

WARRIOR FRAMEWORK

USER GUIDE

DOCUMENT HISTORY

Revision	Description	Author	Date
1	Ninja (1.0)	Brian Jishi	9/8/2015
2	Ninja (1.5)	Sathyamoorthy Radhakrishnan	1/7/2016
3	Ninja (1.6)	Sathyamoorthy Radhakrishnan	1/20/2016
4	Ninja (2.0)	Sanika Kulkarni	9/29/2016
5	Ninja (2.9)	Brian Jishi	3/16/2017

WARRIOR FRAMEWORK

REVISION: 5

TABLE OF CONTENTS

WARRIOR FRAMEWORK USER GUIDE

1.0	WHAT IS WARRIOR FRAMEWORK?.....	7
1.1	WHAT MAKES WARRIOR FRAMEWORK GENERIC?.....	7
1.2	WHAT NOT TO DO IN WARRIOR FRAMEWORK?	9
2.0	WARRIOR FRAMEWORK INSTALLATION	10
3.0	WARRIOR FRAMEWORK TOOLS.....	14
3.1	WARHORN: AN OVERVIEW	14
3.2	KATANA: AN OVERVIEW	15
3.3	IRONCLAW: AN OVERVIEW	16
3.3.1	VERIFICATION OF CASE XML FILES:.....	16
3.3.2	VERIFICATION OF SUITE XML FILES:.....	16
3.3.3	VERIFICATION OF PROJECT XML FILES:	16
3.3.4	RUNNING IRONCLAW.....	16
3.4	MOCKRUN: AN OVERVIEW	17
3.5	VIRTUAL MACHINE WITH WARRIOR FRAMEWORK ENVIRONMENT: AN OVERVIEW.....	Error! Bookmark not defined.
4.0	UNDERSTANDING THE WARRIOR FRAMEWORK DIRECTORY STRUCTURE	18
4.1	ACTIONS DIRECTORY.....	18
4.2	PRODUCTDRIVERS DIRECTORY	19
4.3	FRAMEWORK DIRECTORY	20
4.4	TOOLS DIRECTORY	28
4.4.1	THE ADMIN DIRECTORY	28
4.4.2	JIRA DIRECTORY	28
4.4.2.1	AUTO DEFECT CREATION	29
4.4.3	THE REPORTING DIRECTORY	30
4.4.4	THE XSD DIRECTORY.....	31
4.4.5	THE W_SETTINGS FILE.....	31
4.5	DOCS DIRECTORY	32
4.6	WARRIOR FRAMEWORKCORE DIRECTORY	33
4.7	WARRIORSPACE DIRECTORY.....	Error! Bookmark not defined.

WARRIOR FRAMEWORK

REVISION: 5

4.7.1	CASES DIRECTORY	34
4.7.2	SUITES DIRECTORY	34
4.7.3	PROJECTS DIRECTORY	35
4.7.4	DATA DIRECTORY	35
4.7.5	CONFIG_FILES DIRECTORY	36
4.7.6	EXECUTION DIRECTORY	36
5.0	UNDERSTANDING THE BUILT-IN KEYWORDS.....	37
5.1	CLI ACTIONS PACKAGE	37
5.2	CLOUDSHELL ACTIONS PACKAGE.....	37
5.3	COMMON ACTIONS PACKAGE	37
5.4	DEMO ACTIONS PACKAGE:	37
5.5	NETCONF ACTIONS PACKAGE	37
5.6	NETWORK ACTIONS PACKAGE	37
5.7	REST ACTIONS PACKAGE	38
5.8	SELENIUM ACTIONS PACKAGE	38
5.9	SNMP ACTIONS PACKAGE	38
6.0	UNDERSTANDING THE CASE	39
7.0	UNDERSTANDING THE SUITE	46
8.0	UNDERSTANDING THE PROJECT	52
9.0	UNDERSTANDING THE INPUT DATA FILE (IDF)	55
10.0	THE DATA REPOSITORY.....	57
11.0	CASE DESIGN FEATURES.....	58
11.1	RUNNING KEYWORDS IN SEQUENCE	58
11.2	RUNNING KEYWORDS IN PARALLEL.....	58
11.3	DESIGNING A NEGATIVE KEYWORD	58
11.4	KEYWORDS DESCRIPTION	58
11.5	CONDITIONAL KEYWORDS EXECUTION	59
11.6	DESIGNING A NEGATIVE CASE	59
11.7	RUNNING A CASE IN SEQUENCE	59
11.8	RUNNING A CASE IN PARALLEL.....	60
11.9	RUNNING A CASE MULTIPLE TIMES	60
11.10	RUNNING A CASE UNTIL FAILURE/PASS.....	60
11.11	ERROR HANDLING IN WARRIOR FRAMEWORK.....	61

WARRIOR FRAMEWORK

REVISION: 5

12.0 HOW TO CREATE KEYWORDS	63
13.0 HOW TO CREATE A CASE.....	70
14.0 HOW TO CREATE A SUITE.....	71
15.0 HOW TO CREATE A PROJECT.....	72
16.0 HOW TO CREATE AN INPUT DATA FILE	73
17.0 EXECUTING THE CASE	74
17.1 THROUGH CLI	74
17.1.1 RUN A CASE.....	74
17.1.2 RUN MULTIPLE CASES	74
17.1.3 RUN A SUITE.....	74
17.1.4 RUN MULTIPLE SUITES	74
17.1.5 RUN A PROJECT	74
17.1.6 RUN A COMBINATION OF CASE, SUITE, AND PROJECT.....	74
17.1.7 SCHEDULE EXECUTION.....	75
17.1.8 RUN KEYWORDS IN A CASE IN PARALLEL, CASES IN SEQUENCE.....	75
17.1.9 RUN KEYWORDS IN A CASE IN SEQUENCE, CASES IN PARALLEL.....	75
17.1.10 RUN MULTIPLE TIMES	75
17.1.11 RUN UNTIL FAILURE	75
17.1.12 RUN A PROJECT	75
17.1.13 EXECUTION BASED ON CATEGORY.....	76
17.1.14 JIRA BUG REPORTING.....	77
17.1.15 CREATING SUITES	78
17.1.16 ENCRYPT PASSWORDS	80
17.2 THROUGH KATANA	80
18.0 UNDERSTANDING THE RESULTS	81
19.0 GUIDELINES TO CREATE A WARRIOR FRAMEWORK-KEYWORDS REPOSITORY	83
20.0 GUIDELINES TO CREATE A WARRIORSPACEREPOSITORY	84

KATANA USER GUIDE

1. KATANA: AN INTRODUCTION.....	86
1.0 WHERE IS KATANA LOCATED?	86
1.1 THE PREREQUISITES	86

WARRIOR FRAMEWORK

REVISION: 5

1.2	HOW TO INSTALL KATANA?	86
1.3	HOW TO RUN KATANA?.....	86
2.	GETTING STARTED WITH KATANA	88
3.	CREATING CASES WITH KATANA.....	92
4.	CREATING SUITES WITH KATANA.....	109
5.	CREATING PROJECTS WITH KATANA.....	120
6.	CREATING INPUT DATA FILES WITH KATANA.....	128
7.	EXECUTING WARRIOR FRAMEWORK THROUGH KATANA.....	145

WARHORN USER GUIDE

1.	WHAT IS WARHORN?.....	151
2.	HOW DOES WARHORN WORK?	152
3.	PREREQUISITES	154
4.	HOW TO INSTALL WARHORN?.....	155
5.	THE DIRECTORY STRUCTURE.....	156
6.	HOW TO CONFIGURE THE DEFAULT CONFIGURATION FILE?	157
6.1	THE <WARHORN> TAG.....	157
6.2	THE <WARRIOR FRAMEWORK> TAG.....	158
6.3	THE <KATANA> TAG	159
6.4	THE <DRIVERS> TAG.....	160
6.5	THE <WARRIOR FRAMEWORKSPACE> TAG	161
7.	HOW TO RUN WARHORN?.....	162
8.	THE LOG FILES	163

1.0 WHAT IS WARRIOR FRAMEWORK FRAMEWORK?

Warrior Framework is a generic automation framework that automates software tasks and processes, in addition to functional, performance and solutions testing. It is designed and implemented by Fujitsu Network Communications.

Warrior Framework is a keyword driven framework that uses xml based test data that is generated by a web interface. Warrior Framework was designed to allow users to effectively create reusable generic keywords and leave the execution intelligence out to be handled by Warrior Framework. The execution of the keyword will be handled by the framework and designed in the cases, suites and project files. Warrior Framework cases, suites and projects are created in xml. These files may be edited in any xml editor or using Katana. Katana is a web based interface that simplifies editing and creation of xml cases, suites and projects. Katana provides keyword searching, keyword documentation review and keyword, case and project execution instructions.

With Warrior Framework, testers can execute keywords and cases sequentially or in parallel with a simple selection. In addition, users can implement conditions and loops for keyword execution, design keyword, case, and suite error handling by making selections in the xml file, eliminating the need for duplicate and tedious python scripts.

Warrior Framework was designed for automation of end to end solutions. Its architecture allows for testing multiple devices under test within a single case, suite or project file. For example, from a single case, Warrior Framework can execute keywords on a windows based system, a Network Element, a web-based application and a Linux based VM. Warrior Framework also allows the integration of test sets such as Wireshark, Spirent and others within the case. In addition, Warrior Framework can also integrate cases created in other frameworks and execute them in conjunction with Warrior Framework cases. Warrior Framework's ability to integrate multiple systems, test sets, and frameworks within a single execution simplify the design and execution of end to end testing.

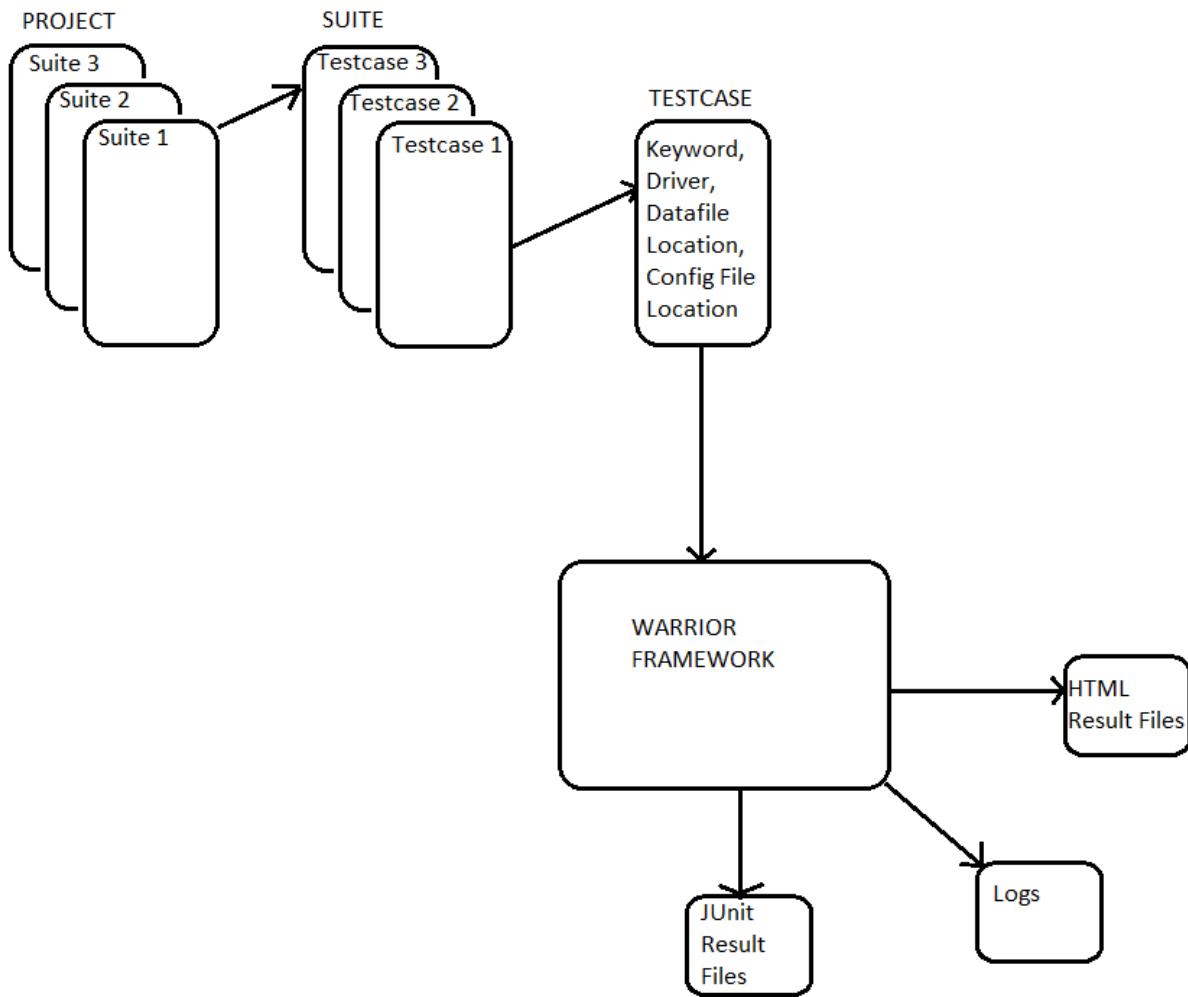
1.1 WHAT MAKES WARRIOR FRAMEWORK GENERIC?

Warrior Framework's structure can be essentially broken down in the following components:

1. Framework
2. Keywords
3. Cases (also, Suites and Projects)

4. Data Files

A diagrammatic representation of Warrior Framework:



The above diagram gives a quick overview of what each component does in Warrior Framework. The case stores the data file location and other execution details like which driver and keyword to use, whether a particular keyword impacts the case results, and so on. The data file stores the system information, the driver acts an interface between the case and the keyword, and keyword executes the command sent to it by the case. At the end, Warrior Framework generates html result files, JUnit result files, xml result files and Logs that provide a complete documentation of each and every keyword's execution results.

A project is a collection of Suites, which in turn is a collection of Cases.

Every component here is independent of every other component. Exactly this, makes Warrior Framework a customizable automation framework. Given a set of cases and data files, changing the system

information in the data file or using a different data file altogether, lets you test different systems by changing just a single file. Using a different case, or by changing the existing case, you can test different scenarios without writing a single line in Python.

Warrior Framework comes with a set of Core Keywords that help you get started with automation. Those keywords may not be sufficient for you if you have specific requirements, in that case, Warrior Framework allows you to write your own drivers and keywords, and those new, specific keywords can be used in your cases.

1.2 WHAT NOT TO DO IN WARRIOR FRAMEWORK?

Warrior Framework was designed to allow users to effectively create reusable generic keywords and leave the execution intelligence out of the keywords to be handled by Warrior Framework. In order for your automation efforts to be successful using Warrior Framework, please follow the following recommendations:

- Do not write keywords as one large python script to be executed. Design your effort and break up the script into re-usable keywords. Warrior Framework is not a python scripting tool. If your goal is to write cases as python scripts, Warrior Framework should not be used.
- Use the recommended Warriorspace directory structure for creating cases, suites and projects. It will simplify the use of Warrior Framework.
- Do not combine product drivers into one product driver. These drivers were designed to allow for smaller keyword files grouped by functionality or any logical demarcation you choose. If you combine into one driver, your keyword files will be large over time.

2.0 WARRIOR FRAMEWORK INSTALLATION

Warrior Framework comes with an installation tool – Warhorn ([Warhorn User Guide: Section 1](#)) – that automates the process of installing Warrior Framework on your system. Warhorn will install Warrior Framework, the necessary keywords and Warriorspaces, and Warrior Framework recommended prerequisites for you when it is executed. More information about Warhorn can be obtained at the end of this document in the Warhorn User Guide ([Section 1](#))

If you still want to proceed with installing Warrior Framework without using Warhorn, then here are the steps for the manual install:

The Warrior Framework automation framework is located in the following git repository:

```
http://rtx-swtl-git.fnc.net.local/scm/war/Warrior\_Framework\_main.git
```

Development activities on Warrior Framework will be done on the master branch of the above repository. Stable tested versions of Warrior Framework will be marked as tags in the master branch at regular intervals.

After cloning specific version check the release notes available in `Warrior_main/release_notes.txt` for details specific to the release.

Procedure to get a specific version of Warrior Framework:

Clone Warrior Framework main

```
$ git clone http://rtx-swtl-git.fnc.net.local/scm/war/Warrior_main.git
```

Go to `Warrior_main` directory

```
$ cd Warrior_main
```

To check the list of versions available, execute "git tag --list" command

```
$ git tag -list
```

Response:

Page 10 of 163

```
v1.0  
v1.1  
v1.2  
v1.3  
v1.4  
v1.5  
$
```

To check the current version you are at, execute "git branch" command

```
git branch
```

Response:

```
* master  
$
```

- * indicates the active version.

- In the above example master is the active version.

- If the active version is master it means you are using a version of Warrior Framework framework that is under development. It is not a stable tested version.

4. To switch to a specific version from master, execute 'git checkout <version_name>' command.

```
$ git checkout v1.4
```

Response:

```
Note: checking out 'v1.4'.  
You are in 'detached HEAD' state. You can look around, make experimental  
changes and commit them, and you can discard any commits you make in this  
state without impacting any branches by performing another checkout.  
If you want to create a new branch to retain commits you create, you may do  
so (now or later) by using -b with the checkout command again. Example:  
git checkout -b <new-branch-name>
```

WARRIOR FRAMEWORK

REVISION: 5

```
HEAD is now at 146313d... reduce command timeout, wait after timeout=60
$
```

Execute git branch command to verify the active version.

```
git branch
```

Response:

```
* (HEAD detached at v1.4)
master
$
```

- * indicates the active version

Switch from one version to another (current=v1.4, switch to v1.3)

```
$ git checkout v1.3
```

Response:

```
Previous HEAD position was 146313d... reduce command timeout, wait after
timeout=60
HEAD is now at ecb6373... WAR-180, handle and prompt on timeout
$
```

Executing this command would let you verify the active version:

```
$ git branch
```

Response:

```
* (HEAD detached at v1.3)
master
$
```

Switch to master branch again.

```
$ git checkout master
```

Response:

```
Previous HEAD position was ecb6373... WAR-180, handle and prompt on timeout
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
$
```

Executing this command would let you verify the active version:

```
$ git branch
```

Response:

```
* master
$
```

3.0 WARRIOR FRAMEWORK TOOLS

Warrior Framework comes with a set of handy tools that help in easing up the Warrior Framework setup, case, suite, and project creation, and their verifications.

3.1 WARHORN: AN OVERVIEW

Warhorn is an installation tool that automates the process of installing Warrior Framework and external Keywords and Warriorspace for you. Warhorn also installs the pre-requisites that Warrior Framework recommends you need install. Warhorn allows you to customize your local Warrior Framework to suit your requirements.

A more detailed and step-by-step documentation on how to install and run Warhorn can be found in

[Warhorn User Guide: Section 1](#)

3.2 KATANA: AN OVERVIEW

Katana is Warrior Framework's web based case creation and execution tool. From Katana, users can create cases, suites, projects, and input data files. Cases, suites, projects, and input data files in Katana are web based forms that once saved, create the appropriate xml files. Each field in the form has a tooltip explaining its functionality. Katana also allows you to execute cases, suites, and projects from it's execute tab.

To start using Katana, refer to [Katana User Guide: Section 1](#)

3.3 IRONCLAW: AN OVERVIEW

Ironclaw is Warrior Framework's xml verification tool. Ironclaw can be used to verify the structure of the xml files used by Warrior Framework. Ironclaw will automatically detect the xml type (case, suite, project) and do the following as applicable:

3.3.1 VERIFICATION OF CASE XML FILES:

Once Ironclaw identified that the xml is a case xml. It will do the following:

1. Verify conformance to case XSD.
2. Verify that the Input Data File defined in the xml file exists.
3. Verify the validity of keywords: will check if the keyword is in the corresponded actions package.
4. Verify that the driver exists in the Product Drivers folder.

3.3.2 VERIFICATION OF SUITE XML FILES:

Once Ironclaw identified that the xml is a suite xml. It will do the following:

1. Verify conformance to suite XSD.
2. Verify all cases listed in suite exist.
3. Verify each case in the suite as described in [Section 7.0](#) of this User Guide

3.3.3 VERIFICATION OF PROJECT XML FILES:

Once Ironclaw identified that the xml is a project xml. It will do the following:

1. Verify conformance to project XSD.
2. Verify all suites listed in project exist.
3. Verify each suite as described in [Section 8.0](#) of this User Guide

3.3.4 RUNNING IRONCLAW

To run Ironclaw execute the following command:

```
Warrior Framework -ironclaw /path/case.xml
```

3.4 MOCKRUN: AN OVERVIEW

MockRun is a Warrior Framework Tool which mocks the device under test to help automation teams run the Tests without the device (hardware) required for that test or its software simulation. MockRun helps automation teams design, develop, and deliver tests without being dependent on hardware or software availability, thus, helping agile teams increase their velocity of achieving automation goals.

Currently MockRun is supported only on CLI modules, i.e. it mocks and verifies all the Keywords in the CLI driver.

To run MockRun, type in this command:

```
python Warrior Framework -mockrun path/to/case
```

When using `-mockrun`, the test script is validated for Errors and Warnings. These Errors and Warnings are logged in the Execution directory under the ‘mock run’ with a time stamp. To summarize these errors and warnings, you can use the `-summary` option:

```
python Warrior Framework -summary -mockrun path/to/case
```

4.0 UNDERSTANDING THE WARRIOR FRAMEWORK DIRECTORY STRUCTURE

There are in total seven directories and one Warrior Framework executable in the Warrior Framework directory. What to expect in each directory and its utility has been explained in detail below. Whether you are a case developer or a keyword developer, acclimatizing yourself to Warrior Framework directory structure is very important for a good understanding of Warrior Framework. This is the current Warrior Framework directory structure:

Warrior			
Name	Date modified	Type	Size
Actions	5/12/2016 3:03 PM	File folder	
Docs	5/12/2016 3:03 PM	File folder	
Framework	5/12/2016 3:03 PM	File folder	
ProductDrivers	5/12/2016 3:03 PM	File folder	
Tools	5/12/2016 3:04 PM	File folder	
WarriorCore	5/12/2016 3:03 PM	File folder	
Warriorspace	5/12/2016 3:03 PM	File folder	
Warrior	5/12/2016 3:03 PM	File	8 KB

4.1 ACTIONS DIRECTORY

Warrior Framework's Actions directory hosts the keyword driven libraries. The python libraries that contain the keyword functions are located in this directory. Any future keyword function development will need to be added to either an existing library or to a new library within the Actions directory. All these libraries are explained in detail in [Section 5.0](#) of this User Guide

WARRIOR FRAMEWORK

REVISION: 5

Actions				
Name	Date modified	Type	Size	
📁 CliActions	9/12/2016 10:58 A...	File folder		
📁 CloudshellActions	9/12/2016 10:54 A...	File folder		
📁 CommonActions	9/12/2016 10:54 A...	File folder		
📁 DemoActions	9/12/2016 10:58 A...	File folder		
📁 ExampleActions	9/12/2016 10:54 A...	File folder		
📁 NetconfActions	9/12/2016 10:58 A...	File folder		
📁 NetworkActions	9/12/2016 10:54 A...	File folder		
📁 RestActions	9/12/2016 11:11 A...	File folder		
📁 SeleniumActions	9/12/2016 10:58 A...	File folder		
📁 SnmpActions	9/12/2016 10:54 A...	File folder		
📄 _init_	9/12/2016 10:54 A...	Python File	0 KB	

4.2 PRODUCTDRIVERS DIRECTORY

The Warrior Framework driver files reside in the Product Drivers directory. Drivers are python files that make up the execution layer of the Warrior Framework framework. The product drivers will interact with the xml based case and start the execution of the steps based on the product driver indicated in the step. A user should be modifying or creating new product drivers. For a new product, the user must create a new product driver from the driver template provided in the Drivers directory.

