IAS 3.5.0

Test Automation Proposal

|  |  |
| --- | --- |
| *Published by:* | *VASCO Data Security* |
| Author: | [Author] |
| Date of Issue:  Version: | 6/11/2008  [version x.y] |
| **Status:** | [Status] |

Release history

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Name | Comment |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Related documents

Below the relevant project documents and other documents are listed.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Title | Author | publisher | date of issue | version | ISBN |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Audience

This document is an internal Vasco document intended to:

Terms and Abbreviations

This paragraph contains a list of terms and abbreviations used in this document.

|  |  |
| --- | --- |
| Term | Explanation |
|  |  |
|  |  |
|  |  |

Table of Contents

[IAS 3.5.0 1](#_Toc373159834)

[Release history 2](#_Toc373159835)

[Related documents 2](#_Toc373159836)

[Audience 2](#_Toc373159837)

[Terms and Abbreviations 2](#_Toc373159838)

[Table of Contents 3](#_Toc373159839)

[1 Proposed Automation Solution 7](#_Toc373159840)

[1.1 Jenkins 7](#_Toc373159841)

[1.1.1 IAS build job 7](#_Toc373159842)

[1.1.2 Regression test job 8](#_Toc373159843)

[1.1.2.1 Dedicated regression test job 8](#_Toc373159844)

[1.1.2.2 Dynamic regression test job 8](#_Toc373159845)

[1.1.3 SDTF Continuous Integration job 8](#_Toc373159846)

[1.2 Python-based framework - SDTF 10](#_Toc373159847)

[1.2.1 SystemTest 10](#_Toc373159848)

[1.2.1.1 SystemTestCase 11](#_Toc373159849)

[1.2.1.2 SystemTestSuite 12](#_Toc373159850)

[1.2.1.3 SystemTextTestResult 12](#_Toc373159851)

[1.2.1.4 SystemTestLoader 12](#_Toc373159852)

[1.2.2 Logging 13](#_Toc373159853)

[1.2.3 Configuration 13](#_Toc373159854)

[1.2.4 Main: program execution 14](#_Toc373159855)

[1.2.5 Support for automated deployment and install 14](#_Toc373159856)

[1.2.6 Utilities 15](#_Toc373159857)

[1.2.7 Remarks and discussion about the SDTF 15](#_Toc373159858)

[1.2.7.1 Dynamic remapping of test case classes to concrete IDs in the Test Management System 15](#_Toc373159859)

[1.2.7.2 The sample doesn’t demonstrate the need for a special suite class 15](#_Toc373159860)

[1.2.7.3 Load more configuration files? 16](#_Toc373159861)

[1.2.7.4 Deployment 16](#_Toc373159862)

[1.2.8 STAF 16](#_Toc373159863)

[1.2.8.1 Requirements 16](#_Toc373159864)

[1.2.9 Coding guidelines 17](#_Toc373159865)

[1.2.10 Advantages and disadvantages 18](#_Toc373159866)

[2 Appendix: SDTF software setup 19](#_Toc373159867)

[2.1 Virtual Machine Template 19](#_Toc373159868)

[2.2 Revision control 19](#_Toc373159869)

[2.3 Python 20](#_Toc373159870)

[2.4 Eclipse 20](#_Toc373159871)

[2.4.1 PyDev 20](#_Toc373159872)

[2.4.2 Egit 22](#_Toc373159873)

[2.4.2.1 Import project from Git 23](#_Toc373159874)

[2.5 SDTF GIT Structure 26](#_Toc373159875)

[3 Appendix: Epydoc 28](#_Toc373159876)

[3.1 General 28](#_Toc373159877)

[3.2 Classes & Modules 28](#_Toc373159878)

[3.3 Variables 28](#_Toc373159879)

[3.4 Methods and functions 29](#_Toc373159880)

[3.4.1 Parameters 29](#_Toc373159881)

[3.4.2 Return values 29](#_Toc373159882)

[3.5 Cross-references 29](#_Toc373159883)

[4 Appendix: Git Primer 30](#_Toc373159884)

[4.1 Introduction 30](#_Toc373159885)

[4.2 Initial setup 30](#_Toc373159886)

[4.3 Creating a repository 30](#_Toc373159887)

[4.4 Workflow 31](#_Toc373159888)

[4.4.1 Getting information about the working directory and repository 31](#_Toc373159889)

[4.4.1.1 View the project history 31](#_Toc373159890)

[4.4.1.2 View the changes in the working directory 31](#_Toc373159891)

[4.4.2 Checking out a version 31](#_Toc373159892)

[4.4.3 Adding content to the repository 31](#_Toc373159893)

[4.4.4 Committing changes to the repository 32](#_Toc373159894)

[4.4.4.1 Staging files to commit them 32](#_Toc373159895)

[*4.4.4.2* Committing to the local repository 32](#_Toc373159896)

[4.4.4.3 Pushing changes from the local repository to the central repository 32](#_Toc373159897)

[4.4.5 Branching and tagging 32](#_Toc373159898)

[4.5 Convenient features 32](#_Toc373159899)

[4.5.1 Committing only partial changes 32](#_Toc373159900)

[4.5.2 Modifying the last commit 32](#_Toc373159901)

[4.5.3 Stashing 33](#_Toc373159902)

[5 Appendix: Automated IAS install 34](#_Toc373159903)

[5.1 Requirements 34](#_Toc373159904)

[5.1.1 On the Test Control Host 34](#_Toc373159905)

[5.1.2 IAS host (Windows 2K8) 34](#_Toc373159906)

[5.1.2.1 Setup in an environment with AD as data store 34](#_Toc373159907)

[5.1.3 IAS host (Red Hat Enterprise Linux) 35](#_Toc373159908)

[5.1.3.1 Administrator user rights 35](#_Toc373159909)

[5.1.3.2 Install STAF 35](#_Toc373159910)

[5.1.3.3 Mounting 35](#_Toc373159911)

[5.1.3.4 Installing with yum package manager 35](#_Toc373159912)

[5.2 Usage 36](#_Toc373159913)

[5.3 Implementation details 36](#_Toc373159914)

[Appendix [title] 38](#_Toc373159915)

[6 Appendix: Test Automation Landscape 39](#_Toc373159916)

[6.1 VATF framework from Mechelen 39](#_Toc373159917)

[6.1.1 High level overview 39](#_Toc373159918)

[6.1.2 Available modules 39](#_Toc373159919)

[6.1.2.1 Labmanager 40](#_Toc373159920)

[6.1.2.2 SNMP 40](#_Toc373159921)

[6.1.2.3 Radius 40](#_Toc373159922)

[6.1.2.4 GoogleMail 40](#_Toc373159923)

[6.1.2.5 VATF\_digipass 40](#_Toc373159924)

[6.1.2.6 VATF\_generic\_gui 41](#_Toc373159925)

[6.1.2.7 VATF\_logger 41](#_Toc373159926)

[6.1.2.8 VATF\_testlinker 41](#_Toc373159927)

[6.1.2.9 VATF\_remoteshell 41](#_Toc373159928)

[6.1.2.10 VATF\_tracer 41](#_Toc373159929)

[6.1.3 Advantages and disadvantages 41](#_Toc373159930)

[6.2 Cucumber 42](#_Toc373159931)

[6.2.1 What is Cucumber ? 42](#_Toc373159932)

[6.2.2 How does it work ? 42](#_Toc373159933)

[6.2.3 Advantages / disadvantages 43](#_Toc373159934)

[6.3 Brisbane TCL framework 44](#_Toc373159935)

[6.3.1 Overview 44](#_Toc373159936)

[6.3.2 TCL soap framework 44](#_Toc373159937)

[6.3.2.1 Available modules 44](#_Toc373159938)

[6.3.2.2 Seal authentication 44](#_Toc373159939)

[6.3.2.3 Radius authentication 44](#_Toc373159940)

[6.3.2.4 Digipass 45](#_Toc373159941)

[6.3.2.5 Logging 45](#_Toc373159942)

[6.4 TCL dpadmincmd framework 45](#_Toc373159943)

[6.4.1 Available components 45](#_Toc373159944)

[6.4.1.1 Test\_all 45](#_Toc373159945)

[6.4.1.2 Logging 45](#_Toc373159946)

[6.4.2 Advantages and disadvantages 45](#_Toc373159947)

[6.5 Selenium 46](#_Toc373159948)

[6.5.1 Advantages 47](#_Toc373159949)

[6.5.2 Disadvantages 47](#_Toc373159950)

[6.6 TestComplete 47](#_Toc373159951)

[6.6.1 Advantages 48](#_Toc373159952)

[6.6.2 Disadvantages 49](#_Toc373159953)

[6.7 Gui testing 49](#_Toc373159954)

[6.7.1 AutoIt 49](#_Toc373159955)

[6.7.1.1 Advantages 49](#_Toc373159956)

[6.7.1.2 Disadvantages 50](#_Toc373159957)

[6.7.2 Sikuli 50](#_Toc373159958)

[6.7.2.1 Advantages 50](#_Toc373159959)

[6.7.2.2 Disadvantages 50](#_Toc373159960)

[6.7.3 Comparison 50](#_Toc373159961)

# Proposed Automation Solution

The purpose of the automation solution is to support a hybrid approach so that we can support the technologies and programming environments that are already in use such as Tcl, Selenium and AutoIt. These different tools will be integrated together by starting them from Jenkins. Shell mechanisms will be used to pass information between the separate tools.

We recognize the need to have a common framework to tie everything together and offer common services such as logging, reporting to TestLink, deployment and configuration amongst product specific interaction. It has been decided, based on experience, appropriateness and our understanding that Python will be used to build a lightweight framework.

## Jenkins

Jenkins Continuous Integration (CI) server is an application that monitors executions of repeated jobs, such as building a software project or jobs run by cron. It can dispatch these jobs to different machines and chain these executions together. Jobs can be started manually via its web interface, programmatically via its REST API or automatically triggered by e.g. a commit in a source code repository.

These job types will be considered, each with a different setup (note that the two first types are there for the testing of IAS, while the last one is actually there to test the SDTF):

1. IAS build: build a release of IAS and run a validation suite against it (i.e. continuous integration for IAS).
2. Regression: this job type has two subtypes: dedicated regression tests run if the validation suite succeeded and dynamic regression tests run on demand.
3. SDTF continuous integration: run unit tests and maybe some system tests whenever SDTF source code is checked in.

These jobs and their relationships are summarized in Figure 1. The LabManager configurations necessary to support this are summarized in Figure 2.

### IAS build job

A build job requires 2 slaves running constantly: one build server (named e.g. BRI\_BUILD\_SRV) and a dedicated Jenkins slave (named e.g. DED\_JE\_SLVA, the trailing A marks it as a slave for build and validate type jobs).

