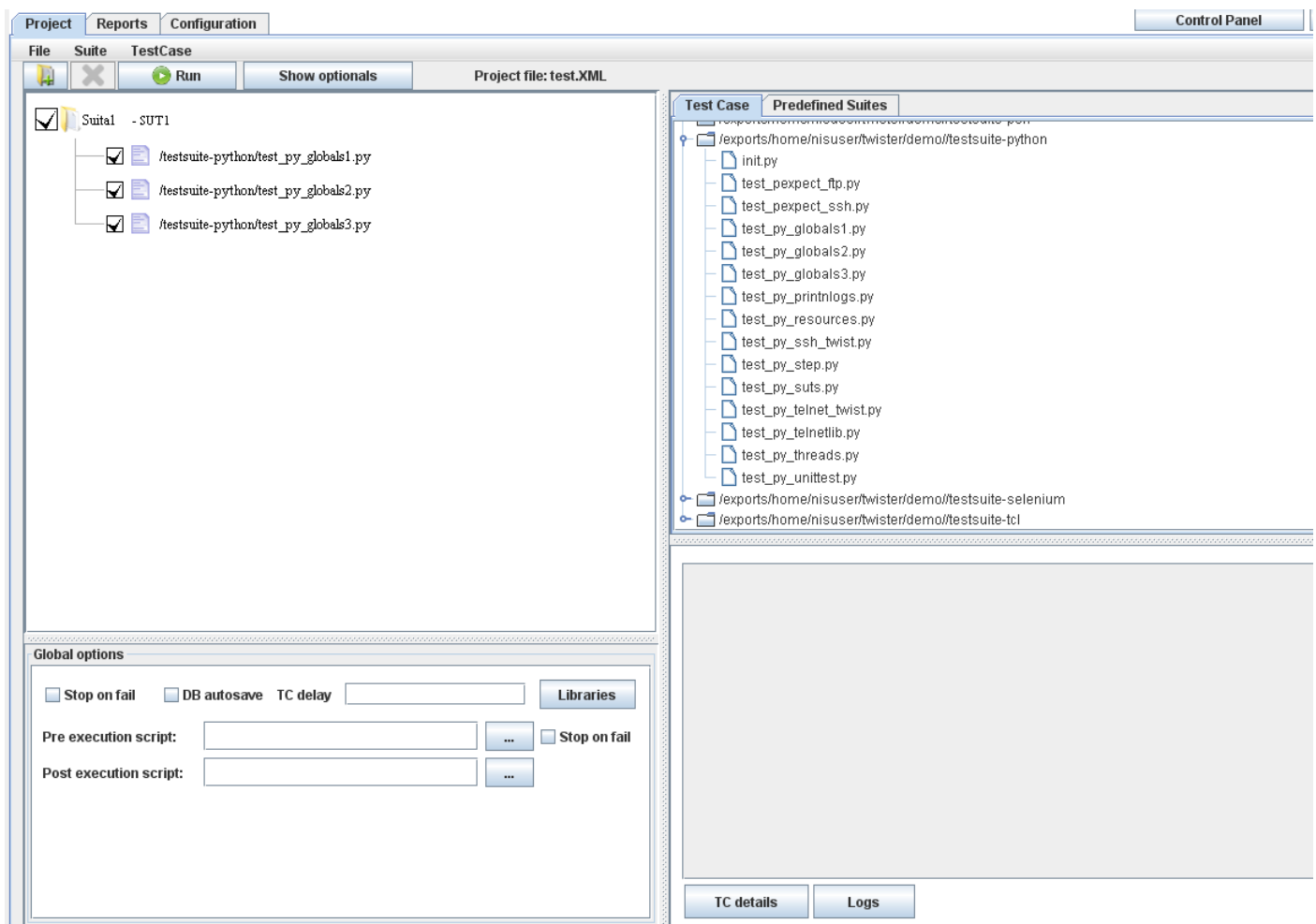
TWISTER USER GUIDE

A different configuration, with TCL scripts:

Luxoft
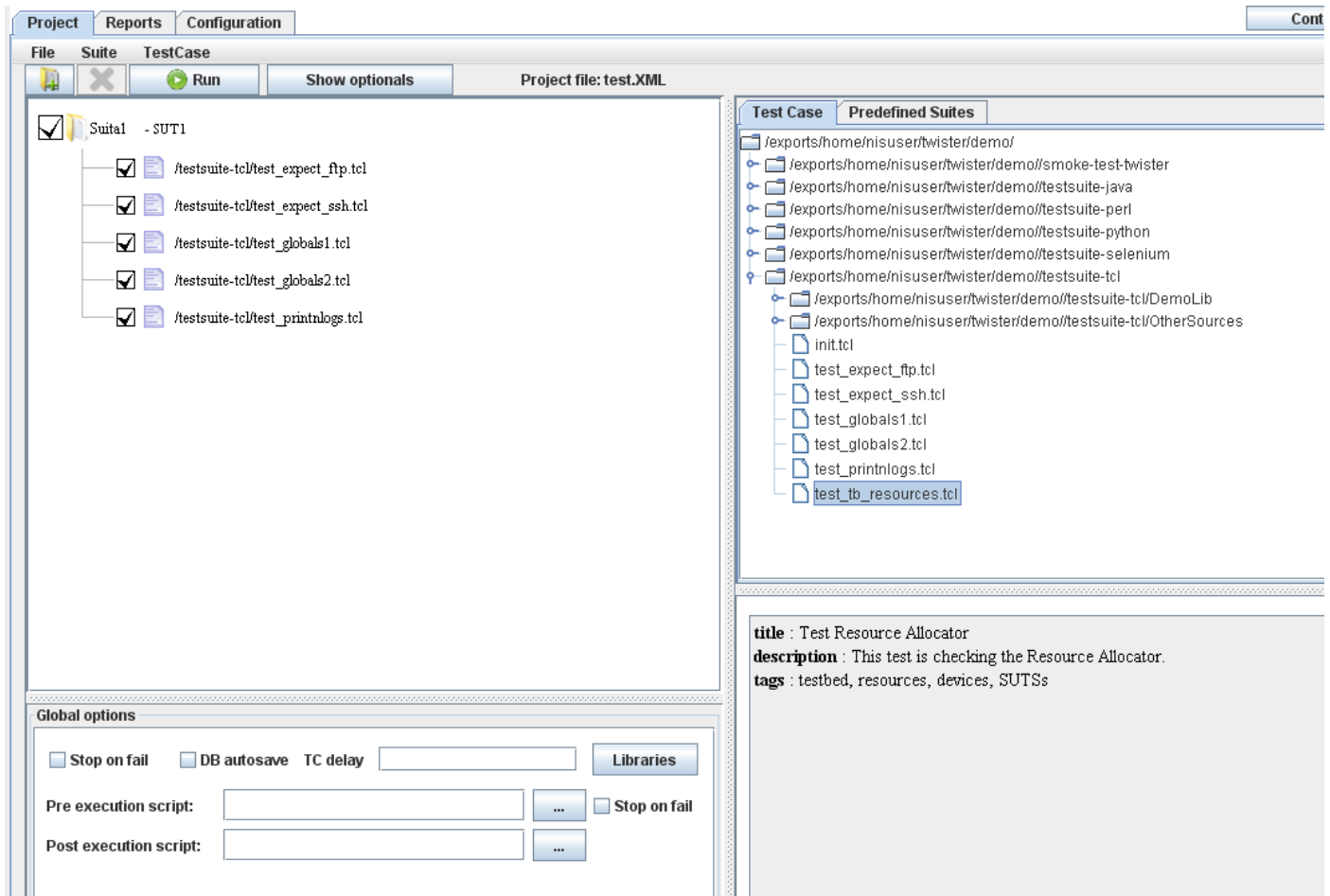
TWISTER USER GUIDE

While running:

- You can check test lists with their statuses. By default, all tests are in pending, unless they recently ran, in which case the most recent status is displayed;
- Logs for the tests. The logs can be cleaned, exported, or searched for keywords.

Here the Central engine is stopped; in this case you can see the most recent statuses:

At the top, there are two buttons, which control the Central Engine: **Run/ Pause** and **Stop**.

Also at the top, is the status of the Central Engine, the time of the last start, time elapsed and the user that started it.

TWISTER USER GUIDE

While the Central engine is running:

Luxoft

9.1   The Reports Tab

When clicking on it, the reports page will open in a new tab.

For your convenience whenever you will click the **Home** link the database configuration (including username, password, database, fields and reports) are reloaded. The other links use the cached version. So if you change your settings in db.xml you don't have to restart the CE, you just need to click Home button.

Reporting home (this will look different, depending on the configuration!)
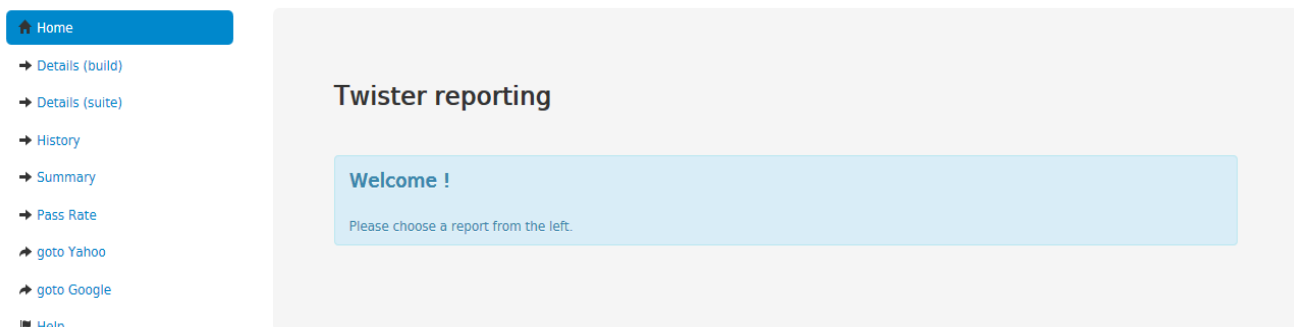
A report with user chosen fields;
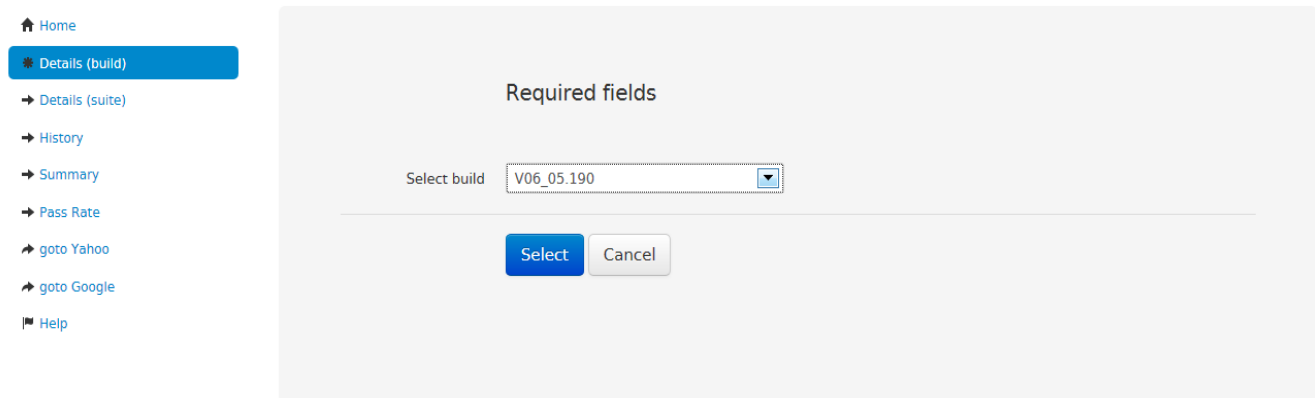
The same report, after the user chose the build;

The configuration tab

Here, you can configure:

- Central Engine port (default 8000). This is the port the applet will use for connection;
- the path of the test files, logs files, project files
- the path of the database xml, e-mail xml, EP names;
- additional path for python library files, and the path for predefined suite files
- the names of the log files;
- e-mail configuration, database configuration;
- devices configuration for Test Bed;
- SUTs configuration;
- Test Configurations settings;
- global variables, injected in all Twister test files;
- services configuration;
- plugins configuration;
- panic-detect: checks errors in CLI logs.
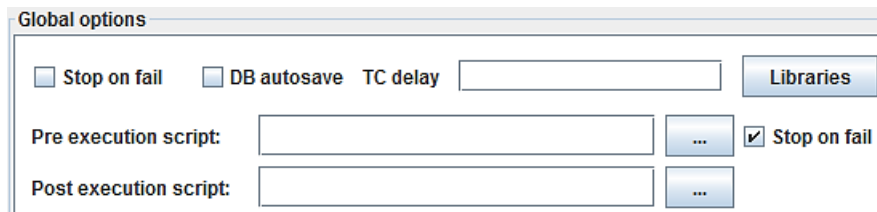
More about this in `**Configure the framework**` section.

10 How to define the suites and add the tests

When starting the interface, you must first *select* or *create* a **project file**. This file will save your suites, script files and suites configurations.

Each project has a set of global settings:

- **Stop on fail** = if a test that is mandatory will fail, or will crash, all the project will fail ;
- **DB Autosave** = after all the tests from all suites finish execution, the Central Engine will automatically save the results into database, without asking the user ;
- **TC Delay** = after each test, the EP will wait X seconds, before starting to execute the next test ;
- **Pre execution script** = this defines a script that can be executed in command line interface; the script from this path will be executed by the Central Engine *before* running any test. The script can be written in any language (Perl, TCL, binary executable, etc.). You cannot use a normal Twister test as a Pre/ Post exec script! If you write a script in an interpreted language, don't forget to add `#!/usr/bin/env ... ` *your language* ` on the first line. If the file is not executable, CE will automatically run `chmod +x` on the file ;
- **Post execution script** = this defines a script that can be executed in command line interface; the script from this path will be executed by the Central Engine *after* running all tests. It has the same specifications as the Pre execution script ;

After setting the project file, click on (Add suite). The required fields are: **the name** of the suite and **one or Test beds** (the workstation(s) where the tests from this suite will run).
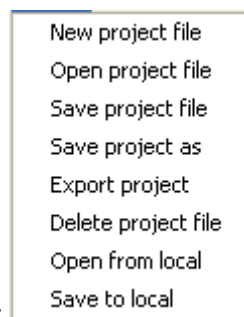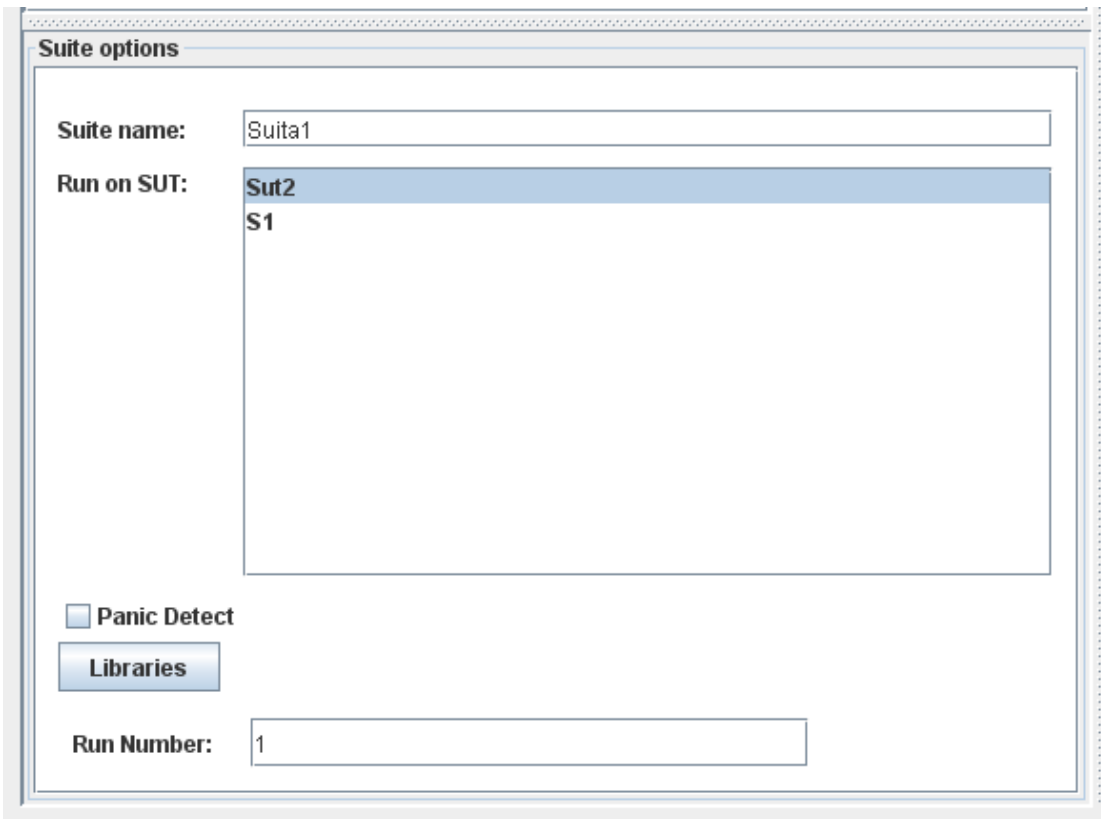
In order to save the **project file**, use the *File menu :* .