ProductDrivers				
Name	Date modified	Type	Size	
📄 _init_	5/12/2016 3:03 PM	Python File	1 KB	
📄 cli_driver	5/12/2016 3:03 PM	Python File	1 KB	
📄 common_driver	5/12/2016 3:03 PM	Python File	1 KB	
📄 demo_driver	5/12/2016 3:03 PM	Python File	1 KB	
📄 driver_template	5/12/2016 3:03 PM	Python File	1 KB	
📄 netconf_driver	5/12/2016 3:03 PM	Python File	1 KB	
📄 network_driver	5/12/2016 3:03 PM	Python File	1 KB	
📄 rest_driver	5/12/2016 3:03 PM	Python File	1 KB	
📄 snmp_driver	5/12/2016 3:03 PM	Python File	1 KB	

The following are the existing drivers:

<u>Product Driver</u>	<u>Description</u>

cli_driver	Product driver that invokes execution of all keyword functions that can be used for command line interface testing.
common_driver	Product driver that invokes execution of all the common keyword functions that can be re-used among multiple products.
demo_driver	Product driver that invokes execution of all demo test related keyword
driver_template	Sample driver to be used in creating a new product driver. The instructions on how to create the new product driver are detailed as part of the comments in the code.
netconf_driver	Product driver that invokes execution of all keyword functions that can be used for netconf interface testing.
network_driver	Product driver that invokes execution of all keyword functions that can be used for all network related testing.
rest_driver	Product driver that invokes execution of all keyword functions that can be used for REST interface testing.
snmp_driver	Product driver that invokes execution of all SNMP keywords.

4.3 FRAMEWORK DIRECTORY

Warrior Framework's system layer is in the Framework directory. The generic framework libraries that support the functional and execution layer are located in this directory. These libraries provide services such as data access, reading, writing, and log file creation, parsing and test results processing. Full documentation for all the modules can be obtained in the Warrior Framework_docs repository located at

http://rtx-swtl-git.fnc.net.local/projects/WAR/repos/Warrior_docs/browse

in main/source/index.html.

This Framework directory has two subdirectories: ClassUtils, and Utils.

Framework				
Name	Date modified	Type	Size	
ClassUtils	5/12/2016 3:03 PM	File folder		
Utils	5/12/2016 3:03 PM	File folder		
init	5/12/2016 3:03 PM	Python File	1 KB	

Further, the ClassUtils directory has two subdirectories – WNetwork and WSelenium and eight files out of which one is netconf.py. This particular file was added in as a replacement for the ncclient pre-requisite that Warrior Framework had until v1.8. All the other files are class files that provide specific utilities for objects created by you. Every file has been described in detail below.

ClassUtils				
Name	Date modified	Type	Size	
WNetwork	5/12/2016 3:03 PM	File folder		
WSelenium	5/12/2016 3:03 PM	File folder		
init	5/12/2016 3:03 PM	Python File	0 KB	
configuration_element_class	5/12/2016 3:03 PM	Python File	8 KB	
json_utils_class	5/12/2016 3:03 PM	Python File	7 KB	
netconf	5/12/2016 3:03 PM	Python File	20 KB	
netconf_utils_class	5/12/2016 3:03 PM	Python File	12 KB	
rest_utils_class	5/12/2016 3:03 PM	Python File	12 KB	
snmp_utility_class	5/12/2016 3:03 PM	Python File	3 KB	
ssh_utils_class	5/12/2016 3:03 PM	Python File	15 KB	
testdata_class	5/12/2016 3:03 PM	Python File	37 KB	

1. configuration_element_class.py

This module provides the following utilities for the ConfigurationElement object. The functions in this class provide utilities for finding a match against a regular expression, function for replacing variables with their values, functions for parsing data, getting a node in the tree and so on.

2. json_utils_class.py

This module provides API for JSON related operations. This class provides you with functions that can sort a JSON object, calculate difference between two JSON objects and files, writes JSON to a file, and print JSON.

3. netconf.py

This file was added in as a replacement for the ncclient pre-requisite that Warrior Framework had until v1.8. It is recommended that this file not be modified.

4. netconf_utils_class.py

These are API for operations related to NetConf Interfaces. The Requests package is used here. This class provides functionalities to open and close an SSH connection to a Netconf system. It also provides functionalities to get, edit, copy, and delete configuration from the datastore, lock, unlock, validate, and get configuration system, create and wait for subscriptions.

5. rest_utils_class.py

API for operations related to REST Interfaces. The Requests package is used in this module. This module provides you with HTTP methods such as post, get, put, patch, delete, options, and head.

6. snmp_utils_class.py

This is SNMP utility module using the python PYSNMP module. This module provides functionalities for generating commands, creating a communityData object, creating a UDP transport object, creating an MIB object, to check for Octet string.

7. testdata_class.py

This module has all testdata related class and methods that provide utilities for substituting the [VAR_SUB] patterns, resolving, getting, and validating the iteration patterns, expand, validate, and verify iteration patterns provided in each verification search provided for the command and more.

8. WNetwork

The WNetwork directory has all the utilities for a Network provided by Warrior Framework.

WNetwork			
Name	Date modified	Type	Size
__init__	5/12/2016 3:03 PM	Python File	0 KB
base_class	5/12/2016 3:03 PM	Python File	1 KB
connection	5/12/2016 3:03 PM	Python File	1 KB
diagnostics	5/12/2016 3:03 PM	Python File	3 KB
file_ops	5/12/2016 3:03 PM	Python File	22 KB
logging	5/12/2016 3:03 PM	Python File	3 KB
network_class	5/12/2016 3:03 PM	Python File	1 KB

8.1 base_class.py

Page 22 of 163

Base class for Network package. This class is inherited in all classes of the Network package. Inheriting this class avoids the problem of calling object with **args/**kwargs while using super method (while invoking the methods of the Network package using an instance of Network class)

8.2 connection.py

This is the Warrior Framework Network connectivity module. This has three defined functions: connect, connect_ssh, connect_telnet.

8.3 diagnostics.py

Warrior Framework Network diagnostics module contains utilities to ping and trace the route from the remote host

8.4 file_ops.py

This is the Warrior Framework Network File operations module which contains functionalities to start FTP and SFTP on the remote, FTP and SFTP from the remote host, and check if a certain file exists on remote host.

8.5 Logging.py

This module collects the response on a connected session as a separate thread. It can start and stop the thread, retrieve status its status, and collect logs generated by the thread.

8.6 Network_class.py

This is class that inherits all other classes in the Network package. Instance of this class may be used to invoke the methods of all the other classes in this package. While using the instance of this class, if the Base classes have same method name, then the method will be executed based on python's inheritance MRO (Method Resolution Order). Execute 'Network.__mro__' to find out the current MRO. Order is from left to right of the MRO.

9. WSelenium

The WSelenium package provides utilities for operations related to browser functionalities and element locators

WSelenium				
Name	Date modified	Type	Size	
__init__	5/12/2016 3:03 PM	Python File	0 KB	
browser_mgmt	5/12/2016 3:03 PM	Python File	8 KB	
element_locator	5/12/2016 3:03 PM	Python File	7 KB	
element_operations	5/12/2016 3:03 PM	Python File	4 KB	
wait_operations	5/12/2016 3:03 PM	Python File	2 KB	

9.1 browser_mgmt.py

This is the Selenium browser management library. It provides functionalities for opening and closing a browser, resizing the browser window, going back and reloading pages and so on.

9.2 Element_locator.py

This is the Selenium element locator library. It provides you with utilities that get elements by name, xpath, link text, partial link, ID, tag name, class name, and CSS selector.

9.3 Element_operation.py

This is the Selenium element operations library. It provided functionalities for getting and clicking an element, performing an element action, sending keys, clearing text and so on.

9.4 Wait_operation.py

This is the wait or sleep operations library with functionalities for waiting till a particular stage of the HTML element is reached, and for getting a function provided by the locator.

10. Utils Directory

The Utils directory contains modules that provide various utilities. They have been described briefly below.

Utils				
Name	Date modified	Type	Size	
__init__	5/12/2016 3:03 PM	Python File	1 KB	
cli_Utils	5/12/2016 3:03 PM	Python File	35 KB	
config_Utils	5/12/2016 3:03 PM	Python File	2 KB	
data_Utils	5/12/2016 3:03 PM	Python File	37 KB	
datetime_utils	5/12/2016 3:03 PM	Python File	2 KB	
demo_utils	5/12/2016 3:03 PM	Python File	1 KB	
dict_Utils	5/12/2016 3:03 PM	Python File	3 KB	
driver_Utils	5/12/2016 3:03 PM	Python File	1 KB	
email_utils	5/12/2016 3:03 PM	Python File	4 KB	
encryption_utils	5/12/2016 3:03 PM	Python File	1 KB	
file_Utils	5/12/2016 3:03 PM	Python File	19 KB	
import_utils	5/12/2016 3:03 PM	Python File	1 KB	
list_Utils	5/12/2016 3:03 PM	Python File	2 KB	
print_Utils	5/12/2016 3:03 PM	Python File	4 KB	
rest_Utils	5/12/2016 3:03 PM	Python File	20 KB	
string_Utils	5/12/2016 3:03 PM	Python File	5 KB	
telnet_Utils	5/12/2016 3:03 PM	Python File	4 KB	
testcase_Utils	5/12/2016 3:03 PM	Python File	10 KB	
xml_Utils	5/12/2016 3:03 PM	Python File	21 KB	

10.1 cli_Utils.py

This is the API for cli related operations. It has function that can connect and disconnect to SSH and Telnet sessions, send command and get the response of that command, get object session, connection port, and the dictionary containing all the responses, send ping and source ping, and more.

10.2 config_Utils.py

This module provides functionalities for setting the Logs directory, Log file, the datarepository, the Result file, the Debug file, and the JUnit file.

10.3 data_Utils.py

This is the API for testdata related functionality. This module provides utilities to get the system data, system and subsystem name, session ID, the variable configuration file, object from data repository, command details, command parameters, verification details, the variable configuration file details, REST

data, Netconf data, verify command response and response across multiple systems and other such functions. Please refer to the Warrior Framework_docs repository for full documentation.

10.4 datetime_Utils.py

This is the utility for date time functionalities such as getting the current time stamp.

10.5 demo_Utils.py

This API is for demo purposes only.

10.6 dict_Utils.py

This is the utility for functionalities related to dictionaries. This module provides utilities to convert string into dictionaries.

10.7 driver_Utils.py

This module gathers the argument information about the keywords, executes the keywords and reports the keyword status back to the product driver. This module will be deprecated in the next release of Warrior Framework.

10.8 email_Utils.py

Provides utility to send email using SMPT

10.9 encryption_Utils.py

This file provides utilities for data encryption.

10.10 file_Utils.py

This file provides all utilities for file manipulation, creation, and deletion.

10.11 import_Utils.py

This file provides utilities for package verifications and imports.

10.12 list_Utils.py

This is the library to hold all APIs related to list operations

10.13 print_Utils.py

This is the Warrior Framework Frameworks print library that provides utilities to print outputs on to the console and simultaneously marking them as ‘Information’, ‘Error’, ‘Exception’, and ‘Debug’.

10.14 rest_Utils.py

This is the REST Utils library that provides utilities for all REST related functionalities like resolving the value of allow_redirects, timeouts, certificate, stream, data and other attributes that the user can provide through the case or the datafile.

10.15 string_Utils.py

This is the library to hold all APIs related to string operations

10.16 telnet_Utils.py

This library provides all telnet related utilities which can open a communication using the telnet protocol, connecting to an NE and closing it, writing and reading to and from the terminal, and getting the output until the prompt and returning the results.

10.17 case_Utils.py

This module contains methods for case results related operations. The utilities provided by this module can report a substep status, open and close a file, create a root tag with the name Case, create tags with a particular name under the case tag, update the keyword result file, get value of impact, context, and onError from the case, and more such functionalities. Please refer to the full documentation available in the Warrior Framework_docs repository.

10.18 xml_Utils.py

This module contains methods for all xml related functionalities that can create a subelement, get value of an XML node by tag name and attribute, get the first and the last child of an XML node, get the root and the tree from a file, get child of an XML parent node, and so on. Please refer the Warrior Framework_docs repository for a complete documentation.

4.4 TOOLS DIRECTORY

The tools directory contains directories containing files specific to Warrior Frameworks integration with other tools. There are four directories contained under this directory – admin, Jira, Reporting, Xsd and one file – w_settings.xml

Tools				
Name	Date modified	Type	Size	
admin	5/12/2016 3:03 PM	File folder		
Jira	5/12/2016 3:03 PM	File folder		
Reporting	5/12/2016 3:03 PM	File folder		
Xsd	5/12/2016 3:03 PM	File folder		
__init__	5/12/2016 3:03 PM	Python File	0 KB	
w_settings	5/12/2016 3:03 PM	XML Document	1 KB	

4.4.1 THE ADMIN DIRECTORY

Currently, the admin directory consists of a single file - the secret key file.

admin				
Name	Date modified	Type	Size	
secret.key	5/12/2016 3:03 PM	KEY File	1 KB	

The secret.key is a plain text file that stores your AES 64 bit encoded key that encodes and decodes any sensitive information like passwords that may have to be entered into the XML file.

4.4.2 JIRA DIRECTORY

Jira				
Name	Date modified	Type	Size	
jira_config	5/12/2016 3:03 PM	XML Document	1 KB	

The Jira directory contains a jira.xml file to be configured by each installation. The XML file can contain multiple systems each being a separate Jira project. One project can be defined as a default project. When

executing Warrior Framework through the CLI, if a Jira project is not specified, Warrior Framework will use the project defined as default.

To define a project as default, it should be marked as default="true". Only ONE project can be default. If multiple projects are defined as default, Warrior Framework will use the first project marked as default. If no project is marked default, then Warrior Framework will use the first project in the xml file. In the screen shot below the last project is marked default:

```
<?xml version="1.0" ?>

<jira>

    <system name="warrior" >
        <url>https://demojiraplat.atlassian.net/</url>
        <username>admin</username>
        <password>satmu1323*</password>
        <Assignee>Automatic</Assignee>
        <project_key>WAR</project_key>
    </system>

    <system name="project1">
        <url>https://demojiraplat.atlassian.net/</url>
        <username>admin</username>
        <password>satmu1323*</password>
        <Assignee>Automatic</Assignee>
        <project_key>PROJ</project_key>
    </system>

    <system name="project2" >
        <url>https://demojiraplat.atlassian.net/</url>
        <username>admin</username>
        <password>satmu1323*</password>
        <Assignee>Automatic</Assignee>
        <project_key>PROJ2</project_key>
    </system>

    <system name="project3" default="true">
        <url>https://demojiraplat.atlassian.net/</url>
        <username>admin</username>
        <password>satmu1323*</password>
        <Assignee>Automatic</Assignee>
        <project_key>PROJ3</project_key>
    </system>

</jira>
```

4.4.2.1 AUTO DEFECT CREATION

Automatically create JIRA defects against the default JIRA project from jira_config file:

```
./Warrior Framework -ad path/to/tc1.xml path/to/tc2.xml path/to/ts1.xml
path/to/ts2.xml path/to/proj1.xml
```

Automatically create JIRA defects against the provided JIRA project from JIRA config file:

```
./Warrior Framework -ad -jiraproj jiraproj name path/to/tc1.xml  
path/to/tc2.xml path/to/ts1.xml
```

Create defects in default JIRA project for all the defects JSON files present in defects_directory1, defects_directory2:

```
./Warrior Framework -ujd -ddir path/to/defects_directory1  
path/to/defects_directory2
```

Create defects in default JIRA project for the defects JSON files defects_json1 and defects_json2:

```
./Warrior Framework -ujd -djson path/to/defects_json1 path/to/defects_json2
```

Create defects in provided JIRA project for all the defects JSON files present in defects_directory1 and defects_directory2

```
./Warrior Framework -ujd -jiraproj jiraproj_name -ddir  
path/to/defects_directory1 path/to/defects_directory2
```

Create defects in provided JIRA project for the defects JSON files path/to/defects_json1 and path/to/defects_json2

```
./Warrior Framework -ujd -jiraproj jiraproj_name -djson  
path/to/defects_json1 path/to/defects_json2
```

4.4.3 THE REPORTING DIRECTORY

This contains the files needed for the HTML file generation and reporting. It is imperative that these files not be modified.

Reporting				
Name	Date modified	Type	Size	
html_results_template	5/12/2016 3:03 PM	Chrome HTML Do...	1 KB	
wjunit_to_xunit	5/12/2016 3:03 PM	XSL Stylesheet	3 KB	

4.4.4 THE XSD DIRECTORY

This directory contains the XSDs required by Ironclaw to verify if the Cases, Suites, and Projects are correct. It is recommended that these files not be modified.

Name	Date modified	Type	Size
warrior_project.xsd	5/12/2016 3:03 PM	XSD File	3 KB
warrior_suite.xsd	5/12/2016 3:03 PM	XSD File	4 KB
warrior_testcase.xsd	5/12/2016 3:03 PM	XSD File	9 KB

4.4.5 THE W_SETTINGS FILE

The w_settings.xml file that comes with Warrior Framework is used to set the settings for Warrior Framework features.

```

<?xml version="1.0" encoding="UTF-8"?>
<Default>
    <Setting name="def_dir">
        <Logsdir/>
        <!--
            Optional, uses default, support absolute path, relative path and environment variable(${ENV.DEF_LOGSDIR})
        -->
        <Resultsdir/>
        <!--
            Optional, uses default, support absolute path, relative path and environment variable(${ENV.DEF_RESULTSDIR})
        -->
    </Setting>
    <Setting name="mail_to">
        <smtp_host/>
        <!--
            Optional, use FNC smtp host smtp.fnc.fujitsu.com to notify test case execution results
        -->
        <sender/>
        <!--
            Optional, use sender's email such as Warrior_PBO@fnc.fujitsu.com
        -->
        <receiver/>
        <!--
            Optional, use receiver's email such as FNC.User@fnc.fujitsu.com
        -->
        <subject/>
        <!--
            Optional, use this as subject line such as WARRIOR , additional test case execution results Pass/Fail and Test Execution File shall be appended to this
        -->
    </Setting>
</Default>

```

In the w_settings.xml you can add a path to the Logs directory and Results directory – so when you run Warrior Framework, it would automatically use the paths given here, rather than the default.

Inside the <Setting name=mail_to>, you can add the information about the host, sender, receiver, and subject so that after every Warrior Framework run, you will receive an email with the results and logs attached to it. If these details are wrong or left unfilled, Warrior Framework will not be able to send an email.

WARRIOR FRAMEWORK

REVISION: 5

4.5

DOCS DIRECTORY

Warrior Framework docs are generated using sphinx. The html files generated by Sphinx are located in the Docs directory of Warrior Framework. Using a browser, open the Warrior Framework_index.html file located in the docs directory of Warrior Framework to view the keywords. You will then be able to view and search for keywords and review the keyword documentation. The docs also will display the python libraries within the Framework directory.

Docs				
Name	Date modified	Type	Size	
source	5/12/2016 3:03 PM	File folder		
make	5/12/2016 3:03 PM	Windows Batch File	8 KB	
Makefile	5/12/2016 3:03 PM	File	8 KB	
Run_Warrior_with_CYGWIN	5/12/2016 3:03 PM	Adobe Acrobat D...	468 KB	
Warrior_Framework_UserGuide_rev7	5/12/2016 3:03 PM	Adobe Acrobat D...	887 KB	

WARRIOR FRAMEWORK

REVISION: 5

source

Name	Date modified	Type	Size
Actions.CliActions.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.CommonActions.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.CseActions.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.DemoActions.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.ExampleActions.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.NetconfActions.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.NetworkActions.Connection.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.NetworkActions.Diagnostics.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.NetworkActions.FileOps.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.NetworkActions.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.QtpActions.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.RestActions.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.SnmpActions.rst	5/12/2016 3:03 PM	RST File	1 KB
Actions.SshActions.rst	5/12/2016 3:03 PM	RST File	1 KB
conf	5/12/2016 3:03 PM	Python File	10 KB
Framework.ClassUtils.rst	5/12/2016 3:03 PM	RST File	2 KB
Framework.ClassUtils.WNetwork.rst	5/12/2016 3:03 PM	RST File	2 KB
Framework.ClassUtils.WSelenium.rst	5/12/2016 3:03 PM	RST File	2 KB
Framework.rst	5/12/2016 3:03 PM	RST File	1 KB
Framework.Utils.rst	5/12/2016 3:03 PM	RST File	4 KB
modules.rst	5/12/2016 3:03 PM	RST File	1 KB
warrior_index.rst	5/12/2016 3:03 PM	RST File	1 KB

4.6 WARRIOR FRAMEWORKCORE DIRECTORY

This directory contains the python files that compose Warrior Framework's execution engine. Do not make any changes to the files in this directory.

4.7 WARRIORSPACE DIRECTORY

Warrior Framework uses this directory as the default location for case, data files and execution results will reside.

WARRIOR FRAMEWORK

REVISION: 5

Warriorspace

Name	Date modified	Type	Size
Config_files	5/12/2016 3:03 PM	File folder	
Data	5/12/2016 3:03 PM	File folder	
Execution	5/12/2016 3:03 PM	File folder	
Projects	5/12/2016 3:03 PM	File folder	
Suites	5/12/2016 3:03 PM	File folder	
Testcases	5/12/2016 3:03 PM	File folder	

4.7.1 CASES DIRECTORY

This directory will include the case.xml files that define the keywords to be executed. This is the default space provided by Warrior Framework for you to save your cases. You can use a different directory to save the cases.

Testcases

Name	Date modified	Type	Size
samples	5/12/2016 3:03 PM	File folder	
demo_tc1	5/12/2016 3:03 PM	XML Document	3 KB
demo_tc2	5/12/2016 3:03 PM	XML Document	2 KB
demo_tc3	5/12/2016 3:03 PM	XML Document	2 KB
Testcase_Template	5/12/2016 3:03 PM	XML Document	3 KB

samples

Name	Date modified	Type	Size
cli_tc_sample	5/12/2016 3:03 PM	XML Document	2 KB
netconf_tc_sample	5/12/2016 3:03 PM	XML Document	3 KB
rest_tc_sample	5/12/2016 3:03 PM	XML Document	18 KB
testset_tc_sample	5/12/2016 3:03 PM	XML Document	4 KB

4.7.2 SUITES DIRECTORY

This directory will include the suite.xml files that define the suites to be executed. This is the default space provided by Warrior Framework for you to save your suites. You may use a different location.

WARRIOR FRAMEWORK

REVISION: 5

Suites

Name	Date modified	Type	Size
 demo_suite	5/12/2016 3:03 PM	XML Document	2 KB
 Suite_Template	5/12/2016 3:03 PM	XML Document	2 KB

4.7.3 PROJECTS DIRECTORY

This directory will include the project.xml files that define the projects to be executed. This is the default space provided by Warrior Framework for you to save your projects. You may use a different location.

Projects

Name	Date modified	Type	Size
 demo_project	5/12/2016 3:03 PM	XML Document	1 KB
 Project_Template	5/12/2016 3:03 PM	XML Document	2 KB

4.7.4 DATA DIRECTORY

The Data directory will contain the input data files as indicated in the case.xml. If the case.xml does not define a file, Warrior Framework will search for data file with the same name as the case_Data.xml. If a data is not found, Warrior Framework will assume that the case does not require a data file.