The definition of the build server is outside the scope of this document. Jenkins needs a way to know when the build finished and if it succeeded as this is the condition to meet before starting the validation part of the job.

The dedicated Jenkins slave will also act as a test control host (TCH). This means SDTF is installed and STAF (Software Testing Automation Framework, see Section 1.2.8) is running. All other machines deployed by the test script running on the TCH will probably need to run STAF, but in most cases, they aren’t expected to have SDTF installed. The TCH deploys the test environment, runs the tests and reports the results to TestLink and to Jenkins.

The validation suite is composed of smaller steps exposed as separate building blocks so they can be mixed and matched to support the hybrid approach. These building blocks will include:

* deploy\_test\_env script: deploy the test environment (called TEST\_ENV\_CONF , this script runs from the TCH).
* install\_and\_configure script: ask the TCH to install IAS and configure it.
* val\_suite.py: a validation suite implemented in Python using the SDTF, to be run from the TCH. The validation suite must be able to report its success/failure status in TestLink but also to the Jenkins server because it must know if it has to start the dedicated regression test job.

### Regression test job

These are more intensive tests, running in parallel (to test several platforms at once) in their own environments that are created on demand to reduce the load on LabManager. The Jenkins slave dedicated to these jobs is DED\_JE\_SLVB (the trailing B marks it as a slave for regression tests). The load on that machine is expected to be low since it is only used to request the deployment of the TCH and handing these requests for test suites to start. This requires the implementation of another script:

* deploy\_tch\_and\_test\_env script: this script deploys the TCH, then instructs it to start the test environment (see deploy\_test\_env\_script described in Section 0). There is no need to communicate the end result of the script back to Jenkins.

#### Dedicated regression test job

This kind of job is started automatically, but only if the validation part of the build suite succeeded.

#### Dynamic regression test job

This kind of job is started by human intervention depending on test plans, on the progress of the automation efforts. Ad-hoc scripting will probably evolve from there to a more automated approach.

### SDTF Continuous Integration job

This job uses a dedicated Jenkins slave (named e.g. DED\_JE\_SLVC, with the C marking the job type) that will run a test suite to validate SDTF (unit tests and maybe some systems tests). This Jenkins slave will also be a TCH (i.e. run STAF and have SDTF installed).

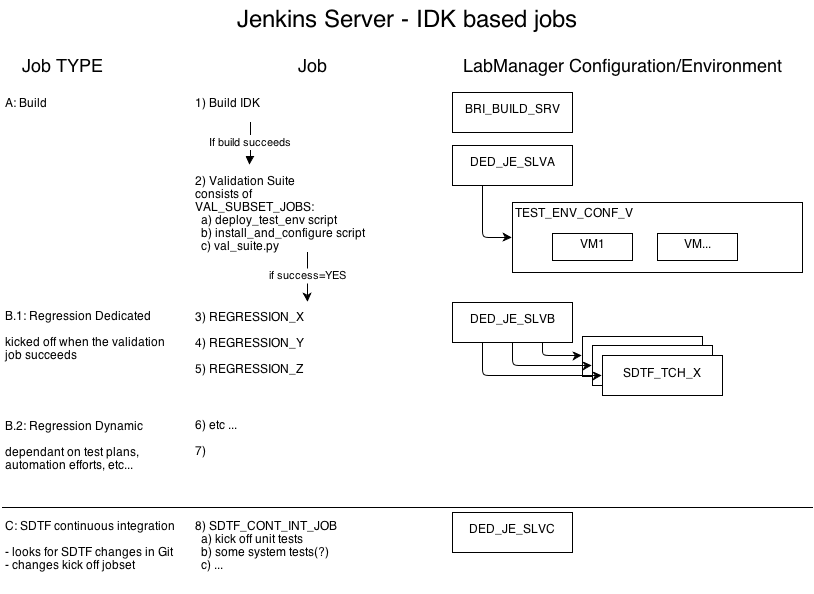


Figure 1: Jenkins jobs for IAS and SDTF testing

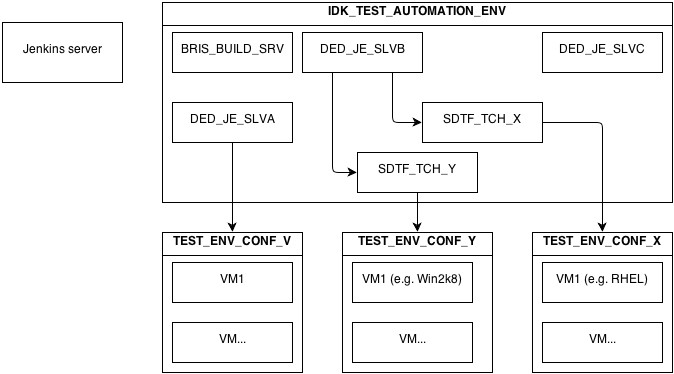


Figure 2: LabManager configurations deployed for the IDK based jobs

## Python-based framework - SDTF

SDTF (Software testing framework) is WQA’s Proof of Concept (POC) of a test framework. It is written in Python and based upon Python’s unittest library ([http://docs.python.org/library/unittest.html](http://pydev.org)).

The deviations from and extensions to the unittest library are needed to provide special features such as:

* A shared context object to pass information between test cases and test suites
* Inserting the test results in the Test Management System (TestLink in our case)

The link between the code implementing the test cases and the Test Management System is made by the docstring of the test cases: currently, this is the concatenation of an ID and a short description that must match the entry in the Test Management System.

### SystemTest

SystemTest contains our extensions to the standard unittest framework and provides on top of the standard unittest functionality:

* Suite fixtures: a suite setUp / tearDown method called only once per test suite. Typically ‘expensive’ operations like deployment/installation/… can be done in here.
* Ability to pass a context to the TestCase
* An extended TextTestResult
* A modified TestLoader

Defined in framework/core/systemtest.py. The relationship between these classes and their unittest ancestors is shown in Figure 3.

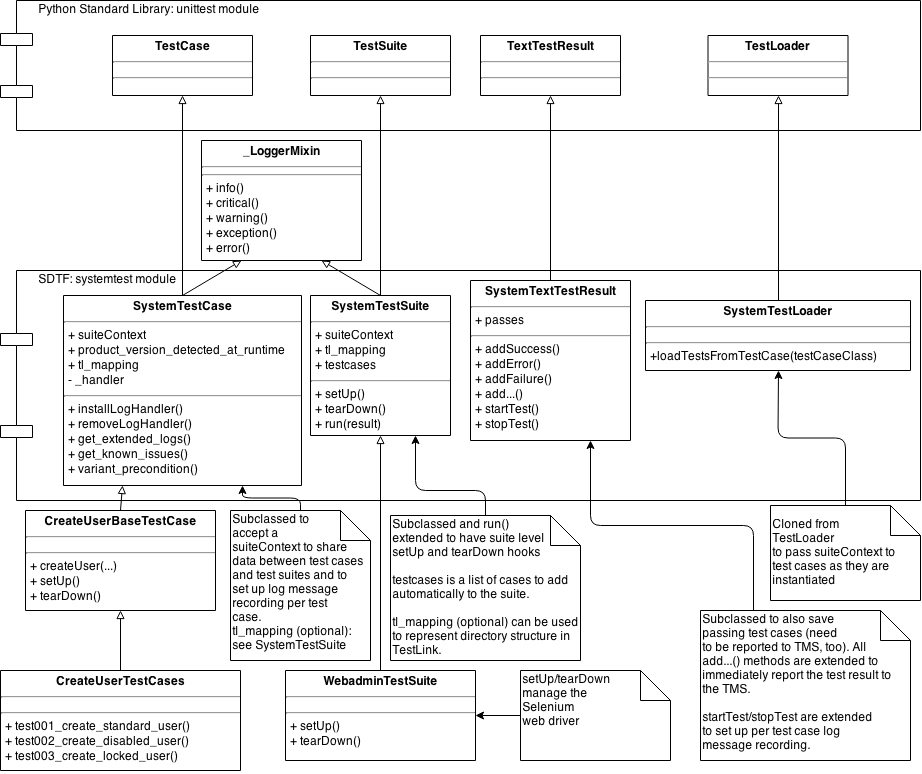


Figure 3: Class diagram of the systemtest module with an example concrete test case and test suite

#### SystemTestCase

SystemTestCase is a subclass of the unittest.TestCase class, extending it with a suiteContext attribute and the ability to store any log message during the test case execution. The log messages are recorded by a custom subclass[[1]](#footnote-1) of the logging.Handler (from the Python standard library) class. This logging handler (stored in the \_handler attribute) is added to the logger at runtime by the object of type SystemTextTestResult created by the test runner. As a convenience, the standard logging methods (info, warning, error, critical and exception[[2]](#footnote-2)) have been added to SystemTestCase so that e.g. self.info(‘Log message’) does the right thing.  
All test cases must (indirectly) inherit from SystemTestCase and (for future extensibility) make sure not to override the SystemTestCase setUp and tearDown methods (inherited from unittest.TestCase): they may be extended, but the subclass’ methods should call their parent’s methods.  
A SystemTestCase has a special product\_version\_detected\_at\_runtime member that can be set only once to a value different from “None” during the lifetime of the test case object. This member is used during test case reporting to TestLink. Setting it is optional. The benefit of setting it is that if set and if it is equal to the build name specified on the command line and if that particular build doesn’t already exist in TestLink, then the build is automatically created in TestLink before reporting the test case result.  
A test case can override the get\_extended\_logs function to return a list of file names that contain additional information about the test run. Currently, these file names are appended to the notes of the test execution in TestLink for blocked and failing test cases. The get\_extended\_logs function e.g. copies the files from the different machines in the environment IAS was installed in to a central location and reports back the names of these copies.  
In some ways, a SystemTestCase class is also a suite: it will be instantiated once for each method whose name starts with `test’ (the instances are test cases). Hence, a SystemTestCase class can also define a tl\_mapping attribute similar to the tl\_mapping attribute in SystemTestSuite.  
Compared with unittest.TestCase, SystemTestCase has an extra initialization step: there is sometimes a need to initialize each instance (one for each test method inside a test case class) slightly differently. However, a test case’s \_\_init\_\_ method’s signature is fixed and the \_\_init\_\_ method is (in the source code) far away from the different test methods of the same class. The variant\_precondition method is run right after the \_\_init\_\_ method with the method name as parameter and all keyword parameters provided to the with\_variant\_precondition decorator. The default implementation of variant\_precondition turns the keyword parameters into instance attributes (for more details, see [doc/variant\_precondition.org](https://git.vasco.com/wqa/sdtf/raw/master/doc/variant_precondition.org), first committed in [a5298381e83c5ca1](https://git.vasco.com/wqa/sdtf/blob/a5298381e83c5ca1cc0eef5173095e95fc1e5966/doc/variant_precondition.org)).  
The get\_known\_issues method is called for test cases that signal anything but a success. The return values are used to update TestLink with related issue IDs. Care must be taken when implementing get\_known\_issues not to attribute issue IDs when other issues are present for which no issue ID is known.