You can download the project file locally, to share it with your team members.

A suite is basically a folder, where one or more script/ test files can be added, that will be executed by one, or more SUTs and implicitly by one, or more Execution Processes.

Each suite can also have some properties attached to it, like `release`, `build`, `comments`, etc. These fields are defined in `DB.xml` file (*more about this in the database configuration section*) and will look different, depending on the configuration (text box, drop down list, path to a script, etc.) ; these `meta` properties are used while saving the results into database.

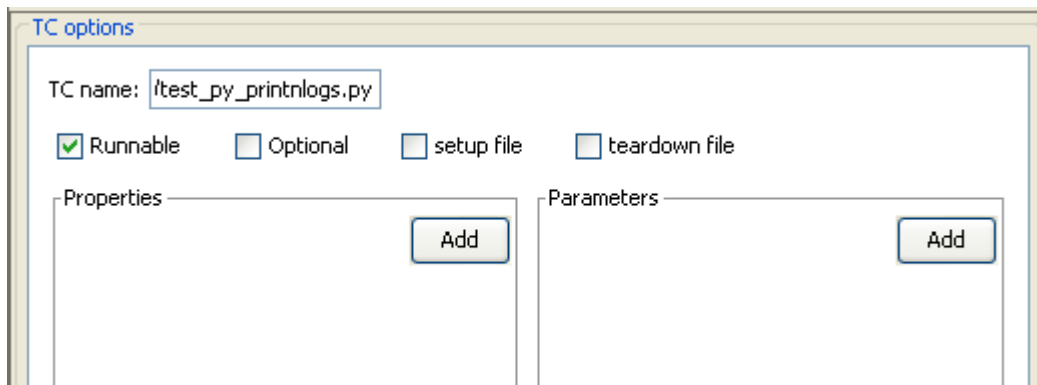A suite can be exported for later usage by right clicking it and selecting *Export*:

The content of the suite, including the test cases and its sub suites, is saved in an xml file in the *Predefined Suites* path. These exported suites can be loaded in other project using drag & drop from the *Predefined Suites* tab into the suite definition window.

The script/ test files and the suites can be re-arranged anytime, using drag & drop, or can be deleted.

Each script/ test file has a few properties too:

- A test that is not *Runnable* will be sent to **EP**, but will never execute. You can use this option to transfer configuration files on the EP, without executing them ;
- *Optional* tests that fail, will not stop the execution when the global *Stop on Fail* checkbox is active ;
- *Setup* tests are executed at all times (they have to be runnable) and if they *fail*, the entire Suite is considered *Failed*. Setup files are NOT saved into the database!
- *Teardown* tests are executed at all times (they have to be runnable). They are used to clean-up a suite, even if the suite was aborted because of a failed Setup. Teardown files are NOT saved into the database!

The user can define a list of parameters and properties for every test case.
Both the parameters and the properties are sent to the tests and can be accessed during execution.

The properties: `type`, `suite`, `file`, `config_files`, `dependency`, `status`, `Runnable`, `Optional`, `setup_file`, `teardown_file` and `param` are reserved for internal use and must NOT be used; more about this in `**How to write Twister tests**` section.

TWISTER USER GUIDE

11 How to run the test files

After all the suites and scripts are set, click on [▶ Run], to start the execution.

What will happen is that, all **checked** scripts from all suites will run, in order. The execution doesn't stop if one of the scripts fails, excepting the case when the test that fails is *mandatory* and the *Stop on Fail* checkbox is checked.

If the **Run** button is disabled, it means that the Central Engine service is not running, so the execution cannot start.
If the **Run** button is enabled, you can start the execution. When you start the execution, the **Stop** button will activate, to allow stopping the Central Engine and killing all the running processes and the **Run** button will become **Pause**, allowing to pause after the current tests finishes its execution.

If the Central Engine was started recently, by default all the files will be in state   *pending* .
If a previous run was completed, the most recent status is displayed (*pass, failed, etc.*).

If the execution is stopped prematurely, all files that did not execute (were pending), have status NOT EXECUTED.

The states for the files and their respective icons are:

- (status **working**) while the file is running;
- (status **pending**) before the file was executed;
- (status **pass**) if the execution was successful. This status can be set manually;
- (status **fail**) if the execution failed. This status can be set manually;
- (status **skipped**) if the file was marked as skip (*runnable=false*), or the file doesn't exist;
- (status **aborted**) if the file was stopped while running. This status can be set manually;
- (status **timeout**) if the file took too much time to run. This status can be set manually;
- (status **not executed**) if the file was supposed to run, but didn't;
- (status **waiting**) if the file depends on another file and the dependency didn't finish its execution, so this file is waiting.
- (status **invalid**) if the test case returned user defined invalid scenario.

The statuses: *working* and *pending* should NOT be sent, because they are intermediary statuses and will produce confusion with a finishing status. When a test is completed, the icon will change to Pass, Fail or any other status sent from a test.

While the tests are running, the logs from the left will update, showing the live output. All the history of result can be seen in the `log_summary.log`.

The logs can be cleaned, exported, or searched for keywords, by clicking the buttons from the bottom.

After all the tests are run, if the e-mail is configured, CE sends an e-mail with the report and then, all tests are saved into the database, excepting the *Setup* and *Teardown* files.

Following it's an example of distributed execution of tests.

First you define your SUTs:
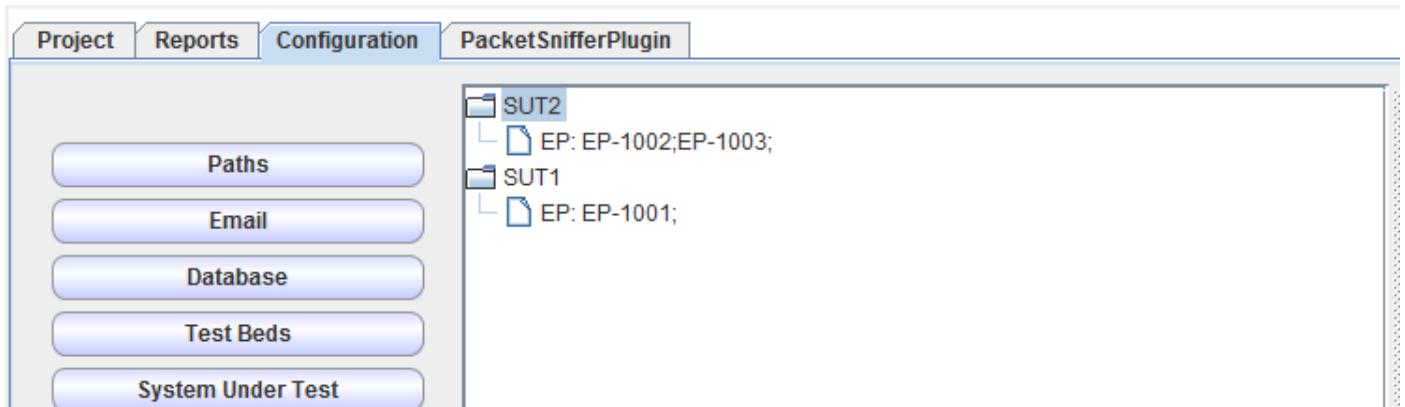
If you don't set any EPs on the SUTs, one local EP will be automatically assigned for you.

After, you define your suites and tests to be run:

When running the tests you can see that the Suite2 who did run on SUT2 which had 2 EPs assigned it was run in the same time on 2 different EPs:
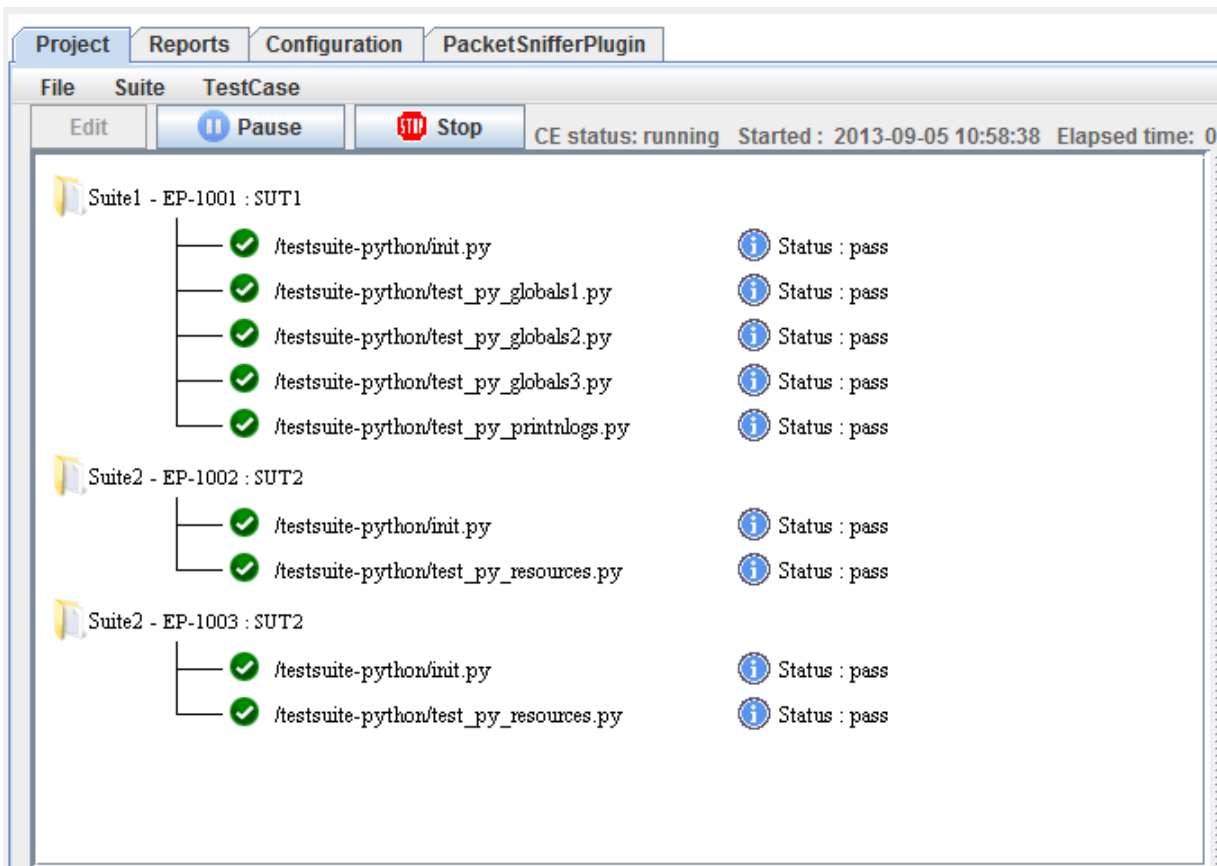
12 How to change the Central Engine log verbosity

You can see the Central Engine logs in 3 ways: in the applet, while running the tests, in the web interface and directly on the server, in file `/opt/twister/server_log.log`.

However, the logs are by default set on Warning and don't show much information.

If you need to set the verbosity, there are 2 ways:

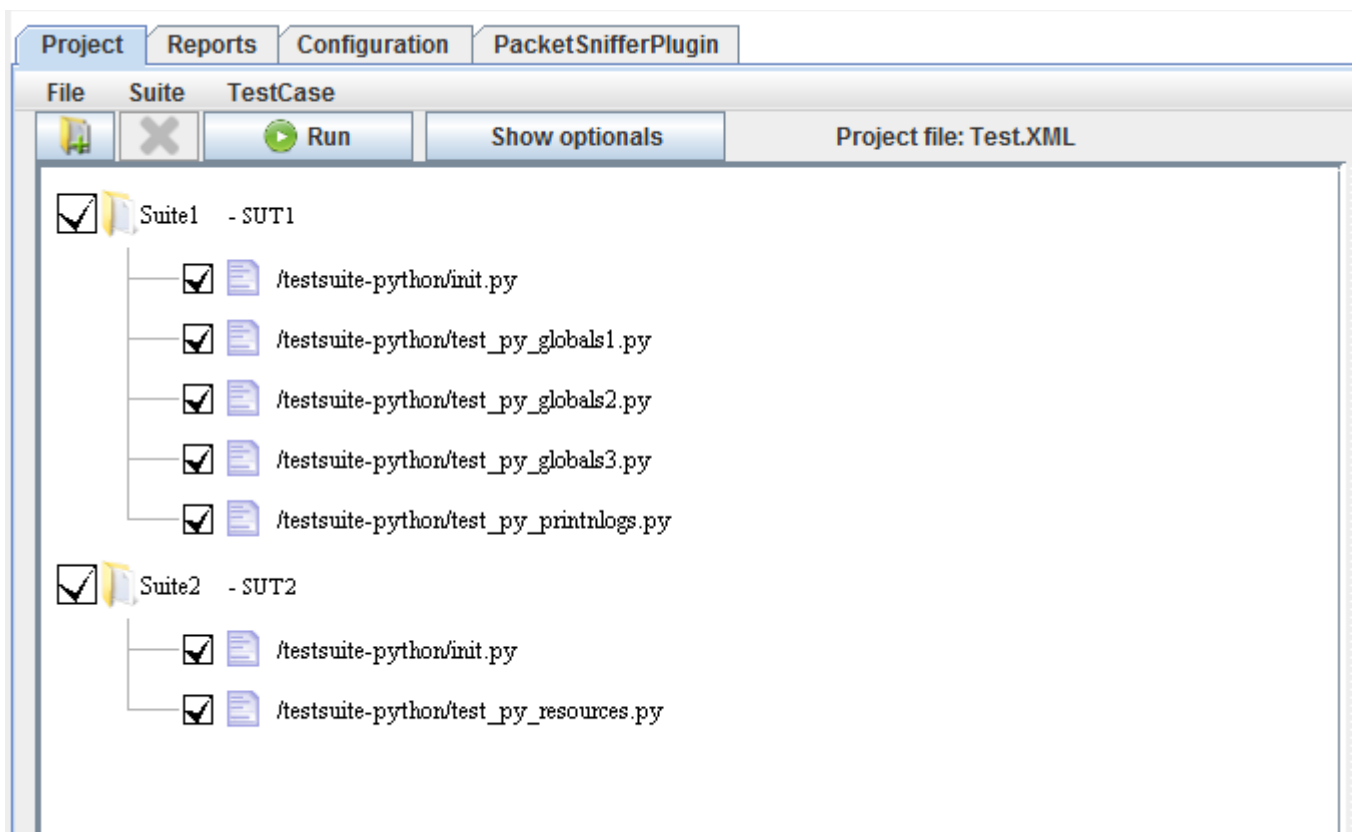- in the settings file `/opt/twister/config/server_init.ini `, section `verbosity` (this will set the default verbosity, when the Central Engine starts). You must use a value from: FULL, DEBUG, INFO, WARNING, ERROR, CRITICAL;
- live, during runtime, using the script from the repository `twister/bin/set_log_level.py` (the script is used like this: `python set_log_level.py X ` , where X is value from: FULL, DEBUG, INFO, WARNING, ERROR, CRITICAL. If you don't enter any value, the script will display the current log level (`python set_log_level.py`).

13 Command line interface

The Command-line script can be found in twister/bin, both on server and client side.

This script can be used to:

- start, stop, pause the execution;
- show all users that are running tests;
- display what EPs are enabled for your user;
- show what is the start time for this run, suites list and tests list;
- show execution summary status: how many test cases are planned for execution, how many were executed, how may passed, how many failed;
- show execution details status: the same, plus status per test case;
- queue tests during run time (the user is not forced to stop the execution in order add more files for execution). Queuing a file while the execution is stopped has no effect - it will be discarded when the execution is started. Instead, the project file has to be updated in order to include that file.
- dequeue tests during run time, before they are executed. DE queuing a file while the execution is stopped has no effect.