Data

Name	Date modified	Type	Size
 samples	5/12/2016 3:03 PM	File folder	
 demo_tc_Data	5/12/2016 3:03 PM	XML Document	3 KB

samples

Name	Date modified	Type	Size
 input_datafile_sample	5/12/2016 3:03 PM	XML Document	7 KB
 netconf_Data_sample	5/12/2016 3:03 PM	XML Document	1 KB
 rest_datafile_sample	5/12/2016 3:03 PM	XML Document	13 KB
 testset_Data_sample	5/12/2016 3:03 PM	XML Document	1 KB

4.7.5 CONFIG_FILES DIRECTORY

The Config_files directory will contain the configuration files as indicated in the input data files.

Config_files				
Name	Date modified	Type	Size	
samples	5/12/2016 3:03 PM	File folder		
config_readme	5/12/2016 3:03 PM	Text Document	1 KB	
demo_tc_testdata	5/12/2016 3:03 PM	XML Document	1 KB	

samples				
Name	Date modified	Type	Size	
netconf_config_data_sample	5/12/2016 3:03 PM	XML Document	2 KB	
netconf_myconfig_sample	5/12/2016 3:03 PM	XML Document	1 KB	
testdata_sample	5/12/2016 3:03 PM	XML Document	28 KB	
testset_config_sample	5/12/2016 3:03 PM	XML Document	4 KB	
variable_config_sample	5/12/2016 3:03 PM	XML Document	2 KB	

4.7.6 EXECUTION DIRECTORY

The Execution directory will be automatically created by Warrior Framework under Warriorspace directory. If the Execution directory already exists, Warrior Framework will use the existing directory. During the execution of a case, suite or project, Warrior Framework will create as subdirectory named after the time/date of the execution. Under the time/date directory, a project, suite or case directory will be created and the result and log files will be created under the applicable directory.

Execution				
Name	Date modified	Type	Size	
Execution_readme	5/12/2016 3:03 PM	Text Document	1 KB	

5.0 UNDERSTANDING THE BUILT-IN KEYWORDS

Warrior Framework provides the user with a set of built-in generic keywords.

The documentation for these modules can be found here: warrior_fw/docs/build/html/Actions.html

5.1 CLI ACTIONS PACKAGE

The CliActions package is the first Actions package that you will arrive at after opening the Actions directory. The CliActions class has methods (keywords) related to actions that can be performed on any command line interface.

5.2 CLOUDSHELL ACTIONS PACKAGE

The CloudshellActions class has methods (keywords) related to actions that can be performed on a CloudShell system.

5.3 COMMON ACTIONS PACKAGE

The CommonActions package contains the keywords that are common to all products that are developed. The CommonActions class has methods (keywords) that are common for all the products.

5.4 DEMO ACTIONS PACKAGE:

The DemoActions package that has all demo case related keywords. These are the keywords used in the Warrior Framework Demo cases. They are there for demo purposes only. The DemoActions class has methods (keywords) related to actions used in demo KW.

5.5 NETCONF ACTIONS PACKAGE

The NetconfActions package contains all netconf related keywords. The NetconfActions class has methods (keywords) related to actions performed on any basic netconf interface.

5.6 NETWORK ACTIONS PACKAGE

The NetworkActions package contains three subpackages – The Connection package, the Diagnostics package, and the FileOps package.

5.7 REST ACTIONS PACKAGE

The RestActions package contains the RestActions class. This class has all the keywords related to REST operations.

5.8 SELENIUM ACTIONS PACKAGE

The SeleniumActions package contains keywords related to Selenium operations.

5.9 SNMP ACTIONS PACKAGE

The SnmpActions package has the implementation of the standard SNMP protocol commands for SNMP v1 and v2c. It has the CommandSnmpActions class that contains all the keywords related to SNMP

6.0 UNDERSTANDING THE CASE

A Warrior Framework Case is an XML file written using specific tags that have been pre-defined by Warrior Framework. To correctly write a Warrior Framework Case, understanding what each and every tag means and the correct structure of the case is very important. A case can be created using Katana. Please refer to [Katana User Guide: Section 3](#) a complete guide to creating a case using Katana.

A simple case looks something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Testcase>
    <Details>
        <Name>Sample_Testcase</Name>
        <Title>Sample_Testcase</Title>
        <Engineer>Jo Smith</Engineer>
        <Date>2016-05-18</Date>
        <Time>11:12</Time>
        <State>New</State>
        <InputDataFile>../Data/Sample_Input_Data_file.xml</InputDataFile>
        <Datatype>Custom</Datatype>
        <default_onError action="next"/>
        <Logdir/>
        <Resultsdir/>
        <ExpectedResults>PASS</ExpectedResults>
        <Category>Iterative</Category>
    </Details>
    <Steps>
        <step Driver="cli_driver" Keyword="connect" TS="1">
            <Arguments>
                <argument name="system_name" value="CLI[INT]"/>
                <argument name="ip_type" value="ip"/>
                <argument name="prompt" value=".*(%|#|\$)"/>
                <argument name="session_name" value="session5"/>
            </Arguments>
            <onError action="abort"/>
            <Description>Connect to the ssh/telnet port of the system</Description>
            <iteration_type type="" />
            <Execute ExecType="Yes"/>
            <context>positive</context>
            <impact>impact</impact>
            <runmode type="rup" value="4"/>
        </step>
        <step Driver="cli_driver" Keyword="disconnect" TS="2">
            <Arguments>
                <argument name="system_name" value="CLI[INT]"/>
                <argument name="session_name" value="session5"/>
            </Arguments>
            <onError action="abort"/>
            <Description>Disconnects/Closes session established with the system/subsystem</Description>
            <iteration_type type="" />
            <Execute ExecType="Yes"/>
            <context>positive</context>
            <impact>impact</impact>
            <runmode type="rmt" value="5"/>
        </step>
    </Steps>
    <Requirements>
        <Requirement/>
    </Requirements>
</Testcase>
```

Tags in a Case:

<Case></Case> forms the root tag of the case. This root tag helps Warrior Framework identify this XML file as a Case. This tag has three child tags – Details, Steps, and Requirements.

1. Details

The <Details></Details> signifies that everything included inside this tag gives Warrior Framework information about the Case. This tag has thirteen child tags – Name, Title, Engineer, Category, Date, Time, State, InputDataFile, Datatype, default_OnError, Logsdir, Resultsdir, and ExpectedResults.

1.1. Name

The tag holds the Case name. It must be unique. The name of the file and the text of this tag should match. This is a mandatory field.

1.2. Title

The title lets you add a title or a description for this case so that you can refer to it later on, to understand the purpose of this case. This is a mandatory field.

1.3. Engineer

This tag stores the Case developer's name. This is a mandatory field.

1.4. Category

Feature or Category. This field will be used to group cases quickly. You can classify your case under any Category (like Sanity, or Alarm). This is an optional field, but it is highly recommended that you categorize your cases.

1.5. Date

The Date tag stores the date of creation of this Case. This is an optional field, only defined just so that you can date stamp a Case.

1.6. Time

The Time tag stores the time of creation of this Case. This is an optional field, only defined just so that you can time stamp a Case. This is an [optional](#) field.

1.7. State

This tag contains the state of the Case, i.e., whether the Case is a new, released, test-assigned, or anything else that you might consider appropriate for the Case. This is an [optional](#) field.

1.8. InputDataFile

Location of the data file to be used by the keyword. If the case does not need to use a data file, then a No_Data is required. If this is left blank, Warrior Framework will expect to find a data file named casename_data.xml in the Data directory of the Warriorspace.

1.9. Datatype

Determines the interaction of the case with the data file. If Iterative is selected, the case data file needs to use the system data file format. The steps of the case will run against the data in each system of the data file sequentially. If Custom is selected, the user can use a custom defined data file, with custom variables assigned to each keyword to access the input data file. The argument within a step, field will be used to access the data from the data file. If Hybrid is selected, each step gets run according to what has been specified against it. For more details on the Hybrid mode, please see the “Iteration_type Tag”. [The default is ‘Iterative’](#).

1.10 Default_OnError

Default error handling for the case. This is where you set what happens if a case step fails. User can override the default by selecting the on error in the step section. If the step has an on error selection, that will be used instead of the default. The Default_OnError options are: Next, Goto, and Abort.

- Selecting Next will result in the next step being executed if the step fails.
- Selecting Goto and the step number to go to will result in execution of the step indicated. The execution of steps will continue from that step going forward.

- Selection Abort will terminate the execution of the case upon failure or error. The step status would be set to FAIL

a. The Actions attribute

This is an attribute of the Default_OnError. This is an optional attribute. If not set, Warrior Framework will default to “next”, that is, it would proceed to the next Case. The options for this attribute are next, abort, and goto.

b. Value attribute

This is an attribute of the default_OnError tag but it is necessary only if the action attribute is set to “goto”. This attribute accepts value as a step number, that is, the number of the step that you want to jump to in case the current step throws an Error. Warrior Framework will then directly go to executing the step corresponding to the number given if the current step does not pass.

1.10. Logsdir

Location of directory where log files will be stored. If field is left blank Warrior Framework will use the default location under the execution folder of the Warrior Frameworkspace. This is therefore, an optional field.

1.11. Resultsdir

Location of directory where result files will be stored. If field is left blank, Warrior Framework will use what is in the case, if that has been left blank, Warrior Framework will use the location defined in the w_setting.py file, if that is also left blank, then Warrior Framework will use the default under the Execution directory. This is, therefore, an optional field.

1.12. ExpectedResults

The ExpectedResults tag is for your benefit, so that you know if a particular case is supposed to Pass or Fail or Error out. This is an optional field.

2 Requirements

User can add as many requirements as needed. These requirements are for informational purposes only. The requirements entered in the case will be shown in the xml result file.

2.1 Requirement

Every requirement needed for this case must be added inside a <Requirement></Requirement>. This is an optional field.

3 Steps

3.1 Step

User can add as many steps as needed but at least one step is required. Each step will have a Driver selection and a Keyword selection. User needs to indicate the product driver to be used in the step and the keyword.

3.1.1 Arguments

Section within the step where the arguments are detailed.

3.1.1.1 Argument

The argument tag is where you can pass arguments directly to the keywords through the Case. The name of the argument should be the value of the attribute 'name' in this tag, while the value to be passed for that argument should be the value of the 'value' attribute. The argument tag containing the system name only when the Custom mode is selected for the case.

3.1.2 OnError

Set what happens if the step fails. This will override the default of the case. Options are Next, Goto, Abort and Abort_as_error. The behavior is as follows:

- Selecting Next will result in the next step being executed if the step fails.
- Selecting Goto and the step number to go to will result in execution of the step indicated. The execution of steps will continue from that step going forward.
- Selection Abort will terminate the execution of the case if the step does not pass. The step status would be set to FAIL

- Selection Abort_as_error will terminate the execution of the case if the step does not pass. The step status would be set to ERROR.
- This is an optional field.

3.1.3 Description

Text description for each step can be specified within this tag. This is an optional field but it is highly recommended that you fill it out.

3.1.4 Iteration_type

This tag is required only when the Hybrid mode is selected. This can have three options – once_per_tc, once_per_iter, and standard. The ‘standard’ mode is the default where the step would run ‘normally’ – like how it would run, had it been an Iterative or Custom Datatype mode. The ‘once_per_tc’ runs that particular step only once per case while the ‘once_per_iter’ mode lets that particular step for every iteration that the set of steps go through.

3.1.5 Execute

This is an optional field that decides when a step should be executed. If this field is not set, the default will be to execute the keyword. The ExecType attribute has the options: If, If Not, Yes, No.

If: The keyword will be executed if the condition below is met, if the condition is not met, the Else action will be executed.

If Not: The keyword will be executed if the condition below is not met, if the condition is met, the Else action will be executed.

Yes: The Keyword will be executed without any conditions.

No: The keyword will not be executed.

<Rule> The condition is set in this element.

3.1.6 Context

Indicates if the step is a positive or negative test scenario. The default is ‘positive’.

3.1.7 Impact

Used to decide if status of step will or will not impact the case status. Options are Impact and noimpact with [impact being the default](#).

3.1.8 Runmode

This tag can be filled up to run a case multiple times, until failure, or until pass. This tag has two attributes, 'type' and 'value'. In type, you can specify either 'RMT', 'RUF', or 'RUP', and in 'value', you can specify the maximum number of times you want the step to run. This is an [optional](#) tag.

7.0 UNDERSTANDING THE SUITE

A suite is a collection of case with execution instructions. A Warrior Framework Suite is an XML file written using specific tags that have been pre-defined by Warrior Framework. To correctly write a Warrior Framework Suite, understanding what each and every tag means and the correct structure of the suite is very important. A suite can be created using Katana. Please refer to [Katana User Guide: Section 4](#) a complete guide to creating a suite using Katana.

A typical suite looks something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<TestSuite>
    <Details>
        <Name>Sample_Suite</Name>
        <Title>Sample_Suite</Title>
        <Engineer>Jo Smith</Engineer>
        <Date>05/17/2016</Date>
        <Time>13:33:58</Time>
        <type exectype="sequential_testcases"/>
        <State>New</State>
        <default_onError action="next"/>
        <Resultsdir/>
        <IDF>../Data/Sample_Input_Data_File.xml</IDF>
    </Details>
    <Requirements>
        <Requirement/>
    </Requirements>
    <Testcases>
        <Testcase>
            <path>../Testcases/Sample_Testacase.xml</path>
            <context>positive</context>
            <runtype>sequential_keywords</runtype>
            <onError action="goto" value="2"/>
            <runmode type="rmt" value="2"/>
            <impact>impact</impact>
        </Testcase>
        <Testcase>
            <path>../Testcases/demo_tc2.xml</path>
            <context>positive</context>
            <runtype>sequential_keywords</runtype>
            <onError action="next" value="" />
            <runmode type="ruf" value="5"/>
            <impact>impact</impact>
        </Testcase>
    </Testcases>
</TestSuite>
```

Tags in a suite:

<Suite></Suite> forms the root tag of the suite. This root tag helps Warrior Framework identify this XML file as a Suite. This tag has three child tags – Details, Cases, and Requirements.

1.0 Details

The <Details></Details> signifies that everything included inside this tag gives Warrior Framework information about the Suite. This tag has thirteen child tags – Name, Title, Engineer, Date, Time, Type, State, InputDataFile, Datatype, default_OnError, Logsdir, Resultsdir, and ExpectedResults.

1.1 Name

The tag holds the Suite name. It must be unique. The name of the file and the text of this tag should match. This is a **mandatory** field.

1.2 Title

The title lets you add a title or a description for this case so that you can refer to it later on, to understand the purpose of this case. This is a **mandatory** field.

1.3 Engineer

This tag stores the Case developer's name. This is a **mandatory** field.

1.4 Date

The Date tag stores the date of creation of this Suite. This is an **optional** field, only defined just so that you can date stamp a Suite.

1.5 Time

The Time tag stores the time of creation of this Suite. This is an **optional** field, only defined just so that you can time stamp a Suite.

1.6 Type

The execType attribute is used to determine how the cases within the suite should be executed. The options are 'Parallel, Sequential, Run_Multiple_Time, Run_Until_Fail, Run_Until_Pass.' The

Max_Attempts attribute is used with Run_Multiple, Run_Until_Fail, and Run_Until_Pass to determine how many times to run the suite or what is the max attempts value if a case does not fail. The [default execute type is sequential](#).

The Parallel option indicates that all the cases within the suite are to be run in parallel, all at the same time.

The Sequential option indicates that all the cases within the suite are to be run in sequence, one after the other.

The Run Multiple option indicates that the sequence of cases (i.e. suite) will be executed as many times as indicated in the max attempts attribute.

The Run until Fail option indicates that the sequence of cases (i.e. suite) will be executed until the suite Fails or the max attempts have been reached.

The Run until Pass option indicates that the sequence of cases (i.e. suite) will be executed until the suite Pass or the max attempts have been reached.

1.7 State

This tag contains the state of the Suite, i.e., whether the Suite is a new, released, test-assigned, or anything else that you might consider appropriate for the Case. This is an [optional](#) field.

1.8 default_OnError

Default error handling for the suite. This is where you set what happens if a case fails. User can override the default by selecting the on error in the case section. If the case has an on error selection, that will be used instead of the default.

1.8.1 Action attribute

This is an attribute of the default_OnError. This is an optional attribute. If not set, Warrior Framework will [default to the “next”](#), that is, it would proceed to the next case. You will be able to set different onError setting per case if needed. That will be done at the case level. The options for this attribute are next, abort, and goto.

1.8.2 Value attribute

This is an attribute of the default_OnError tag but it is necessary only if the action attribute is set to "goto".

This attribute accepts value as a case number, that is, the number of the case that you want to jump to in case the current case throws an Error. Warrior Framework will then directly go to executing the case corresponding to the number given if the current case throws an Error.

1.9 Resultsdir

Location of directory where result files will be stored here. If field is left blank Warrior Framework will use what is in the case, if that is left blank then, Warrior Framework will use the location defined in the w_setting.py file, if that is also left blank, then Warrior Framework will use the default under the Execution directory. This is, therefore, an optional field.

1.10 IDF

Path to Data file associated with the suite. If a data file is listed, that data file will overwrite the data file used in the cases in the suite. If a data file is not listed, the data files within the cases will be used. This is therefore, an optional field.

2.0 Requirements

User can add as many requirements as needed.

2.1 Requirement

Every requirement needed for this case must be added inside a <Requirement></Requirement>. This is an optional field.

3.0 Cases

User can add as many cases as needed but at least one case is required.

3.1 Path

This tag is for the location of the case. You can enter a relative path. This is a mandatory field.

3.2 Context

The context of each case can be either positive or negative. This determines if the case is expected to Pass or Fail. The default is 'positive'.

3.3 runtype

This runtype tag determines if the sequential_keywords, or parallel_keywords. Default is sequential keywords. If sequential_keywords is selected, the keywords (or steps) within the case will all run in sequence. If the parallel_keywords is selected, then the keywords (or steps) within the case will all run in parallel.

3.4 onError

Set what happens if the case fails. This will override the default of the suite. Set what happens if the step fails. This will override the default of the suite. Options are Next, Goto, Abort and Abort_as_error. The behavior is as follows:

- Selecting Next will result in the next case being executed if the case fails.
- Selecting Goto and the case number to go to will result in execution of the case indicated. The execution of cases will continue from that case going forward.
- Selection Abort will terminate the execution of the suite if the case does not pass. The case status would be set to FAIL
- Selection Abort_as_error will terminate the execution of the suite if the case does not pass. The case status would be set to ERROR.
- This is an optional field.

3.5 Runmode

This tag has two attributes - "type" and "value". The value provided to the type attribute indicates whether this particular case should run multiple times (RMT), run until failure (RUF), or, run until pass (RUP). The attribute value takes in the number of attempts that this case should go through before

completing the execution of this case – in case of RUP and RUF, the value given in this attribute would be treated as the maximum number of attempts that Warrior Framework would execute this case till it passes/fails, while for RMT, Warrior Framework would run the case for as many times as indicated. This is an [optional](#) field.

3.6 impact

Used to decide if status of case will or will not impact the suite status. Options are Impact and noimpact with [impact being the default](#).

8.0 UNDERSTANDING THE PROJECT

A Project is a collection of suites. A Warrior Framework Project is an XML file written using specific tags that have been pre-defined by Warrior Framework Framework. To correctly write a Warrior Framework Project, understanding what each and every tag means and the correct structure of the project is very important. A project can be created using Katana. Please refer to [Katana User Guide: Section 5](#) a complete guide to creating a project using Katana.

A typical Project looks something like this:

```
▼<Project>
  ▼<Details>
    <Name>Sample_Project</Name>
    <Title>Sample_Project</Title>
    <Engineer>Jo Smith</Engineer>
    <State>New</State>
    <Date>05/17/2016</Date>
    <Time>14:19:30</Time>
    <default_onError action="next"/>
  </Details>
  ▼<Testsuites>
    ▼<Testsuite>
      <path>../Suites/Sample_Suite.xml</path>
      <onError action="next" value="">
        <impact>impact</impact>
      </onError>
    </Testsuite>
    ▼<Testsuite>
      <path>../Suites/Suite_Template.xml</path>
      <onError action="next" value="">
        <impact>impact</impact>
      </onError>
    </Testsuite>
    ▼<Testsuite>
      <path>../Suites/demo_suite.xml</path>
      <onError action="next" value="">
        <impact>impact</impact>
      </onError>
    </Testsuite>
  </Testsuites>
</Project>
```

A project is a collection of suites with execution instructions. A project template and a sample project are located in Warrior Frameworkspace/projects directory. The root node is `<Project></Project>`. This root node has two children – the `<Details></Details>` tag, and the `<Suite></Suite>` tag

1.0 Details

This tag contains all the details about the Project. The following are its child tags:

1.1 Name

This tag contains the project name. It must be unique. This name should match the file name. This is a **mandatory** tag.

1.2 Title

This is the project title or description. You can describe what this project does and what all kind of suites it contains in this tag. This is also a [mandatory](#) tag.

1.3 [The Engineer](#)

This tag contains the name of the Project designer. This is a [mandatory](#) tag.

1.4 [State](#)

This tag contains the state of the Project, i.e., whether the Project is a new, released, test-assigned, or anything else that you might consider appropriate for the Case. This is an [optional](#) field.

1.5 [Date](#)

The Date tag stores the date of creation of this Project. This is an [optional](#) field, only defined just so that you can date stamp a Project.

1.6 [The Time](#)

The Time tag stores the time of creation of this Project. This is an [optional](#) field, only defined just so that you can time stamp a Project.

1.7 [default_OnError](#)

This is the default behavior if any of the suite fails. User can override the default by selecting the on error in the suite section. If the suite has an on error selection, that will be used instead of the default.

1.7.1 Actions attribute

This is an attribute of the default_OnError. This is an [optional](#) attribute. If not set, Warrior Framework will [default to the “next”](#), that is, it would proceed to the next suite. You will be able to set different onError setting per suite if needed. That will be done at the suite level. The options for this attribute are next, abort, and goto.

1.7.2 Value attribute

This is an attribute of the default_OnError tag but it is [necessary only if the action attribute is set to “goto”](#). This attribute accepts value as a suite number, that is, the number of the suite that you want to jump to

in case the current suite throws an Error. Warrior Framework will then directly go to executing the suite corresponding to the number given if the current suite throws an Error.

2.0 Suite

You can add as many suites you want in a project, but at least one suite is required.

2.1 Path

This tag contains information about the location of the suite. You can enter a relative path. This is a mandatory field.

2.2 onError

Set what happens if the suite fails. This will override the default of the project. Options are Next, Goto, Abort and Abort_as_error. The behavior is as follows:

- Selecting Next will result in the next suite being executed if the suite fails.
- Selecting Goto and the suite number to go to will result in execution of the suite indicated. The execution of suites will continue from that suite going forward.
- Selection Abort will terminate the execution of the suite if the suite does not pass. The suite status would be set to FAIL
- Selection Abort_as_error will terminate the execution of the suite if the step does not pass. The suite status would be set to ERROR.
- This is an optional field.

2.3 Impact

This tag will determine if the failure of this suite will impact the overall pass/fail status of the project. Setting it to impact will cause the project to fail if the suite fails. Setting it to no impact will not cause the project to fail if the suite fails. The default is 'impact'.