#### SystemTestSuite

Subclass of unittest.TestSuite, extending it with a setUp, a tearDown method and a suiteContext attribute. The run method is extended to actually call the suite’s setUp and tearDown methods.  
As a convenience, the standard logging methods (info, warning, error, critical and exception[[3]](#footnote-3)) have been added to SystemTestCase so that e.g. self.info(‘Log message’) does the right thing.  
The suites can be used to represent the directory structure of the test case organization in TestLink: if the suite class defines a member tl\_mapping, it is remembered at test execution time (see SystemTestCase.safe\_path\_when\_run() to query those values). This is used during reporting to TestLink to distinguish test cases with the same name but defined in different folders.  
When defined, the testcases member in a suite is used to automatically add instantiate test cases and add them to the suite at suite initialization time.

#### SystemTextTestResult

Subclass of TextTestResult, also adding the passed test cases to the testResult object, because passes must be recorded in the Test Management System as well. All addXXX methods have been extended to immediately report the test case result to the test management system (framework/core/tms.py, itself currently using framework/core/testlink.py). For TestLink, this action is able to create the build ID automatically in TestLink for the selected test plan if needed, but only if the new build ID has been confirmed during the test case execution (by setting the product\_version\_detected\_at\_runtime member of the test case).  
The logging handler recording the test case specific log messages is added to the logger at runtime by the SystemTextTestResult method startTest and removed from the logger by the SystemTextTestResult method stopTest. Another possibility would have been to add it to the setUp/tearDown method of SystemTestCase. However, there would have been the risk of subclasses forgetting to call their parent’s setUp/tearDown methods. Since SystemTextTestResult is less likely to be subclassed, the risk is reduced.

#### SystemTestLoader

The unittest framework works with the concept of a suite (unittest.TestSuite) that is run by a test runner (unittest.TestRunner). Building the suite can be automated by a TestLoader instance: based on naming conventions, the test loader will instantiate given test cases and add them to the suite. Unfortunately, the instantiation of the test case can’t be parameterized to allow to pass the suite context[[4]](#footnote-4). To solve this problem, the unittest.TestLoader was subclassed (to use its auxiliary utility functions) and one of the loadXXX methods was copy pasted then adapted to pass the suiteContext. All other methods have been overwritten to raise a NotImplementedError.

### Logging

All messages printed during tests should use a logger. Currently, the logging infrastructure from the Python standard library is used. The logger object to be used is stored in the configuration data in the suiteContext and a selected subset of logging methods is also defined on the test cases and test suites. During test case execution, any message printed to standard output (not standard error!) is captured and also sent to the logger. However, the advantage of using the logger directly is that filename and linenumber information is recorded, helping to know where the message originated.

This central logger object is instantiated in the framework/core/stflogging.py module. This module also defines a decorator ‘traced’ that is useful to trace function calls: any function or method definition preceded by ‘@stflogging.traced’ will log the call and its arguments to the central logger object.

### Configuration

The configuration is loaded from two sources: one that is central to the framework and one that is linked (by its place in the file system hierarchy) to the executed test project. The framework/core/config.py module defines the configuration settings that are common to all tests. Currently, these are:

* TestLink login information (URL and DevKey)
* Virtualization information (Virtualization system URL, username and password, …)
* The central logger instance

The configuration object of type SDTFConfiguration that can be instantiated to access this information accepts two optional parameters: a filename and an arbitrary Python value. These are used to build the test specific configuration that should be added to the configuration object. The object looks for an SDTF\_config.py file, stopping at the first hit. First the same directory as the filename it received as input parameter is searched, then the search walks up the file system hierarchy to its root (this is the file system root on UNIX systems and a drive name on Windows system). When an SDTF\_config.py file is found, it is loaded as a module and the function (which must be defined) get\_config\_dictionary from that module is called with the arbitrary Python value (the second parameter passed on object creation) as parameter. This function must return a dictionary whose values are added to the configuration object using the keys as attribute names. Examples of test specific configuration information include:

* Test plan name
* IP addresses of servers or URLs to test

Building this configuration object and inserting it in the suiteContext is automated by the framework.core.main.Main class responsible for test program execution.

This module also contains a convenience class (called ConfigObject) to easily create objects containing only data attributes with minimal typing overhead.

### Main: program execution

The code to load the configuration, build a test suite, running it and reporting the results to the Test Management System is fairly repetitive. It has been concentrated in the Main class defined in framework/core/main.py. This class can be instantiated without any parameters: sensible defaults are provided to implement the usual case, i.e. starting a suite specified by its name on the command line. Some of the optional arguments are the filename of the script being run (to look for the corresponding SDTF\_config.py file) and the test cases to run (an iterable of test case classes or one test case class, each containing one or more tests that can be discovered by the SystemTestLoader).

This class also handles the instantiation of the configuration class and the parsing of the command line arguments. This will give all test scripts a common command line interface:

<script name> <build> <platform> [suite] [--config.sub.var=value …]

The build and platform information are needed to link the test results to the TMS. The suite must be the (or a unique abbreviation of the) name of a test suite to run, loaded from the module defined in the script. Additional configuration values can be specified at the end of the command line. (See special parameters)

#### Special parameters

* **--help:** If `--help’ is present anywhere in the command line arguments, a help text is printed and nothing is done. The [docstring](http://freeradius.org/radiusd/man/radtest.html) of the script is used in the help text.
* **--dump\_fqtcn:** The optional --dump\_fqtcn parameter will print a list of all testcases in the provided suite along with their (dynamic) index. After printing the list, nothing is done.
* **--dump\_fqsn:** The optional --dump\_fqsn parameter will print a list of all subsuites in the provided suite along with their (dynamic) index. After printing the list, nothing is done.
* **--tests\_to\_run**: The optional --tests\_to\_run parameter allows you to run a specific subset of tests inside a suite. You can separate testcase indexes by a ',' or by a '-' if you want to run a range of tests. Example :
  + --tests\_to\_run="1,2-5,3,10"
    - Will run testcases : 1, 2, 3, 4, 5 and 10
* **--suites\_to\_run**: The optional --suites\_to\_run parameter allows you to run specific subsuites of a suite. Similar usage to the --tests\_to\_run parameter. Indexes out of bound are ignored.

### Support for automated deployment and install

The SDTF supports the infrastructure described in Section 1.1. It contains lists of available configurations that can be cloned from the virtualization system’s library.

This list is subdivided per product, e.g. the list with the configurations for testing IAS will be maintained in framework/ias/\_environments.py. This list contains elements describing the environments (initial development efforts will use W2K8\_ODBC, W2K8\_AD and RHEL as examples):

1. Name in the virtualization system’s library
2. Information to map this configuration to a TestLink platform name
3. A list of machines, each element containing the following information:
   1. The name of the machine
   2. Its external IP address (filled in at deployment time)
4. The default virtualization type (Labmanager/vCloud/None) to be used for this environment

The case of IAS testing is special in that there are several axes of variation (OS, backend or data store and the client or plugin: the authentication test suites are expected to be along the line of ODBC\_RadiusClient, EDB\_RadiusClient (embedded database as data store), ADDS\_RadiusClient (Active Domain as data store), ADBE\_RadiusClient (AD as backend), LDAPBE\_RadiusClient, ODBC\_Plugin1 and so on. It is expected that there will be a suite for each combination, with two parameters specializing their inherited behavior for the setup task and possibly the tests to execute.

There will thus also be a list of clients or plugins in framework/dpap/platforms.py and assorted deploy and install routines. These will clone or deploy the clients into an already deployed configuration (containing the IAS server under test).

There is a naming convention for the predefined configurations in the virtualization system’s library and for the names of the configurations deployed into the workspace (note that the name in the workspace is derived automatically from the name in the library and the current state of the workspace):

1. Library configurations: <product>\_<platform>\_<options> (for IAS, options is the data store or backend used)
2. Workspace configurations: SDTF\_<name in library>[\_<number>] (the optional number is added to make the configuration name unique if necessary)

### Utilities

There is a module in the framework named utils that contains small self-contained submodules with their documentation in the source code itself. This section just serves as a reminder to look through it to see what could be useful, such as:

* myip.get\_my\_ip([optional arguments]): get the external IP address of the computer the script is running on
* debug: defines functions to use in the [Python debugger](http://www.python.org/dev/peps/pep-0263/)[[5]](#footnote-5)
* shell: defines functions useful when working with external processes

### Remarks and discussion about the SDTF

#### Dynamic remapping of test case classes to concrete IDs in the Test Management System

This is handled by defining the tl\_mapping member in the suites inside which test cases are run. These tl\_mapping values are collected at test execution time to represent the place in the test plan hierarchy where the test case belongs and thus find its ID in TestLink. The path described by the tl\_mapping values of the suites containing sub-suites and cases should exactly match the layout defined in TestLink.

Original text (October 2012): The test cases should be written to certain interfaces, allowing e.g. to reuse the same code to test Web admin interfaces using Internet Explorer of Firefox. However, these test cases will have different IDs in the Test Management System. There is no obvious way to do this yet, though the solution will probably involve a mapping of Python test case and interface type (IE or Firefox in the example above) to concrete IDs used to identify the test case in the Test Management system. This mapping could be maintained in the test specific configuration (SDTF\_config.py).

#### The sample doesn’t demonstrate the need for a special suite class

I.e. SystemTestSuite may not even be needed if we look at the setupClass and tearDownClass method of unittest.TestCase and override/extend those. This would allow to get rid of the SystemTestSuite definition, of all specific code to make sure that the SystemTestLoader passes a suiteContext to the test suites it instantiates, would concentrate the user code in less classes (no SystemTestSuite subclass) and would reduce the number of parameters to pass to Main.

On the other hand, this forces us to use a SystemTestCase subclass descendant with methods named testXXX to group test cases.

#### Load more configuration files?

Currently, only two configuration files are used: a central configuration file and the first SDTF\_config.py file found as explained in the Configuration section. The idea of walking up the file hierarchy was born to allow different suites for separate functional areas of the same product to share the same configuration while being located in separate folders for clarity. However, this forces the different suites to share their configuration data. Maybe any sdtf\_config.py file found during the search should be loaded (the uppermost first, with deeper situated files overriding the values)?

#### Deployment

The SDTF should also deploy the needed environment. This is future work: a proof of concept already exists in suites/ias/demo/resources.