You can use `**./cli.py --help**` anytime, to see the usage information.

```
Usage: cli.py --server <ip:port> --command [...parameters]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  --server=SERVER       The Central Engine IP : port (default= 127.0.0.1:8000)
  --login=LOGIN         Login on the Central Engine with user : password
                        (default= user:password)
  -u, --users           Show active and inactive users.
  --eps                 Show active and inactive Eps.
  --stats               Show stats.
  --details             Show detailed status for All files.
  --status-details      Show detailed status for running, finished, pending,
                        or all files.
  -q QUEUE, --queue=QUEUE
                        Queue a file at the end of a suite. Specify queue like
                        `Suite:file_path`.
  -d DEQUEUE, --dequeue=DEQUEUE
                        Un-Queue 1 or more files. Specify like `EP,
                        EP:suite_id, EP:Suite, or EP:file_id`.
  --run-test=RUN_TEST   Add a file in a suite, run blocking and then show the
                        log. Specify the fname and the SUT.
  --sut=SUT             Use this with the `run-test` option.
  -s SET, --set=SET     Set status: start/ stop/ pause. (Must also specify a
                        config and a project)
  -c CONFIG, --config=CONFIG
                        Path to FWMCONFIG.XML file.
  -p PROJECT, --project=PROJECT
                        Path to PROJECT.XML file.
```

TWISTER USER GUIDE

Commands:

- ./cli.py -u   - Show active and inactive users ;
- ./cli.py --eps   - Show active and inactive Eps ;
- ./cli.py --stats   - Show minimal stats ;
- ./cli.py --details   - Show detailed stats, per ep + suite + test ;
- ./cli.py --status-details <running| finished| pending| all> ;
- ./cli.py --queue <Suite1:some_path/some_file.py>
  - Queue a file at the end of a suite. Must specify queue in the form of `suite:file_path` ;
- ./cli.py --dequeue <EP | EP:suite_id | EP:Suite | EP:file_id>
  - Un-queue a file from a project, before it is executed ;
- ./cli.py -s stop | ./cli.py -s start -p ~/twister/config/testsuites.xml --config ~/twister/config/fwmconfig.xml
  - start, pause, or stop the central engine.

All the commands need the **--server** and **--login** flags to be able to connect to the Central Engine!

Examples:

./cli.py --version
Showing cli.py version:

```
tscguest@tsc-server:/opt/twister/bin$ python cli.py --version
cli.py  2.014

tscguest@tsc-server:/opt/twister/bin$
```

./cli.py –server <…> --login <…>
It must be used in conjunction with other command:

```
tscguest@tsc-server:/opt/twister/bin$ python cli.py --server tsc-server:8000 --login usr:passwd

Hello, user `tscguest`.
Authentication successful!

You didn't specify a command ! Exiting !

tscguest@tsc-server:/opt/twister/bin$
```
You can see below how it is used with other commands.

./cli.py –u
It will show you the users that have Twister installed and active/inactive users.
If you don't specify a server and login it will use the default values 127.0.0.1:8000 and user:password.

```
tscguest@tsc-server:/opt/twister/bin$ python cli.py -u --server tsc-server:8000 --login
usr:passwd

Hello, user `tscguest`.
```

TWISTER USER GUIDE

```
All users that have installed Twister: `bogdan, croq, nisuser, nisuser2, tscguest`.
All active users: `croq, nisuser, tscguest`.

tscguest@tsc-server:/opt/twister/bin$
```

./cli.py --eps
It will show you a status of the EPs and their state (active/inactive).

```
tscguest@tsc-server:/opt/twister/bin$ python cli.py --eps --server tsc-server:8000 --login
usr:passwd

Hello, user `tscguest`.

Your Execution-Processes are:
EP-1001 : service not running
EP-1002 : service not running
EP-1003 : service not running

tscguest@tsc-server:/opt/twister/bin$
```

./cli.py --stats
It will show you the status of the tests executed on the server:

```
tscguest@tsc-server:/opt/twister/bin$ python cli.py --stats --server tsc-server:8000 --login
usr:passwd

Hello, user `tscguest`.

User status : stopped
Last started: 2013-08-29 11:35:45
Time elapsed: 0:00:56

Passed  : 5
Failed  : 2
Pending : 0
Working : 0
Aborted : 0
No Exec : 0
Other   : 0
> Total : 7
Pass rate: 71.43%

tscguest@tsc-server:/opt/twister/bin$
```

./cli.py --details
It will show you detailed status of all tests:

```
tscguest@tsc-server:/opt/twister/bin$ python cli.py --details --server tsc-server:8000 --login
usr:passwd

Hello, user `tscguest`.

User status : stopped
Last started: 2013-08-29 13:13:10
```

```
Time elapsed: 0:00:09

Your Suites are:
  - on EP-1001 :
    - (id 101) S1
      - [pass] (id 1001) /home/tscguest/twister/demo/testsuite-python/init.py
      - [pass] (id 1002) /home/tscguest/twister/demo/testsuite-python/init.py
      - [pass] (id 1003) /home/tscguest/twister/demo/testsuite-python/init.py

  - on EP-1002 :
    - nothing here

  - on EP-1003 :
    - nothing here

tscguest@tsc-server:/opt/twister/bin$
```

./cli.py --status-details

It will show you the detailed status of the tests based on a filter (all, running, finished, pending).

```
tscguest@tsc-server:/opt/twister/bin$ python cli.py --status-details all --server tsc-
server:8000 --login usr:passwd

Hello, user `tscguest`.

User status : stopped
Last started: 2013-08-29 13:13:10
Time elapsed: 0:00:09

Your Suites are:
  - on EP-1001 :
    - (id 101) S1
      - [pass] (id 1001) /home/tscguest/twister/demo/testsuite-python/init.py
      - [pass] (id 1002) /home/tscguest/twister/demo/testsuite-python/init.py
      - [pass] (id 1003) /home/tscguest/twister/demo/testsuite-python/init.py

  - on EP-1002 :
    - nothing here

  - on EP-1003 :
    - nothing here

tscguest@tsc-server:/opt/twister/bin$
```

./cli.py –q

It will add/queue a test at the end of a specified suite.
This command it's working only when the tests are already running.

```
tscguest@tsc-server:/opt/twister/bin$ python cli.py -q
S1:/home/tscguest/twister/demo/testsuite-python/test_py_printnlogs.py --server tsc-server:8000
--login usr:passwd

Hello, user `tscguest`.
```

```
Test `/home/tscguest/twister/demo/testsuite-python/test_py_printnlogs.py` was queued in suite
`S1`.

tscguest@tsc-server:/opt/twister/bin$ python cli.py --details --server tsc-server:8000 --login
usr:passwd

User status : running
Last started: 2013-08-29 13:40:37
Time elapsed: 0:00:27

Your Suites are:
  - on EP-1001 :
    - (id 101) S1
      - [pass] (id 1001) /home/tscguest/twister/demo/testsuite-python/init.py
      - [pass] (id 1002) /home/tscguest/twister/demo/testsuite-python/test_py_globals1.py
      - [pass] (id 1003) /home/tscguest/twister/demo/testsuite-python/test_py_globals2.py
      - [pass] (id 1004) /home/tscguest/twister/demo/testsuite-python/test_py_globals3.py
      - [pass] (id 1005) /home/tscguest/twister/demo//testsuite-python/test_py_printnlogs.py
      - [working] (id 1006) /home/tscguest/twister/demo//testsuite-python/test_pexpect_ftp.py
      - [pending] (id 1007) /home/tscguest/twister/demo/testsuite-python/test_py_printnlogs.py

  - on EP-1002 :
    - nothing here

  - on EP-1003 :
    - nothing here

tscguest@tsc-server:/opt/twister/bin$
```

./cli.py –d
It will delete/de queue a test, a suite or all of the tests from a suite or EP.
This command it's working only when the tests are already running.

```
tscguest@tsc-server:/opt/twister/bin$ python cli.py --details --server tsc-server:8000 --login
usr:passwd

User status : running
Last started: 2013-08-29 14:04:41
Time elapsed: 0:00:07

Your Suites are:
  - on EP-1001 :
    - (id 101) S1
      - [pass] (id 1001) /home/tscguest/twister/demo/testsuite-python/init.py
      - [pass] (id 1002) /home/tscguest/twister/demo/testsuite-python/test_py_globals1.py
      - [pass] (id 1003) /home/tscguest/twister/demo/testsuite-python/test_py_globals2.py
      - [pending] (id 1004) /home/tscguest/twister/demo/testsuite-python/test_py_globals3.py
      - [pending] (id 1005) /home/tscguest/twister/demo//testsuite-python/test_py_printnlogs.py
      - [pending] (id 1006) /home/tscguest/twister/demo//testsuite-python/test_pexpect_ftp.py

  - on EP-1002 :
    - nothing here

  - on EP-1003 :
    - nothing here

tscguest@tsc-server:/opt/twister/bin$
```

```
tscguest@tsc-server:/opt/twister/bin$ python cli.py -d EP-1001:1006 --server tsc-server:8000 --
login usr:passwd

Test `1006` was removed from the project.

tscguest@tsc-server:/opt/twister/bin$ python cli.py --details --server tsc-server:8000 --login
usr:passwd

User status : running
Last started: 2013-08-29 14:04:41
Time elapsed: 0:00:17

Your Suites are:
  - on EP-1001 :
    - (id 101) S1
      - [pass] (id 1001) /home/tscguest/twister/demo/testsuite-python/init.py
      - [pass] (id 1002) /home/tscguest/twister/demo/testsuite-python/test_py_globals1.py
      - [pass] (id 1003) /home/tscguest/twister/demo/testsuite-python/test_py_globals2.py
      - [pass] (id 1004) /home/tscguest/twister/demo/testsuite-python/test_py_globals3.py
      - [working] (id 1005) /home/tscguest/twister/demo//testsuite-python/test_py_printnlogs.py

  - on EP-1002 :
    - nothing here

  - on EP-1003 :
    - nothing here

tscguest@tsc-server:/opt/twister/bin$
```

./cli.py –s <status>
It will start/stop/pause the execution of a project file.
You will have to specify the config and the project file you will be using.

```
tscguest@tsc-server:/opt/twister/bin$ python cli.py -s start --server tsc-server:8000 --login
usr:passwd -c ~/twister/config/fwmconfig.xml -p ~/twister/config/testsuites.xml

Starting... running

tscguest@tsc-server:/opt/twister/bin$ python cli.py --details --server tsc-server:8000 --login
usr:passwd

Hello, user `tscguest`.

User status : running
Last started: 2013-08-29 11:35:45
Time elapsed: 0:00:08

Your Suites are:
  - on EP-1001 :
    - (id 101) S1
      - [pass] (id 1001) /home/tscguest/twister/demo//testsuite-python/init.py
      - [working] (id 1002) /home/tscguest/twister/demo//testsuite-python/test_pexpect_ftp.py
      - [pending] (id 1003) /home/tscguest/twister/demo//testsuite-python/test_pexpect_ssh.py
      - [pending] (id 1004) /home/tscguest/twister/demo//testsuite-python/test_py_globals1.py
      - [pending] (id 1005) /home/tscguest/twister/demo//testsuite-python/test_py_globals2.py
      - [pending] (id 1006) /home/tscguest/twister/demo//testsuite-python/test_py_globals3.py
      - [pending] (id 1007) /home/tscguest/twister/demo/testsuite-python/init.py
```

Luxoft

```
- on EP-1002 :
  - nothing here

- on EP-1003 :
  - nothing here
```

14 Twister configuration

14.1 - Twister configuration files

Twister has a few configuration files, in XML, ini and Json format.

There are 2 types of configurations: per user (located at */$USER_HOME/twister/config*) and global for all users (by default located at */opt/twister/config*).

- fwmconfig.xml: the **master framework config**. Contains the paths to all the other config files. Config saved per user;
- epname.ini: contains all the EPs for the current user. **This must be edited manually**. Config saved per user;
- email.xml: contains all the information necessary to send an e-mail from Twister. Config saved per user;
- db.xml: contains all the information about saving and reading from the MySQL database. Config saved per user;
- globals.xml: contains all global variables, per user;
- plugins.xml: contains information about all plugins, per user;
- testsuites.xml: contains all suites and all files for the current run. Saved per user;
- resources.json: contains all the resources from Resource Allocator server, all testbeds and devices. It's a global config;
- services.ini: contains all the services from Service Manager. It's a global config. This must be edited manually;
- server_init.ini: contains the current server type (**no_type**, **development**, or **production**), the **server location** description and the **verbosity**. In `production` mode, all user roles are enabled. In `no_type` mode, all roles are enabled, but changing users is disabled;
- users_and_groups.ini: contains all users, groups and roles;
- twister.key: contains a user defined string that is used to encrypt/decrypt the passwords for db.xml and email.xml files. The passwords from these files cannot be decrypted without the key and the user can protect this file to restrict access for other users.. It is saved per user and it must be edited manually.

Most of the files are generated automatically from the Java applet. They **should not** be edited manually, unless specified otherwise.

14.2 - Configure the paths

★*All the paths below refer to the computer where the **Central Engine** is running*.

Test case source path represents the folder where all test files are located. The files here can be dragged inside suites, in the first tab (suites).

Projects path is the folder where the profiles are saved in the first tab; usually this doesn't need to be changed.

EP Name file stores the list of EPs (the workstations where tests will run). An `EP` is just a name to uniquely identify a computer, it can be any string.

Logs path is the folder where all the logs are written. There are 5 major logs: log running, log debug, log summary, log info, and log CLI. Each of the logs will be saved in the logs path, with the name defiled in the configuration. Usually, the logs don't need to be changed.

E-mail XML path, ***Database XML path*** and ***Globals XML path*** are the files that store the information for the next 3 tabs. You can have multiple files, and switch between configurations.

The *Central Engine port* is, of course, the port where the applet connects to the server. The default value is *8000*.

Library path defines a path where user defined python libraries can be found. The user libraries will be used together with the system libraries, stored in */opt/twister/lib*, to execute the test cases.

Predefined suites path this defines a path where user can save suites (different from project files) for future usage. The user can export a suite from a project file and import it in another project file.

14.3 - Configure the e-mail

Here you can configure the parameters required to connect to a SMTP server and send an e-mail.

The Central Engine will send the e-mail every time the execution finishes for ALL the test files.

The most important fields are: SMTP *IP* and *port*, *username*, *password*, *from* and the *e-mail list*.

Optionally, you can change the subject and add a few lines in the message body.

In order to test the connection to the SMTP server, using the user and password, you can send a test e-mail, using the button `Test` in the applet.

Both the subject and the message can contain insert fields from `DB.xml`, from `<insert_section>` tags.

For example, if you defined the fields with IDs `release_id`, `build_id`, `suite`, you can write the subject like:

```
E-mail report for R$release_id B$build_id - $suite [$date]
```

If your release number is `2`, build number is `15` and suite is `Branch Test1`, the subject will be generated like:

```
E-mail report for R2 B15 – Branch Test1 [2012.03.23 13:24]
```

You can also use one of the following tags:

- **$date**
- **$twister_user**
- **$twister_ce_hostname**
- **$twister_ce_ip**
- **$twister_ce_os**
- **$twister_ce_python_revision**
- **$twister_ce_type**
- **$twister_ep_hostname**
- **$twister_ep_ip**
- **$twister_ep_os**
- **$twister_ep_python_revision**
- **$twister_pf_fname**
- **$twister_rf_fname**
- **$twister_server_location**

Twister is providing a test button so the user can verify if the system can generate the email with the configured parameters. Activating the test button sends the email. The button is working regardless of the Enable/Disable checkbox. The user can disable email, send a test email, and then later enable email.

When you press the Test button, in case you entered wrong parameters/info the CE will pass through the SMTP error messages from the email server to an alert (pop-up in the GUI).

The email password is stored in a non-human readable format and it's encrypted with the user key from users_and_groups config file.

14.4 - Configure the database

By default, all the database information is stored in `DB.xml` file. This file can be changed from the interface, in the **Paths tab**.

The user can have multiple configuration file and switch between then.

The database file contains information about how to connect to the database, information for saving results into database and information to generate the reports.

The structure of the db.xml file has 3 sections, delimited by XML tags:
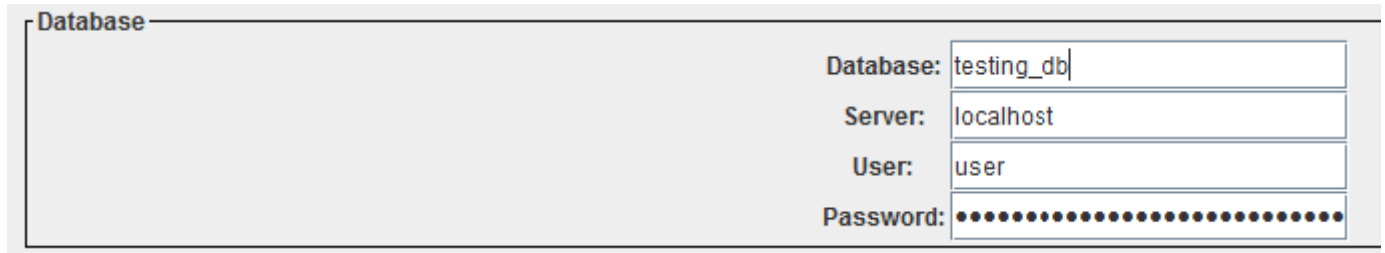- database connection section (db_config)
- statements for saving results into database (insert_section)
- statements for reports (reports_section)

11.4.1 Database connection

The user can define the information on how to connect to the database in the Configuration -> Database section.
The database connection information is stored in db.xml file between <db_config></db_config> tags.