9.0 UNDERSTANDING THE INPUT DATA FILE (IDF)

A typical datafile looks something like this:

```

▼<credentials>
  ▼<system name="http_bin_1">
    <url>http://httpbin.org/post</url>
    <cookies>{"cookie_name": "this_cookie"}</cookies>
  </system>
  ▼<system name="http_bin">
    ▼<subsystem name="bin_1">
      <url>http://httpbin.org/put</url>
      <user>Jo</user>
      <password>536ett</password>
      <content_type>json</content_type>
      <expected_response>200</expected_response>
    </subsystem>
    ▼<subsystem name="bin_2">
      <url>http://httpbin.org/delete</url>
      <content_type>json</content_type>
      <expected_response>500</expected_response>
    </subsystem>
    ▼<subsystem name="bin_3">
      <url>http://httpbin.org/patch</url>
      <expected_response>200</expected_response>
      <user>Jo</user>
      <password>536ett</password>
    </subsystem>
  </system>
  ▼<system name="http_bin_3">
    <url>http://httpbin.org/get</url>
    <user>Jo</user>
    <password>536ett</password>
    <content_type>json</content_type>
    <expected_response>200</expected_response>
    <params>{'key1': 'value1', 'key2': ['value2', 'value3']}</params>
  </system>
</credentials>
```

This is a system based format that is recommended by Warrior Framework. This format is required for most of the built-in keywords and for the iterative and hybrid mode.

Here, the root of the data file is `<credentials></credentials>`. The system-based format requires all the child nodes of the `<credentials></credentials>` to be `<system></system>`. Now, all the information that is required can be filled inside the `<system></system>`.

A `<system></system>` can have multiple subsystems inside it. These subsystems should be added in as `<subsystem></subsystem>` child tags under the `<system></system>` tags and the corresponding information required should be added as child tags to the `<subsystem></subsystem>` tags. This can also be added as attribute name and value in the `<subsystem></subsystem>` tags.

This data file is provided as the input datafile in Warrior Framework Case.

This data files supports the following

- System with data for the system
- A system with subsystem with data only for the subsystems.

The case of system with subsystem with data for both system and subsystem is NOT supported. In such a case user should create the data for the system as a separate subsystem within the system. Substituting values from environment variables:

Substituting values from environment variables is supported in input datafile

To reference the environment variables use the pattern `${ENV.variable_name}` where `variable_name` is the name of the variable in the environment settings of the Operating System

E.g. `${ENV.IPADDR}` will be replaced with the value of `IPADDR` variable in the environment settings

To create a datafile in Katana, you can refer to [Katana User Guide: Section 6](#) of this document.

10.0 THE DATA REPOSITORY

Warrior Framework provides a data repository within a case, suite, and project. The data repository within a case contains all the data returned by a keyword stored as dictionary entry. The data returned by the keywords is stored automatically in the data repository and is accessible to the other keywords within the case.

Warrior Framework provides a data repository within a suite as well. The data repository within a suite contains all the case statuses. The status of each executed case in that suite is available for every other case to access.

Similarly, Warrior Framework provides a data repository within a project. The data repository within a project contains all the suite statuses. The status of each executed suite in that project is available for every other suite to access.

Warrior Framework Framework expects all data returned by a keyword should be returned in a python dictionary. This returned dictionary will be stored in the case, suite, or project data repository as a key-value pair. Keywords will have to access the case data repository to get the data returned from a previous keyword, cases in a suite will have access to case statuses returned from the previous cases, and suites in a project will have access to suite statuses returned from the previous suites.

If a previous keyword has returned `api_resp (string)`, `session_id (object)` to the case data repository. In order to use these values in the next keyword you would have to access that in the keyword using this function:

```
Utils.data_Utils.get_object_from_datarepository
```

And use it in the keyword as:

```
api_resp = Utils.data_Utils.get_object_from_datarepository('api_resp')
session_id = Utils.data_Utils.get_object_from_datarepository('session_id')
```

11.0 CASE DESIGN FEATURES

11.1 RUNNING KEYWORDS IN SEQUENCE

Warrior Framework keywords defined within a case will run sequentially by default. When executing a case in Warrior Framework, the keywords will execute sequentially.

11.2 RUNNING KEYWORDS IN PARALLEL

Warrior Framework keywords defined within a case can be executed in parallel. Follow the steps below to run the keywords in parallel:

1. Create a case to contain the keywords that need to be run in parallel.
2. Create a suite or modify an existing one.
3. In the case block of the Suite, add the case created in step 1
4. Set the runtype tag to: <runtype>parallel_keywords</runtype>

11.3 DESIGNING A NEGATIVE KEYWORD

Negative keywords are keywords that are expected to fail. If a negative keyword fails, the keyword will be marked as a Pass in the case results. All keywords in Warrior Framework are positive keywords by default unless marked as negative. Follow the steps below to mark a keyword as a negative one:

1. Create a case to contain the keywords to be executed.
2. After selecting the driver and keyword, set the context tag to Negative.
<context>negative</context>
3. If the context tag is not present, Warrior Framework will default keyword's context to be positive.

11.4 KEYWORDS DESCRIPTION

While creating a keyword, a user has the option of setting a WDesc variable within the keyword that would be set to a string describing the keyword behavior. The string assigned to WDesc will be read by Warrior Framework at execution and displayed with the keyword name in the case xml result file. Katana, Warrior Framework's Web based case creation tool will also read the WDesc string and display it in the step description field.

11.5 CONDITIONAL KEYWORDS EXECUTION

Warrior Framework allows the user to set conditions for each keyword's execution. The user will be able to set the execution condition to always run, to never run, to run if a condition is met, or to run if a condition is not met. Follow the procedure below to set the execution conditions:

1. Create a case to contain the keywords to be executed.
2. After selecting the driver and keyword, set the execute tag for each keyword
 - a. <Execute ExecType='If, If Not, Yes, No' > <Rule Condition= 'variable' Condvalue='1' Else='next, abort, goto' Elsevalue='2' /> </Execute>
 - b. Execute type : Yes = always run , No= to never run, If = run if a condition is met, If Not = to run if a condition is not met
 - c. Rule Condition: The variable being evaluated for the condition.
 - d. Condvalue: The value expected for the condition to be met.
 - e. Else: Action to be taken if condition is not met
 - f. Elsevalue: Step number in case Else was goto.

11.6 DESIGNING A NEGATIVE CASE

Negative cases are cases that are expected to fail. If a negative case fails, the cases will be marked as a Pass in results. All cases in Warrior Framework are positive cases by default unless marked as negative. Follow the steps below to mark a case as a negative one:

1. Create a suite to contain the cases to be executed.
2. After selecting the case path, set the context tag to Negative. <context>negative</context>
3. If the context tag is not present, Warrior Framework will default case context to be positive.

11.7 RUNNING A CASE IN SEQUENCE

Warrior Framework cases defined within a suite can be executed in parallel. Follow the steps below to run the case in parallel:

1. Create the case that you would like to run in sequence.

2. Create a suite or modify an existing one.
3. Set the type tag in the suite : <type exectype ='Sequential' />
4. In the case block of the Suite, add the case created in step 1
5. Execute the suite. The cases will run in sequence.

11.8 RUNNING A CASE IN PARALLEL

Warrior Framework cases defined within a suite can be executed in parallel. Follow the steps below to run the case in parallel:

1. Create the case that you would like to run in parallel.
2. Create a suite or modify an existing one.
3. Set the type tag in the suite : <type exectype ='Parallel' />
4. In the case block of the Suite, add the case created in step 1
5. Execute the suite. The cases will run in parallel.

11.9 RUNNING A CASE MULTIPLE TIMES

Warrior Framework cases defined within a suite can be executed Multiple Time. Follow the steps below to run the case/s multiple times:

1. Create a suite or modify an existing one.
2. Set the type tag in the suite : <type exectype = 'Run_Multiple', Max_Attempts='5' />
3. Execute the suite. The cases in the suite will run multiple times as indicated in the Max_Attempts attribute.

11.10 RUNNING A CASE UNTIL FAILURE/PASS

Warrior Framework cases defined within a suite can be executed until failure. Follow the steps below to run the case/s multiple times until failure:

1. Create a suite or modify an existing one.

2. Set the type tag in the suite : `<type exectype = 'Run_Until_Failure, Max_Attempts='5' />`. If you want to run the suite until it passes, set the type tag in the suite :`<type exectype = 'Run_Until_Pass, Max_Attempts='5' />`
3. Execute the suite. The cases in the suite will run until the suite fails or the suite is executed as indicated in the Max_Attempts attribute.

11.11 ERROR HANDLING IN WARRIOR FRAMEWORK

Warrior Framework allows error handling to be handled at the keyword, case, suite, and project level. The error handling can be set to instruct Warrior Framework what to do in case of a failure. The options are to proceed to the next step, case or suite, abort execution or go to another case, step or suite. If the error handling instruction is not present, Warrior Framework will default to proceed to next step/case/suite as applicable.

1. Case Error Handling: User can set a default error handling instruction for the entire case using the `default_OnError` tag. `<default_OnError action = 'goto' value='9' />`. If this tag is not present, Warrior Framework will default to proceed to next step.
2. Step Error Handling: When an error is encountered during execution of a keyword, the keyword will fail. User has the option of indicating what to do in case of a keyword failure on a step by step basis. Each step can have its own error handling. Use the following tag to set the step error handling :`<onError Action='next, abort, goto' Value='9' />`
 - a. The step error handling will supersede the case default error handling.
 - b. If step error handling is not defined, the default error handling will be used. If the default error handling is not set as well, Warrior Framework will default to go to next step in case of a step failure.
3. Suite Error Handling: User can set a default error handling instruction for the entire suite using the `default_OnError` tag. `<default_OnError action = 'goto' value='9' />`. If this tag is not present, Warrior Framework will default to proceed to next case.

- a. Within the suite, a user has the option of indicating what to do in case of a case failure on a case by case basis. Each case can have its own error handling. Use the following tag to set the case error handling :`<onError Action='next, abort, abort_as_error, goto' Value='9' />`
 - b. The case error handling will supersede the suite default error handling.
 - c. If case error handling is not defined, the default error handling will be used. If the default error handling for the suite is not set as well, Warrior Framework will default to go to next case in case of a case failure.
4. Project Error Handling: User can set a default error handling instruction for the entire Project using the `default_OnError` tag. `<default_OnError action = 'goto' value='9' />`. If this tag is not present, Warrior Framework will default to proceed to next suite.
 - a. Within the project, a user has the option of indicating what to do in case of a suite failure on a suite by suite basis. Each suite can have its own error handling. Use the following tag to set the suite error handling :`<onError Action='next, abort, abort_as_error, goto' Value='9' />`
 - b. The suite error handling will supersede the project default error handling.
 - c. If suite error handling is not defined, the default error handling for the project will be used. If the default error handling is not set as well, Warrior Framework will default to go to next suite in case of a suite failure.

12.0 HOW TO CREATE KEYWORDS

Keywords in Warrior Framework are created in Action files that are imported by the product drivers. If you are creating a new keyword file, do the following:

- Create a directory under the Actions directory. Use the following naming convention:
NameofyourchoiceActions
- Inside the directory newly created, create your python file following the naming convention:
nameofyourchoice_Actions.py

Warrior Framework uses standard Python programming for its keyword development, however Warrior Framework requires users to adhere to the following rules:

1. The keywords can be methods of a class or they can be an independent function in a library
2. When using classes for keywords:
 - Users should always use the default `__init__` for a class as shown below, and can extend it to add more attributes within `__init__`
 - Multiple classes are allowed in a single file
 - Class inheritance is not supported for keyword classes
 - In Warrior Framework, method names are keywords and hence the method names should be unique within the driver's packages.
 - Users are allowed to use a mix of classes and independent functions in a single actions file as long as their names are unique.
 - Static methods with decorators are not currently supported by the Warrior Framework framework for the Action classes.
 - Each keyword (along with its driver) is a step in the Warrior Framework case. A keyword can have multiple sub-steps that are implemented in the code.

Sample Keyword Class:

```
""" This is the actions file, keywords are programmed here """
""" Import any python library you want to use below this line """

import time

""" Import Warrior Framework Framework Utilities below this line"""
""" As a recommendation always use the below three lines in every actions
file"""

import Framework
import Framework.Utils as Utils
from Framework.Utils.print_Utils import print_info, print_debug,
print_warning, print_error

""" WARRIOR FRAMEWORK KEYWORD TEMPLATE AND RULES """

""" Warrior Framework uses standard Python programming for its keyword
development, however Warrior Framework requires the users to adhere to the
following rules:

Supported Keyword types

The keywords can be methods of a class or they can be a independent function
in a library

When using classes for keywords:
Users should always use the default __init__ for a class as shown below,
and can extend it to add more attributes within __init__
Multiple classes are allowed in a single file
Class inheritance is not supported for keyword classes
In Warrior Framework, method names are keywords and hence the method names
should be unique within the driver's packages.
Users are allowed to use a mix of classes and independent functions in a
single actions file as long as their names are unique.
Static methods with decorators are not currently supported by the Warrior
Framework framework for the Action classes.

Each keyword (along with its driver) is a step in the Warrior Framework
case. A keyword can have multiple sub-steps that are implemented in the code
"""

class CliActions(object):

    """
    Default    init    field must be used when using classes for keywords.
    """
    def __init__(self):
        self.resultfile = Utils.config_Utils.resultfile
        self.datafile = Utils.config_Utils.datafile
```

WARRIOR FRAMEWORK

REVISION: 5

```
self.logsdir = Utils.config_Utils.logsdir
self.filename = Utils.config_Utils.filename
self.logfile = Utils.config_Utils.logfile

""" Sample method (keyword) with a single substep and some of the most
commonly used functions from the data_Utils is given below.

*** Please remember this is only a sample keyword to help a beginner in
Warrior Framework Framework, and may not work exactly ***

"""

def connect_ssh(self, system_name, session_name=None,
expected prompt='.*'):

"""
KEYWORD DOCUMENTATION: A recommended style for keyword documentation
is given below

Connects to the ssh port of the the given system and creates a
pexpect session object for the system

:Arguments:
1. system_name(string) = Name of the system from the input
   datafile
2. session_name(string) = name of the session to the system
3. expected_prompt(string) = prompt expected in the terminal

>Returns:
1. status(bool)= True / False
2. session_id (dict element)= key, value

key = if session_name is not none = system_name_session_name,
if session_name is none = system_name

value = a pexpect session object for the system.

"""

"""

""" THE DESCRIPTION (optional):

All keywords can have a description variable that gives details of
what is being performed by the keyword
It should be a string enclosed within double or single quotes, no
variable substitutions to be used here as it is used as just a text
for reporting purpose. This string will be used in result file for
reporting a step. If this variable is missing Warrior Framework will
just
report the keyword name in step description of the result file.

"""

wdesc = "Connect to the ssh port of the system and creates a session
and return it if successful"

"""


```

WARRIOR FRAMEWORK

REVISION: 5

```
PSUBSTEP (optional) :
```

```
All keywords can start with the below function-call to pSubStep with  
a single argument describing in short the sub-step's functionality  
If this function-call is missing Warrior Framework will not report  
any substep
```

```
    in the case result file"""
```

```
    Utils.case_Utils.pSubStep('Connect to ssh of  
                            {0}'.format(system_name))
```

```
"""
```

SOME USEFUL Information:

Warrior Framework provides a data repository within a case and a suite. The data repository within a case contains all the data returned by a keyword stored as dictionary entry. The data returned by the keywords is stored automatically in the data repository and is accessible to the other keywords within the case. The process by which keywords can access the data returned by other keywords is described in item #4 below.

Warrior Framework also provides a data repository within a suite. The data repository within a suite contains all the data returned by the keywords of the cases within the suite stored as dictionary entry. The data returned by the keywords is stored automatically in the data repository and is accessible to the other keywords within the suite. The process by which keywords can access the data returned by other keywords is described in item #4 below.

1. Custom Logfile for a Keyword:

```
-----  
Warrior Framework framework creates a logfile by default with the name of  
the case located in Warrior Frameworkspace\Execution\DateTime\logs.
```

If the keyword is written as class method then it is available to the keyword as self.logfile.

If the keyword is written as an independent function then it is available for the user in the case data repository.

However user may want to create a separate logfile for each keyword, to create a custom logfile for a keyword use the below function from file_Utils

```
logfile = Utils.file_Utils.getCustomLogFile(self.filename,  
                                         self.logsdir,'NE_TL1_{0}'.format(system_name))
```

2. Adding Notes to case result file:

```
-----  
Use case_Utils.pNote to add notes to the result xml file refer to  
case_Utils.py for more functions related to case result file reporting
```

```
Utils.case_Utils.pNote("Logfile= %s" % logfile)
```

3. Get details from input datafile:

 Warrior Framework supports the use of a system input datafile. A sample system data file is located in Warrior Frameworkspace\Data. The get_credentials method from data_Utils can be used to retrieve data from an xml tag in the system file.

For eg: In-order to get the ip, ssh_port, username, password, prompt of a system called 'dpoe03' from the input datafile.

```
credentials = Utils.data_Utils.get_credentials(self.datafile, 'dpoe03',
['ip', 'ssh_port', 'username', 'password', 'prompt'])
```

4. Sharing data between keywords:

 In Warrior Framework Framework all data to be returned by a keyword should be returned in a python dictionary, and this data will be stored in a the case or suite data repository as a key-value pair. Keywords will have to access the case data repository to get the data returned from a previous keyword.

For eg:

Let's say a previous keyword has returned ne_location(string), session_id(pexpect object) to the case data repository,order to use these values in the next keyword use 'Utils.data_Utils.get_object_from_datarepository' and pass the key to it.

```
ne_location = Utils.data_Utils.get_object_from_datarepository('ne_location')
session_id = Utils.data_Utils.get_object_from_datarepository('session_id')
```

Sample code of the keyword

"""

```
wdesc = "Connect to the ssh port of the system and creates a session and return it if successful"

Utils.case_Utils.pSubStep('Connect to ssh of
{0}'.format(system_name))

output_dict = {}
logfile = Utils.file_Utils.getCustomLogFile(self.filename,
self.logsdir, 'NE_TL1_{0}'.format(system_name))
Utils.case_Utils.pNote(system_name)
Utils.case_Utils.pNote(logfile)
Utils.case_Utils.pNote(Utils.file_Utils.getDateTime())

if session_name is None:
    session_id = system_name
elif session_name is not None:
    session_id = system_name + session_name

credentials = Utils.data_Utils.get_credentials(self.datafile,
system_name, ['ip', 'ssh_port', 'username',
'password', 'prompt'])
```

```
ne_location =
    Utils.data_Utils.get_object_from_datarepository('ne_location')

    if credentials is False:
        status = credentials
    else:
        if credentials['prompt'] is not None:
            expected_prompt = credentials['prompt']
            session_object =
                Utils.cli_Utils.connectSSH(credentials['ip'],
                credentials['ssh_port'], credentials['username'],
                credentials['password'], logfile,
                prompt_expected=expected_prompt)

            if type(session_object) is pexpect.spawn:
                output_dict[session_id] = session_object
                status = True
            else:
                status = False

    """
    Reporting status of a substep:
    -----
    After executing a substep in a keyword, use
    Utils.case_Utils.report_substep_status(status) to report the
    status of that substep to the result file.

    If a keyword has multiple sub-steps use this for each substep in the
    keyword.

    """
    Utils.case_Utils.report_substep_status(status)

    """
    Returning Keyword status and data:
    -----
    Each keyword should return the status of the keyword in a return
    command in the keyword function.

    Any Data generated in the keyword can also be returned as a key
    value pair in a python dictionary.

    This key-value pair will be stored in the data repository for use
    by other keywords.

    """
    return status, output_dict

"""
RETURN TYPES

1. Supported return types from keywords.
   a. Only status: True or False only
   b. Only dictionary.
```

WARRIOR FRAMEWORK

REVISION: 5

- c. status, dictionary
- d. dictionary, status
- 2. If a keyword does not return a status, it will be declared as failure in the case results.
- 3. When a keyword returns a dictionary the case-datarepository will be updated with the dictionary returned by the keyword.
- 4. If a keyword returns any unsupported value the keyword will be Declared as failure.

```
*** Sample Return statement ***
"""
```

```
return status, output_dict
```

13.0 HOW TO CREATE A CASE

A Warrior Framework Case is an XML file written using specific tags that have been pre-defined by Warrior Framework Framework. To correctly write a Warrior Framework Case, understanding what each tag means and the correct structure of the case is very important. To understand the significance each tag, please refer to [section 6.0](#) of this User Guide.

You can use the Warrior Framework Tool – Katana to create a new case or, you can create a case directly. To create a create case without Katana, you should:

1. Open an xml editor – like Notepad++, Gedit, or any other that is installed on your system
2. Start creating tags – for reference, you can refer to the final figure in [Katana User Guide: Section 3](#) which shows a complete case, or you can follow the Katana steps ([Katana User Guide: Section 3](#)) and go on creating the tags and adding their respective values in your editor. Be sure that you close every tag that you open and that every attribute value is enclosed in double quotation marks. You can refer to -

<http://www.xmlobjective.com/category/guide/>

to better understand how to create a valid xml file.

3. Once you are satisfied with the Case that you have created, it is recommended that you run this file through Ironclaw ([Section 3.3](#)) so that you are sure that this is a valid Case.

14.0 HOW TO CREATE A SUITE

A Warrior Framework Suite is an XML file written using specific tags that have been pre-defined by Warrior Framework Framework. To correctly write a Warrior Framework Suite, understanding what each tag means and the correct structure of the Suite is very important. To understand the significance each tag, please refer to [Section 7.0](#) of this User Guide.

You can use the Warrior Framework Tool – Katana to create a new suite or, you can create a suite directly. To create a create case without Katana, you should:

1. Open an xml editor – like Notepad++, Gedit, or any other that is installed on your system
2. Start creating tags – for reference, you can refer to the final figure in [Katana User Guide: Section 4](#) which shows a complete suite, or you can follow the Katana steps ([Katana User Guide: Section 4](#)) and go on creating the tags and adding their respective values in your editor. Be sure that you close every tag that you open and that every attribute value is enclosed in double quotation marks. To better understand how to create a valid xml file, you can refer to :

<http://www.xmlobjective.com/category/guide/>

3. Once you are satisfied with the suite that you have created, it is recommended that you run this file through Ironclaw ([Section 3.3](#)) so that you are sure that this is a valid suite.

15.0 HOW TO CREATE A PROJECT

A Warrior Framework Project is an XML file written using specific tags that have been pre-defined by Warrior Framework Framework. To correctly write a Warrior Framework Project, understanding what each tag means and the correct structure of the Project is very important. To understand the significance each tag, please refer to [section 8.0](#) of this User Guide.

You can use the Warrior Framework Tool – Katana to create a new project or, you can create a project directly. To create a create case without Katana, you should:

1. Open an xml editor – like Notepad++, Gedit, or any other that is installed on your system
2. Start creating tags – for reference, you can refer to the final figure in [Katana User Guide: Section 5](#) which shows a complete project, or you can follow the Katana steps ([Katana User Guide: Section 5](#)) and go on creating the tags and adding their respective values in your editor. Be sure that you close every tag that you open and that every attribute value is enclosed in double quotation marks. You can refer to -

<http://www.xmlobjective.com/category/guide/>

to better understand how to create a valid xml file.

3. Once you are satisfied with the suite that you have created, it is recommended that you run this file through Ironclaw ([Section 3.3](#)) so that you are sure that this is a valid project.

16.0 HOW TO CREATE AN INPUT DATA FILE

A Warrior Framework Input Data File is an XML file written in the system based format that acts as a data store for the Warrior Framework Case. To correctly write a Warrior Framework Input Data File, understanding structure of the Input Data File is very important. To understand this, please refer to [section 9.0](#) of this User Guide.