### STAF

The Software Testing Automation Framework (STAF - [http://staf.sourceforge.net/](http://redmine.vasco.com/projects/vatf/wiki/VatfPom)) is an open source, multi-platform, multi-language framework designed around the idea of reusable components, called services (such as process invocation, resource management, logging, and monitoring). There is a wrapper around the PySTAF bindings in framework/core/STAF.py that implements STAF handles that are registered and unregistered automatically and translates the return codes into exceptions with a more descriptive output.

A POC has been performed that successfully demonstrated the following functionalities from within Python:

* Obtain information about OS on a remote computer
* Launch a process on a remote host and get its return code, stdout, …
* Copy files to/from a remote host.

#### Requirements

STAF needs to be installed on each machine of the configuration. The installation procedure on test hosts is the following:

1. Install STAF (accept all defaults and make sure that "Start STAF on user login" is enabled).
2. Arrange the VM to automatically log in to the account used to install STAF (see [Autologon](http://technet.microsoft.com/en-us/sysinternals/bb963905.aspx) from Windows Sysinternals)
3. Edit the C:\STAF\bin\STAF.cfg file to trust all clients with level 5 (this is wide open!). Add the line  
   trust level 5 default

### Coding guidelines

This section is a work in progress. The most important reference is [PEP-8](http://git-scm.com/book/ch3-2.html)[[6]](#footnote-6), the Style Guide for Python Code defined by the Python developers.

What follows are comments on that document:

* Implementing core modules: care should be taken to make the module easy to import.
  1. If the module is spread over several files in a directory, consider writing an \_\_init\_\_.py file to automatically import the submodules or from the submodules themselves (see e.g. framework/ias/\_\_init\_\_.py or framework/utils/\_\_init\_\_.py)
  2. Avoid exporting names without good reasons to minimize the risk of name clashes:
     + Either define and maintain an \_\_all\_\_ list at the top of the module of names that should be visible by default or prefix names that shouldn’t be exported with an underscore (this also holds for imports: use “import xxx as \_xxx” or “from xxx import yyy as \_yyy” where necessary).
     + Avoid importing blindly all names from other modules (i.e. do not use “from xxx import \*”) because these names will then be exported, too.
* Using modules: the number of exported names from core modules should be small, but only import the names you need. See also this [link](http://www.python.org/dev/peps/pep-0257/)[[7]](#footnote-7).
* Naming conventions: the goal is to follow it for new code. Unfortunately, the unittest module from the standard library doesn’t adhere to it and names its methods e.g. setUp and tearDown instead of set\_up and tear\_down. This implies that e.g. the SystemTestCase class and its descendants will also violate the naming convention.
* Character sets: VASCO’s product are used in many different countries and allows many character sets to be used. Recommendations :
  1. If possible, avoid non ASCII chars in your source files. Otherwise, use UTF-8 encoding for your source files (Add a # -\*- encoding: utf-8 -\*- to the file, see [PEP 0263 -- Defining Python Source Code Encodings](https://github.com/nu7hatch/gmail)[[8]](#footnote-8)).
  2. Learn about Unicode in Python (see [Pragmatic Unicode](http://git-scm.com/book/ch2-2.html)[[9]](#footnote-9)).
* Documentation convention: the [epytext](http://epydoc.sourceforge.net/epytext.html)[[10]](#footnote-10) markup language is used. Use the doc/gendocs.py script to update the API documentation in doc/api/index.html. See Section 2 for more information about epydoc and its epytext markup.
* Future work: Testing SDTF. Library modules should consider integrating (using the if \_\_name\_\_ == “\_\_main\_\_” idiom) some usage examples. It would be even better to structure these usage examples as test cases so that SDTF can be tested as well.

### Advantages and disadvantages

Advantages of using Python

* Object oriented
* Platform independent (+ implementations like IronPython, Jython … available)
* Big standard library
* Aims at clear, readable code

SDTF advantages

* Based on proven concepts (unittest module)
* Still young, so can incorporate insights from other projects from the start

SDTF disadvantages

* The framework is still a work in progress

# Appendix: SDTF software setup

## Virtual Machine Template

There is a virtual machine template in LabManager called SDTF\_TCH that comes preinstalled with SDTF in c:\SDTF and with desktop shortcuts to start a shell where all environment variables are set to be able to fetch updates from the central repository and to commit and push. Read the text shown by the bash shell to configure your SSH key for access to the central repository.

When cloning this template into a configuration to use it, take into account that it will reboot automatically once (see the desktop background) before being fully usable. When the desktop background changes to usage instructions, you can click away the “Network Locations” and “Microsoft Security Essentials” dialogs. You may also need to activate the windows license and update Java.

## Revision control

Git ([http://git-scm.com](http://docs.python.org/2/library/pdb.html)) is used as a revision control system. The Git for Windows client ([http://msysgit.github.com](http://staf.sourceforge.net/)) is used to access the central repository ([git@git.vasco.com:stf.git](http://docs.python.org/library/unittest.html)) using the SSH protocol.

The utilities bundled with Git for Windows assume that there is a HOME environment variable that names a directory where to put configuration information (Git configuration, SSH configuration). Use either git-bash.bat or git-cmd.bat to start a shell that sets up the environment correctly (i.e. the PATH to the binaries and the HOME directory).

Generate a private/public key pair (without pass phrase to make it more convenient) using:

ssh-keygen -t rsa -C firstname.lastname@vasco.com

*(Use the default location proposed by the tool)*.

Then you can copy-paste your **public** key (%HOME%/.ssh/id\_rsa.pub) to the central repository (log in to [https://git.vasco.com](http://www.python.org/dev/peps/pep-0008/) and select the `SSH Keys’ tab in your profile). It should look something like

ssh-rsa AAA<...>== <username>@<computername>

You are now ready to e.g. clone[[11]](#footnote-11) the repository (but see also the discussion of Eclipse and its **EGit** plugin to access the source code).

Configure your email address and name to include in the commit messages:

git config --global user.name ´John Doe´  
git config --global user.email ´John.Doe@vasco.com´

## Python

Version 2.7 of the language is used. There are various precompiled installation binaries available on the internet (e.g. <http://portablepython.com>). Make sure to use the 32 bit version of Python even if you are on a 64 bit platform. [virtualenv](http://git-scm.com) is used as it is the recommended way to get the Selenium bindings for Python and can be used to install the [SOAP client library](http://msysgit.github.com). All these installation steps are implemented in install\_sdtf.py. Once the repository is checked out, the necessary libraries (stored in the resources directory) need to be installed:

python install\_sdtf.py

The script might prompt (via the UAC dialog) for an administrator password to install STAF and issue instructions about DLLs to register to enable AutoIt. Note that the script is only supported for Windows at the moment (using either bash from the Git for Windows project or cmd.exe).

The installation script will create three additional files in the project directory: sdtf-activate, sdtf-activate.bat and sdtf-deactivate.bat. These files are necessary to setup the environment (and for resetting it to its original state) and they are used as described in Table 1. Note that it is best to do this before starting the IDE (from the same shell) in the case the IDE picks up his paths pointing to the Python interpreter from the environment.

Table 1: Setting up and resetting the environment for SDTF

|  |  |  |
| --- | --- | --- |
|  | Windows cmd.exe | Git for Windows bash |
| Before doing anything with SDTF (once in every shell!) | path\to\sdtf\sdtf-activate | source path/to/sdtf/sdtf-activate |
| To undo (optional) | path\to\sdtf\sdtf-deactivate | sdtf-deactivate |

## Eclipse

Eclipse can be used as the development environment. There are two useful plugins: PyDev and EGit.

### PyDev

PyDev adds support for Python development. Installation in Eclipse:

* *Manually*: select Help > Install New Software… to start a wizard. In the wizard, `Add...’ [http://pydev.org](http://git-scm.com/book/en/Git-Tools-Stashing)/updates as a new software site, then install the `PyDev for Eclipse’ plugin.
* *Automatically*: Download & install it using the Eclipse Marketplace

In Window > Preferences, in the PyDev tab, select Interpreter – Python and point Eclipse to the Python interpreter installed in the previous step, or auto configure it.

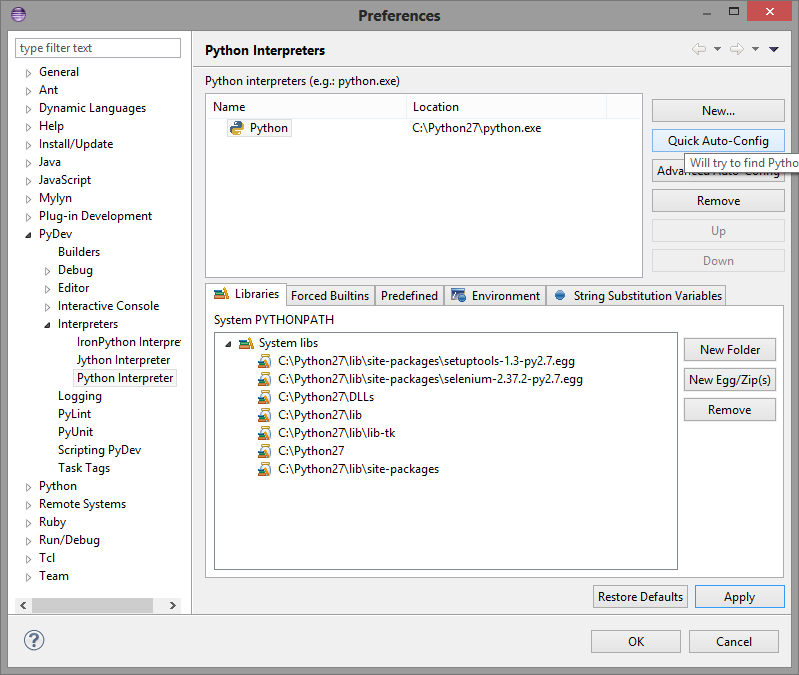


Figure 4: Ensure the location of Python Interpreter is set correct

### Egit

eGit adds support for Git to Eclipse. Installation in Eclipse:

* *Manually*: select Help > Install New Software… to start a wizard. In the wizard, `Add...’ http://download.eclipse.org/egit/updates as a new software site, then install the `Eclipse EGit’ plugin.
* *Automatically*: Download & install it using the Eclipse Marketplace

Set the ssh private as follows:

* Go to Window > Preferences,
* In General > Network Connections > SSHv2 tab,
* Set your SSH2 home to the directory where ssh-keygen stored the private key. *This will make EGit share the same key as the command line Git for Windows client to access the central repository git.vasco.com.stf.git.*

Set the correct git configuration as follows:

* In Window > Preferences, in the Team > Git > Configuration tab,
* Check the proper location for the User Settings is set   
  *(it should be the Windows file name of the .gitconfig in the directory that Git for Windows considers to be your home directory, i.e. the value of the HOME environment variable followed by ‘\.gitconfig’).*
* Check if the System Settings location has correctly found the etc\gitconfig file in the directory where Git for Windows was installed.