The information about database name, server name and username can be changed as well by editing the db.xml file. The user's password is stored in the db.xml file in an encrypted form so it MUST be changed using the Twister GUI.

11.4.2 Saving results into database

Based on db.xml file, the Twister framework can be configured to identify what are the results and internal variables that need to be stored into database. The framework doesn't impose a fixed structure of the tables available in database, so the user must specify the information about the tables that store the results and what are the SQL statements used to save the results.

The information about results saving into database is stored in db.xml file between <insert_section> </insert_section> tags.

The *insert section* defines a list of SQL queries that will execute every time the execution finishes for ALL the test files. All queries are executed for each and every test file.

The insert queries use the information from the fields described above. A file can only access the fields defined in his parent suite.

The internal variables exported by Twister for database savings are:
- **$twister_user** = the name of the user that ran the tests;
- **$twister_ce_os** = the operating system of the computer where Central Engine runs;
- **$twister_ep_os** = the operating system of the computer where Execution Process runs;
- **$twister_ce_ip** = the IP of the Central Engine;
- **$twister_ce_hostname** = the host name of the Central Engine;
- **$twister_ep_ip** = the IP of the Execution Process;
- **$twister_ep_hostname** = the host name of the Execution Process;
- **$twister_ep_name** = EP name, defined in **Suites tab**;
- **$twister_rf_fname** = the path to Twister resources file (default is `resources.json`);
- **$twister_pf_fname** = the name of Twister project file

- **$twister_ce_python_revision** = python version from Central Engine;
- **$twister_ep_python_revision** = python version from Execution Process;
- **$twister_suite_name** = suite name, defined in **Suites tab**;
- **$twister_tc_name** = the file name of the current test;
- **$twister_tc_full_path** = the path + file name of the current test;
- **$twister_tc_title** = the title, from the **Suites tab**;
- **$twister_tc_description** = the description, from the **Suites tab**;
- **$twister_tc_status** = the final status of the test: pass, fail, skip, abort, etc;
- **$twister_tc_crash_detected** = if the file had a fatal error that prematurely stopped the execution;
- **$twister_tc_time_elapsed** = time elapsed;
- **$twister_tc_date_started** = date and time when the running started;
- **$twister_tc_date_finished** = date and time when the running finished;
- **$twister_tc_log** = the complete log from execution.
- **$twister_ce_type** = the type of the server
- **$twister_server_location** = the host location of the server
- **$twister_tc_revision** = only for test cases hosted in ClearCase, shows the revision number;

The user can define additional variables that can be saved into database by editing the db.xml file. The syntax used to define a new variable is the following:

```
<field ID="<ID_Field>" SQLQuery="<SQL_query>" Label="<GUI_Label>" Type="<UserText| UserSelect|
UserScript| DbSelect>" GUIDefined="true| false" Mandatory="true| false" Level="Project | Suite"
/>
```

Each field must contain the following tags:

- ID: represents the name of the field and MUST be unique;
- Type: there are 4 types of fields: UserSelect, DbSelect (where you must define an SQL query that will generate a list of value in the interface; the user will select 1 value and that will be saved; the difference between them is that DbSelect will never appear in the interface), UserScript (when specifying this field, for each Suite defined in the **Suites tab**, the user must provide the path to a script that will run for each file that is inserted in the database. This script can be written in any programming language and all the output printed on the screen *stdout* will captured and written in the database) and UserText (free text, can be any text);
- SQLQuery: this is required only for *UserSelect* and *DbSelect* fields. The query must be defined in such a way that the values will be unique (eg: by using SELECT DISTINCT id, name FROM …) and should select 2 columns. The first column will be the **ID** and second will be the **description** of the respective ID;
- GUIDefined: if a field is not GUI defined, it will not appear in the **Suites tab**, when editing suites. DbSelect fields are never visible in the interface;
- Mandatory: if a field is mandatory, each suite from the **Suites tab** must have a value for this field. If the user doesn't choose a value, he will not be able to save the profile, or start the execution of the project. Mandatory fields must be GuiDefined;

- **Label**: a short text that describes the field and will appear in the interface; labels are not necessary for DbSelect fields, because they are not visible in the interface.
- **Level**: an optional parameter that can be used to indicate at what level (per project or per suite) is the field set; by default, the user defined fields are set at suite level.

Note: Not all the fields must have values at once; depending on the type of variable that user wants to define; only a subset of these fields must have defined values.

These user defined variables can be used in queries using `$variable_name` if the field type is in {UserText| UserSelect| UserScript} or `@dbselect_field_name@` if the field type is DbSelect.

The SQL queries used to save the results into the database are stored between <sql_statement> </sql_statement> tags.

Examples *of SQl queries for database inserts*:

```
<sql_statement>
INSERT INTO gg_regression
(suite_name, test_name, status, date_start, date_end, build, machine)
VALUES
( '$twister_suite_name', '$twister_tc_name', '$twister_tc_status',
'$twister_tc_date_started', '$twister_tc_date_finished', '$release.$build',
'$twister_ep_name' )
</sql_statement>
```

Or

```
<sql_statement>
INSERT INTO results_table1
VALUES
( @res_id@, $release_id, $build_id, $suite_id, $station_id, '$twister_tc_date_finished',
'$twister_tc_status', '$comments' )
</sql_statement>
```

14.4.2.1 Defining a UserText variable

In this scenario, the user can define a custom variable that can be populated in GUI as free text and saved into database. This scenario applies in situations where user needs additional information to be saved into the database and the information must to be entered manually by the user.

To define such a variable, the following syntax must be used:
```
<field ID="<ID_Var>" SQLQuery="" Label="<GUI_Label>" Type="UserText" GUIDefined="true"
Mandatory="true " />
```

When such a variable is defined, a text box will appear automatically in GUI and it has to be populated for every defined suite.

For instance, let's say the user defines the field "Run Number":

```
<field ID="Run_number" SQLQuery="" Label="Run Number" Type="UserText" GUIDefined="true"
Mandatory="true " />
```

The GUI will present a field called "Run Number" and the user HAS to enter an alphanumeric value.

In order to be saved in the database, the variable must be referenced in the SQL query according to the ID field ('$Run_number').

For our example, the SQL query could look like:

```
<sql_statement> INSERT into results(run_number) values ( '$Run_number')</sql_statement>
```

14.4.2.2 Defining a UserScript variable

In this scenario, the user can define a custom variable that can be populated by an external executable script that is selected in GUI. This scenario applies in situations where user needs additional information to be saved into the database and the information can be obtained as an output of an script.

To define such a variable, the following syntax must be used:

```
<field ID="<ID_Var>" SQLQuery="" Label="<GUI_Label>" Type="UserScript" GUIDefined="true"
Mandatory="true " />
```

When such a variable is defined, a text box will appear automatically in GUI and the user can browse and select an executable script for the file system.

For instance, let's say the user defines the field "Location":

```
<field ID="ID_Location" SQLQuery="" Label="Location" Type="UserScript" GUIDefined="true"
Mandatory="true " />
```

The GUI will present a field called "Location" and the user HAS to select a script by pressing the Script button.
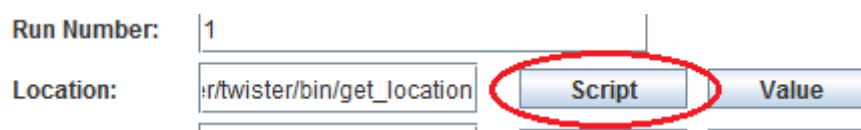
To verify the output of the selected script, the user can press Value button and a window will show the result.
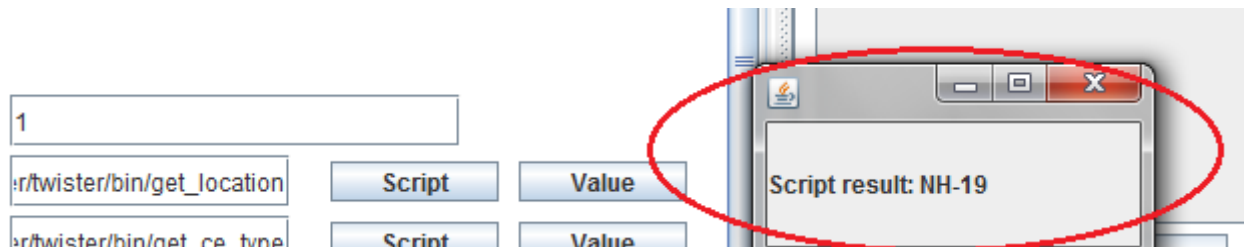
In order to be saved in the database, the variable must be referenced in the SQL query according to the ID field ('$ID_Location').

For our example, the SQL query could look like:

```
<sql_statement> INSERT into results (location) values ( '$ID_Location')</sql_statement>
```

14.4.2.3 Defining a UserSelect variable

In this scenario, the user can define a custom variable that can be populated by executing an SQL query on the database and selecting on of multiple options. This scenario applies in situations where user needs additional information to be saved into the database and the information can be obtained from database.

To define such a variable, the following syntax must be used:

```
<field ID="<ID_Var>" SQLQuery="<SQL Query>" Label="<GUI_Label>" Type="UserSelect"
GUIDefined="true" Mandatory="true " />
```

When such a variable is defined, a text box will appear automatically in GUI and the user can selected from the options offered by the SQL query that he defined.

For instance, let's say the user defines the field "EP Name":

```
<field ID="EP_Name" SQLQuery="select distinct ep_name from reports" Label="EP Name"
Type="UserSelect" GUIDefined="true" Mandatory="true " />
```

The GUI will present a field called "EP Name" and the user HAS to select one of the options by pressing the Database button.
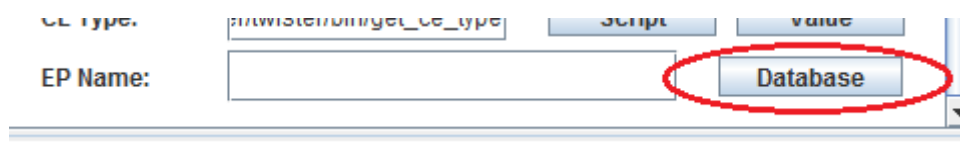
When the Database button is pressed, a new window will open and the user can select from the options.
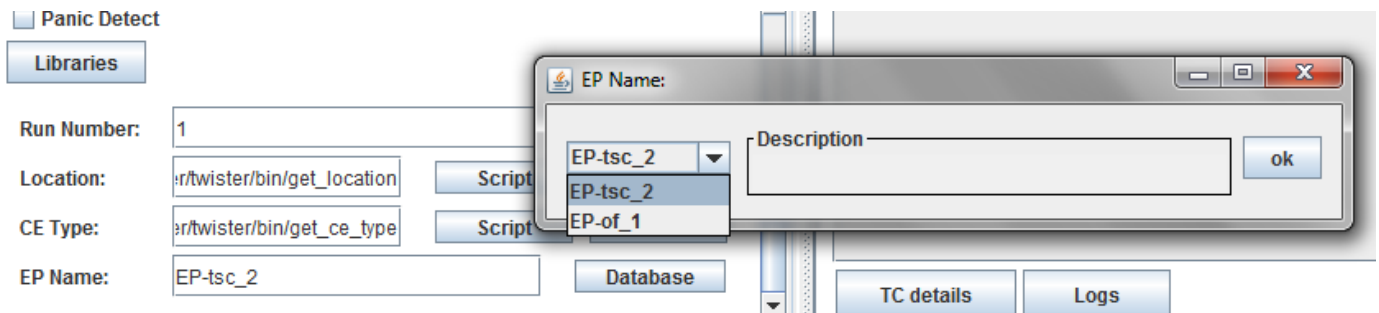
In order to be saved in the database, the variable must be referenced in the SQL query according to the ID field ('$EP_Name').

For our example, the SQL query could look like:

```
<sql_statement> INSERT into results(ep_name) values ( '$EP_Name')</sql_statement>
```

14.4.2.4 Defining a DbSelect variable

In this scenario, the user can define a custom variable that can be populated automatically by the framework by executing an SQL query on the database. The output of the SQL query must contain a single record. This scenario applies in situations where user needs additional information to be saved into the database and the information can be obtained automatically from database by executing a defined query.

To define such a variable, the following syntax must be used:

```
<field ID="<ID_Var>" SQLQuery="<SQL Query>" Label="" Type="DbSelect" GUIDefined="false"
Mandatory="true " />
```

When such a variable is defined the GUI doesn't show anything. No action from user is required, the variable is populated automatically.

For instance, let's say the user defines the field "Max_Value":

```
<field ID="Max_Value" SQLQuery="select MAX(value) from reports" Label="EP Name" Type="DbSelect"
GUIDefined="false" Mandatory="true " />
```

No GUI action is required from user.

In order to be saved in the database, the variable must be referenced in the SQL query according to the ID field but with a different syntax (@Max_Value@).
For our example, the SQL query could look like:

```
<sql_statement> INSERT into results(value) values (@Max_Value@)</sql_statement>
```

11.4.3 Creating reports

Based on db.xml file, the Twister framework can be configured to generate reports using the information saved into the database. . The framework doesn't impose a fixed structure of database and the reports, so the user must specify the information about the tables that store the results and what are the SQL statements used to generate the reports.

The information about results saving into database is stored in db.xml file between < reports_section ></ reports_section> tags.

In this section you can define the *fields*, the *reports* and the *redirects*.

The **fields** must have the following properties:

- ID: represents the name of the field and MUST be unique;
- Type: there are 2 types of fields: UserSelect (where you must define an SQL query) and UserText (free text, you can write anything);
- SQLQuery: this is required only for UserSelect fields. The query should select two columns: the first is the ID and the second is a name, or a description of the respective ID. If the table where you have the data doesn't have any description associated with the ID, you can use only the ID;
- Label: a short text that describes the field, when the user is asked to select a value.

***Examples** of report fields*:

```
<field ID="Dates" Type="UserSelect" Label="Select date:"
SQLQuery="SELECT DISTINCT date FROM results_table1 ORDER BY date" />

<field ID="Statuses" Label="Select test status:" Type="UserSelect"
SQLQuery="SELECT DISTINCT status FROM results_table1 ORDER BY status" />

<field ID="Releases" Label="Select release" Type="UserSelect"
SQLQuery="SELECT DISTINCT SUBSTRING(build, 1, 6) AS R FROM results_table1 ORDER BY R" />

<field ID="Other" Type="UserText" Label="Type other filters:" SQLQuery="" />
```

The reports must have the properties:

- ID: represents the name of the report and MUST be unique;
- Type: there are 4 types of reports: Table (an interactive table is generated; the table can be sorted and filtered dynamically), PieChart, BarChart and LineChart (they show both the chart and the table; for PieChart report, the SQL query must be defined in such a way that the first column is a string describing the data, and the second column is an integer or float data; BarChart and LineChart must also have the query generate 2 columns, the first is a number and the second is a label or another number);
- SQLQuery: all reports must define an SQL query. If the type of report is Table, it can select any number of fields (although it's recommended to use a maximum of 10, to fit on the screen without

having to scroll to the right). If the report is a chart, you must select only 2 columns. The query can use any, or none of the fields described above. Each field name **must be surrounded by `@`**. When a field is used in the query, the reporting framework will require the user to choose a value, before displaying the report.

Examples or reports:

```
<report ID="Details (build)" Type="Table"
SQLQuery="SELECT * FROM  results_table1  WHERE build='@Build@' ORDER BY id" />

<report ID="Details (suite)" Type="Table"
SQLQuery="SELECT * FROM  results_table1  WHERE build='@Build@' AND suite_name='@Suite@' " />

<report ID="Summary" Type="PieChart"
SQLQuery="SELECT status AS 'Status', COUNT(status) AS 'Count' FROM  results_table1  WHERE build=
'@Build@' group by status " />

<report ID="Pass Rate" Type="LineChart"
SQLQuery="SELECT Build, COUNT(status) AS 'Pass Rate (%)' FROM  results_table1  WHERE Build LIKE
'@Release@%' AND status='Pass' GROUP BY Build"
SQLTotal="SELECT Build, COUNT(status) AS 'Pass Rate (%)' FROM  results_table1  WHERE Build LIKE
'@Release@%' GROUP BY Build" />
```

An optional field called Folder can be used for a report to group it in a specific category.