You can use the Warrior Framework Tool – Katana to create a new input data file or, you can create it directly using any available XML editor. To create a create case without Katana, you should:

1. Open an xml editor – like Notepad++, Gedit, or any other that is installed on your system
2. Start creating tags – for reference, you can refer to the final figure in [Katana User Guide: Section 6](#) which shows a complete project, or you can follow the Katana steps ([Katana User Guide: Section 6](#)) and go on creating the tags and adding their respective values in your editor. Be sure that you close every tag that you open and that every attribute value is enclosed in double quotation marks. You can refer to -

<http://www.xmlobjective.com/category/guide/>

to better understand how to create a valid xml file.

17.0 EXECUTING THE CASE

17.1 THROUGH CLI

Warrior Framework cases can be executed in multiple ways. Below is a list of CLI execution options:

17.1.1 RUN A CASE

```
Warrior Framework /path/case.xml
```

17.1.2 RUN MULTIPLE CASES

```
Warrior Framework /path/case1.xml /path/case2.xml
```

17.1.3 RUN A SUITE

```
Warrior Framework /path/suitename.xml
```

17.1.4 RUN MULTIPLE SUITES

```
Warrior Framework /path/suite1.xml /path/suite2.xml
```

17.1.5 RUN A PROJECT

```
Warrior Framework /path/projectname.xml
```

17.1.6 RUN A COMBINATION OF CASE, SUITE, AND PROJECT

```
Warrior Framework /path/case.xml /path/suite1.xml /path/projectname.xml
```

17.1.7 SCHEDULE EXECUTION

```
Warrior Framework -scheduletime datetime /path/to/case1.xml  
/path/to/case2.xml /path/to/case3.xml
```

17.1.8 RUN KEYWORDS IN A CASE IN PARALLEL, CASES IN SEQUENCE

```
Warrior Framework -kwparallel -tcsequential /path/to/case1.xml  
/path/to/case2.xml /path/to/case3.xml
```

17.1.9 RUN KEYWORDS IN A CASE IN SEQUENCE, CASES IN PARALLEL

```
Warrior Framework -tcpparallel /path/to/case1.xml /path/to/case2.xml  
/path/to/case3.xml
```

17.1.10 RUN MULTIPLE TIMES

```
Warrior Framework -RMT <numeric value> /path/to/case1.xml /path/to/case2.xml  
/path/to/case3.xml
```

17.1.11 RUN UNTIL FAILURE

```
Warrior Framework -RUF <numeric value> /path/to/case1.xml /path/to/case2.xml  
/path/to/case3.xml
```

17.1.12 RUN A PROJECT

```
Warrior Framework -RUP <numeric value> /path/to/case1.xml /path/to/case2.xml  
/path/to/case3.xml
```

17.1.13 EXECUTION BASED ON CATEGORY

Warrior Framework CLI supports execution of cases by category. The category field within the cases will be read to determine if a case is a match. By providing a category and a test directory, Warrior Framework will search the cases in the directory for cases that match the give category and they allow the user to execute the cases.

Arguments:

- runcat (mandatory): provide a list of case categories to be searched for and executed.
- tcdir (optional): list of directories to search for case xml files, default=cwd.
- suitename (optional): name of suite xml file to be created.
- suitelocn (optional): path to location where the suite xml file will be created, default=cwd

Format:

```
./Warrior Framework -runcat cat1 cat2 cat3 -tcdir path/to/dir1 path/to/dir2  
path/to/dir3           -suitename          suite_file_name           -suitelocn  
location_to_create_suite_xml_file
```

Commands:

Search the cwd for cases that matches at least one of the provided categories and executes them as individual cases. Note: The cases will not be grouped into a suite using this command.

```
./Warrior Framework -runcat cat1 cat2 cat3
```

Search the provided directories path/to/dir1 path/to/dir2 path/to/dir3 for cases that matches atleast one of the provided categories cat1 cat2 cat3, and executes them as individual cases.

Note: No suite created automatically in this case.

```
./Warrior Framework -runcat cat1 cat2 cat3 -tcdir path/to/dir1 path/to/dir2  
path/to/dir3
```

Search the cwd for cases that matches at least one of the matching categories cat1 cat2 cat3

Creates a suite xml with the provided suite name in the cwd and executes the suite.

```
./Warrior Framework -runcat cat1 cat2 cat3 -suitename suite_file_name
```

Search the cwd for cases that matches at least one of the provided categories cat1 cat2 cat3

Creates a suite xml with the provided suite name in the provided suite locationn and executes the suite.

WARRIOR FRAMEWORK

REVISION: 5

```
./Warrior Framework -runcat cat1 cat2 cat3 -suitename suite_file_name -  
suitelocn location_to_create_test_suite_xml_file
```

Search the provided directories path/to/dir1 path/to/dir2 path/to/dir3 for cases that matches at least one of the provided categories cat1 cat2 cat3. Creates a suite xml with the provided suite name in the cwd and executes the suite.

```
./Warrior Framework -runcat cat1 cat2 cat3 -tcdir path/to/dir1 path/to/dir2  
path/to/dir3 -suitename suite_file_name
```

Search the provided directories path/to/dir1 path/to/dir2 path/to/dir3 for cases that matches at least one of the provided categories cat1 cat2 cat3. Creates a suite xml with the provided suite name in the provided suite location and executes the suite.

```
./Warrior Framework -runcat cat1 cat2 cat3 -tcdir path/to/dir1 path/to/dir2  
path/to/dir3 -suitename suite_file_name -suitelocn  
location_to_create_suite_xml_file
```

17.1.14 JIRA BUG REPORTING

For JIRA bug reporting, configure Warrior Framework/ Tools/Jira/jira_config.xml file

Default JIRA project definition:

Warrior Framework will use the first project it finds in the JIRA config file marked default="true". Do not have multiple default projects. 2. If no projects are marked as default="true", then it is the first project in the jira_config.xml

Commands:

Automatically creates JIRA defects against the default JIRA project from JIRA config file.

```
./Warrior Framework -ad path/to/tc1.xml path/to/tc2.xml path/to/ts1.xml  
path/to/ts2.xml path/to/proj1.xml path/to/proj2.xml
```

Automatically creates JIRA defects against the provided JIRA project from JIRA config file.

```
./Warrior Framework -ad -jiraproj jiraproj name path/to/tc1.xml  
path/to/tc2.xml path/to/ts1.xml path/to/ts2.xml path/to/proj1.xml  
path/to/proj2.xml
```

Creates defects in default JIRA project for all the defects JSON files present in defects_directory1, defects_directory2, defects_directory3

```
./Warrior Framework -ujd -ddir path/to/defects_directory1
path/to/defects_directory2 path/to/defects_directory3
```

Creates defects in default JIRA project for the defects JSON files path/to/defects_json1 path/to/defects_json2 path/to/defects_json3

```
./Warrior Framework -ujd -djson path/to/defects_json1 path/to/defects_json2
path/to/defects_json3
```

Creates defects in provided JIRA project for all the defects JSON files present in defects_directory1, defects_directory2, defects_directory3

```
./Warrior Framework -ujd -jiraproj jiraproj_name -ddir
path/to/defects_directory1
path/to/defects_directory2
path/to/defects_directory3
```

Creates defects in provided JIRA project for the defects JSON files path/to/defects_json1 path/to/defects_json2 path/to/defects_json3

```
./Warrior Framework -ujd -jiraproj jiraproj_name -djson path/to/defects_json1
path/to/defects_json2 path/to/defects_json3
```

17.1.15 CREATING SUITES

Warrior Framework offers CLI support for creation of a default suite from a list of cases.

Arguments:

-cs (mandatory): used to setup suite creation from command line, takes no values

-suitename (mandatory): name of suite xml file to be created.

-suitelocn (optional): path to location where the suite xml file will be created, default=cwd

-cat (optional): to add cases by Category

-tcdir (optional): The case directory

filepaths: list of case xml files, not required when used to create suite with –category.

Format:

WARRIOR FRAMEWORK

REVISION: 5

```
./Warrior Framework -createsuite -suitename suite_file_name -suitelocn  
location_to_create_test_suite_xml_file path/to/tc1.xml path/to/tc2.xml  
path/to/tc2.xml
```

```
./Warrior Framework -createsuite -cat cat1 cat2 cat3 -suitename  
suite_file_name -suitelocn location_to_create_test_suite_xml_file -tcdir  
path/to/dir1 path/to/dir2 path/to/dir3
```

Commands:

Creates a suite with provided name and with the provided list of cases in the cwd, with default values for other suite parameters.

```
./Warrior Framework -createsuite -suitename suite_file_name path/to/tc1.xml  
path/to/tc2.xml path/to/tc2.xml
```

Creates a suite with provided name and provided list of cases in the provided suitelocn, with default values for other suite parameters.

```
./Warrior Framework -createsuite -suitename suite_file_name-suitelocn  
location_to_create_test_suite_xml_file path/to/tc1.xml path/to/tc2.xml  
path/to/tc2.xml
```

Create a suite with provided name in the cwd. Search the cwd for cases matching atleast one of the provided categories cat1 cat2 cat3.

```
./Warrior Framework -createsuite -suitename suite_file_name -cat cat1 cat2  
cat3
```

Search the cwd for cases matching at least one of the provided categories cat1 cat2 cat3.

Create a suite with provided name in the provided suitelocn.

```
./Warrior Framework -createsuite -suitename suite_file_name -suitelocn  
location_to_create_test_suite_xml_file -cat cat1 cat2 cat3
```

Searches the provided list of case directories [<path/to/dir1> <path/to/dir2> <path/to/dir3>] for cases matching at least one of the provided categories cat1 cat2 cat3. Create a suite with provided name in the provided suite location.

WARRIOR FRAMEWORK

REVISION: 5

```
./Warrior Framework -createsuite -suitename suite_file_name -suitelocation_to_create_test_suite_xml_file -cat cat1 cat2 cat3 -tcdir path/to/dir1 path/to/dir2 path/to/dir3
```

17.1.16 ENCRYPT PASSWORDS

Run this command to encrypt a password:

```
Path/to/Warrior Framework -encrypt plain_text_password
```

You can add ‘sudo’ in front of the command, if you want run it as an admin. This command will give you an encoded password.

```
The encrypted text for plain_text_password is:
```

```
25298e188e659bba82672dcb317215847c68db0f39998c7f92deeb455e86648c
```

Encrypt a password with a specific Secret Key

To encrypt a plain text password using a specific key, run the following command:

```
Path/to/Warrior Framework -encrypt plain_text_password -secretkey  
IamaNinjaWarrior Framework
```

This would encrypt plain_text_password by using “IamaNinjaWarrior Framework” as the secret key. The secret key should be 16 letters – counting special characters and spaces – in length.

17.2 THROUGH KATANA

Warrior Framework can also be executed through Katana. Please refer to [Katana User Guide: Section 4](#) for all the details.

18.0 UNDERSTANDING THE RESULTS

When a case, suite, or a project is run, it creates a result directory. Being able to understand this result directory is a big asset in debugging and understanding your keywords and/or the cases, suites, and projects. The results directory would of the same name that the case, suite, or project that was run and it would be time stamped. This directory would be created either in the path mentioned in the w_settings file - if that is left empty, then in the path mentioned in the case, suite, or project - if that is left empty, then in the default directory – the Execution folder in Warrior Framework space.

If the results directory is the first one in the destination for that particular case, suite, or project, then the time stamp would not be added. This results directory will have three subfolders underneath in – Defects, Logs, and Results. If the case (or suite, or project) passed with no defects, error, or exceptions, then the defects folder would be empty.

The Logs folder would contains logs from all the system that were used by the case and also the consoleLogs. The consoleLogs contain all the activity that went on, on the console when the case (or suite or project) was running. These consoleLogs contain all the information you need to know what happened during the execution – but, consoleLogs can be tedious to read, so Warrior Framework has the HTML result file inside the Results directory.

The Results directory has one subfolder – Keyword Results – and three sub-files inside it. Out of those three sub-files, one is an HTML file which can opened in a browser. It would look something like this:

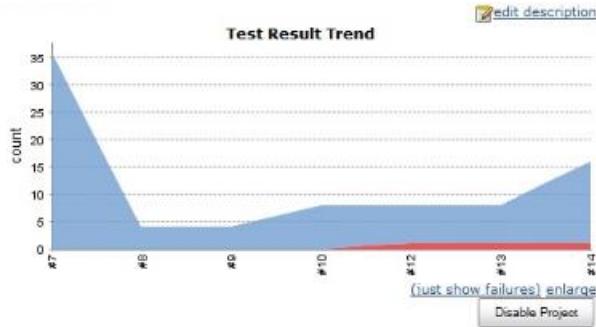
Type	Name	Info	Timestamp	Duration	Status	Results (Logs & Defects)	Summary					
							Kw Count	Kw Passed	Kw Failed	Kw Errors	Kw Exceptions	Kw Skipped
Testcase	tc_global_row_multi_subsys	cli cmd attr. defined globally	2016-05-26 12:34:36	16.0	PASS		3	3	0	0	0	0
Keyword	connect_ssh	1. system_name = rxtxalp01_multi_subsys[cli_m2] 2. prompt = labadmin@rxtxalp01: 3. int timeout = 50	2016-05-26 12:34:38	7.0	PASS							
Keyword	send_commands_bytestdata_rownum	1. system_name = rxtxalp01_multi_subsys[cli_m2] 2. row num = ls_dir3	2016-05-26 12:34:45	6.0	PASS							
Keyword	disconnect	1. system_name = rxtxalp01_multi_subsys[cli_m2]	2016-05-26 12:34:51	1.0	PASS							

This HTML result file gives a visual appeal to the result files. Since this is a screenshot of an HTML result file of a case, only the case and the keywords are showing in it, but if it were a suite or a project, all the cases inside the suite, and all the suites inside the project would be displayed.

Here you can see, that just by glancing that the result file, you can determine which keywords passed and which did not. The results are also segregate the keywords based on the final execution state of that keyword, so you know exactly how many passed, failed, threw an Error or an Exception.

Clicking on the Result Icon in the Results, Logs, and Defects column redirects you to the corresponding file containing the result xml file of that particular entity. Clicking of the Logs icon would direct you to logs, and clicking on the defects icon would direct you to the corresponding defects file.

Then, there is the JUnit result file – this file is generated for the benefit of external tools such as Jenkins which can read the JUnit files generated by case executions to display the results graphically.



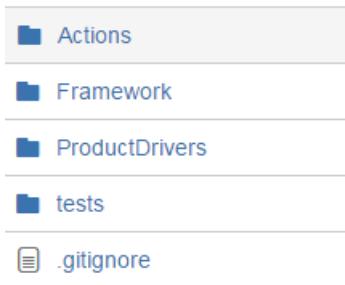
This JUnit file is in the correct JUnit format. This file provides a very high level view of the execution and its result.

The XML result file is the third file found under the Results directory. This result file is internal to Warrior Framework that has more details about the execution than the JUnit result file. This file is similar to the JUnit file but in a format that has been decided by Warrior Framework.

19.0 GUIDELINES TO CREATE A WARRIOR FRAMEWORK-KEYWORDS REPOSITORY

Any product specific keywords that are developed should have a particular file structure for Warrior Framework to work as expected. To adhere to this file structure, it is important that any Warrior Framework Keywords repository be structured correctly. The following are a few guidelines that would help you create a brand new Warrior Framework-Keywords repository:

1. The repository name must be the name of the product for which you are developing the Warrior Framework Keywords. This would be the root of the repository.
2. Inside the root of the repository, a directory structure like this must be maintained:



3. The Actions folder must contain all the keywords, the Framework folder must contain all the associated utils, and the ProductDrivers must contain all the product drivers.
4. The tests directory should be used only for uploading the unit tests related to this keyword repository.
Do NOT upload Warrior Framework tests in this directory.

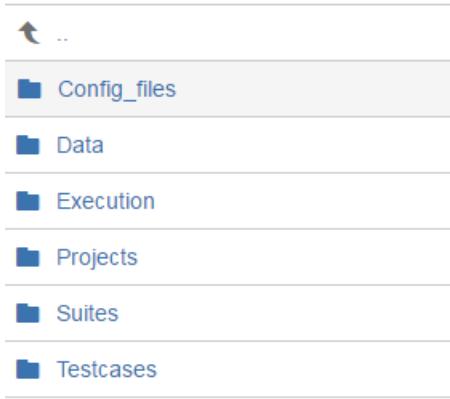
20.0 GUIDELINES TO CREATE A WARRIORSPACEREPOSITORY

Any product specific Warrior Frameworkspaces that are developed are recommended to have a file structure so that writing and managing tests is simplified. For this, it is important that the Warriorspacerepository be structured well. The following are a few guidelines that would help you create a brand new Warriorspacerepository:

1. The repository name should be the name of the product for which you are developing the Warrior Frameworkspace. This would be the root of the repository.
2. Inside the root of the repository, a directory structure like this should be maintained:



3. Inside this Warriorspace directory, this file structure is recommended:



4. The Config_files folder should contain all the Testdata files and the Variable Config files, the Data folder should contain all the Input Data files, the Projects folder should contain all the Project files, the Suites should contain all the Suite files, and the Cases should contain all the Case files.
5. The Execution directory is not necessary, and can be skipped completely while creating the repository.

KATANA
USER GUIDE

1. KATANA: AN INTRODUCTION

Katana is Warrior Framework's web based case creation and execution tool. From Katana, users can create cases, suites, projects, and input data files. Cases, suites, projects, and input data files in Katana are web based forms that once saved, create the appropriate xml files. Each field in the form has a tooltip explaining its functionality. Katana also allows you to execute cases, suites, and projects from the execute tab.

1.0 WHERE IS KATANA LOCATED?

Katana is currently located at:

<http://rtx-swtl-git.fnc.net.local/projects/WAR/repos/katana/browse>

1.1 THE PREREQUISITES

Katana requires git to be installed – this is required solely to clone Katana into your machine.

If you are working on Windows, you will need to install python27 before you can run the Katana server.

If you are working on any Linux based machines, you'll need xterm available on your machine. Xterm is required only when you are executing Warrior Framework through Katana. If you don't intend to execute Warrior Framework through Katana, but only need Katana to build your Cases, Suites, and Projects for you, then you don't need xterm installed.

1.2 HOW TO INSTALL KATANA?

Katana can be cloned into any directory as desired. Open your terminal, 'cd' into the directory of your choice and then type in this:

```
git clone http://USERNAME@rtx-swtl-git.fnc.net.local/scm/war/katana.git
```

Don't forget to replace the USERNAME with your actual username.

1.3 HOW TO RUN KATANA?

Katana is easy to run. Open your terminal and cd into the directory where you have cloned Katana. Then, type in:

```
cd katana/Katana
```

Once you are inside the Katana directory, type in:

```
python katana.py
```

This starts up Katana on the localhost at the port 5000. You can access Katana by opening any browser and typing in:

```
localhost:5000
```

If you wish to start up Katana on a different port, you can use the following command:

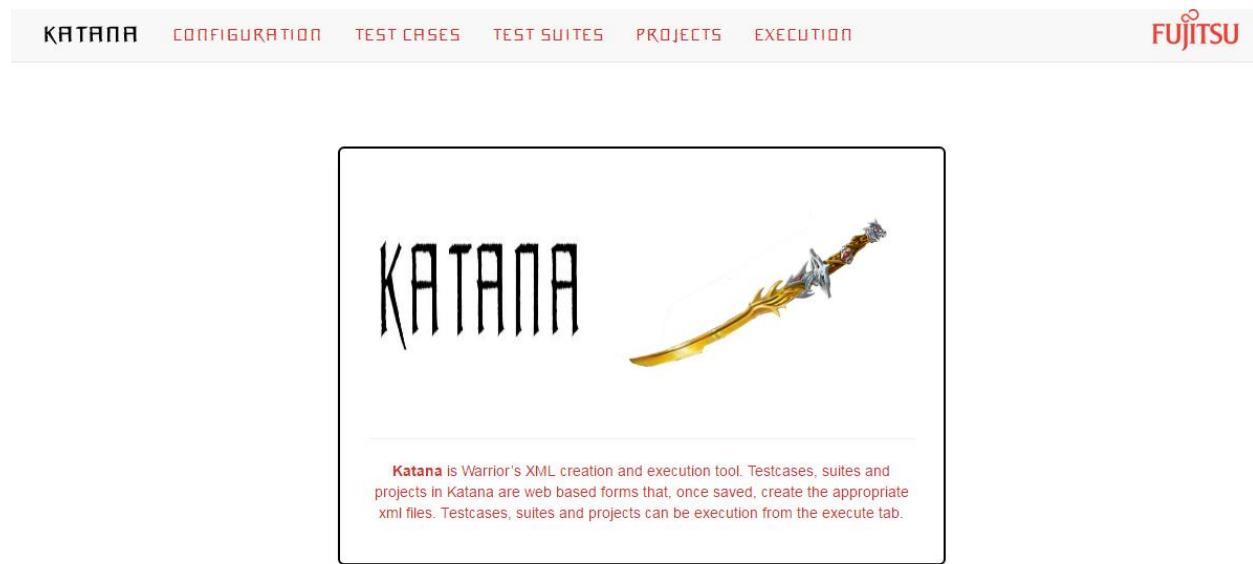
```
python katana.py -p 5001
```

This starts up Katana on the localhost at the port 5001. Now, you can access Katana by opening any browser and typing in:

```
localhost:5001
```

2. GETTING STARTED WITH KATANA

Katana starts up on the localhost and this page shows up



Katana has 7 tabs on the top in total. To configure Katana to your system, you will need to click on the second tab – the configuration tab – and this page would show up.

CONFIGURATION

On this page you may define the following Katana configuration parameters.

Name of the Engineer

All testcase created/modified in this session will be tagged with this Engineer's name.

Current location of the Warrior Framework Directory

The directory where the Python Action & Driver files will be read from.

Test Case Directory

The test case directory is where the XML files created by this application are stored.

Test Suite Directory

The test suite directory is where the XML files created by this application are stored.

Project Directory

The project directory is where the project XML files created by this application are stored.

Input Data File Directory

The input data file directory is where the data files created by this application will be stored.

Python Path

Enter the path to the Python that you want to use while executing Warrior - if left empty, the default Python will be used.

Now, the information on this page is supposed to be filled in once by the User when Katana is fired up for the first time. Each field is explained below:

Name of the Engineer

This field should contain your name.

Name of the Engineer All testcase created/modified in this session will be tagged with this Engineer's name.
Jo Smith

Current Location of the Warrior Framework Framework Directory

This field should contain the path to the Warrior Framework directory in your machine.

Current location of the Warrior Framework Directory The directory where the Python Action & Driver files will be read from.
/home/jSmith/warior_main/Warrior

Case Directory

This field should contain the path to the Case directory in your machine. This directory may or may not be inside your Warrior Framework directory

Test Case Directory The test case directory is where the XML files created by this application are stored.
/home/jSmith/warior_main/Warrior/Warriorspace/Testcases

To understand what Warrior Framework Cases are, refer to [Warrior Framework User Guide: Section 6.0](#).
To create cases in Katana, refer to [Katana User Guide: Section 3](#)

Suite Directory

This field should contain the path to the Suite directory in your machine. This directory may or may not be inside your Warrior Framework directory

Test Suite Directory The test suite directory is where the XML files created by this application are stored.
/home/jSmith/warior_main/Warrior/Warriorspace/Suites

To understand what Warrior Framework Suites are, refer to [Warrior Framework User Guide: Section 7.0](#).
To create suites in Katana, refer to [Katana User Guide: Section 4](#)

Project Directory

This field should contain the path to the Project directory in your machine. This directory may or may not be inside your Warrior Framework directory

Project Directory
The project directory is where the project XML files created by this application are stored.
/home/jSmith/warior_main/Warrior/Warriospace/Projects 

To understand what Warrior Framework Projects are, refer to [Warrior Framework User Guide: Section 8.0](#). To create projects in Katana, refer to [Katana User Guide: Section 5](#)

Input Data Directory

This field should contain the path to the Data directory in your machine. This directory may or may not be inside your Warrior Framework directory

Input Data File Directory
The input data file directory is where the data files created by this application will be stored.
/home/jSmith/warior_main/Warrior/Warriospace/Data 

To understand what Warrior Framework Input Data Files are, refer to [Warrior Framework User Guide: Section 9.0](#). To create input data files in Katana, refer to [Katana User Guide: Section 6](#)

Python Path

Here, a path to an alternate Python that you may want to use for executing Warrior Framework can be added here. If this field is left empty, then the default Python on your system would be used.