#### Import project from Git

File > Import… > unfold Git > select “Projects from Git” > Click Next > URI > Click Next. Now you are in the Source Git Repository wizard.

* The wizard should look as in **Figure 5**, then click **Next**.
* After some network traffic, a list of available branches should appear, **select** them **all** (this means that the complete history of the project will be copied to your local clone of the repository).  
  Then click **Next**
* ***Select*** a ***destination folder*** where the repository will be *(for instance C:\GIT\)*.   
  Then Click **Next**.
* Once the repository has been downloaded, select the **‘Use the New Project wizard’** radio button and click **Finish**.
* New Project wizard appears. **Unfold PyDev** and then **select** the **PyDev Project** option and click **Next**.

Figure 5: Import Projects from Git URI wizard screenshot

* Type a ***project name*** *(such as “stf”)*,  
  Uncheck “**Use default**”,   
  **Select** the **directory** of the repository,  
  **Select** the **python** Interpreter,  
  Click **Finish**.

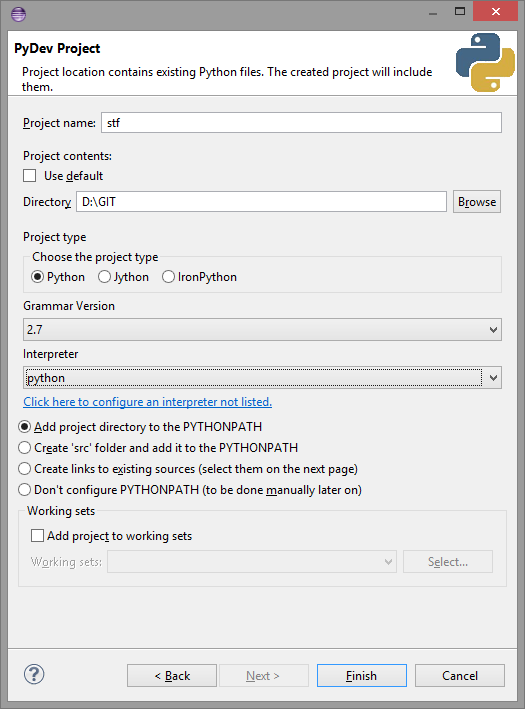


Figure 6: PyDev Project Settings

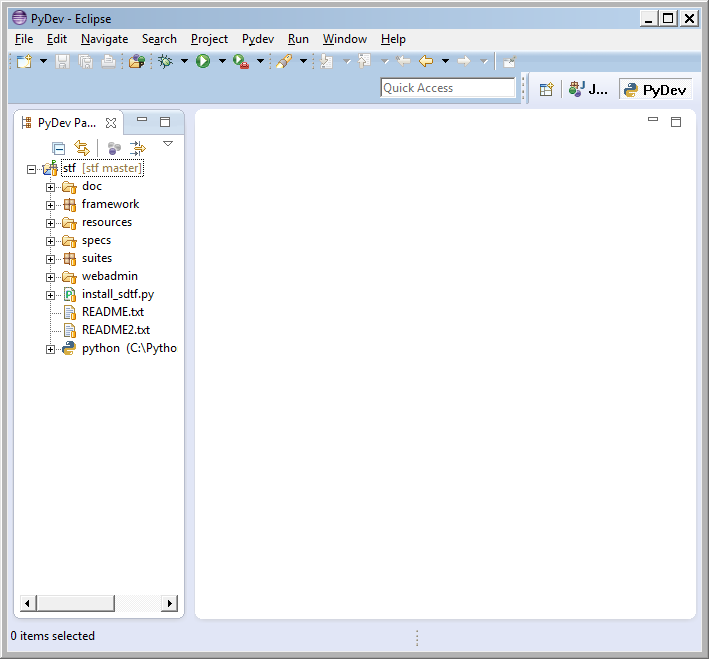
Your eclipse repository should now look as inFigure 7. 

Figure 7: Repository after install procedure

## SDTF GIT Structure

Below you will find the proposed structure of the SDTF project in GIT:

* SDTF
  + **framework**: contains the SDTF framework components. Consists of a core package, tools package and a package for each product. E.g.
    - core: SDTF core modules like systemtest.py, stflogging.py, main.py, testlink.py, testEnvironment.py …
    - IAS: IAS related modules (e.g. radius client)
      * Servertools – *This is where we would store Brisbane’s serverTools automation.*
    - Product\_y: …
    - Product\_z: …
    - Digipass: product Digipass related modules (e.g. digipass.py)
    - Utils: non‑product related utility modules (e.g. smtp module)
  + **install\_sdtf.py**: a script to install SDTF correctly. This will unpack files from the resources subdirectory and create batch files to setup the virtual environment (see [http://virtualenv.org](mailto:git@git.vasco.com:stf.git)) called sdtf-activate.bat, sdtf-activate.bat, sdtf-activate.bat
  + **resources**: generic resources required by the framework (installers, python modules, … needed to setup a TCH)
    - IEDriverServer\_Win32\_2.25.2.zip
    - python-suds-0.4.tar.gz
    - …
    - STAF3410-setup-win32.exe
  + **runtime**: directory created by install\_sdtf.py to install SDTF components into
  + **specs**: test specifications per product/functionality (note: follows the same structure as the suites)
    - IAS
      * Webadmin
        + Webadmin\_spec.xml
      * Functionality Y
        + Functionality\_Y\_spec.xml …
      * …
    - Product Y …
  + **suites**: automated test suites per product/functionality. Each functionality can have its own set of resources. E.g:
    - IAS
      * Demo
        + Resources

Any resources needed for testing this functionality (e.g. csv with set of users/DP, dpx files, …)

* + - * + Suite\_userAdmin.py
        + Suite\_digipassAdmin.py
        + …
        + Testcase\_userAdmin.py
        + Testcase\_digipassAdmin.py
        + …
      * ***TCL\_administration:*** This is where we would put the TCL soap admin framework as is.
      * ***TCL\_dpadmincmd:*** This is where we would put the TCL dpadmincmd framework as is.
      * Installation
        + Resources: This is where we would put Brisbane’s installation automation scripts
      * Webadmin
      * *Functionality y*
        + *…*

# Appendix: Epydoc

## General

Epydoc is a syntax used for generating HTML documentation for Python. All Epydoc tags should be written in a docstring (triple double quotes).

Paragraphs in descriptions need to use one whitespace between them in order to be parsed properly and can contain markup like italicization or bold text with I{italic text} and B{bold text}. Example:

“””Description of a B{module}

More detailed I{description} of the module”””

Another useful tag for markup is C{} with which you can differentiate code from text.

These can also contain (un)ordered lists, all lists should be indented. You can create sub lists by adding another indentation:

“””

Example of an ordered list:

1. Item 1

2. Item 2

3. Item 3

Example of an unordered list:

-Item 1

-Subitem 1

-Item 2

-Item 3

“””

## Classes & Modules

Class and module descriptions are generally written after the class-header or at the beginning of the module.

## Variables

You can make a distinction between three types of variables, module variables, class variables and instance variables using @var, @cvar and @ivar respectively.

These need to be declared under the variable so that the parser recognizes the name and default value. @type can be used to define the type of the variable. Below is an example:

z = 14

"""@var: Module variabele"""

class Test:

"""

test klasse

"""

x = 1

"""

@type: int

@cvar: klasse var

"""

def \_\_init\_\_(self, val):

self.var = val

"""

@type: int

@ivar: instantievar

"""

## Methods and functions

Method and function descriptions are placed below the declaration; here you can also define what parameters the method needs and what it returns.

### Parameters

Parameters can be defined using @param, default values are automatically detected.

@param paramName: description

You can set the type of the parameter by using @type, of course you have the possibility to cross-reference here as well.

@type paramName: L{ExampleClass}

### Return values

Return values work much the same way as parameters.

@return: Returns a value between x and y

You can define the return type by using @rtype, again cross-referencing can be used.

@rtype: L{ExampleClass}

It’s also possible to define what exceptions are thrown by using @raise, with this you can also describe when this exception occurs. The type of exception is automatically cross-referenced.

@raise MyException: Occurs when the wrong value is given

## Cross-references

Classes and functions can be referenced from within descriptions, it’s also possible to reference to URL’s.

Referencing to an object can be done as follows:

L{ExampleClass/ExampleFunction}

Or if you want a better description:

L{See the ExampleClass<ExampleClass>}

In the same way URL’s can be built:

U{http://www.example.com} or U{Go to URL<http://www.example.com>}

# Appendix: Git Primer

## Introduction

This document is a very short introduction to the Git version control system showing its most used commands inside a normal workflow and also shows some convenient features that are slightly more advanced. An excellent book (Pro Git) is available in print (ISBN 1430218339) and online ([http://git-scm.com/book](https://git.vasco.com)) and this document links to the appropriate sections of the book where applicable. Its second chapter is an introduction to the basics if you are in a hurry and if this document isn’t enough. Another good reference is the site [http://gitref.org](https://help.github.com/articles/generating-ssh-keys).

Git is different from more traditional version control systems (like CVS or SVN) in the fact that every developer has the complete history of the project on his local drive and not simply a snapshot of the version he is currently working on. Even though all repositories are functionally equal, the [workflow](http://www.virtualenv.org/en/latest/index.html) in the development team often designates one repository as the central repository that is viewed as the reference repository by all team members.

Git identifies versions not by a series of automatically incremented numbers but by a unique hexadecimal code representing the version, called the commit ID. When passed as arguments to Git tools, these commit IDs can be abbreviated as long as the abbreviation is unique within the repository.

## Initial setup

Every commit is tagged with a human readable user name and an email address (of the author and of the committer, though these are usually the same). These can be configured globally in the HOME directory of the developer with:

git config --global user.name ´John Doe´  
git config --global user.email ´John.Doe@vasco.com´

## Creating a repository

A repository can be created in an already existing project with [git init](https://fedorahosted.org/suds/#Initializing-a-Repository-in-an-Existing-Directory):

cd project-directory  
git init

This initializes the repository, but the content of project-directory still need to be added to it (see section “Adding content to the repository”).

Another way is to get a repository from an existing project. This operation is called [cloning](http://virtualenv.org#Cloning-an-Existing-Repository) since a complete copy of the repository (and its history) is made locally:

git clone git@git.vasco.com:wqa/sdtf.git

This command will use the SSH protocol to contact the server (see the [github documentation on SSH keys](http://www.effbot.org/zone/import-confusion.htm) if you have trouble with this method). Other methods exist and may be used depending on the server settings.