Example or reports grouped in a category called "Details User":

```
<report ID="Details (Select User)" Folder="Details User" SQLQuery="SELECT DISTINCT tc_user AS
'User',ce_type AS 'CE Type',ce_hostname AS 'CE Host',suite_name AS 'Suite',tc_path AS 'TC Name',tc_name
AS 'TC File',tc_revision AS 'TC Revision',run_nb AS 'Run Number',tc_status AS 'TC Status',
CONCAT(substring (tc_date_finished,3,2), substring(tc_date_finished,6,2),
substring(tc_date_finished,9,2)),',', substring(tc_date_finished,12,8)) AS 'Date Finished' from reports
where tc_user='@EP_user@' AND ce_hostname='@CP_Host@' AND ce_location='@Location@'" SQLTotal=""
Type="Table" />

<report ID="Details (Type User)" Folder="Details User" SQLQuery="SELECT DISTINCT tc_user AS
'User',ce_type AS 'CE Type',ce_hostname AS 'CE Host',suite_name AS 'Suite',tc_path AS 'TC Name',tc_name
AS 'TC File',tc_revision AS 'TC Revision',run_nb AS 'Run Number',tc_status AS 'TC Status',
CONCAT(substring (tc_date_finished,3,2),substring(tc_date_finished,6,2), substring(tc_date_finished,9,2),
',', substring(tc_date_finished,12,8)) AS 'Date Finished' from reports where tc_user like
'%@EP_type_user@%' AND ce_hostname='@CP_Host@' AND ce_location='@Location@'" SQLTotal="" Type="Table" />
```

The user details reports will be grouped as in the following picture:
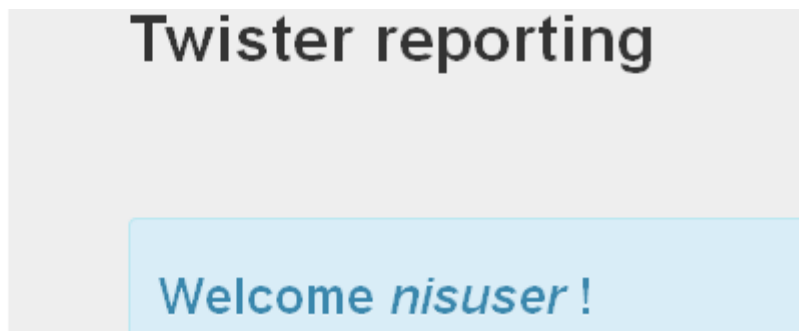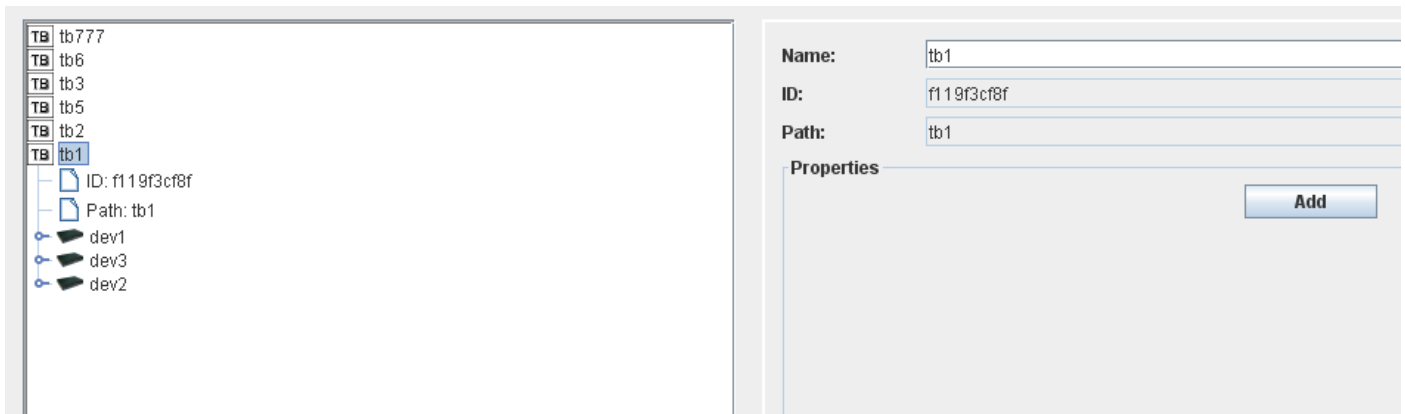
The **redirects** must have the properties:

- ID: represents the name of the redirect and MUST be unique. Ideally, the ID should start with the word `goto`;
- Path: is the full path to a HTML page. It can be a link to a static page, to PhpMyAdmin for the current database, or a user web page served by any web server.

Examples of redirects:

```
<redirect ID="goto PhpMyAdmin" Path="http://my-server/phpmyadmin" />

<redirect ID="goto PHP Report" Path="http://my-server/some-report.php" />
```

14.5 - Configure the devices (TestBed)

The Resource Allocator server is used to view and edit the Testbed and SUT properties.

The testbed is <u>global</u> for all users.

Each device == node == resource must have a name and some properties in the form of {*key* : *value*}.

The name of a resource must be unique in its parent. For example you cannot have more nodes called `Device1` in parent `Testbed1`, but you can have nodes called `Device1` for both `Testbed1` and `Testbed2`. This is important, because each resource can be accessed using its ID, or its full path (just like a Unix file system).

The Resource Allocator server exposes a simple API inside the Twister tests, for accessing the resources:

- **get_resource**(ID or full path, [unicode | str]) - returns a dictionary containing all the node properties. All the keys and values returned from this command will be Unicode strings, by default. You can convert them into normal strings by sending the extra parameter `str`;
- **set_resource**(name, parent ID or full path, properties in dictionary or JSON string)
- This function is used to CREATE and MODIFY nodes. If the resource is created, the ID of the new resource is returned. If the resource is updated, the function returns True.
Example: **set_resource**('module1', '/tb1/device2', '{"ip":"10.0.0.1", "port":"80"}') ;
- **rename_resource**(ID or full path, new name) - renames resources or their properties;
- **delete_resource**(ID or full path) - deletes resources or their properties;
- getResourceStatus(ID or full path) - obsolete.

TWISTER USER GUIDE

Examples:

Let's say there are 3 testbeds: *tb1*, *tb2*, *tb3*. Each testbed has 2 devices: *dev1* and *dev2*. Each device has 3 modules: *mod1*, *mod2*, *mod3*.

To access module 2 from device 1 from testbed 3, you can use: **get_resource**('/tb3/dev1/mod2').
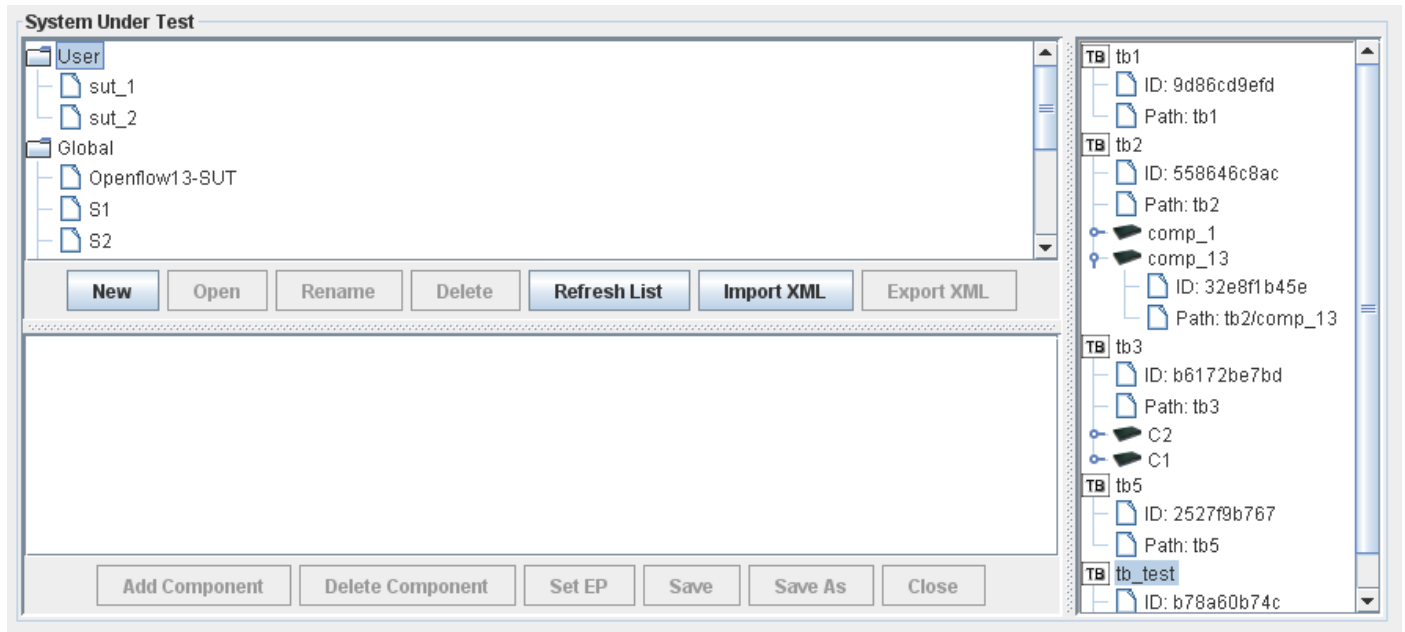To access testbed 1, you can use: **get_resource**('/tb1').

To rename device 2 from testbed 2, you can use: **rename_resource**('/tb2/dev2', 'dev_x2'). Note that the new name is just a string, not the full path.

In **production mode**, to be able to Set, Rename or Delete a resource, a user must have the *CHANGE_TESTBED* role.

14.6 - Configure the SUTs

This is where you can edit the SUTs (systems under test). A SUT is a collection of TestBeds or devices, linked together in a system that will be tested as a whole.

You can create multiple SUTs and every SUT reside in it's own file in JSON format. The SUT's can be system or user type and they reside in shred directory or user specific directory.

You configure the path for shared SUT files in "Configuration->Paths->System SUT files" path field. This field "SUT files path" must be added in Twister GUI.

You configure the path for local SUT files in "Configuration->Paths->User SUT files" path field. This field "SUT files path" must be added in Twister GUI.

All files from shared directory and user directory are listed in "Configuration->System Under Test" section and can be used for testing.

All the SUT files are displayed as a list in upper left window. If no SUT file is opened for edit, the buttons of lower left window are greyed out.

Open a SUT file for edit, using "Open" button. If a SUT file is opened by another user for edit, this is shown as "Reserved by <username>". The GUI doesn't update periodically the list of reserved files. You must hit "Refresh list" button to check the reserved files.

When an SUT is opened and it is already reserved for other user, it is opened as view-only and the buttons in lower left window are greyed out.

When a SUT is opened and it is not already reserved for a different user, it gets reserved and the buttons in lower left window are available.

Rename an SUT file by hitting "Rename" button. You must have proper system rights to change a file. If the file cannot be renamed, a dialog window is displayed with message "Cannot rename SUT file".

Delete a SUT file by hitting "Delete" button. You must have proper system rights to delete a file. If the file cannot be deleted, a dialog window is displayed with message "Cannot delete SUT file"

TWISTER USER GUIDE

Refresh the list of SUT files, by hitting "Refresh list" button.

Import an SUT file from an external XML file, by hitting "Import XML" button. A navigation window opens allowing you to select the XML file to be imported. After import, the new SUT file is shown in upper left window.

Export an SUT file by hitting "Export XML" button. A navigation window opens to allow you to specify the name of the exported file and the directory.

Create a new SUT file by hitting the "New" button. This enables the buttons in lower left window.

Define the content of the new SUT, by drag & drop TB components in lower left window.

Create SUT components using "Add component" button.

Delete SUT components using "Delete component" button.

Save the SUT content using "Save" button. This operation doesn't release the SUT file and the you can continue to edit it.

Select a different name for the SUT file, using "Save As" button. When the SUT file name is changed, the previous file is released and the new created file is reserved for edit. For instance, you can open SUT_1 file for edit. The SUT_1 gets reserved. You change the content of SUT_1 and save it as SUT_2. In this moment, the SUT_1 is released and the new SUT_2 file is reserved for edit.

You can complete SUT editing using "Close" button. When the SUT file is closed, the SUT file is released and the buttons in lower left window become greyed out.

The changes done in SUT, become available for read to other users, as soon as you save them.

If timeout expires and you didn't save the changes, the SUT file is released and all unsaved changes are lost.

If you logout from GUI, the SUT file is released and all unsaved changes are lost.

You can change the SUT only if CHANGE_SUT role is set on user's group.

A new group role LOCK_SUT is created for groups.

An user belonging to a group with LOCK_SUT privilege can lock an SUT so it cannot be changed by other users; lock and unlock options are not available for users that do not belong to a group with LOCK_SUT privilege.

Lock operation is preserved if the user logout from GUI or his GUI timeout expires. The SUT must be unlocked explicitly by the user. The lock operation is cancelled if the CE is restarted.

When an SUT is locked by an user, the message "Locked by: <username>" is displayed at the right of the SUT name.

A SUT cannot be locked as long as it's reserved by an user.

The Resource Allocator server exposes a simple API inside the Twister tests, for accessing the SUTs:

- get_sut(ID or full path, [unicode | str]) - returns a dictionary containing all the SUT properties. All the keys and values returned from this command will be Unicode strings, by default. You can convert them into normal strings by sending the extra parameter `str`;

When you use the get_sut function, for example get_sut('SUT1'), you will receive a hash-map like this:

```
{'path':'SUT1', 'meta':{'epnames': 'EP-1001'}, 'id':'a322908eab', 'children':['5cae3abaef',
'c76db917e6']} ; Each child contains a property called "_id" that is the ID of a Testbed or Device.
```

- set_sut(name, parent ID or full path, properties in dictionary or JSON string)

- This function is used to CREATE and MODIFY SUTs. If the SUT did not exist before, the ID of the new SUT is returned. If the SUT is updated, the function returns True.
Example: set_sut('SUT11', '/', '{"epnames":"EP-1001;EP-1002"}') ;

- rename_sut(ID or full path, new name) - renames SUTs or their properties;
- delete_sut(ID or full path) - deletes SUTs or their properties.

In **production mode**, to be able to Set, Rename or Delete a SUT, a user must have the *CHANGE_TESTBED* role.

14.7 - Define Test Configurations
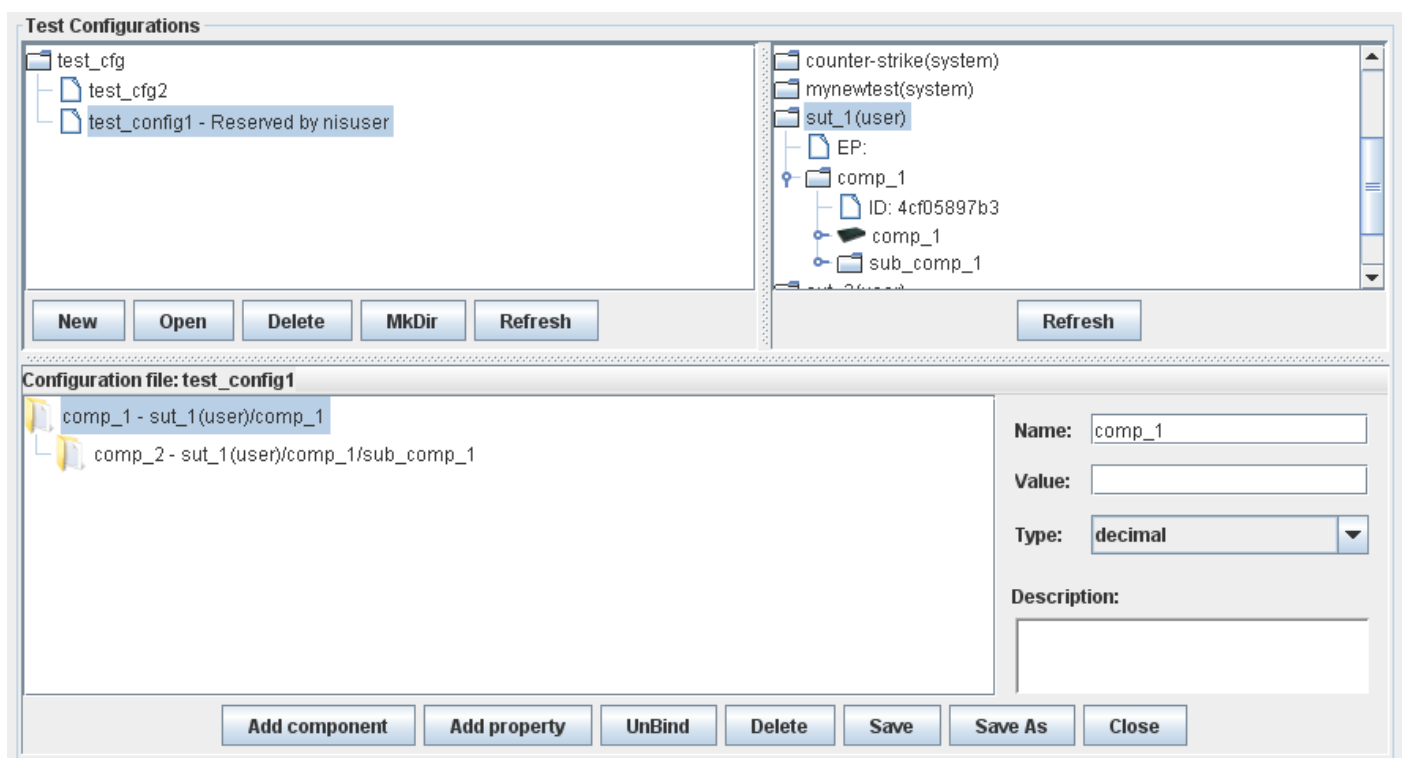
This is where you can edit the test configurations that can be used in test cases. A test configuration is a collection of components and operational parameters that can be applied on the SUTs.