Python Path
Enter the path to the Python that you want to use while executing Warrior - if left empty, the default Python will be used.
/usr/local/bin/python27 

Finally, hit save to save the configuration.

<input type="button" value="Save"/> 

Katana is now ready to be used.

3. CREATING CASES WITH KATANA

Katana allows you to create Cases through its user interface. You have to start up Katana and go to its Cases tab. There, you will see a “New Case” button:



Note: The list of cases below the “New Case” button may differ as Katana would show the Cases saved in your case directory.

On clicking that button, you will be directed to a new page:

KATANA TESTCASE EDITOR

To access help, hover over the fields to view the tooltips.

DETAILS

Name *	Test case form name, will be used as the file name. No spaces.
Test case Title *	Title or Description of this Test case
Category	Testcase Category
Testcase State	▼
Engineer *	Jo Smith
Last updated	2016-09-21 11:48
Default On Error	next ▼
Input Data File	Location of the Input Data File <input type="checkbox"/> No Data File.
Data type *	▼
Log Directory	Path to the Log Directory.
Results Directory	Path to where Results of this testcase are stored.
Expected Results	Your expected value for this testcase.

REQUIREMENTS

STEPS

A Testcase requires at least one Step definition.
Please use the New Step button below to create a step.

 Create another

A step by step guide to create a case is Katana is given below

1. Name

Name *	Test case form name, will be used as the file name. No spaces.
---------------	--

Here enter the name of the case. A name can be anything that you recognize. This is a mandatory field.

Name * 

2. Case Title

Test case Title * 

A case Title (or description) is a field where you can add a descriptive title to identify what this case does. This is a mandatory field.

Test case Title * 

3. Category

Category 

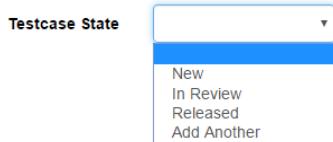
In this field, you can classify this case into a particular “category”. Categorization of cases is important when you have to run all the cases in the same category together. Please refer [Warrior Framework User Guide: Section 17.1.13](#) to better understand how a category works. This is an optional field, although, it is recommended that you categorize your cases.

Category 

4. Case State

Testcase State 

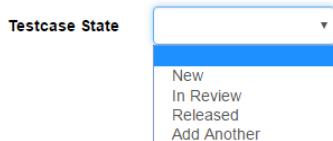
A case state is informative when you have to keep track of which cases are new, released, or in review. These are the in-build categories that are available to you. This is an optional field, but it is recommended that you fill out this field so as to keep track of which case is in which state of development.



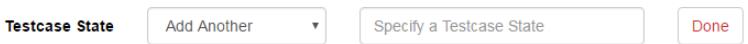
You can select any state of your choice, like “New”. This would mark your case as “New” meaning that this is a newly created case.



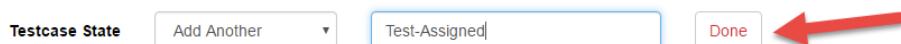
Furthermore, if the available categories do not really fit your case state, you can go ahead and select “Add Another”.



This opens up another field which lets you add a State of your own.



Here you can add in any case state that suits the state in which your case is, such as “Test-Assigned”, and click “Done”.



Your case now has a newly created “Test-Assigned” state. You can also use this newly created states for any other Case that you may create after this.



5. Engineer

This field indicates the name of the Engineer who created this Case. This is a prefilled field but, you can change it if you want to. This is a mandatory field.

Engineer *

Jo Smith

6. Last Updated (Date-Time stamp)

This field indicates the Date and Time on which this Case was last updated. This is a prefilled field, but the contents cannot be changed.

Last updated

2016-05-18

11:19

7. Default on Error

This is field where you can specify what the Case should do it case it errors out. By default, it is “next”, that is, if you don't make a change, the Case will proceed to the next Case available for execution if the current case throws an error.

Default On Error

next ▾

Clicking on the dropdown shows all the options that are available to you to tell Warrior Framework what to do in the case where the current Case errors out. You have the ability to choose any one of them.

Default On Error

next ▾

next
abort
goto

On selecting “goto”, another pops up and lets you add the Case number that Warrior Framework should go to in case of an error.

Default On Error

goto ▾

Step # to go to

Default On Error

goto ▾

5

For further information about what Default on Error does, please go through [Warrior Framework User Guide: Section 6](#)

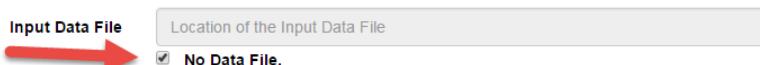
8. Input Data File

This is field wherein you can add the path to a datafile ([Warrior Framework User Guide: Section 9](#)) that you want this case to use. This field is **mandatory only if the “No Data File” checkbox is left unchecked.**



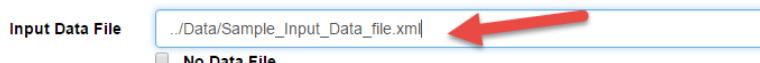
Input Data File Location of the Input Data File
 No Data File.

Once the “No Data File” checkbox is checked, the field grays out and then you don’t have to enter a path to the datafile.



Input Data File Location of the Input Data File
 No Data File.

In the case where you want a datafile included in your Case, uncheck the “No Data File” checkbox and add either an absolute path to your datafile or a path relative from your case. It is recommended that you add a relative path here.



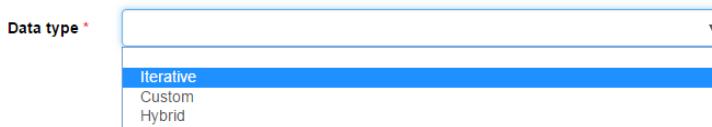
Input Data File /Data/Sample_Input_Data_file.xml
 No Data File.

9. Data type



Data type *

This dropdown field lets you indicate what kind how a case should interact with the datafile ([Warrior Framework User Guide: Section 9.0](#)). This is a **mandatory** field.



Data type *

- Iterative
- Custom
- Hybrid

10. Log Directory

Log Directory

Path to the Log Directory.

This field lets you specify the path to the Logs directory – this directory is where all the logs returned by Warrior Framework would be stored. This is an optional field.

Log Directory

/home/jSmith/Warrior_Logs|



If left empty, the logs would be directed and stored in the default location.

11. Results Directory

Results Directory

Path to where Results of this testcase are stored.

This field lets you specify the path to the Results directory – this directory is where all the results returned by Warrior Framework would be stored. This is an optional field.

Results Directory

/home/jSmith/Warrior_Results|



If left empty, the results would be directed and stored in the default location.

12. Expected Results

This field is where you can specify what kind of a result is expected from the case. This is an optional field as it is an informative field for you.

Expected Results

PASS|



13. Requirements

This field is for you add any requirements that this case may have. This is an optional field. You can add a new requirement by clicking on the “New Requirement” button.

REQUIREMENTS**New Requirement**

This opens up a new field in which you can specify the requirement.

REQUIREMENTS

Specify the test case requirement

OK**Cancel****New Requirement**

Once typed in, you can hit “OK” to save it or “Cancel” to discard all changes.

REQUIREMENTS

Requirement-001

OK**Cancel****New Requirement****REQUIREMENTS**

Requirement-001

OK**Cancel****New Requirement**

You can also delete an added requirement by clicking the minute “x” against the Requirement that you want to delete.

REQUIREMENTS

Requirement-001

New Requirement**14. Step**

Every Case requires at least one step. A new step can be added to the Case by clicking the “New Step” button. The first entire step is mandatory. Every step after that is optional.

STEPS

A Testcase requires at least one Step definition.
Please use the New Step button below to create a step.

[New Step](#)



On clicking the “New Step” button, a new set of fields drop down for you to be filled. All the newly appeared fields are a part of this new “step”.

STEPS

A Testcase requires at least one Step definition.
Please use the New Step button below to create a step.

Drivers * <input type="text" value="Select Driver"/>	Keywords * <input type="text" value="Select Keyword"/>
Description <input type="text"/>	
Execute Type <input type="text" value="Yes"/>	
Arguments	
Signature & Comment	
Impact on Failure <input type="text" value="impact"/>	
Context <input type="text" value="positive"/>	
Run Multiple Times <input type="text" value="Number of times this step should run."/>	
On Error <input type="text" value="next"/>	
<input type="button" value="Save Step"/> <input type="button" value="Cancel"/>	

[New Step](#)

Create another

15. Drivers

Every step must have an associated driver to it. This is a mandatory field.

Drivers *

Katana lets you select a driver by showing all the drivers that you have in your Warrior Framework directory in the dropdown. You can select any driver out of those. You would not be able to select any driver which lies outside of the ProductDrivers directory. This is because Warrior Framework enforces a strict directory structure for the Actions and ProductDrivers.

The list of drivers on your Katana may differ since you may have different drivers in your ProductDrivers directory.

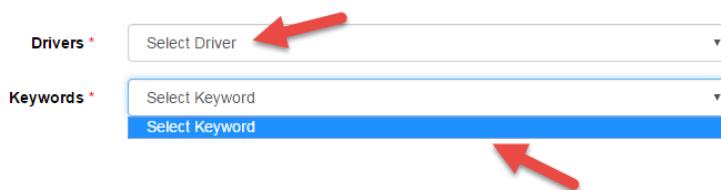


16. Keywords

Every step must have an associated keyword to it. This is a **mandatory** field

Keywords *

If no driver is selected, no keywords will show up in the dropdown. You have to choose the driver before you can choose the Keyword.



If the driver is chosen, a list of Keywords would show up. You can select any Keyword out of those.

Drivers *	<input type="text" value="cli_driver"/>
Keywords *	<input type="text" value="Select Keyword"/> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 2px;"> Select Keyword connect connect_all connect_ssh connect_telnet disconnect disconnect_all send_alltestdata_commands send_command send_commands_bytestdata_rownum send_commands_bytestdata_title send_commands_bytestdata_title_rownum set_session_timeout verify_session_status </div>

17. Description

This field is populated when the Keyword is chosen. The description is for your reference, so that, you know exactly what this keyword does. This is an optional field.

Description	<input type="text" value="Execute a CSE's script from Warrior framework"/>
-------------	--

18. Arguments

This space gets populated when the keyword is selected. All the arguments accepted by the Keyword show up here. You can type in the values associated with each value against that argument name. To identify which are the mandatory arguments, you can go through the next section.

Arguments	<input type="text" value="system_name"/>
	<input type="text" value="cse_logfile_path"/>

19. Signature & Comments

This field also appears prepopulated when the Keyword is selected. This is basically for you benefit, so that you understand what the keyword does when it runs, what kind of arguments it accepts, and what kind of values it returns.

Signature & Comment

```
def execute_cse_script(self, system_name, cse_logfile_path='default'):
```

Executes CSE's run script (cse's wrapper script) on the cse view.
 1. Connects to the Server IP
 2. Sets the view
 3. Go to the cse run script location
 4. Execute the cse run script on the server
 5. Send the log file from the server to Warrior logsdir
 6. Checks the log for the run result

 :Arguments:
 1. system_name(string) = name of the system in the input datafile
 2. cse_logfile_path = full path to the cse logfile, default='default'

 :Returns:
 1. status(bool) = True or False

20. Execute Type

Execute Type

The Execute Type is by default set to 'Yes'. You can change it by clicking on the dropdown. This reveals all the options you have available to you.

Execute Type

If
 If Not
Yes
 No

On clicking on 'If' and 'If Not', two other input boxes show up that let you add your conditions.

Execute Type

Condition

Condition Value

Else

Execute Type

Condition

Condition Value

Else

The third input box is a dropdown consisting of the condition that handles the scenario where the condition in 'If'/'If Not' is not met and hence Warrior Framework would need to know what to do in such a case.

Execute Type	<input type="text" value="If Not"/>
Condition	<input type="text" value="Rule condition"/>
Condition Value	<input type="text" value="Rule condition value"/>
Else	<input type="text" value="next"/> <div style="border: 1px solid #ccc; padding: 5px; background-color: #f9f9f9;"> next abort goto </div>

21. Impact on Failure.

Impact on Failure	<input type="text" value="impact"/>
-------------------	-------------------------------------

On clicking the dropdown you would be able to select from one of the options presented to you. The default is 'impact'.

Impact on Failure	<input type="text" value="impact"/> <div style="border: 1px solid #ccc; padding: 5px; background-color: #f9f9f9;"> impact Impact noimpact </div>
-------------------	---

22. Context

Context	<input type="text" value="positive"/>
---------	---------------------------------------

On clicking the dropdown you would be able to select from one of the options presented to you. The default is 'positive'.

Context	<input type="text" value="positive"/> <div style="border: 1px solid #ccc; padding: 5px; background-color: #f9f9f9;"> positive positive negative </div>
---------	---

23. Run Mode

This field lets you select if you want this keyword to run until pass (RUP), run until failure (RUF), or run multiple times (RMT). This is an optional field.

Run Mode

The Run Mode field is dropdown consisting of values “RUP”, “RUF”, and “RMT”.

Run Mode

RMT
RUP
RUF

Whenever a value from the dropdown is selected, and new field “Value” pops up that lets you fill in the maximum number of times you want to run this step – if RUP or RUF is selected or the number of times you want to run this step if RMT is selected.

Run Mode

RMT

Value *

You can input the number of times you want this particular step to run here.

Value *

4

24. On Error

This tag handles the Error condition. You can click on the dropdown to choose an option. The default is whatever the case onError value is.

On Error

goto

Goto Step #

5

You can click on the dropdown to assign a different error handling for this step.

On Error

goto
next
abort
goto



25. Save Step

Clicking on the 'Save Step' button saves this step. This DOES NOT save the case – only that particular step.

A step must be saved



Once the step is saved, a summary of it would appear on that page. This would look something like this:

STEPS												
#	Driver	Keyword	Iteration Type	Execute	Arguments	onError	Impact	Context	Insert Step	RMT	⊕	
1	cse_driver	execute_cse_script		Type = Yes		next	impact	+	✖	8	⊕	
<i>Execute a CSE's script from Warrior framework</i>												

26. Cancel

On the other hand, clicking on 'Cancel' discards all the changes made to this step.



26. Edit step

Clicking on the driver name lets you edit an already created step.

STEPS												
#	Driver	Keyword	Iteration Type	Execute	Arguments	onError	Impact	Context	Insert Step	RMT	⊕	
1	cse_driver	execute_cse_script		Type = Yes		next	impact	+	✖	8	⊕	
<i>Execute a CSE's script from Warrior framework</i>												

This opens up the step editor once again and you can make the desired changes.

STEPS

#	Driver	Keyword	Iteration Type	Execute	Arguments	onError	Impact	Context	Run Mode	Insert Step	⋮
1	cse_driver	execute_cse_script		Type = Yes		next	impact	+	RMT = 8		

Execute a CSE's script from Warrior framework

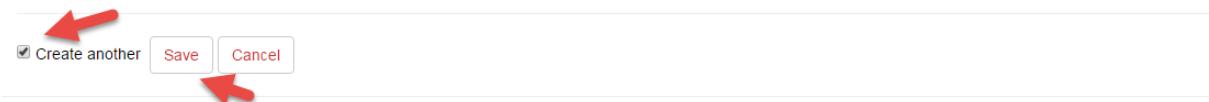
Drivers *	cse_driver				
Keywords *	execute_cse_script				
Description	Execute a CSE's script from Warrior framework				
Execute Type	Yes				
Arguments	<table> <tr> <td>system_name</td> <td><input type="text"/></td> </tr> <tr> <td>cse_logfile_path</td> <td><input type="text"/></td> </tr> </table>	system_name	<input type="text"/>	cse_logfile_path	<input type="text"/>
system_name	<input type="text"/>				
cse_logfile_path	<input type="text"/>				
Signature & Comment	<pre>def execute_cse_script(self, system_name, cse_logfile_path='default'): Executes CSE's run script (cse's wrapper script) on the cse view. 1. Connects to the Server IP 2. Sets the view 3. Go to the cse run script location 4. Execute the cse run script on the server 5. Send the log file from the server to Warrior logsdir 6. Checks the log for the run result :Arguments: 1. system_name(string) = name of the system in the input datafile 2. cse_logfile_path = full path to the cse logfile, default='default' :Returns: 1. status(bool) = True or False</pre>				
Impact on Failure	impact				
Context	positive				
Run Mode	RMT				
Value *	8				
On Error	next				
<input type="button" value="Save Step"/> <input type="button" value="Cancel"/>					

27. Save

Clicking on 'Save' saves the entire Case. If any unsaved step is present when this button is clicked, that unsaved step will get saved.



Checking the box against “Create Another” and then hitting Save, lets you save the case and directly opens up a page where you can start creating a new Case.



28. Cancel

Clicking on ‘Cancel’ discards all the changes performed.

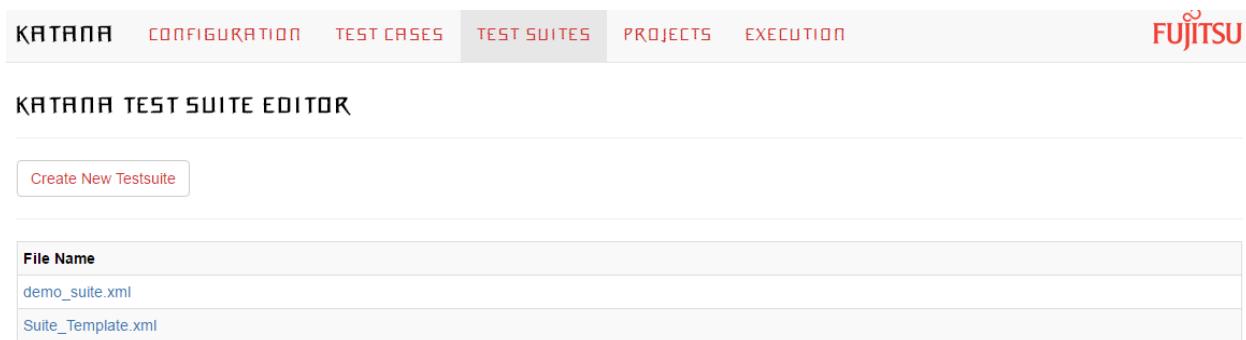


The case thus created would look something like this:

```
▼<Testcase>
  ▼<Details>
    <Name>Testcase_01</Name>
    <Title>Testcase_01_Title</Title>
    <Engineer>Jo Smith</Engineer>
    <Date>2016-05-18</Date>
    <Time>13:25</Time>
    <State>Test-Assigned</State>
    <InputDataFile>../Data/Sample_Input_Data_file.xml</InputDataFile>
    <Datatype>Iterative</Datatype>
    <default_onError action="goto" value="5"/>
    <Logsdir>/home/jSmith/Warrior_Logs</Logsdir>
    <Resultsdir>/home/jSmith/Warrior_Results</Resultsdir>
    <ExpectedResults>PASS</ExpectedResults>
  </Details>
  ▼<Steps>
    ▼<step Driver="cse_driver" Keyword="execute_cse_script" TS="1">
      ▼<Arguments>
        <argument/>
      </Arguments>
      <onError action="next"/>
      <Description>Execute a CSE's script from Warrior framework</Description>
      <iteration_type type="" />
      <Execute ExecType="Yes"/>
      <context>positive</context>
      <impact>impact</impact>
      <runmode type="RMT" value="8"/>
    </step>
  </Steps>
  ▼<Requirements>
    <Requirement>Requirement-001</Requirement>
  </Requirements>
</Testcase>
```

4. CREATING SUITES WITH KATANA

Katana allows you to create Suites through its user interface. You have to start up Katana and go to its Suites tab. There, you will see a “New Suite” button:



Note: The list of suites below the “New Suite” button may differ as Katana would show the Suites saved in your suite directory.

On clicking that button, you will be directed to a new page:

KATANA TEST SUITE EDITOR

To access help, hover over the fields to view the tooltips.

(fields marked with * are mandatory)

Name *	<input type="text" value="Test suite name"/>
Test suite Title	<input type="text" value="Title or Description of this Test case"/>
Type *	<input type="text"/>
Suite State	<input type="text"/>
Engineer	<input type="text" value="Who created/edited this testcase definition"/>
Last updated	<input type="button" value="Creation/up"/> <input type="button" value="Creation/up"/>
Default On Error Action *	<input type="text" value="next"/>
Results Directory	<input type="text"/>
Input Data File	<input type="text" value="Location of the Input Data File"/> <input type="button" value="Path"/>

Requirements	<input type="text" value="Requirement"/> <input type="button" value="delete"/>
---------------------	--

TEST CASES

Test case 1	<input type="button" value="Delete this testcase"/>
Path *	<input type="text" value="Test case file path"/> <input type="button" value="Path"/>
Context	<input type="text" value="positive"/>
Run type	<input type="text" value="sequential_keywords"/>
Run Multiple Times	<input type="text" value="Number of times this Testcase should run"/>
On Error Action	<input type="text"/>
<input checked="" type="checkbox"/> Impact On Failure <input type="button" value="Add/Insert Cases"/>	
<input type="checkbox"/> Create another <input type="button" value="Save"/> <input type="button" value="Cancel"/>	

A step by step guide to create a suite in Katana is given below:

1. Name

Name *	<input type="text" value="Test suite name"/>
---------------	--

Here enter the name of the suite. A name can be anything that you recognize. This is a mandatory field.

Name *

2. Suite Title

Test suite Title

A suite Title (or description) is a field where you can add a descriptive title to identify what this suite does. This is a mandatory field.

Test suite Title

3. Type

Type *

This is a mandatory field that lets you determine how the suite should run. You can click on the dropdown and select the 'type' of the suite.

Type *

- sequential_testcases
- parallel_testcases
- Run_Until_Fail
- Run_Until_Pass
- Run_Multiple

4. Suite State

Suite State

A suite state is informative when you have to keep track of which suites are new, released, or in review. These are the in-build categories that are available to you. This is an optional field, but it is recommended that you fill out this field so as to keep track of which suite is in which state of development.

Suite State

New
In Review
Released
Add Another

You can select any state of your choice, like “New”. This would mark your suite as “New” meaning that this is a newly created suite. Furthermore, if the available categories do not really fit your suite state, you can go ahead and select “Add Another”. This opens up another field which lets you add a State of your own.

Suite State

Add Another

Specify a new Suite State

Done

Here you can add in any case state that suits the state in which your case is, such as “Test-Assigned”, and click “Done”.

Suite State

Add Another

Test-Assigned

Done

Your case now has a newly created “Test-Assigned” state. You can also use this newly created states for any other Case that you may create after this.

Suite State

Test-Assigned

5. Engineer

This field indicates the name of the Engineer who created this Case. This is a prefilled field but, you can change it if you want to. This is a mandatory field.

Engineer

Jo Smith

6. Results Directory

Results Directory

--

This field lets you specify the path to the Results directory – this directory is where all the results returned by Warrior Framework would be stored. This is an optional field.