The most recent version on the main development branch will be checked out in .\sdtf and the project history will be downloaded to .\sdtf\.git.

## Workflow

The commands in this section are used on a regular basis. A graphical user interface is available through the command git gui. The following subsections are still useful to describe the concepts.

### Getting information about the working directory and repository

#### View the project history

The history of the current branch can be viewed by running the following [command](http://git-scm.com/book/en/Git-Basics-Getting-a-Git-Repository) in the working copy:

git log

A graphical viewer is available as well and invoked with gitk.

#### View the changes in the working directory

The working directory is the place where the changes are made and tested. To review the differences with the original version (i.e. the [status](http://nedbatchelder.com/text/unipain.html#Checking-the-Status-of-Your-Files) of the working directory), use

git status

This gives an overview of the changes in the working directory and of the changes ready to be committed. This command also suggests commands to modify the state of the listed files. See the section “Adding content to the repository” for an explanation of the concepts used in committing as the concept of “staged for commit” may be unfamiliar.

If you want to see the [changes in a specific file](http://git-scm.com/book/ch2-2.html#Viewing-Your-Staged-and-Unstaged-Changes), use

git diff *<filename>*

### Checking out a version

If the current version checked out in the working directory isn’t the version you want to work on, you can check out a different version (note that this doesn’t require network access):

git checkout *<commit ID>*

The commit ID is either the hexadecimal code shown by e.g. git log or a unique abbreviation of it, a branch name or a tag name.

### Adding content to the repository

Files that are unknown to Git will show up in the git status output as untracked files, with the suggestion to [add](http://git-scm.com/book/en/Distributed-Git-Distributed-Workflows#Tracking-New-Files) them to the staging area, from where they can be committed to really add them to the repository.

git add *<file1>* ... *<fileN>*

Note that when adding directories, Git recurses down the directory tree and adds the content of the subdirectories as well. Files can be [ignored](http://rpmfind.net/linux/rpm2html/search.php#Ignoring-Files) by Git by editing a .gitignore file.

### Committing changes to the repository

Once some changes are made and tested, they can be committed. In Git, this is a process in two steps: first the changes are added to a [staging area](http://git-scm.com/book/en/Git-Basics-Viewing-the-Commit-History#Staging-Modified-Files) and then the contents of the staging area are [committed](mailto:git@git.vasco.com:stf.git#Committing-Your-Changes) to the local repository. To make these changes visible to other team members, they can be [pushed](http://git-scm.com/book/ch2-5.html#Pushing-to-Your-Remotes) from the local repository to the central repository.

#### Staging files to commit them

git add *<file1>* ... *<fileN>*

#### Committing to the local repository

git commit

The command will let you edit the commit message. Git treats the first line of the commit message specially: it assumes that it is a summary of the commit message when displaying information. The convention is to think of the commit message as an email with the first line being the subject of the email, separated from the body of the email by a blank line.

Using git commit -a is more like svn commit in the sense that any modified file will be committed if it is already tracked. This obviates the need for the git add step staging the files.

#### Pushing changes from the local repository to the central repository

git push

Obviously, others can push their changes to the central repository to. Their changes can be integrated with [git pull](http://git-scm.com/book/ch2-2.html#Fetching-and-Pulling-from-Your-Remotes). Note that this implies a [merge](http://git-scm.com/book/en/Git-Basics-Getting-a-Git-Repository) or might require a local history rewrite (called [rebase](http://gitref.org) in Git parlance).

### Branching and tagging

Besides commit IDs, versions can be given names either as [branches](http://www.nist.gov/el/msid/expect.cfm) (these are names that point to an evolving version of the project) or as [tags](http://git-scm.com/book/ch2-2.html) (names pointing to a fixed version of the project).

git checkout -b *<new branch name>*  
git tag *<tag name>*

## Convenient features

### Committing only partial changes

Sometimes, a file may contain several changes that belong in separate commits. When staging the files for commit with git add, there is a -i option to [interactively add](http://git-scm.com/book/ch2-2.html) only parts of the file to the staging area, thus committing only one part of the introduced changes at a time.

### Modifying the last commit

Git allows its users to rewrite the project history in many ways. This is an advanced feature (and controversial, since a version control system is supposed to keep the history), but in the simplest case it can be very useful. Imagine that just after committing you want to fix something that shouldn’t have been committed or is missing from it, then you can [amend](http://git-scm.com/book/ch2-2.html#Changing-the-Last-Commit) your commit: prepare the staging area to contain what you really wanted to commit and use

git commit --amend

You shouldn’t do this after you have pushed your commit to a remote repository.

### Stashing

Sometimes, you are in the middle of big changes and you need to interrupt your work to work on something else in the same project that is more urgent. Instead of committing unfinished work or cloning a complete new repository, you can [stash](http://git-scm.com/book/en/Git-Branching-Rebasing) your unfinished work away, checkout whatever version needed your urgent attention and return later to the stashed work in progress.

# Appendix: Automated IAS install

IAS can be installed from SDTF on hosts deployed to the virtualization system (vCloud/Labmanager). Both deployment and installation are fully automated. This section discusses the requirements for the VMs where the installation is done and how to use the installation scripts.

## Requirements

### On the Test Control Host

The installation procedure of IAS on Windows hosts used AutoIt3. Preferably, AutoIt3 should be installed and available on the PATH of the test control host starting the deployment and installation: this ensures that the latest version of the installation scripts are pushed to the host where IAS will be installed and makes possible error reports more accurate. If the AutoIt3 executable and its dependencies can’t be found on the Test Control Host, the precompiled installation driver script will be used instead, but this script isn’t always up to date.

### IAS host (Windows 2K8)

The installation process requires the administrative user to be logged in (use AutoLogon from sysinternals.com). STAF needs to be installed and setup to start up automatically and to allow remote clients to copy files and execute programs (see Section 1.2.8).

An example configuration is prepared in the library, ready to be cloned to the workspace: IAS\_W2K8\_ODBC\_Library.

#### Setup in an environment with AD as data store

When the IAS server will use an AD server as data store, extra configuration steps are needed:

1. Add the Remote Serve Administration Tools\Role Administration Tools\AD DS and AD LDS Tools\AD DS Tools\AD DS Snap-Ins and Command-Line Tools using the Server Manager. Then, from the command line, use MMC to &File\Add/Remove Snap-In\Active Directory Users and Computers then exit and save.
2. Give TEST\Administrator SeServiceLogonRight:  
   This is needed for the workaround where the IAS service needs to run as domain administrator to be able to access the AD (to e.g. update login times):
   1. run services.msc
   2. edit the properties of the "Interactive Services Detection"
   3. select the "Log On" tab
   4. select the "This Account" radio button and fill in the account name (TEST\Administrator) and password (Test1234) twice
   5. click Apply
   6. you should get a dialog telling you that the service logon right has been added to the specified user account
   7. undo your changes by clicking the "Local System account" radio button and enabling the "Allow service to interact with desktop" checkbox
   8. click OK
   9. you can check that the TEST\Administrator user has the new SeServiceLogonRight:  
      secedit /export /cfg %TEMP%\secedit.cfg  
      find "SeServiceLogonRight" %TEMP%\secedit.cfg  
      and compare with the output of  
      whoami /all | find /I "TEST\Administrator"

### IAS host (Red Hat Enterprise Linux)

The installation process requires the STAF daemon to run with administrative privileges and [expect](http://git-scm.com/book/ch2-5.html) to drive the console based configuration wizard. An example configuration is in the library: IAS\_RHEL61\_Library.

The setup of this configuration took some time, the remainder of this section lists hints how to accomplish the installation of the various parts.

#### Administrator user rights

The administrator user must be allowed to do anything with sudo (use visudo as root to give him those permissions) without typing his password.

#### Install STAF

Use the installer from sourceforge. Check in System > Administration > Firewall and disable it completely to allow other STAF clients to connect to the machine.

#### Mounting

How to mount the WQA share:

mkdir -p /media/wqa

mount -o user=VASCO/lm-user%Shared1234 //eng-be-file.vasco.com/WQA /media/wqa

You can also add this to /etc/fstab (on one line):

//eng-be-file.vasco.com/WQA /media/wqa cifs user=VASCO/lm-user,password=Shared1234 0 0

How to mount a loopback (ISO):

mount /media/wqa/QC-Projects/01-Identikey/3.4/3.4.SR1/builds/IK34SR1\_GA/IDENTIKEY\_Authentication\_Server\_3.4SR1\_LIN64.iso /mnt -t iso9660 -o loop

#### Installing with yum package manager

* Insert RHEL61 Client ISO image with LabManager
* check in /etc/yum.conf if you have the dvd as repo:

[dvd]

name=Red Hat DVD (mount image first!)

enabled=0

baseurl=file:///media/RHEL\_6.1\ x86\_64\ Disc\ 1

Then you can  
yum --enablerepo=dvd install tcl

If necessary, you can override warnings about package signatures with  
yum --nogpgcheck --enablerepo=dvd install tcl

* If your package is not on the DVD, try [rpmfind](http://git-scm.com/book/ch3-2.html)
* Install with yum  
  yum install <URL>

## Usage

The primary means to identify which version to install is to use the default implied by the build number passed as first parameter on the command line: the script will try to fetch the corresponding ZIP file from [ftp.vasco.com](http://git-scm.com/book/en/Git-Tools-Interactive-Staging):

python suite/ias/installation/suite.py 3.5.0.13 RHEL61\_eDB\_RADIUS-NPS Auto --virt.user=abc --virt.user=def --traces.user=ghi --traces.password=jkl --install.ftp.user=Brisbane --install.ftp.password=mno

(The above command line is all on one line, with the double-dashes stuck to the words following them without a space in between).

The version to install can be also passed on the command-line using the parameter install.media to avoid looking for the build on the FTP server. Its meaning is OS dependent: for installations on W2K8 it should be the path to the root directory of the installation DVD (i.e. a directory containing autorun.exe etc) and for installations on RHEL61 either a directory containing the root directory of the installation DVD (i.e. a directory containing an install.sh shell script) or the path to the DVD’s ISO file.

## Implementation details

The central data structure is the list of environment objects defined in framework.ias.environments. Each element of this list (an object of type Environment) represents an environment on which to run test cases:

1. Its name in the test management system.
2. The name of the virtual environment to deploy to run these tests.
3. A list of machine objects, each representing a host.

There are functions to query this list (get\_environment\_by\_LM\_name and get\_environment\_by\_TMS\_name) and to act on these environments (deploy and install).

The machine objects can have functions attached to them to run certain checks and to attempt to fix encountered problems. Since these functions might need additional information about the environment they are testing and its configuration, both the environment object and the machine object accept a configuration object (see framework.core.config.SDTFConfig) with the set\_config method. This allows to build the list of environments without needing to parse the command line (where the configuration information comes from). Since the environment is a container for the machines, all its functions forward their parameters to all machines it contains.