Every configuration is saved in a separate file and stored in the directory defined in Configuration-> Paths-> Test Configuration Path.

In order to create a new configuration file, press on "New" in the applet. You will be asked to introduce a name for the configuration file. If needed, you have the possibility to organise the configuration files in a hierarchical structure, by creating new directories. To create a new directory, press on "MkDir" in the applet. You'll be asked to introduce a name for the new directory.

TWISTER USER GUIDE

In order to define the configuration needed for testing, the configuration file must be opened using "Open" button. When the configuration file is opened, it is reserved for current user for edit. Other users cannot edit the same configuration file to while it is reserved. They can only open it in view-only mode.

To add a new component in the configuration file, press on "Add Component" in the applet. You will be asked to introduce a name for the new configuration element. The name MUST be unique. If you want to add a parameter for the configuration element, press on "Add parameter" in the applet. You will be asked to introduce a name for the parameter and its value.

You can rename a configuration element by changing it's name in lower right window.
In order to use your config files in association with tests, in the first tab (the suites), you should right click on a file and select from the available config files:
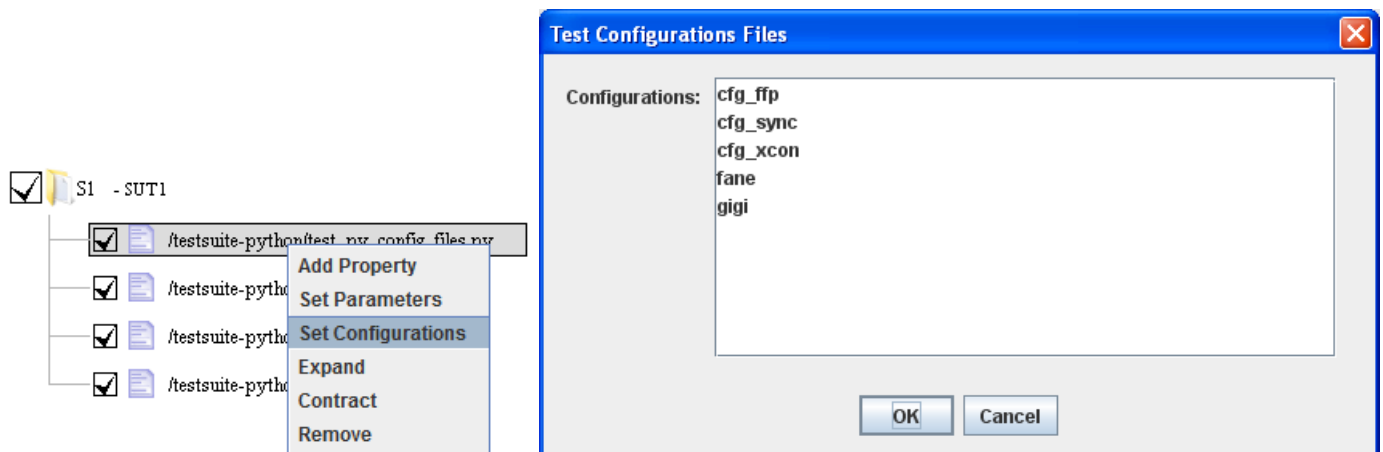
To access the config files from within the test, the list **CONFIG** is exposed (containing all the selected configurations) and the function **get_config**, both available in Python and TCL tests.

The function **get_config** can be used in conjunction with **CONFIG**, or you can specify a custom config path, like this:

```
get_config( CONFIG[0], '/Feature_1/Line_rate' )
get_config( '/home/user/twister/config/test_config',  '/Feature3' )
```

A suite of global params compatible .XML files
You can have hundreds of configuration files organized in folders or subfolders.
In the project tab where you define your suites and tests to be run you can assign one or more configuration files to each test individually.

14.8 – Binding a SUT to a configuration file

TWISTER USER GUIDE

There could be scenario when you want to write a generic test case where you don't want to hardcode the name of the SUT or it's components. You can use some generic names inside the test case (e.g. first_switch) and use the Twister GUI to create the bind to a real device. If later on, the test case must be applied against a different device, there is no need to edit the test case, just use Twister GUI to change the bind to a different device.

You can define what components of configuration file apply to what SUT components. The information about the defined bindings is saved in user home directory file
$TWISTER_CLIENT_PATH/config/bindings/bindings.xml
Every user has own binding file.

The binding is done in Test Configuration section, by drag&drop of an SUT component to a configuration component.
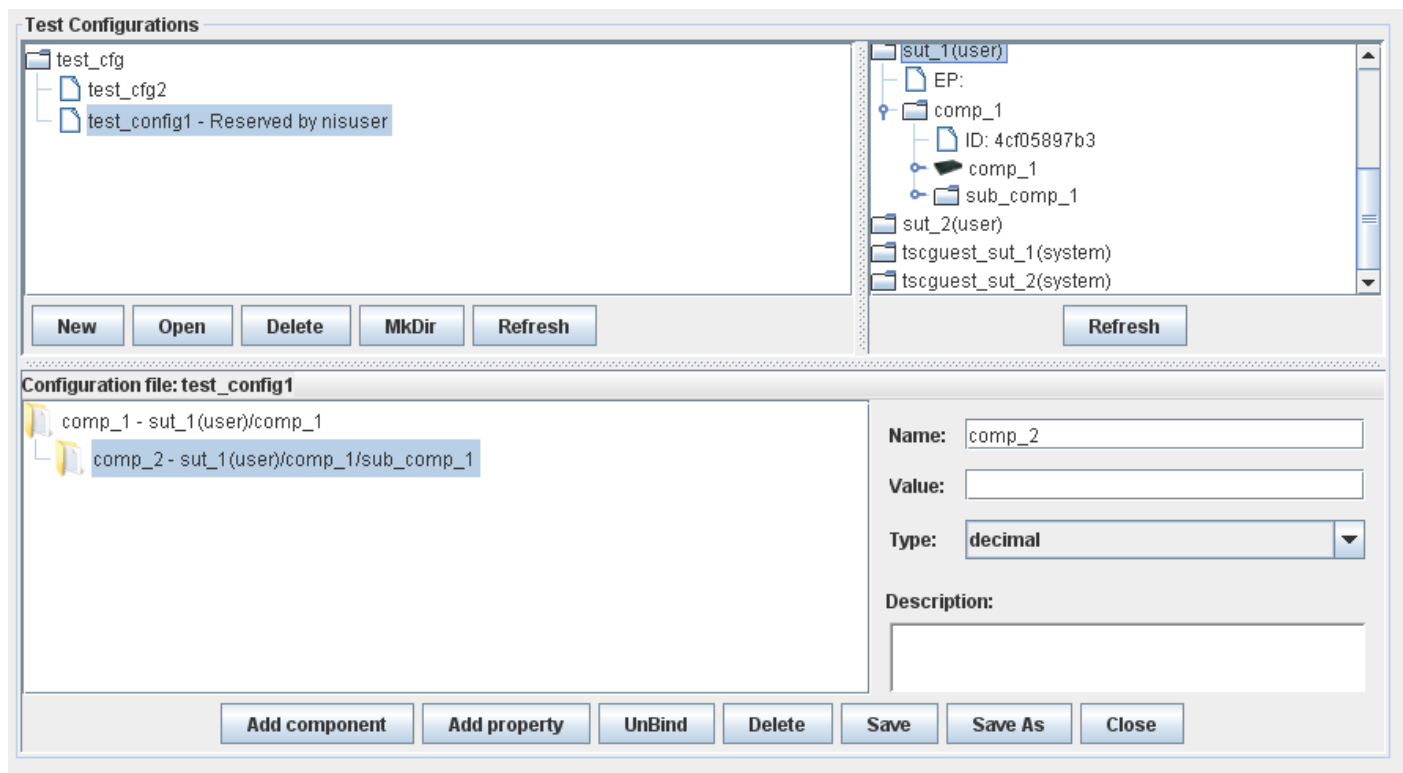
If there is no test configuration file opened for edit, the binding is done as global (DEFAULT). In this scenario, you can bind only the global components and the specific test configuration bindings are available for view only.
If a test configuration file is opened for edit, you can create bindings specific to that file. The other bindings (global or for a different test configuration file) are not available.

Having global and per test configuration file bindings, provides a flexible mechanism to user to define the bindings. User can associate a test configuration file to a test case and another test configuration file to rest of the test case(s). For some of the test cases, the system can use the default binding schema, but for a specific list of test configuration files, the system can use the specific defined bindings and can override the default bindings.

The binding is reflected in the binding.xml file by the following structure:

    [default_binding]
    Component_1/session_ip : SUT1/mgmt/IP
    Component_2/session_port : SUT1/mgmt/port
    Comp_2 : sut_1/comp_3/sub_comp_1
    [test_configuration_1]
    Component_1/TX_port : SUT1/switc_3/port_5
    Component_2/RX_port : SUT1/switch_2/port_4
    Comp_2 : sut_1/comp_1/sub_comp_1

A global binding applies when the specific test configuration file is not defined. If the binding for a test configuration file is defined, it overrides the global binding.

API to retrieve the defined binding is provided. This API is used by test cases to retrieve the binding information. The API returns the SUT node associated with the configuration component. The prototype is

    get_bind_id("component_name","test_config_name") - provide SUT component ID
    get_bind_name("component_name","test_config_name") - provide SUT component name

where:
- component_name is the node name defined in the configuration file (e.g. Component_1/TX_port)
- test_config_name is the name of the configuration file set as parameter for the test case; in case user doesn't set a configuration file for the test case, the configuration_file_name is null (None python type) and the API will check in default_binding section

If the test_config_name is not provided, the API returns the information from default binding.
If the component_name is not found in test_config, it is searched in default binding.

To unbind a component from a SUT, you have to select the component and hit "UnBind" button.

14.9 - Configure the global parameters

Global parameters are variables available in all test files. There is no need to import any library.
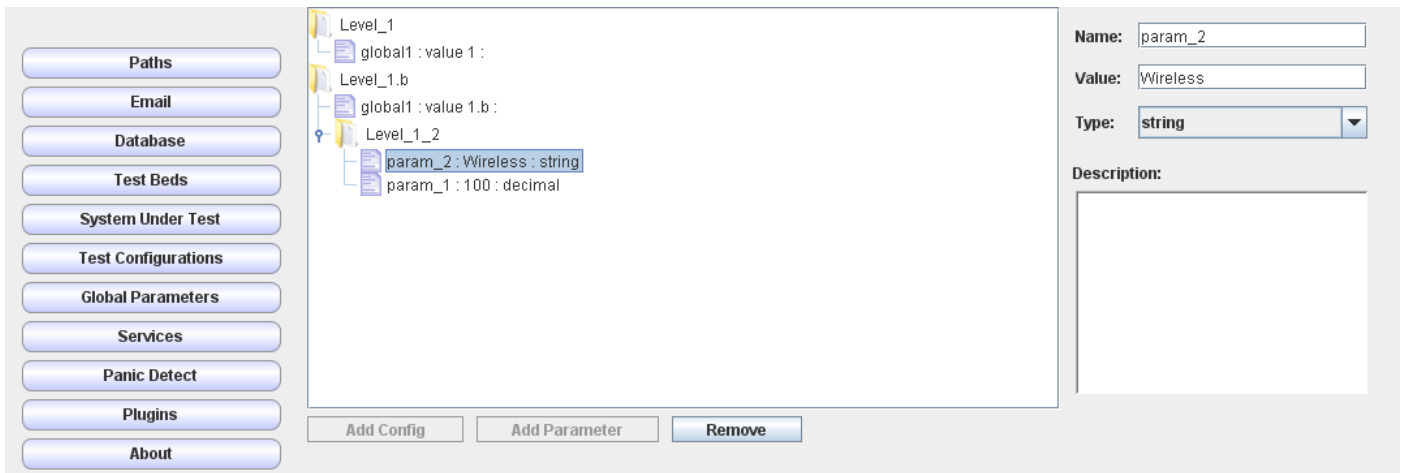
The parameters are stored per user!

The API is very simple: in any test, use **get_global**(name) and **set_global**(name, value). You don't need to import anything.

If the *name* is a folder (not a variable with a value), instead of a value, getGlobal returns a dictionary. If the parameter *name* doesn't exist, getGlobal returns *False*.

The setGlobal function will update, create or delete parameters. If the name exists, it is updated. If it doesn't exist, it is created. If you use **set_global** (name, False), the parameter is deleted.
The changes made by setGlobal are temporary and will RESET every time the tests start running.

There are 3 types of parameters:

1. the default parameters stored in the configuration, that are saved in globals.xml file ;
2. the serializable parameters saved by the test files are shared between all Eps from a user ;
3. the complex, not serializable parameters are stored on the EP that is running the tests.

14.10 - Configure 'panic detect'

Panic detect is a mechanism that allows the users to add `expressions` that will be searched in the test logs. If any of the expressions is detected, the execution STOPS.

An `expression` can be a normal string, or a regex.

This is useful to check for core dumps and critical errors; in these extreme cases, it's useless to continue the execution of any test.

14.11 - Services and Plugins

Print screen with Services (in production mode, using Services requires the *CHANGE_SERVICES* role)

Print screen with Plug-ins (in production mode, using Plug-ins requires the *CHANGE_PLUGINS* role)
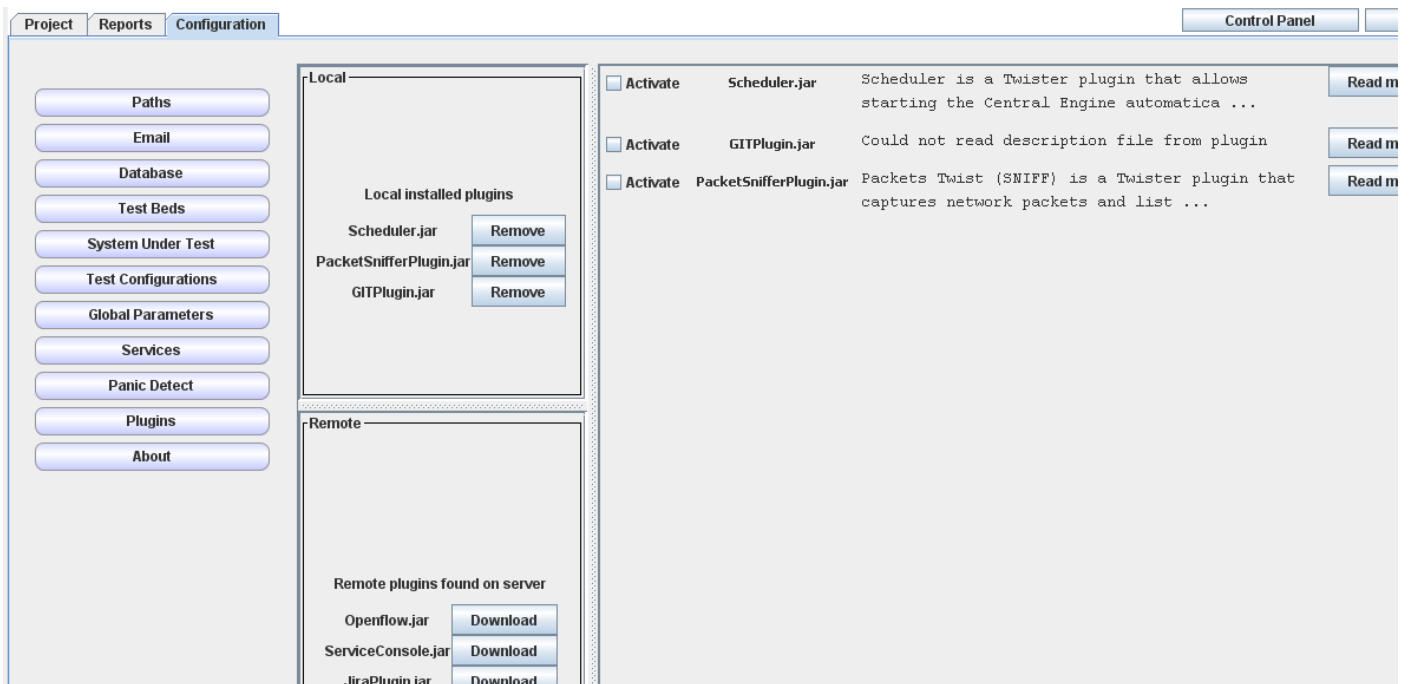
More about this in the `**Twister Libraries, Plugins, Services**` help file.

14.12 - User management

Print screen with Users Management

TWISTER USER GUIDE

Since *version 2.020*, every user that connects to the Central Engine must be registered in Twister, must be part of a group, and implicitly has a set of roles.