Results Directory

/home/jSmith/Warrior_Results



7. Last Updated (Date-Time stamp)

This field indicates the Date and Time on which this Case was last updated. This is a prefilled field, but the contents cannot be changed.

8. Default on Error

This is field where you can specify what the Suite should do it case it errors out. By default, it is “next”, that is, if you don’t make a change, the Suite will proceed to the next Suite available for execution if the current suite throws an error.

Clicking on the dropdown shows all the options that are available to you to tell Warrior Framework what to do in the case where the current Case errors out. You have the ability to choose any one of them.

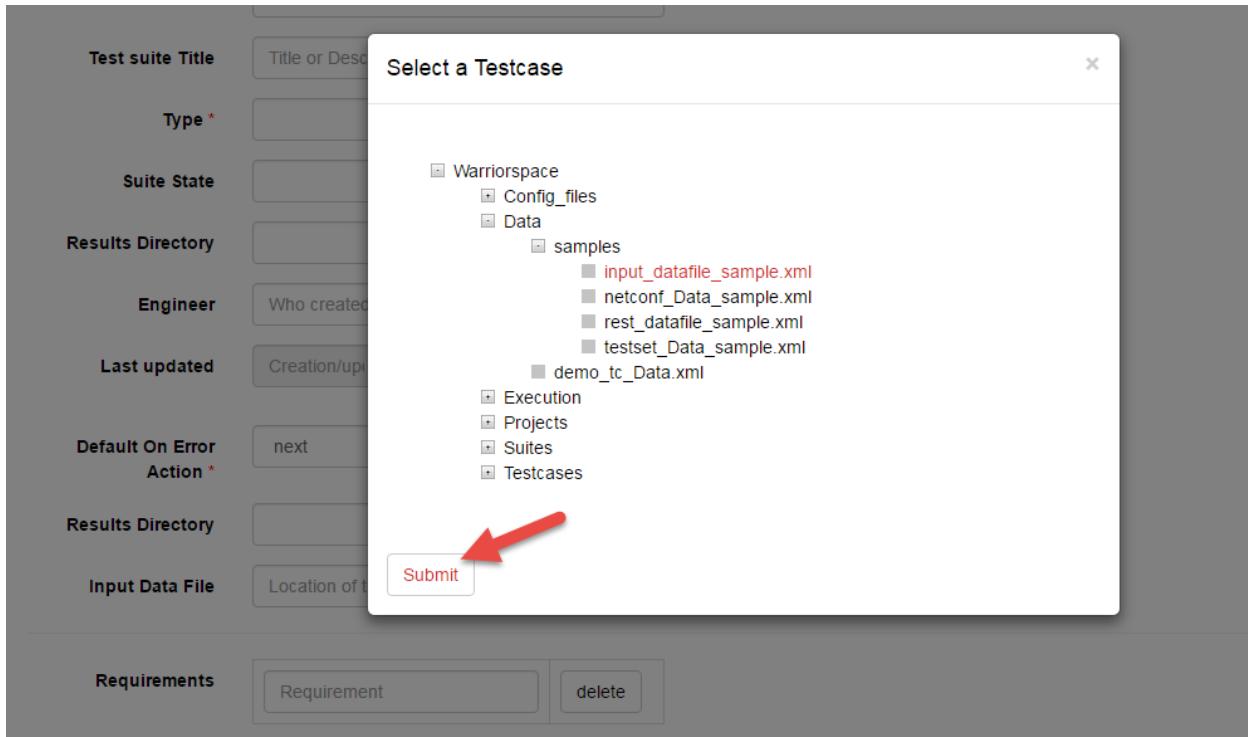
On selecting “goto”, another pops up and lets you add the Case number that Warrior Framework should go to in case of an error.

9. Input Data File

This is field wherein you can add the path to a datafile ([Warrior Framework User Guide: Section 9.0](#)) that you want this case to use. You can click on the 'Path' button to choose a path



Once you choose the Datafile, you can click on 'Submit' to enter that datafile path.



Or you can simply type in a path to your datafile.



10. Requirements

Requirements	Requirement	delete
Add More		

This field is for you add any requirements that this suite may have. This is an optional field. You can add a new requirement by clicking on the “Add More” button.

Requirements
Requirement-001
<input type="button" value="delete"/>
<input type="button" value="Add More"/>



This opens up a new field in which you can specify the requirement.

Requirements
Requirement-001
<input type="button" value="delete"/>
Requirement
<input type="button" value="delete"/>
<input type="button" value="Add More"/>

You can also delete an added requirement by clicking the ‘delete’ button against the Requirement that you want to delete.

Requirements
Requirement-001
<input type="button" value="delete"/>
Requirement
<input type="button" value="delete"/>
<input type="button" value="Add More"/>



After this point, every input is related to the case:

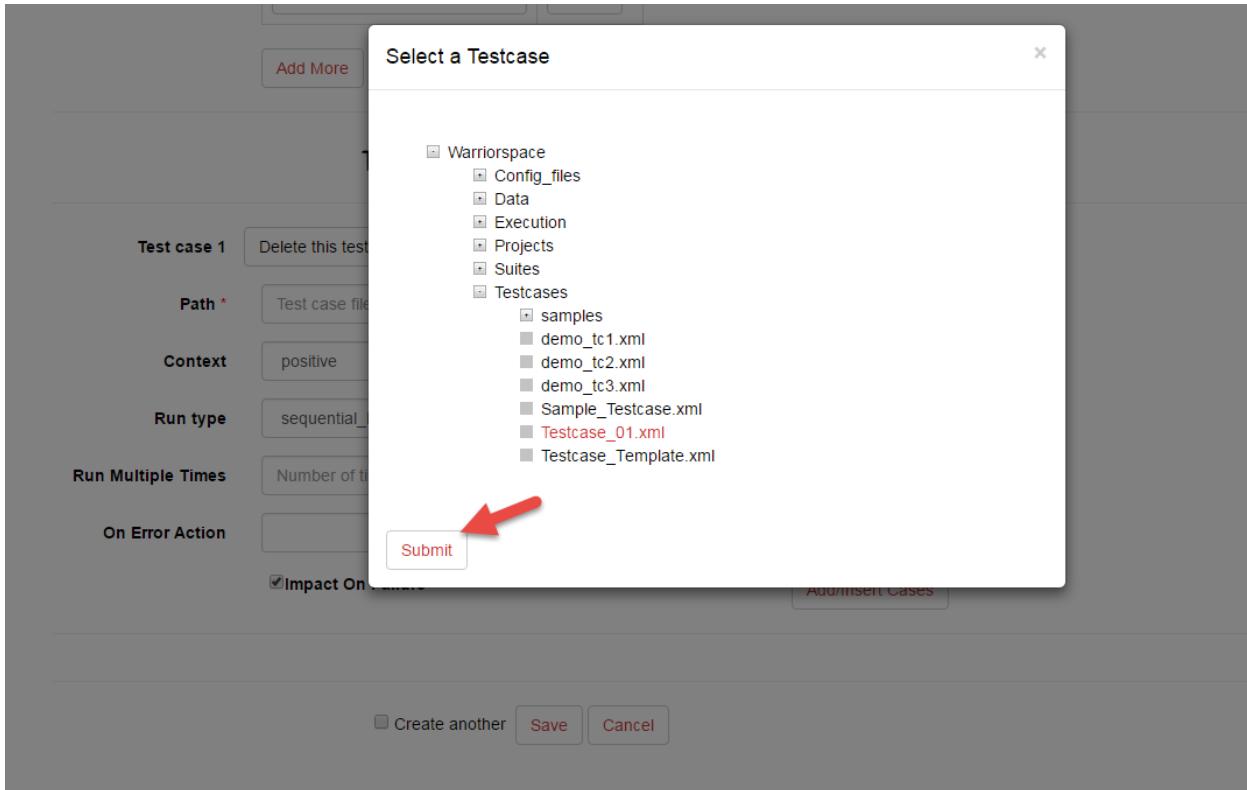
TEST CASES

11. Path

This is the first field in case part of the Suite creation. You can add a path to the case by clicking on the ‘Path’ button.

Path *	Test case file path	<input type="button" value="Path"/>
---------------	---------------------	-------------------------------------





Then, you will be able to select a path to your case, and to enter this path you can hit 'Submit'.

Path *	<input type="text" value="../Testcases/Testcase_01.xml"/>	<input type="button" value="Edit"/>
--------	---	-------------------------------------

You can also manually add a path to your case. This is a mandatory field.

12. Context

Context	<input type="text" value="positive"/>
Context	<input type="text" value="positive"/> <input checked="" type="text" value="positive"/> <input type="text" value="negative"/>

13. Run type

Run type	<input type="text" value="sequential_keywords"/>
----------	--

Run type

sequential_keywords	▼
sequential_keywords	◀
parallel_keywords	▶

14. Run Mode

This field lets you select if you want this case to run until pass (RUP), run until failure (RUF), or run multiple times (RMT). This is an optional field.

Run Mode

RUP	▼
RUF	◀
RMT	▶

The Run Mode field is dropdown consisting of values “RUP”, “RUF”, and “RMT”.

Run Mode

RUP	▼
RUF	◀
RMT	▶

Whenever a value from the dropdown is selected, and new field “Value” pops up that lets you fill in the maximum number of times you want to run this case – if RUP or RUF is selected or the number of times you want to run this case if RMT is selected.

Run Mode

RUP	▼
Value *	

You can input the number of times you want this particular case to run here.

Run Mode

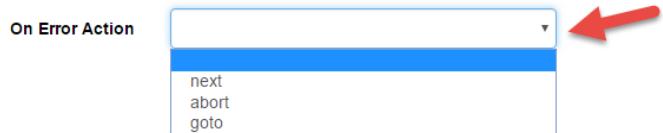
RUP	▼
Value *	5

15. On Error Action

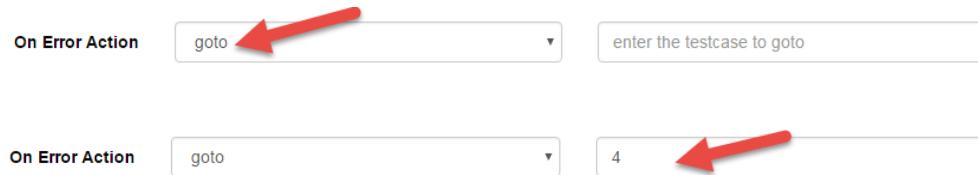
This tag handles the Error condition. You can click on the dropdown to choose an option.

On Error Action

	▼
--	---



If goto is selected, then, another input box appears that lets you add the number of case to which you want to jump in case of failure.



16. Impact on Failure

The Impact on Failure checkbox is checked by default.

Impact On Failure

You can uncheck it if you want the result of this particular case to not affect the entire suite.

Impact On Failure

17. Add/Insert Cases



18. Save

Clicking on the 'Save' button saves this suite.



Checking the 'Create Another' checkbox, lets you save the suite and then open a new fresh suite page for you to start creating a new suite.



19. Cancel

On the other hand, clicking on 'Cancel' discards all the changes made to this suite.

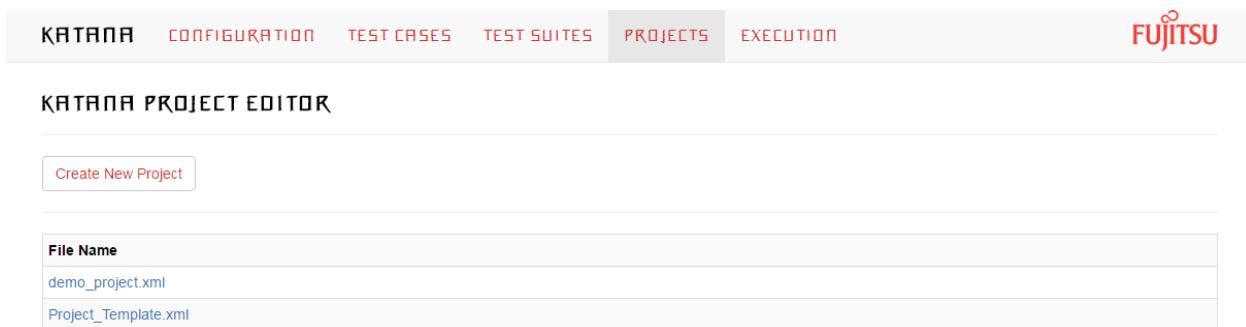


The suite thus created looks something like this:

```
▼<TestSuite>
  ▼<Details>
    <Name>Suite_01</Name>
    <Title>Suite_01_Title</Title>
    <Engineer>Jo Smith</Engineer>
    <Date>05/18/2016</Date>
    <Time>15:33:57</Time>
    <type exectype="sequential_testcases"/>
    <State>Test-Assigned</State>
    <default_onError action="goto" value="3"/>
    <Resultsdir>/home/jSmith/Warrior_Results</Resultsdir>
    <IDF>../Data/samples/input_datafile_sample.xml</IDF>
  </Details>
  ▼<Requirements>
    <Requirement>Requirement-001</Requirement>
    <Requirement/>
  </Requirements>
  ▼<Testcases>
    ▼<Testcase>
      <path>../Testcases/Testcase_01.xml</path>
      <context>positive</context>
      <runtype>sequential_keywords</runtype>
      <onError action="goto" value="4"/>
      <runmode type="RUP" value="5"/>
      <impact>impact</impact>
    </Testcase>
  </Testcases>
</TestSuite>
```

5. CREATING PROJECTS WITH KATANA

Katana allows you to create Projects through its user interface. You have to start up Katana and go to its Projects tab. There, you will see a “Create New Project” button:



Note: The list of projects below the “Create New Project” button may differ as Katana would show the Projects saved in your project directory.

On clicking that button, you will be directed to a new page:


KATANA PROJECT EDITOR

To access help, hover over the fields to view the tooltips.

(Fields marked with * are mandatory)

Name *	<input type="text" value="Project name"/>
Project Title	<input type="text" value="Title or Description of this project"/>
Engineer	<input type="text" value="Who created/edited this testcase definition"/>
Project State	<input type="text" value=""/>
Last updated	<input type="button" value="Creation/up"/> <input type="button" value="Creation/up"/>
Default On Error Action *	<input type="text" value="next"/>
Results Directory	<input type="text" value=""/>

TEST SUITES

Test Suite 1	<input type="button" value="Delete this Test Suite"/>
Path *	<input type="text" value="Test suite file path"/> <input type="button" value="Path"/>
On Error Action	<input type="text" value="next"/>
Impact On Error <input checked="" type="checkbox"/> <input type="button" value="Add/Insert Suites"/>	
<input type="checkbox"/> Create another <input type="button" value="Save"/> <input type="button" value="Cancel"/>	

A step by step guide to create a project in Katana is given below:

1. Name

Name *	<input type="text" value="Project name"/>
--------	---

 Here enter the name of the project. A name can be anything that you recognize. This is a mandatory field.

Name *	<input type="text" value="Project_01"/>
--------	---

2. Project Title

Project Title

Title or Description of this project

A suite Title (or description) is a field where you can add a descriptive title to identify what this suite does.

This is a mandatory field.

Project Title

Project_01_Title|



3. Engineer

Engineer

Sanika Kulkarni



This field indicates the name of the Engineer who created this Case. This is a pre-filled field but, you can change it if you want to. This is a mandatory field.

4. Project State

Project State



A project state is informative when you have to keep track of which projects are new, released, or in review. These are the in-build categories that are available to you. This is an optional field, but it is recommended that you fill out this field so as to keep track of which project is in which state of development.

Project State

- New
- In Review
- Released
- Add Another

You can select any state of your choice, like “New”. This would mark your suite as “New” meaning that this is a newly created project. Furthermore, if the available categories do not really fit your project state, you can go ahead and select “Add Another”. This opens up another field which lets you add a State of your own.

Project State

Add Another



Specify a new Suite

Done



Here you can add in any project state that suits the state in which your project is, such as “Test-Assigned”, and click “Done”.



The screenshot shows a user interface for managing project states. A dropdown menu labeled "Project State" contains the option "Test-Assigned". To the right of the dropdown is a button labeled "Done" with a red arrow pointing to it, indicating the action to be taken after adding the state.

Your project now has a newly created “Test-Assigned” state. You can also use this newly created states for any other Project that you may create after this.



The screenshot shows a dropdown menu labeled "Project State" with "Test-Assigned" selected. This indicates that the previously added state is now available for use in other projects.

5. Last Updated (Date-Time stamp)

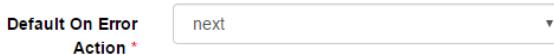
This field indicates the Date and Time on which this Case was last updated. This is a prefilled field, but the contents cannot be changed.



The screenshot shows a dropdown menu labeled "Last updated" with two options: "Creation/upd" and "Creation/upd". This is a standard date-time picker interface.

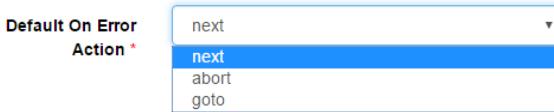
6. Default on Error

This is field where you can specify what the Project should do it case it errors out. By default, it is “next”, that is, if you don’t make a change, the Project will proceed to the next Project available for execution if the current suite throws an error.



The screenshot shows a dropdown menu labeled "Default On Error Action" with "next" selected. This indicates the default behavior for handling errors in the project.

Clicking on the dropdown shows all the options that are available to you to tell Warrior Framework what to do in the case where the current Project errors out. You have the ability to choose any one of them.



The screenshot shows a dropdown menu labeled "Default On Error Action" with "next" selected. A list of options is displayed: "next", "abort", and "goto". The "next" option is highlighted with a blue background, indicating it is the currently selected action.

On selecting “goto”, another pops up and lets you add the Project number that Warrior Framework should go to in case of an error.

Default On Error Action *

enter the suite to goto

Default On Error Action *

7. Results Directory

Results Directory

This field lets you specify the path to the Results directory – this directory is where all the results returned by Warrior Framework would be stored. This is an optional field.

Results Directory

Beyond this, all the inputs refer to the suites that would be included in the project.

TEST SUITES

8. Delete this Suite

Test Suite 1

This button lets you delete a Suite that has been added to the project. But the Project needs at least one suite in it.

9. Path

Path *

This is the first field in case part of the Suite creation. You can add a path to the case by clicking on the 'Path' button.

The screenshot shows the KATANA PROJECT EDITOR interface. In the center, there's a modal window titled "Select a new Path". Inside the modal, there's a tree view of paths under "WarriorSpace". The tree includes "Config_files", "Data", "Execution", "Projects", "Suites", and "Testcases". Under "Suites", files like "demo_suite.xml", "Sample_Suite.xml", "Suite_01.xml", and "Suite_Template.xml" are listed. At the bottom of the modal is a "Submit" button. A red arrow points from this "Submit" button back to the "Path" input field in the main editor area below.

Then, you will be able to select a path to your case, and to enter this path you can hit 'Submit'.

Path *	<input type="text" value="./Suites/Suite_01.xml"/>	<input type="button" value="Edit"/>
--------	--	-------------------------------------

You can also manually add a path to your case. This is a mandatory field.

10. On Error Action

On Error Action	<input type="button" value="next"/> ▾
-----------------	---------------------------------------

This tag handles the Error condition. You can click on the dropdown to choose an option. If goto is selected, then, another input box appears that lets you add the number of suite to which you want to jump in case of failure.

On Error Action	<input type="button" value="next"/> ▾
-----------------	---------------------------------------

next
next
 abort
 goto

On Error Action	goto	enter the suite to goto
-----------------	------	-------------------------

On Error Action	goto	2
-----------------	------	---

11. Impact on Failure

The Impact on Failure checkbox is checked by default.

Impact On Error

You can uncheck it if you want the result of this particular case to not affect the entire suite.

Impact On Error

12. Add/Insert Suites



Clicking on this button lets you add another suite to this project.

12. Save

13. Clicking on the 'Save' button saves this project.



Checking the 'Create Another' checkbox, lets you save the project and then open a new fresh project page for you to start creating a new project.



14. Cancel

15. On the other hand, clicking on 'Cancel' discards all the changes made to this project.

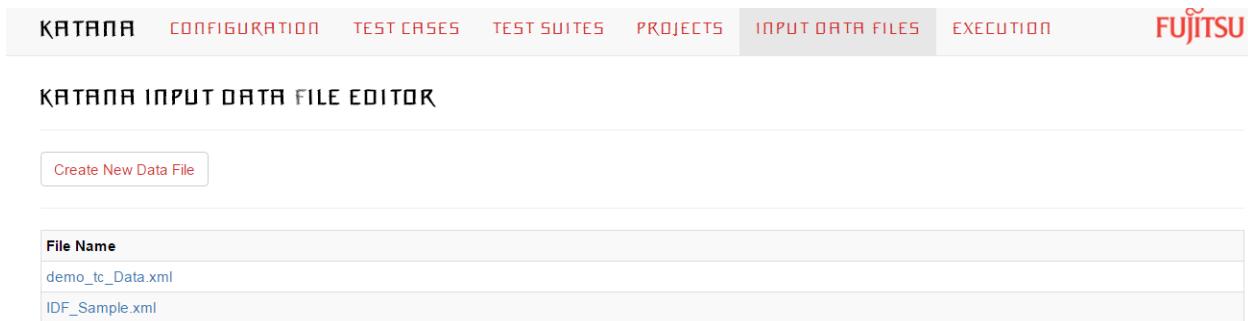


The Project thus created looks something like this:

```
▼<Project>
  ▼<Details>
    <Name>Project_01</Name>
    <Title>Project_01_Title</Title>
    <Engineer>Jo Smith</Engineer>
    <State>Test-Assigned</State>
    <Date>05/31/2016</Date>
    <Time>18:03:53</Time>
    <default_onError action="next"/>
  </Details>
  ▼<Testsuites>
    ▼<Testsuite>
      <path>../Suites/Suite_01.xml</path>
      ▼<Execute ExecType="Yes">
        <Rule Condition="" Condvalue="" Else="next" Elsevalue="" />
      </Execute>
      <onError action="goto" value="5"/>
      <impact>impact</impact>
    </Testsuite>
  </Testsuites>
</Project>
```

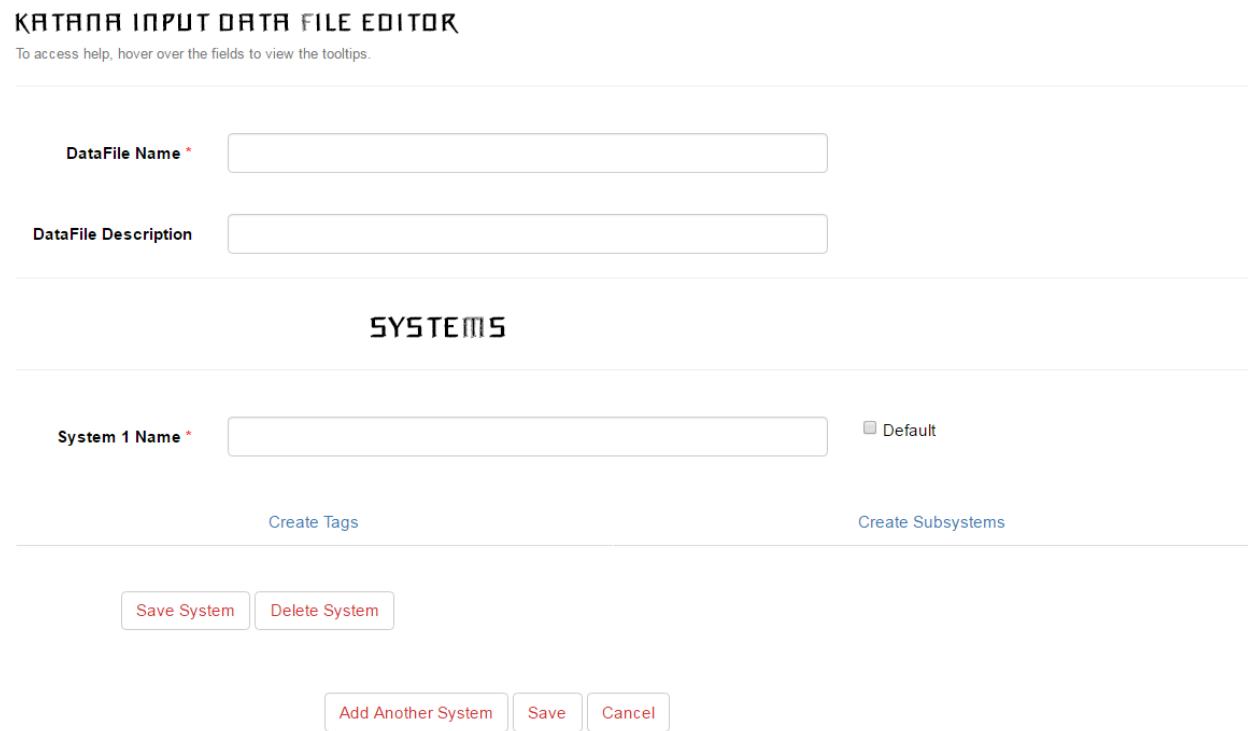
6. CREATING INPUT DATA FILES WITH KATANA

Katana allows you to create Input Data Files through its user interface. You have to start up Katana and go to its Input Data Files tab. There, you will see a “Create New Data File” button:



Note: The list of input data files below the “Create New Data Files” button may differ as Katana would show the input data files saved in your Data directory.