Named sets of checks or fixes (e.g. all machines running STAF will share STAF-related checks) can be created by subclassing Machine (see STAFMachine or LDAPMachine) or by using a function to add the checks to a Machine instance (see IASMachine).

A difficult part of the code is the aspect of copying Environment and Machine objects: Machine objects are mutated to add the externalIP once they are deployed but this should not be reflected in the environments list: the deploy function makes a copy of the desired Environment object and its Machine objects (in the \_normalize\_environment\_parameter function). This requires collaboration of the objects through their shallow\_copy method. When defining sub-classes, care needs to be taken to implement the shallow\_copy method of the sub-class.

Appendix [title]

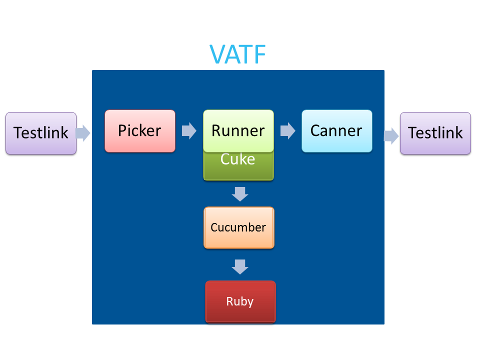
[content]

Source code

# Appendix: Test Automation Landscape

## VATF framework from Mechelen

### High level overview

VATF is the automation framework developed in Mechelen.  
Core technologies used are Ruby, Cucumber and Selenium webdriver.

* Picker : retrieves information on which tests to execute on which platform from TestLink
* Runner: runs the test cases retrieved by picker (by virtue of Cuke) and places their result in a specified folder.
* Cuke: runs Cucumber test cases within a configured VATF environment
* Canner : imports the results from obtained from Runner into TestLink (HTML output file, possible screenshots, detailed VATF log file)

The framework is able to deploy a configuration in LabManager.

With proper configuration, VATF allows for 1:1 mapping of Testlink test cases to local test features based on TestLink unique IDs.

### Available modules

A set of modules have been implemented in Ruby. The following chapters give an overview of each one of these.

#### Labmanager

Allows controlling Labmanager via its SOAP interface.  
Supported operations include listing, deploying, cloning, snapshotting, reverting and getting information from LM configurations and machines.

#### SNMP

Provides a wrapper around the snmpwalk command line tool and as allows querying information from SNMP MIBs.

#### Radius

Provides Radius client authentication functionality and is basically a wrapper around the radtest command line tool ([http://freeradius.org/radiusd/man/radtest.html](http://git-scm.com/book/ch2-6.html)).

This module currently only supports response-only PAP authentication. However, the underlying tool supports chap/mschap en eap-MD5 as well. Adding support for these is straight forward.

Advantages:

* Being a wrapper around a command line utility, it seems very easy to update

Disadvantages:

* radtest does not allow differentiating between mschapv1 and mschapv2, so another command line tool might be more useful.

#### GoogleMail

Provides the functionality to interact with a Gmail mailbox.   
This module is based upon the ‘gmail’ gem ([https://github.com/nu7hatch/gmail](ftp://ftp.vasco.com))

Amongst its functionalities are:

* Listing, finding, deleting mails (+ all functionalities provided by the gmail gem)
* Searching a mail for specific content (using regex matching), extracting links from a mail message …

#### VATF\_digipass

This gem provides Digipass functionality.  
Basically, supports:

* Importing DPX into a file system based db.
* Use the imported DB records (OTP, C/R and signature are supported)
* Querying information from the Digipasses that have been imported

Each Digipasses can be assigned a friendly name for convenience.

Advantage:

* This approach allows us to use any Digipass, so it is not tied to Demo tokens only.

#### VATF\_generic\_gui

A layer on top of Selenium Webdriver (and Capybara) that reduces the maintenance tasks in case of GUI updates. It makes use of a design pattern called Page Object Model to represent a web GUI as an object.  
This module provides:

* Easy mapping and handling of any Web GUI using the [Page Object Model test design pattern](http://git-scm.com/book/ch6-4.html)
* A generic API to perform most common Web GUI interactions (clicking, filling in , navigating to, complete a form with validation, ....)
* Support for the most common browsers Firefox,Chrome, IE …

Advantage:

* Could reduce maintenance if the page objects are well defines

Disadvantage:

* Learning curve could be steep

#### VATF\_logger

Class providing logging facilities, based upon the standard Ruby logger. The messages have associated levels that indicate their importance.

#### VATF\_testlinker

A Ruby wrapper around the TestLink XMLRPC API, thus allowing access to your TestLink test projects, plans, cases, and results using Ruby.  
It supports both TestLink APIs 1.0 Beta 5 (from TestLink 1.8.x) and 1.0 (from TestLink 1.9.x).

#### VATF\_remoteshell

Generic class for performing shell actions on a remote system via SSH or SCP  
Provides a few methods to run commands on remote host and to copy files in between hosts using the scp protocol.

#### VATF\_tracer

Vatf\_tracer is a subclass of VATF\_remoteshell, used for live tailing of logfiles on remote systems using multiple threads.

Its main methods are ‘start\_tracing’, ‘stop\_tracing’ and get\_traces.

### Advantages and disadvantages

Advantages of using Ruby

* Object oriented
* Platform independent (+ implementations like IronRuby, JRuby … available)
* Big standard library
* Lots of extensions (‘gems)
* Aims at clear, readable code

VATF advantages

* Quite some components are in place already and could be re-used. Especially the VATF\_digipass component seems powerful.

VATF disadvantages

* Currently, quite some of the components make use of standard Linux tools (like ps, which, tail …) behind the scenes. Examples of modules doing this are SNMP, Radius, VATF\_remoteshell, VATF\_tracer.
* Some components (like VATF\_generic\_gui) could have a steep learning curve. However, apparently QA Mechelen is reworking and documenting things over here.
* It could be that Cucumber is not flexible enough for us to build up a distributed test framework.

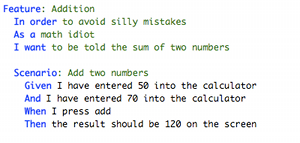
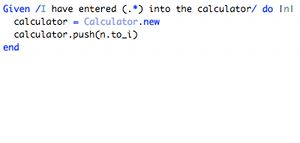
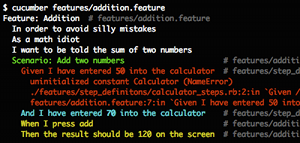
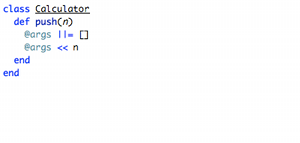
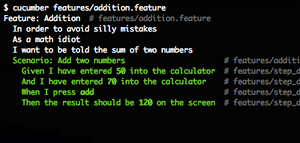
## Cucumber

### What is Cucumber ?

Cucumber makes it possible to software development teams to just describe how software should behave in plain text. The text is written in a business-readable domain-specific language and serves as documentation, development-aid and acceptance tests all packed in 1 format. These stories are typically written before anything else and verified by business analysts, domain experts, etc. non-technical stakeholders. The production code is then written outside-in, to make the stories pass.

Mechelen has chosen to make use of the Cucumber framework in order to allow non‑developers to add/modify test cases. This results in test cases that need to be written in pseudo-English (i.e. Gherkin). Files containing these test cases are called ‘feature’ files in the Cucumber jargon. Associated cucumber step files (written in Ruby) are then used to interface between the feature files and the actual Ruby API calls.

### How does it work ?

1. Describe the behavior in plain text.   
   
2. Write a step definition in the underlying programming language.
3. Run it and watch it fail.  
   
4. Write code to make the step pass.  
   
5. Run again and see the step pass, if it doesn't pass, go back to step 4.

### Advantages / disadvantages

Advantages

* Test-driven development
* A clear text describing what is happening
* Test specification could be written in Cucumber language.

Disadvantages

* No object orientation (everything needs to be repeated).
* No value if people reading and writing the specs are developers.
* There are several methods to describe something and not everyone uses the same way.
* You can’t run Cucumber stories with any other language but English.
* Not agile.
* Maintenance gets higher as stories grow.
* It’s easy in development phase, but in maintenance phase it’s just an extra cost and doesn’t add extra value.
* Extra layer means an extra possibility where it can break.

## Brisbane TCL framework

### Overview

Brisbane uses the TCL scripting language for their automation efforts.   
Their automation framework can be split up in 2 parts based on how they interact with IAS:

* TCL soap framework: aims at testing IAS by interfacing with IASs built-in SOAP API
* TCL dpadmincmd framework: aims at testing IAS by interfacing it using its TCL interface (dpadmincmd)

Both of the above are able to report test results in TestLink, by uploading an XML results file generated by the testrunner.

### TCL soap framework

The ‘TCL soap framework’ aims at testing IAS by interfacing it using its built-in SOAP API. Given the nature of the SOAP protocol, the TCL soap framework is able to run tests on a remote host.

#### Available modules

##### runTest

Main tool for launching tests suites. It will take every script listed in a test suite (i.e. plain text file containing a list of TCL scripts to execute with their arguments) and execute these sequentially.  
All test case results are stored in an XML file that can be uploaded to TestLink.  
  
SOAP interface (IAS)

The SOAP bindings are defined by orgunitAdminCmd, policyAdminCmd, userAdminCmd and reportAdminCmd.

Each of these functions can construct a range of SOAP messages, send it to IAS and treat the result.   
A command line tool ‘SoapClient2.exe’ is used by each one of these functions to perform the actual SOAP communication.  
  
SOAP authentication

Soapauthcmd provides the abilitiy to perform SOAP authentications. It is similar to the SOAP interface discussed above.

#### Seal authentication

Sealauthcmd provides a wrapper around the ‘SealClient6.exe’ utility and as such provides a way to perform seal authentication.

#### Radius authentication

Radius authentication functionality is provided by RadiusAuthCmd.tcl.  
This is a wrapper around the ‘RadiusClient4.exe’ tool.  
It currently only supports PAP as authentication protocol and Response Only authentication.  
  
The RadiusClient4.exe however supports a wider range of authentication protocols / methods, so this wrapper could be easily extended.

Note: RadiusClient4.exe seems to support more features than radtest does (CLI tool used in Mechelens VATF)

#### Digipass

Digipass functionality is currently provided by the RadiusClient4.exe command line tool.   
This tool can generate an OTP to be used as a password, but appears to be limited to generating a Response-Only OTP for a Demo Digipass.

#### Logging

Standard logging facilities (with multiple log levels) are provided by the testUtils module.