When installing **Twister Server** for the first time, you must go in the server install folder and edit the `config/users_and_groups.ini` file. In the section [**users**], the first user by default is [[**admin**]] ; this name must be replaced with **your system user**. This way, you will become a Twister Administrator and will be able to create, edit, or delete other Twister users.

This manual edit must be done only the first time, the rest of the users should be changed using the GUI (the applet). You should also make sure that no one can view the *users_and_groups* file except for the ROOT user and make sure that Central Engine runs as ROOT, so it will be able to read the file (or if you plan to update the users using the GUI, Central Engine must also be able to write).

The roles are enabled only when the server type is `**production**`, in the file `config/server_init.ini`. If the server type is `**development**`, users have all the roles, except for CHANGE_USERS.

If a user cannot be found in *users_and_groups* file, **he CANNOT** use Twister. In `users_and_groups.ini` file, a Twister user has a title (his own name) and 3 fields:

- **groups_production** (a list of groups that this user belongs to when the server is configured in production mode. The groups will give him a set of Roles. The group can be 0 *zero*, which means zero Roles);

- **groups_development** (a list of groups that this user belongs to when the server is configured in development mode. The groups will give him a set of Roles. The group can be 0 *zero*, which means zero Roles);
- **timeout** (the amount of seconds until the applet automatically logs out the user);

The timeout and key are very important in any server mode (development or production).

The groups have a set of unique roles. This is the complete list:

- RUN_TESTS   - Can run tests (server + applet)
- EDIT_TC   - Can edit test files (applet)
- CREATE_PROJECT   - Can create new projects (applet)
- CHANGE_PROJECT   - Can change defined projects (applet)
- DELETE_PROJECT   - Can delete projects (applet)
- VIEW_REPORTS   -Can open the reports (server + applet)
- CHANGE_FWM_CFG - Can change his main config (applet)
- CHANGE_GLOBALS   - Can change global parameters (applet)
- CHANGE_DB_CFG   - Can change database config (applet)
- CHANGE_EML_CFG   - Can change e-mail config (applet)
- CHANGE_PLUGINS   - Can load/ unload plugins (applet)
- CHANGE_TESTBED   - Can change the global testbed (server + applet)
- CHANGE_SUT   - Can change the global SUTs (server + applet)
- CHANGE_SERVICES   - Can start/ stop services (server + applet)
- CHANGE_USERS   - Can create, change and delete users (server + applet)

You can add a new user by pressing on "Add User" button. A new window opens.
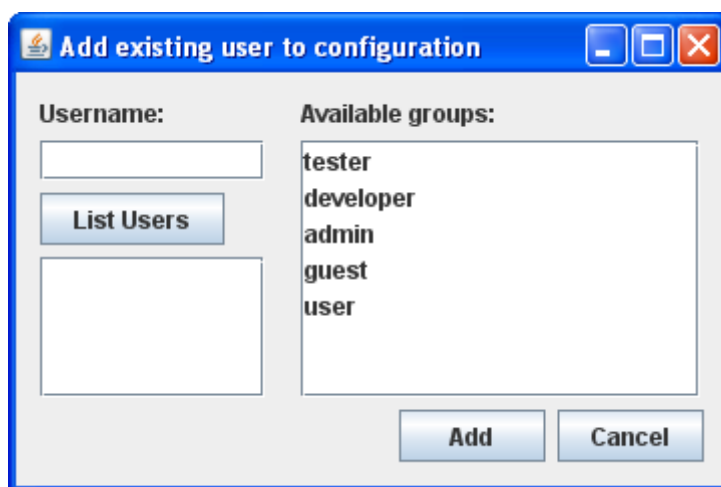
You can insert the name of the user in the "Username" text box or you can select the username from the available users by pressing "List Users" button.
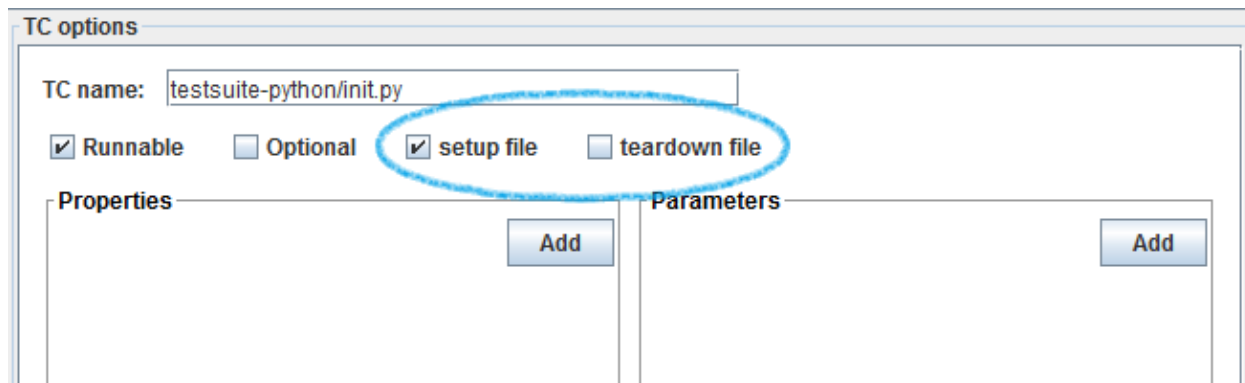
One you enter a username (or get it from list of users) and select it's group, you must press "Add" button

to add the user. The server will validate the username and if it's not a valid user you will get an error message.

14.13 - Set-up and Tear-down controls

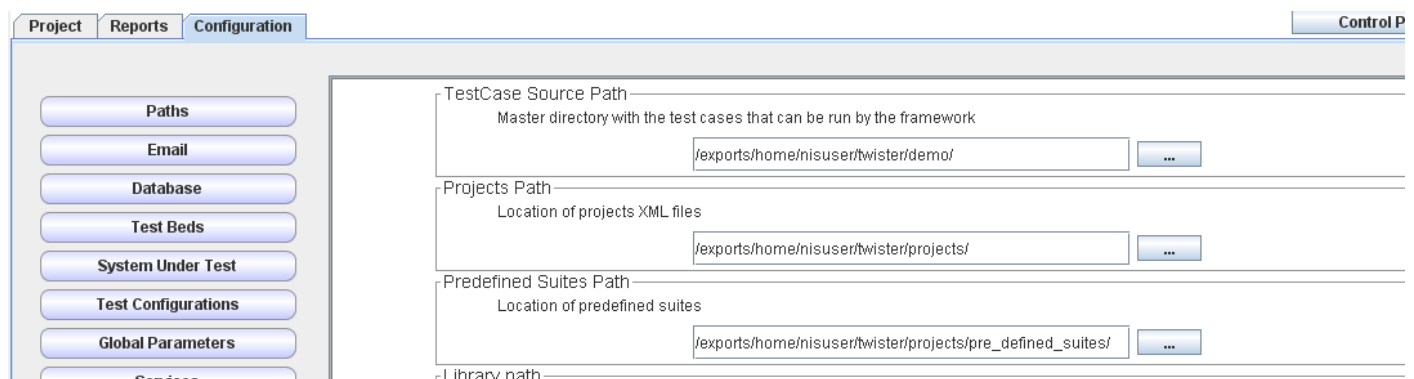Setup and Teardown controls are used to mark scripts as setup or teardown scripts.

The setup scripts can be used to setup a configuration on a network and the teardown to clear the configuration.

- *Setup* scripts are executed at all times (they have to be runnable) and if they *fail*, the entire Suite is considered *Failed*. Setup files are NOT saved into the database!
- *Teardown* scripts are executed at all times (they have to be runnable). They are used to clean-up a suite, even if the suite was aborted because of a failed Setup. Teardown files are NOT saved into the database!
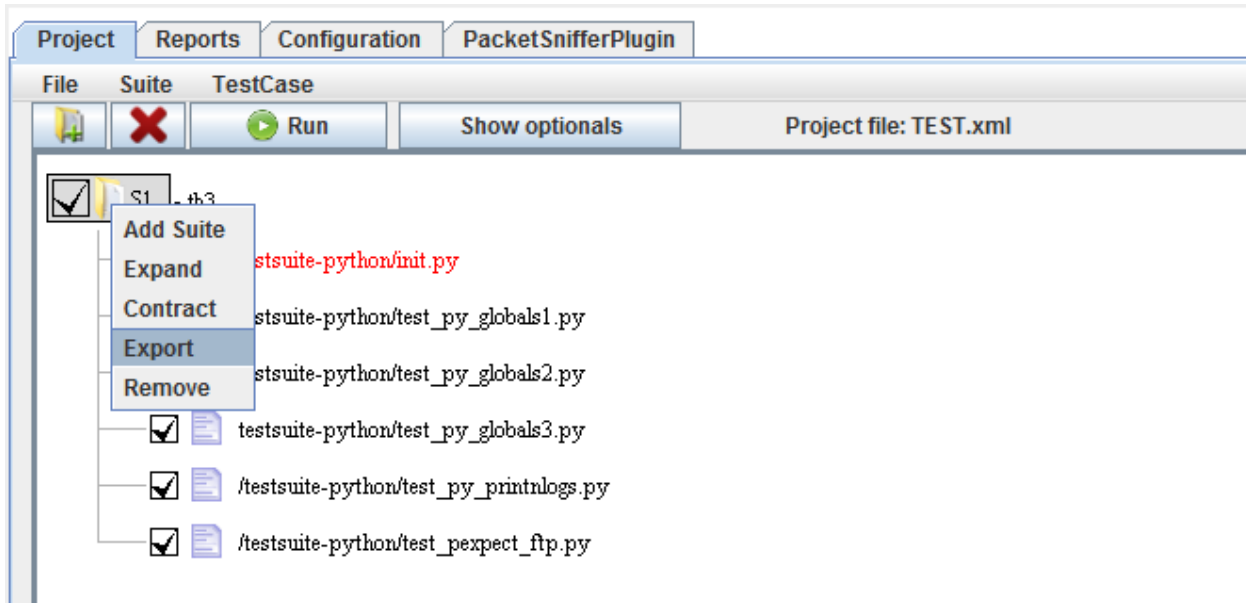
14.14 - Test case/Suites management

You can save your suites for a later use as predefined suites.
Before using this feature you must define the path where the predefined suites will be saved. To do that, go to Configuration-> Paths tab.
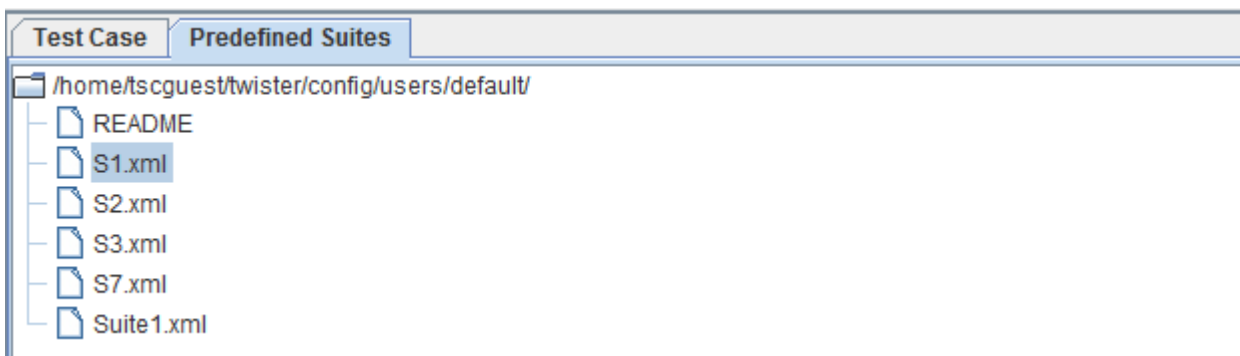
After you define your suite you can save it by right clicking on the suite that you want to save and enter a name for it.

The saved suites can be found in the **Predefined Suites** tab. You can drag and drop them in the main window where you define your tests to be run.

14.15 - User levels

This feature is an enhanced user management with CE typing and multiple user groups: "developer" and "tester", and session management.

    A. CE server typing

Twister is supporting two types of servers: <u>production</u> and <u>development</u>. This information is displayed on GUI to allow the user to find out on what server types he's connected.

    i. The information is stored in a CE configuration file (**server_init.ini**);

ii.   The default value of ce_server_type is no_type. If you use `no_type` you will have access to all the Twister features;

iii.   As administrator you can change control of the file through UNIX group permissions;

iv.   After the ce_server_type is changed, a full restart of CE is required.

Example of server_init.ini:

```
# Valid types: no_type, production, development
ce_server_type = "production"
# The server location can be any string
ce_server_location = "TwisterLand"
# How much logs will be displayed (FULL,DEBUG,INFO,WARNING,ERROR,CRITICAL)
Verbosity = INFO
```

B.   User groups

The twister GUI, it's providing a panel where the groups are managed. By default, there are 3 twister user groups: admin, developer and tester. These can be easily changed.

i.   The twister admin is created by the Unix/NIS admin.

ii.   The twister admin can add, delete users (identified by unix_name such as tfisher), and set the users group.

iii.   The users can be moved from one group to another by the twister admin.

iv.   If a user belongs to the tester group, there will be some parts of the GUI that are greyed out and the user doesn't have access to that functionality.

v.   The GUI login credentials (username/ password) are exported to the test cases as internal variables.

TWISTER USER GUIDE

The Unix/NIS users will be created by the IT team and not from Twister GUI.
Once a user has a Unix/NIS account, the Twister admin will be able to get the list of Unix/NIS accounts and select which one can be added into a group.

If a Unix/NIS account is not present in any Twister groups, he'll not be able to run tests even if he has a UNIX system/NIS account!

C. About screen.

The Twister framework is providing a CE configuration file to allow enterprise-specific information to be displayed in the **Welcome** and **About** screens.

To configure what it is displayed in the About screen you have to create/edit a logo.txt file which must be placed in the main directory where the GUI is loaded from (ex: /var/www/twister).
You can also add an image to the About screen which must be placed in the same directory as logo.txt and must be named logo.png. The image will be resized automatically by the GUI applet and displayed at this dimension W=230px, H=100px.

Logo example:
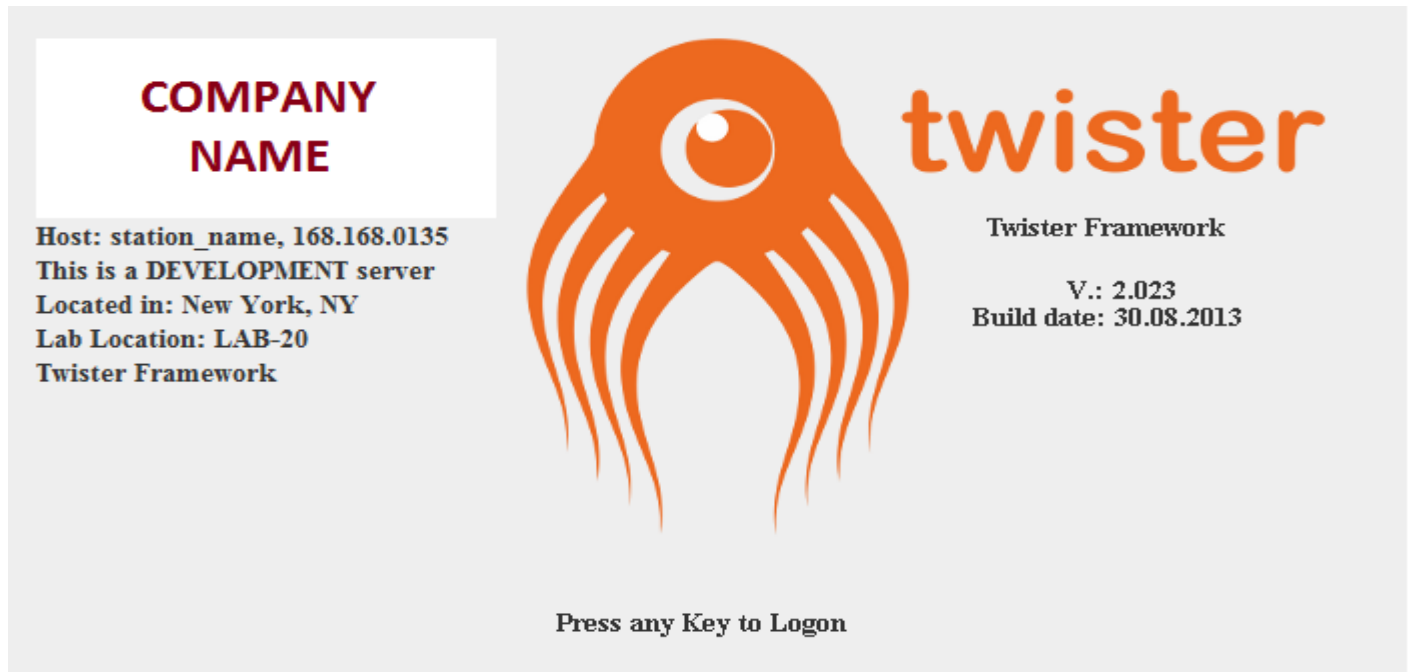
D. User Information

In the GUI, Twister is showing the activated users and the groups they belong to.
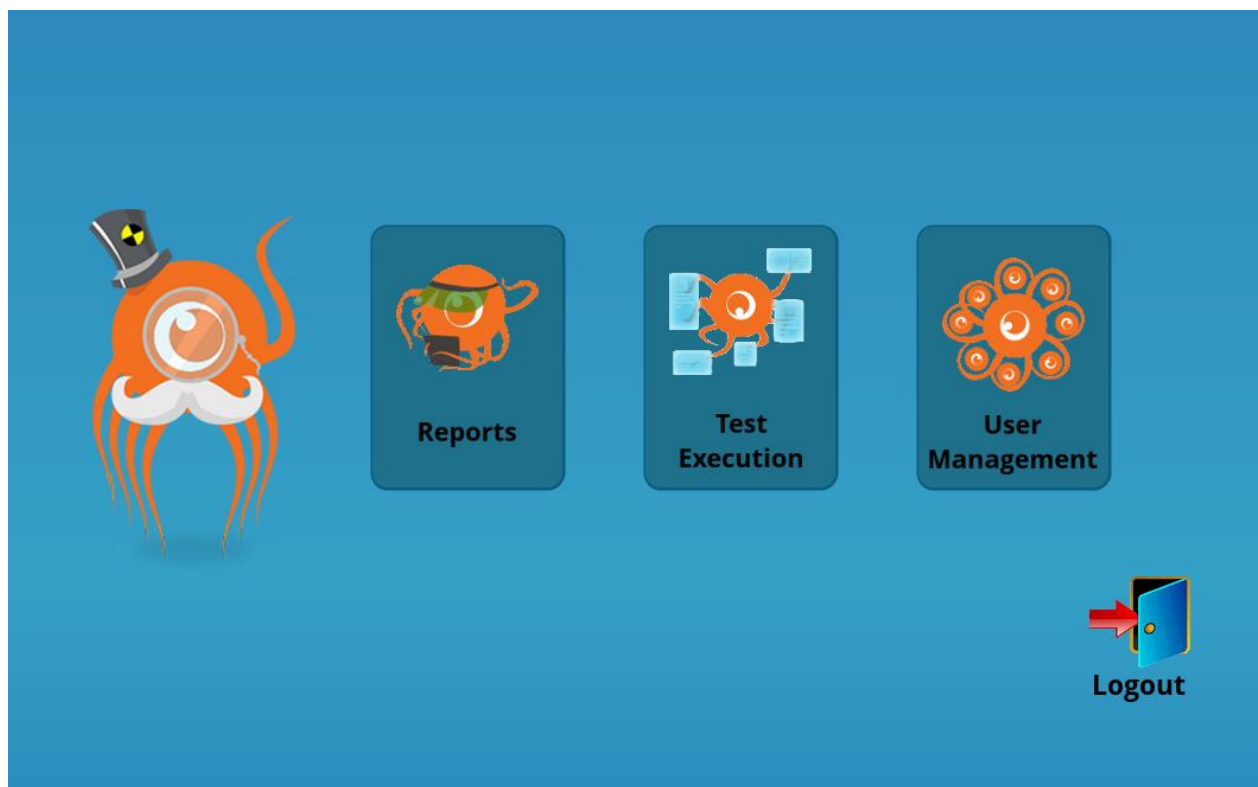
TWISTER USER GUIDE

E. Welcome Screen

Twister is providing a welcome screen same as the about screen from Configuration -> About screen, and indicates that the user should `Press any Key to Logon`.
Doing that the login pop-up is presented. The welcome screen is also presented when post-logout, Timeout, Cancel and login failure.

After you login the Control Panel screen will be shown.

F. Session Termination

i. Twister has a Logout button which returns you to the Welcome screen.
ii. Twister has an inactivity timer - User is logged off if no activity for programmed time. (Off, 15 min, 30 min) set by the admin.
The value can be set by the admin in the User management panel for every user individually. The default value is 0 which mean that the timeout is OFF.

14.16 - Test case details descriptors

In a test case you can define any numbers of Meta-tags.

For example:

<title>This is the title.</title>

<description>This test prints and sends some logs.</description>

<tag></tag> displays **Tag:**<stuff>

<meta></meta> displays **Meta:<**stuff>

#<setup></setup> displays **Setup:**<stuff>

An example of how it can be used you can find it in *~/twister/demo/testsuites-python/init.py* file.

14.17 - Library management

When you install Twister the default path where the libraries are stored is /opt/twister/lib.
Besides this path every user can define a custom library path. You can use the both locations in the same time.
To define the custom path location, go to Configuration -> Paths and set the path to the library location.

In a python test to import a library you must add the following line:

```
import <library_file>
```

The CE will first search for this library in the default library location and then in the custom library path.

14.18 - Log customization

The client log files are reset every time a new execution is started. This overwrites the logs.

Details:
  i. In Log Path in the Configuration Paths window, allow specification of a secondary (archive) /path for log files.
  ii. The use of this secondary /path can be Enabled/Disabled by checkbox. Disabled is the default setting and makes Twister operate in the current fashion.
  iii. When the secondary folder is Enabled: Prior to starting a new execution, the logs are closed, and moved to the secondary /path.
  iv. With the move, all file_names will have the same UNIX timestamp appended such as:

 log_running.log.1370288853
        Through
log_CLI.log. 1370288853
  v. The user can specify the same /path for both primary and backup.

14.19 - EP start-up

This feature provides a mechanism to allow the start of EPs on demand, when they are needed by the user. This feature is providing a registration mechanism between the EPs and the CE. The registration is for EPs to CE, so the CE will know what EPs are available for a user.

To manage the client you will need to run *start_client* script which is installed in client path.

NOTE: start_client script only works in **Linux**.

TWISTER USER GUIDE

The script can be started manually by the user, or automatically at system boot. Another mechanism to start the script automatically is to set this in a *.profile* file that is executed when the user logs-in on that machine. In the same time, the *.profile* file is executed automatically when a user logs into Twister GUI.

The *start_client* script is accepting the following options:
- Start – to start the client process
- Start silent - to start the client process and to disable printing of messages
- Stop – to stop the client process
- Restart – to restart the client process
- Status – to show the status of the client process

There can be 4 use cases for test case running, depending on local or remote machine and existence of epname.ini file.

1. Local testing, automated EP

This is the scenario when the system under test is connected directly to the server machine and the test cases are executed on this machine. There is no need to define epname.ini file.
When Twister client is started, it searches for epname.ini file. If this file doesn't exist, the Twister client assumes that server is running on the same machine and the server port is set to 8000.
The client automatically allocates an internal EP. The name of the EP is set to **<hostname>_auto**, where hostname is the name of the machine.
When testing is done with internal EP, there is no possibility to run parallel testing for the suites that are allocated to automated EP. All test cases are executed sequentially even if there are multiple suites allocated to the automated EP.
The **<hostname>_auto** name will be reflected in the log saved by the server into database.
The **<hostname>_auto** name will be available in the test cases as an internal variable.
User doesn't have to associate the **<hostname>_auto** to the SUT. This is done automatically if the user doesn't define an association. The server uses <hostname>_auto in this case.

2. Local testing, defined EPs

This is when the epname.ini file is defined and populated. It's the same behaviour as `**Remote testing, defined EPs**`. The automated EP is not created in this scenario.

3. Remote testing, automated EP

This is the scenario when the system under test is connected to a client machine different than server machine and the test cases are executed on client machine.
The epname.ini file must contain information about the IP address or the hostname of the server machine. If it doesn't, the client will try to connect on localhost:8000.

```
[AUTO]
AUTO_CE_IP   = <ip/hostname>
AUTO_CE_PORT = 8000
```

The information about CE_IP and CE_PORT is the only content of epname.ini. No information about EP name or EP enabled/disabled is needed.

When Twister client is started, it opens the epname.ini file. If information about server machine is found in epname.ini, it creates an automated EP and registers it as **<hostname>_auto**, where *hostname* is the client machine name.

The EP name will be reflected in GUI in the tab that shows the CLI logs received from EP.

When testing is done with internal EP, there is no possibility to run parallel testing for the suites allocated to the automated EP. All test cases are executed sequentially, even if there are multiple suites allocated to the automated EP.

The **<hostname>_auto** name will be reflected in the log saved by the server into database.

The **<hostname>_auto** name will be available in the test cases as an internal variable.

User has to associate **the <hostname>_auto** to the SUT, using the Twister GUI.

4. Remote testing, defined EPs

This is when the epname.ini file is defined and populated. The automated EP is not created in this scenario. The *epname.ini* file contains a tag called *EP_HOST* for every EP set for the user. This tag is used to indicate the host machine where that EP can be started and it is used to allow the user to control on what machine the EP is activated. This way, the user can have a common *epname.ini* file for many machines and he can control what EPs to be started on what machines.

The EP will register on the Central Engine as **<hostname>_EP-name**.

The **<hostname>_ EP-name** name will be reflected in the log saved by the server into database.

The **<hostname>_ EP-name** name will be available in the test cases as an internal variable.

Example:

```
[EP-1001]
CE_IP   =       11.126.32.20
CE_PORT =       8010
EP_HOST =       11.126.32.20
ENABLED =       1
```

The EP_HOST can be entered as a hostname, or IP address.

The EP_HOST tag is optional, so if it's commented out, or missing, the EP can be started on any machine without exception.

Enabled tag = 1/ 0 tag is also optional.

If the user changes the *epname.ini* file, the client process and the GUI must be restarted so the user can get the latest changes. CE restart is not needed, since the registration mechanism will update the available EPs.

If the CE is stopped, the client process will continuously try to register the EPs with the CE and it will succeed when the CE is up and running.

14.20 - EPs on Windows

Execution Processes can be run on Windows, in "portable mode" (without installing Twister), with roughly the same features as on Linux.
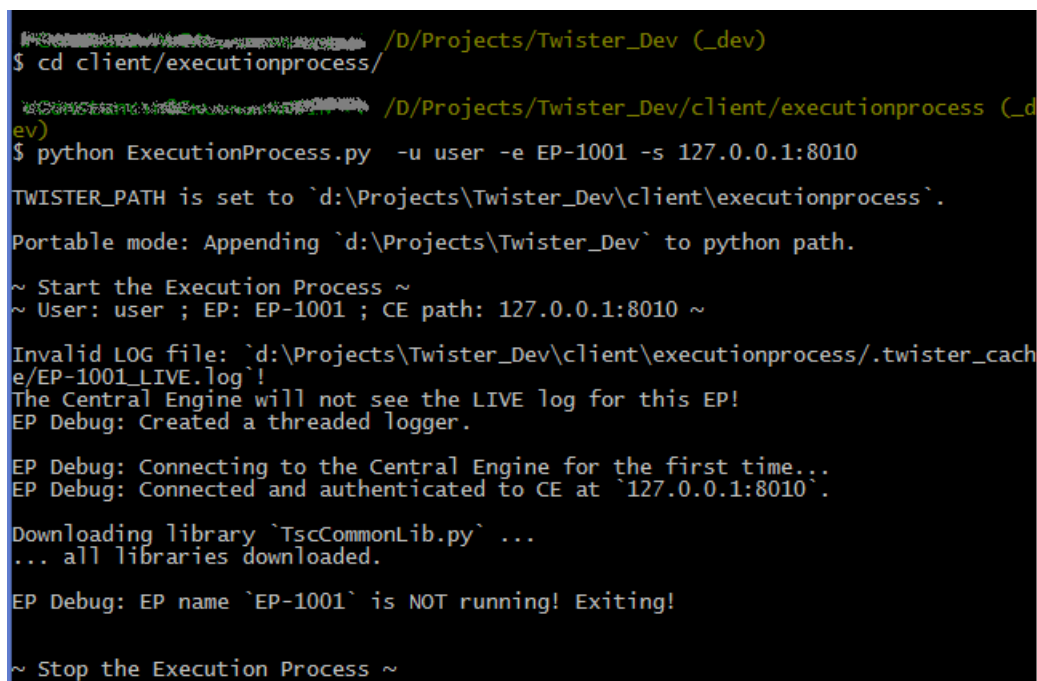
However, you need to understand the limitations of the Windows EP:

- The TCL tests cannot work with Expect, because Expect doesn't work in Windows;
- The Python tests that use pExpect will not work in Windows;
- The Sniffer will not work, because the Scapy package doesn't work in Windows;
- ***Other packages that normally cannot work in Windows, will not work in the tests that you run on the Windows EP***.

You can run any Python test that works on Windows, ex: Selenium, unit testing, FTP, Telnet, or SSH on machines that support Telnet or SSH connections, etc.

To run an EP on Windows, you must open the command line, navigate to `twister/client/executionprocess` folder and run the command:

`python **ExecutionProcess.py -u** *your_user* **-e** *EP-name* **-s** *127.0.0.1:8010*`.

If the respective EP is not started, or doesn't have anything to do, the script will exit immediately.

In order to start the EP, create a suite in the interface in the normal way and assign the suite a SUT that uses this EP name.

TWISTER USER GUIDE

15 How to write tests

Twister framework can run **Python**, **TCL** and **Perl** (limited) tests.

Writing a **Twister test** is just like writing a normal Python 2.7 test, or TCL 8.5 test, or Perl test, with a few exceptions.

Twister inserts a few helper variables and functions, which are not available in the usual environment. So **if the a test uses the helper variables and functions**, the script will become incompatible with the original Python/ TCL/ Perl language and will be executed only from Twister.

Variables inserted in all Twister tests:

- **USER** : the username running the current test ;
- **EP** : the name of the Execution Process running the current test ;
- **SUITE_NAME** : the name of the suite that contains the current test ;
- **FILE_NAME** : the full path of the test file from the machine that runs the Central Engine ;
- **SUT** : the current System Under Test ;
- **PROPERTIES** : a dictionary containg all the properties defined in the applet, for this **testcase**;
- **CONFIG** : a list containing all the config files defined in the applet, for this testcase ;
- **PROXY** : this is a pointer to the **Central Engine RPyc** server, allowing to access many CE functions. It is made for development and should not be used directly inside tests. Instead of using this pointer directly, you should use the exposed functions **from below**, which are heavily tested and will never change.

And a few functions:

- **Log_msg**(logType, message) : this function sends a message in a special log and will not appear in the CLI. Valid log types are : *logRunning*, *logDebug* and *logTest*. It is used for sending debug messages from the tests.
- **get_global**(path) : get a global parameter;
- **set_global**(path, new_value) : set a global parameter;
- **get_config**(config_file_path, variable_path) : get a config, using the full path to a config file and the full path to a config variable from that file.Very similar with **getGlobal** function;
- **get_resource**(ID or full path, [unicode | str]) : get a resource;
- **set_resource**(ID or full path, new name) : create, or update a resource;
- **rename_resource**(ID or full path, new name) : rename one resource or property of a resource;
- **delete_resource**(ID or full path) : delete one resource or property of a resource;

- **get_sut**(ID or full path, [unicode | str]) : get a SUT properties;
- **set_sut**(ID or full path, new name) : create, or update a SUT;
- **rename_sut**(ID or full path, new name) : rename one SUT or property of a SUT;
- **delete_sut**(ID or full path) : delete one SUT or property of a SUT;
- **py_exec** *some_python_command* : this function works **only in TCL** tests and allows running Python commands, or calling functions and objects from global parameters, defined in the previous tests.

In order to access the parameters sent from the interface, a Python test can use `**import sys ; sys.argv**`. The parameters are passed as a list and can be accessed using usual python variable arguments mechanism (using sys.argv).

To access a property defined in the applet, called `**var1**`, a Python test can use the following syntax (it returns the value of property):

```
PROPERTIES['var1']
```

The properties: `type`, `suite`, `file`, `dependency`, `status`, `Runnable`, `Optional`, `setup_file`, `teardown_file`, `config_files` and `param` are reserved for internal use and must NOT be used.

16 Performance and troubleshooting

The Central Engine and the Reporting Server are instances of Python CherryPy and were tested with 750 simultaneous connections, without crashing, or losing connection.

★An article concerning python web servers: *http://nichol.as/benchmark-of-python-web-servers*.

Even if the Central Engine is fast enough, for a smooth experience, it's not recommended to run more than 100 Execution Processes on one Central Engine instance. If you need more, you can simply open another instance of CE, on a different port and connect the rest of the clients on the new one.

The Execution Processes should be running on different workstations and their performance depends on the hardware of the respective machine.

All services have logs that describe every operation that is being executed. If something fails, it will be easy to know where exactly the error was produced.

On the server side, you can check the logs from */opt/twister/server_log.log*, or */opt/twister/logs/* folder. On the client side, you can check the logs from */$USER_HOME/twister/.twister_cache/*. Every EP has its own log.

If you notice a crash, or wish to report a bug, you can use the `*create_bug_report.py*` script from twister repo folder.