## TCL dpadmincmd framework

The ‘TCL soap framework’ aims IASs built-in support for TCL scripting (dpadmincmd tool). As the dpadmincmd needs to run locally on the IAS machine, tests cannot be ran remotely

### Available components

#### Test\_all

Test\_all is this frameworks test runner and will run all TCL test scripts it finds. Basically, it launches the dpadmincmd tool that comes with IAS and passes a TCL test script as argument.  
  
Test\_all will generate an XML file with test case results that can be uploaded to TestLink. Additionally, a style sheet allows the XML file to be rendered into a human-readable format in a browser.

#### Logging

Standard logging facilities (with multiple log levels) are provided by the testUtils module.

### Advantages and disadvantages

Advantage of using TCL

* Platform independent

Disadvantages of using TCL

* Not object oriented and might not be ideal for setting up a larger test automation framework.
* The lack of a rich set of data structures
* TCL might not be as well supported as popular languages like Ruby/Python. E.g. it would be typically easier to find Ruby/Python bindings vs. Tcl bindings for a certain product.

## Selenium

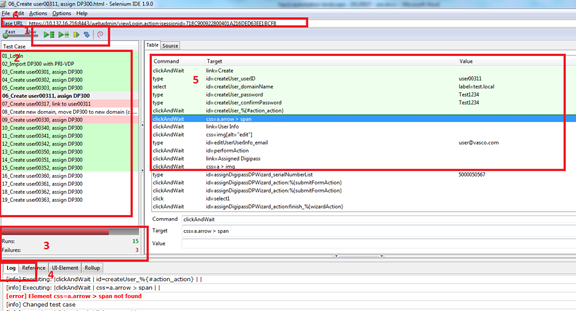
Selenium is a portable software testing framework for web applications. Selenium provides a record/playback tool for authoring tests without learning a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including C#, Java, Groovy, Perl, PHP, Python and Ruby. The tests can then be run against most modern web browsers. Selenium deploys on Windows, Linux, and Macintosh platforms. It can be used for unit, regression, smoke, integration and acceptance testing.

Selenium projects:

* Selenium IDE: Firefox add-on that makes it easy to record and playback tests in Firefox.
* Selenium core: tests run directly in a browser, just as real users do. Need to copy selenium Core & tests directly into application web server
* Selenium RC: Selenium remote control is a system that allows simulating web browsers locally using almost any programming language and testing framework.
* Selenium Grid: allows several Selenium remote control servers to be accessed in parallel by Selenium Grid server.

Selenium IDE usage:

1. Install Selenium as a Firefox plug-in
2. In Firefox browser go to “WebDeveloper>Selenium IDE”



At the main workspace of the Selenium IDE present:

1. Action buttons: run whole test suite, run test case, pause/resume, step, apply rollup rules.
2. List of test cases to be executed. List of test cases can be saved as a test suite.
3. Results of the latest execution with numbed of failed test cases.
4. Log of the execution.
5. Code of the test case.
6. Base URL of the product to be tested

### Advantages

* Selenium is free and open source tool (and thus flexible)
* Selenium can test across multiple browsers
* Selenium uses language of choice

### Disadvantages

* Test cases are code so need constant maintenance and re-factoring
* Dynamic testing, like Data-Driven Testing is not possible with the Selenium IDE
* Selenium IDE is not able to read or write external files or open popups for user interaction
* Slow when testing all the edge cases

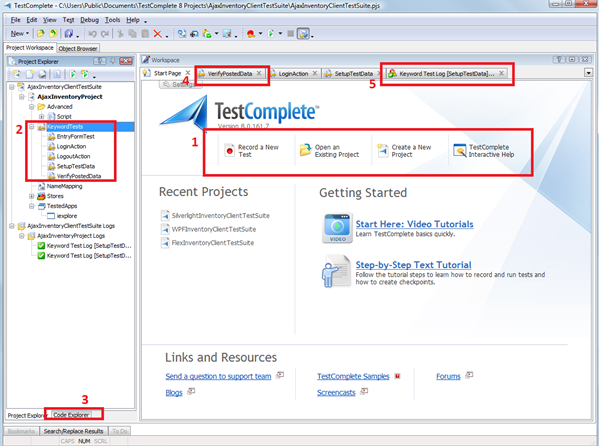
## TestComplete

TestComplete is an automated testing tool that lets testers create, manage and run tests for any Windows, Web or Rich Client software.

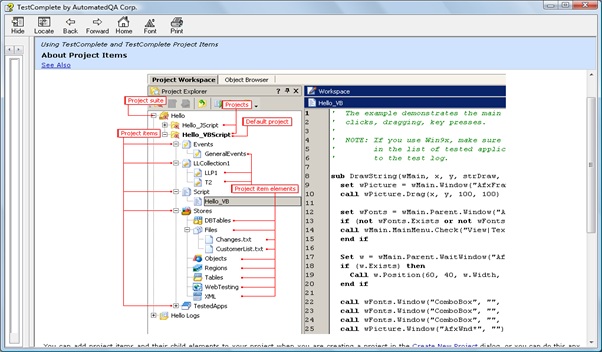
TestComplete automates following types of testing: functional, unit, load, client/server (distributed) regression and data-driven testing.

TestComplete is used to create and automate many different software test types. Record and playback test creation records a tester performing a manual test and allows it to be played back over and over again as an automated test. Recorded tests can be modified later by testers to create new tests or enhance existing tests with more use cases.

1. On the screenshot below you can see the main page of the tool from which you can:
2. Record a New Test/Open an Existing Project/Create New Project
3. See the list of all keyword tests
4. At the page Code Explorer to see a list of all scripted tests
5. To check details of the test itself
6. To see log of the test execution



Screenshot 1 - Homepage of Testcomplete



Screenshot 2 - TestComplete Project Workspace

### Advantages

* Validate Applications During Automated Testing: TestComplete provides various checkpoints that help automated testers easily validate their applications behavior. These checkpoints can verify an application’s objects and their property values, web pages, files and XML documents, images, data in tabular controls, databases and many more elements, both visual and non-visual. TestComplete’s checkpoints can be added while recording and editing your automated tests.
* Easily Create Automated Tests: TestComplete’s keyword tests, or otherwise known as keyword-driven testing, does not require programming or scripting skills and allows even non-technical and inexperienced testers start automating tests instantly, “right out of the box” and create powerful automated tests with minimal effort. Furthermore, TestComplete provides screen previews for each automated test operation, which clearly illustrates the automated test flow and enable testers to add new automated test operations without even having to access the original application.
* Automate Test Cases with Scripts: TestComplete enables experienced test engineers to automate sophisticated testing scenarios using automated test scripts. Test cases can be coded in VBScript, JScript, C++Script, C#Script or DelphiScript and tuned in TestComplete’s industrial-strength automated test script editor, which provides code completion, automated syntax checking and other features for efficient automated test scripting. Automated test scripts can be run independently or can be integrated into keyword tests to make your keyword-driven testing more powerful and flexible.
* Data-driven tests. This functionality is saving a lot of time while recording tests, as different combinations of data can be taken by TestComplete from the external document (.xsl or .csv file) to the test

### Disadvantages

* Cost of the product comparing to the open source tools
* Programming background is highly required for the scripting automation
* Constant update of tests in order to keep them up to date is time consuming

## Gui testing

Not all applications have an API to communicate with them from inside a programming language. To test these kind of applications we have to add an extra layer of abstraction. This is where Autoit and Sikuli come in place. Let’s look at the features of these applications and make a comparison.

### AutoIt

AutoIt is a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting. It uses a combination of simulated keystrokes, mouse movement and window/control manipulation in order to automate tasks in a way not possible or reliable with other languages (e.g. VBScript and SendKeys). AutoIt is also very small, self-contained and will run on all versions of Windows out-of-the-box with no annoying “runtimes” required!

AutoIt was initially designed for PC “roll out” situations to reliably automate and configure thousands of PCs. Over time it has become a powerful language that supports complex expressions, user functions, loops and everything else that veteran scripters would expect.

#### Advantages

* Uses the windows API, easier to integrate
* Has a separate dll that you can use in different programming languages.

#### Disadvantages

* Only available on windows.

### Sikuli

Sikuli is a visual technology to automate and test graphical user interfaces using screenshots. Sikuli includes Sikuli Script, a visual scripting API for Jython(a library for Java), and Sikuli IDE, an integrated development environment for writing visual scripts with screenshots easily. Sikuli Script automates anything you see on the screen without internal API's support. You can programmatically control a web page, a Windows/Linux/Mac OS X desktop application, or even an iphone or android application running in a simulator or via VNC.

#### Advantages

* Usable on multiple platforms, even smartphones.

#### Disadvantages

* Using screenshots, uses image recognition and is going to be slower on large tests.
* Slow startup because of the java virtual machine needs to startup every time for every suite.

### Comparison

|  |  |  |
| --- | --- | --- |
|  | Autoit | Sikuli |
| Cross platform (mobile included) | Only windows | All platforms |
| Method | Windows API | Screenshots |
| Python Bindings | Yes | Possible via xml-rpc |
| TCL Bindings | Yes | Possible via xml-rpc |
| Ruby Bindings | Yes | Possible via xml-rpc |
| Compilable | Yes | No |
| Programming Language | BASIC | Java |
| License costs | 0.00 $ | 0.00 $ |

1. \_SystemTestLoggingHandler, defined in framework/core/stflogging.py [↑](#footnote-ref-1)
2. Notice that the logger’s debug method is not in the list because it would clash with the unittest.TestCase.debug method. [↑](#footnote-ref-2)
3. Notice that the logger’s debug method is not in the list because it would clash with the unittest.TestSuite.debug method. [↑](#footnote-ref-3)
4. There is a way involving the runtime creation of classes wrapping the test case (by simple inheritance) and pulling the suiteContext out of a closure to add it to the original test case \_\_init\_\_ parameters, but it is more advanced Python code that is more difficult to understand and maintain. [↑](#footnote-ref-4)
5. http://docs.python.org/2/library/pdb.html [↑](#footnote-ref-5)
6. http://www.python.org/dev/peps/pep-0008/ [↑](#footnote-ref-6)
7. http://www.effbot.org/zone/import-confusion.htm [↑](#footnote-ref-7)
8. http://www.python.org/dev/peps/pep-0263/ [↑](#footnote-ref-8)
9. http://nedbatchelder.com/text/unipain.html [↑](#footnote-ref-9)
10. http://epydoc.sourceforge.net/epytext.html [↑](#footnote-ref-10)
11. A discussion of the usage of Git is outside of the scope of this document. Look in Section 4 for a primer. An excellent book (Pro Git) is available in print (ISBN 1430218339) and online (<http://git-scm.com/book>) [↑](#footnote-ref-11)