On clicking that button, you will be directed to a new page:



The screenshot shows the 'KATANA INPUT DATA FILE EDITOR' interface. At the top, there's a note: 'To access help, hover over the fields to view the tooltips.' Below this, there are two input fields: 'DataFile Name *' and 'DataFile Description'. Under the heading 'SYSTEMS', there's a 'System 1 Name *' field with a 'Default' checkbox. Below the systems section are buttons for 'Create Tags' and 'Create Subsystems'. At the bottom, there are buttons for 'Save System', 'Delete System', 'Add Another System', 'Save', and 'Cancel'.

A step by step guide for creating an input data file in Katana is given below:

1. DataFile Name

DataFile Name *



This field represents the name of the file that you will be creating. Type in a name for the input data file here

DataFile Name *



2. DataFile Description

DataFile Description

You can add a description for the data file here.

DataFile Description



3. Systems

You can start creating systems inside this Data file

SYSTEMS

System 1 Name *

Default

3.1. System Name

You can add the name of the system in this input field

System 1 Name *



Default

3.2. Default System

System 1 Name *

Default



You can set any system as the default system by checking the “Default” checkbox.

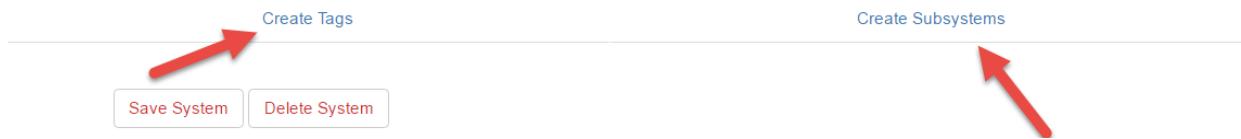
System 1 Name *

Default

If no system has been set as default then the first system would be treated as the default.

3.3. Create Tags and Subsystems

For any system, you can either create Tags for it or subsystems, but not both.



3.3.1. Creating Tags

Clicking on the “Create Tags” would display input boxes and other options to create the tags.



The screenshot shows the "Create Tags" input form. It includes fields for "Tag 1 Name" and "Tag 1 Value", each with a dropdown arrow. Below the fields are buttons for "Add Another Tag" and "Reset". At the bottom are "Save System" and "Delete System" buttons. A red arrow points to the "Tag 1 Name" field.

3.3.1.1. Tag Name

Since, the input data file does not dictate as to which tags should be created for each system, you have to add the tag names that you want. Immediately, on clicking inside the Tag Name input box, you would be able to view all the tag names that are already available to choose from.

Create Tags

Create Subsystems

Tag 1 Name * <input type="text" value="u"/> *	Tag 1 Value * <input type="text"/>	Delete
<div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> conn_type ip password prompt ssh_port telnet_port testdata timeout url username variable_config </div>		

If the tag name already exists in that list, you can select it from the dropdown list; or else you can add a tag name that you want.

Create Tags

Create Subsystems

Tag 1 Name * <input type="text" value="u"/> *	Tag 1 Value * <input type="text"/>	Delete
<div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> url username </div>		

Create Tags

Create Subsystems

Tag 1 Name * <input type="text" value="u"/> *	Tag 1 Value * <input type="text"/>	Delete
<div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> url username </div>		

This is a mandatory field.

3.3.1.2. Tag Value

A corresponding value to the tag created above can be added here.

Create Tags	Create Subsystems
Tag 1 Name * <input type="text" value="username"/> Tag 1 Value * <input type="text" value="Jo"/>  Delete	

3.3.1.3. Add Another Tag

Create Tags	Create Subsystems
Tag 1 Name * <input type="text" value="username"/> Tag 1 Value * <input type="text" value="Jo"/> Delete  Add Another Tag Reset	

Clicking the “Add Another Tag” lets you create another tag for the same system.

Create Tags	Create Subsystems
Tag 1 Name * <input type="text" value="username"/> Tag 1 Value * <input type="text" value="Jo"/>  Delete Tag 2 Name * <input type="text"/> Tag 2 Value * <input type="text"/> Delete Add Another Tag Reset	

3.3.1.4. Copy Tags from System

This functionality lets you copy tags over from another system into the current system. This functionality is visible only when there are more than two systems containing tags in that data file.

You can select the system number that you want to copy the tags from.

Copy Tags from System: <input type="text" value="1"/>	Go
---	---------------------------------

And then hit “Go” to copy over the tags.

Copy Tags from System: <input type="text" value="1"/>	Go 
---	---

3.3.1.5. Reset

Create Tags		Create Subsystems	
Tag 1 Name *	username	Tag 1 Value *	Jo
Tag 2 Name *	timeout	Tag 2 Value *	20
Tag 3 Name *	prompt	Tag 3 Value *	blah
<input type="button" value="Add Another Tag"/> <input type="button" value="Reset"/> 			

The “Reset” button resets the entire system tags for you.

Create Tags		Create Subsystems	
Tag 1 Name *		Tag 1 Value *	
<input type="button" value="Add Another Tag"/> <input type="button" value="Reset"/>			

3.3.1.6. Save System

Create Tags		Create Subsystems	
Tag 1 Name *	username	Tag 1 Value *	Jo
Tag 2 Name *	password	Tag 2 Value *	536ett
Tag 3 Name *	url	Tag 3 Value *	http://httpbin.org/post
Tag 4 Name *	timeout	Tag 4 Value *	10
<input type="button" value="Add Another Tag"/> <input type="button" value="Reset"/>  <input type="button" value="Save System"/> <input type="button" value="Delete System"/>			

Clicking on Save System, saves the system for you.

SYSTEMS

SYSTEMS

#	System Name	System Child-Tags/Attributes	Subsystem Details	Default	
1	system1	username = Jo password = 536ett url = http://httpbin.org/post timeout = 10		Yes	

3.3.1.7. Edit System

SYSTEMS

#	System Name	System Child-Tags/Attributes	Subsystem Details	Default	
1	system1	username = Jo password = 536ett url = http://httpbin.org/post timeout = 10		Yes	

You can click on a saved system name to edit that system.

System 1 Name *

 Default

Create Tags
Create Subsystems

Tag 1 Name *	<input type="text" value="username"/>	Tag 1 Value *	<input type="text" value="Jo"/>	<input type="button" value="Delete"/>
Tag 2 Name *	<input type="text" value="password"/>	Tag 2 Value *	<input type="text" value="536ett"/>	<input type="button" value="Delete"/>
Tag 3 Name *	<input type="text" value="url"/>	Tag 3 Value *	<input type="text" value="http://httpbin.org/post"/>	<input type="button" value="Delete"/>
Tag 4 Name *	<input type="text" value="timeout"/>	Tag 4 Value *	<input type="text" value="10"/>	<input type="button" value="Delete"/>

3.3.1.8. Delete System

You can delete a saved system by clicking on the delete icon next to it.

SYSTEMS

#	System Name	System Child-Tags/Attributes	Subsystem Details	Default	
1	system1	username = Jo password = 536ett url = http://httpbin.org/post timeout = 10		Yes	

You can delete an unsaved system by clicking on the “Delete System” button.

System 1 Name *

 Default

Create Tags
Create Subsystems

Tag 1 Name *	<input type="text" value="username"/>	Tag 1 Value *	<input type="text" value="Jo"/>	
Tag 2 Name *	<input type="text" value="password"/>	Tag 2 Value *	<input type="text" value="536ett"/>	
Tag 3 Name *	<input type="text" value="url"/>	Tag 3 Value *	<input type="text" value="http://httpbin.org/post"/>	
Tag 4 Name *	<input type="text" value="timeout"/>	Tag 4 Value *	<input type="text" value="10"/>	

3.3.2. Create Subsystems

Create Tags
Create Subsystems

Subsystem 1 Name *	<input type="text"/>	<input type="checkbox"/> Default		
Tag 1 Name *	<input type="text"/>	Tag 1 Value *	<input type="text"/>	

Clicking on the “Create Subsystems” tab lets you create subsystems for that System.

3.3.2.1. Subsystem Name

You can add a name to the subsystem in the “Subsystem Name” field. This is a mandatory field.

The screenshot shows the "Create Subsystems" tab selected. At the top left, there is a "Create Tags" button. Below the tabs, there are fields for "Subsystem 1 Name" (containing "subsystem1") and "Default" (with an unchecked checkbox). There are also fields for "Tag 1 Name" and "Tag 1 Value". At the bottom, there are buttons for "Add Another Tag", "Save Subsystem", "Delete Subsystem", "Add Another Subsystem", "Save System", and "Delete System". A red arrow points to the "Subsystem 1 Name" input field.

3.3.2.2. Default Subsystem

Each subsystem has an option to be set as Default. A subsystem can be set as “Default” by checking the Default checkbox against it.

The screenshot shows the "Create Subsystems" tab selected. The "Subsystem 1 Name" field contains "subsystem1" and the "Default" checkbox is checked. Other fields like "Tag 1 Name" and "Tag 1 Value" are present. At the bottom, there are buttons for "Add Another Tag", "Save Subsystem", "Delete Subsystem", "Add Another Subsystem", "Save System", and "Delete System". A red arrow points to the checked "Default" checkbox.

By default, the subsystem is always the default subsystem.

3.3.2.3. Tag Name

Since, the input data file does not dictate as to which tags should be created for each subsystem, you have to add the tag names that you want. Immediately, on clicking inside the Tag Name input box, you would be able to view all the tag names that are already available to choose from.

The screenshot shows a user interface for adding a tag name. At the top left is a text input field labeled "Subsystem 1 Name *" containing "subsystem1". To its right is a checkbox labeled "Default" with a checked mark. Below these is a dropdown menu labeled "Tag 1 Name *". A red arrow points to the left side of this dropdown menu. The dropdown menu itself contains a list of tag names: conn_type, ip, password, prompt, ssh_port, telnet_port, testdata, timeout, url, username, and variable_config. The "ip" option is highlighted with a blue background.

If the tag name already exists in that list, you can select it from the dropdown list; or else you can add a tag name that you want.

This screenshot shows the same interface as above, but with a different selection. The "Tag 1 Name *" dropdown menu now has "ip" selected, indicated by a blue background. The other options are visible but not selected.

This is a mandatory field.

3.3.2.4. Tag Value

A corresponding value to the tag created above can be added here.

The screenshot shows the "Tag 1 Value *" input field at the bottom right of the form. It contains the IP address "197.168.58.92". A red arrow points to the right side of this input field. To the right of the input field is a red "Delete" button.

3.3.2.5. Add Another Tag

Subsystem 1 Name * Default

Tag 1 Name * Tag 1 Value *

Clicking this button, lets you add another tag to the same subsystem.

Subsystem 1 Name * Default

Tag 1 Name * Tag 1 Value *

Tag 2 Name * Tag 2 Value *

3.3.2.6. Copy Tags from Subsystem

This functionality lets you copy tags over from another subsystem into the current subsystem. This functionality is visible only when there are more than two systems containing subsystems in that data file.

You can select the system number that contains the subsystem that you want.

Copy Tags from System: and Subsystem:

You can select the subsystem number that you want to copy the tags from.

Copy Tags from System: and Subsystem:

And then hit "Go" to copy over the tags.

Copy Tags from System:

1

and Subsystem:

2

Go



3.3.2.7. Save Subsystem

Subsystem 1 Name *	<input type="text" value="subsystem1"/>	<input checked="" type="checkbox"/> Default	
Tag 1 Name *	<input type="text" value="ip"/>	Tag 1 Value * <input type="text" value="197.168.58.92"/>	<input type="button" value="Delete"/>
Tag 2 Name *	<input type="text" value="conn_type"/>	Tag 2 Value * <input type="text" value="telnet"/>	<input type="button" value="Delete"/>
<input type="button" value="Add Another Tag"/> <input type="button" value="Save Subsystem"/> <input type="button" value="Delete Subsystem"/>			



“Save Subsystem” saves the subsystem in Katana

System 2 Name *	<input type="text" value="system2"/>	<input type="checkbox"/> Default		
<input type="button" value="Create Tags"/> <input type="button" value="Create Subsystems"/>				
SUBSYSTEMS				
#	Subsystem Name	Child-Tags and Attributes	Default	<input type="checkbox"/>
1	subsystem1	1. ip=197.168.58.92 2. conn_type=telnet	Yes	<input type="checkbox"/>
<input type="button" value="Add Another Subsystem"/>				

3.3.2.8. Edit Subsystem

System 2 Name * Default

[Create Tags](#) [Create Subsystems](#)

SUBSYSTEMS

#	Subsystem Name	Child-Tags and Attributes	Default	
1	subsystem1	1. ip=197.168.58.92 2. conn_type=telnet	Yes	

[Add Another Subsystem](#)

A saved subsystem can be edited by clicking on the subsystem name.

System 2 Name * Default

[Create Tags](#) [Create Subsystems](#)

Subsystem 1 Name * Default

Tag 1 Name * <input type="text" value="ip"/>	Tag 1 Value * <input type="text" value="197.168.58.92"/> Delete
Tag 2 Name * <input type="text" value="conn_type"/>	Tag 2 Value * <input type="text" value="telnet"/> Delete

[Add Another Tag](#) [Save Subsystem](#) [Delete Subsystem](#)

[Add Another Subsystem](#)

3.3.2.9. Delete Subsystem

A saved subsystem can be deleted by clicking on the delete icon against it.

System 2 Name * Default

[Create Tags](#) [Create Subsystems](#)

SUBSYSTEMS

#	Subsystem Name	Child-Tags and Attributes	Default	
1	subsystem1	1. ip=197.168.58.92 2. conn_type=telnet	Yes	

[Add Another Subsystem](#)

An unsaved subsystem can be deleted by clicking the “Delete subsystem” button.

System 2 Name * Default

[Create Tags](#) [Create Subsystems](#)

Subsystem 1 Name * Default

Tag 1 Name * **Tag 1 Value *** [Delete](#)

Tag 2 Name * **Tag 2 Value *** [Delete](#)

[Add Another Tag](#) [Save Subsystem](#) [Delete Subsystem](#)

[Add Another Subsystem](#)

3.3.2.10. Add another subsystem

System 2 Name * Default

[Create Tags](#) [Create Subsystems](#)

SUBSYSTEMS

#	Subsystem Name	Child-Tags and Attributes	Default	
1	subsystem1	1. ip=197.168.58.92 2. conn_type=telnet	Yes	

[Add Another Subsystem](#)



To add another subsystem to the same system, you can click the “Add Another Subsystem” button.

System 2 Name * Default

[Create Tags](#) [Create Subsystems](#)

SUBSYSTEMS

#	Subsystem Name	Child-Tags and Attributes	Default	
1	subsystem1	1. ip=197.168.58.92 2. conn_type=telnet	Yes	

Subsystem 2 Name * Default

Tag 1 Name * Tag 1 Value * [Delete](#)

[Add Another Tag](#) [Save Subsystem](#) [Delete Subsystem](#)

[Add Another Subsystem](#)

[Save System](#) [Delete System](#)

3.3.2.11. Save System

System 2 Name * Default

[Create Tags](#) [Create Subsystems](#)

SUBSYSTEMS

#	Subsystem Name	Child-Tags and Attributes	Default	⊕
1	subsystem1	1. ip=197.168.58.92 2. conn_type=telnet	Yes	⊕
2	subsystem2	1. timeout=20 2. username=Jo 3. password=536ett		⊕

[Add Another Subsystem](#)

 [Save System](#) [Delete System](#)

A system with subsystems can also be saved by clicking the “Save System” button

SYSTEMS

SYSTEMS

#	System Name	System Child-Tags/Attributes	Subsystem Details	Default	⊕
1	system1	username = Jo password = 536ett url = http://httpbin.org/post timeout = 10		Yes	⊕
2	system2		Subsystem 1 Name = subsystem1 Default = Yes Subsystem 1 Child-Tags/Attributes 1. ip = 197.168.58.92 2. conn_type = telnet Subsystem 2 Name = subsystem2 Subsystem 2 Child-Tags/Attributes 1. timeout = 20 2. username = Jo 3. password = 536ett		⊕

3.3.3. Add Another System

Another system can be added to this datafile by clicking the “Add Another System” button.



3.3.4. Save Data File

To save the data file, you can click on the “Save” button.



3.3.5. Cancel Data file creation

To exit data file creation without saving it, click “Cancel”.

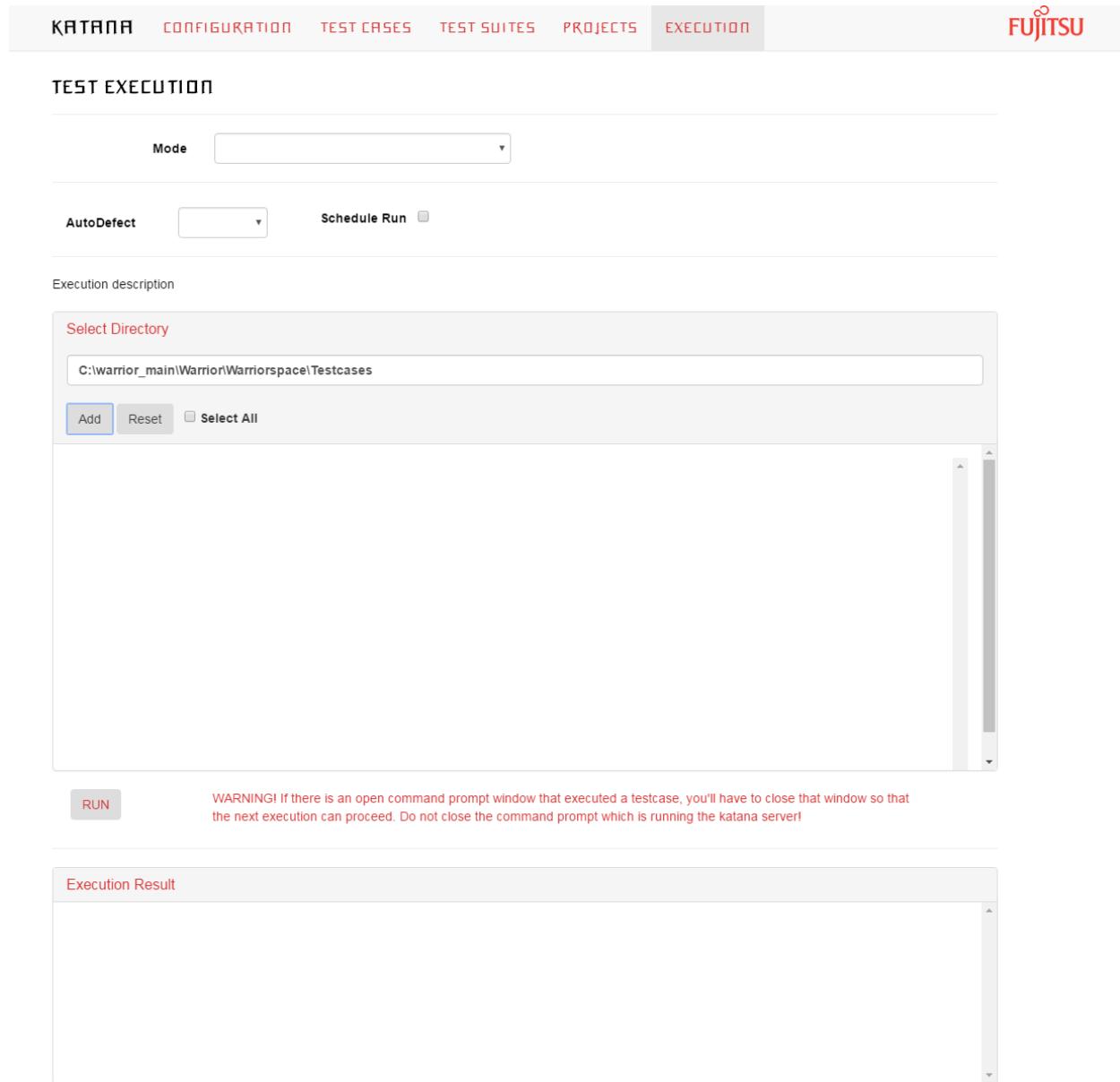


The final Data File thus created looks something like this:

```
<systems>
  <description>This is a sample Data file</description>
  <system default="yes" name="system1">
    <username>Jo</username>
    <password>536ett</password>
    <url>http://httpbin.org/post</url>
    <timeout>10</timeout>
  </system>
  <system name="system2">
    <subsystem default="yes" name="subsystem1">
      <ip>197.168.58.92</ip>
      <conn_type>telnet</conn_type>
    </subsystem>
    <subsystem name="subsystem2">
      <timeout>20</timeout>
      <username>Jo</username>
      <password>536ett</password>
    </subsystem>
  </system>
</systems>
```

7. EXECUTING WARRIOR FRAMEWORK THROUGH KATANA

Warrior Framework can also be executed through Katana. On clicking the ‘Execution’ tab, you will arrive at this page:



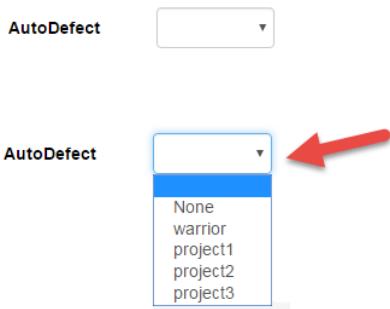
The screenshot shows the Katana Test Execution interface. At the top, there is a navigation bar with tabs: KATANA, CONFIGURATION, TEST CASES, TEST SUITES, PROJECTS, and EXECUTION. The EXECUTION tab is currently selected. In the center, there is a section titled "TEST EXECUTION". It includes a "Mode" dropdown menu, an "AutoDefect" dropdown menu, and a "Schedule Run" checkbox. Below these are sections for "Execution description" and "Select Directory". The "Select Directory" section contains a text input field with the path "C:\warrior_main\Warrior\Warriorspace\Testcases", an "Add" button, a "Reset" button, and a "Select All" checkbox. At the bottom left is a "RUN" button, and next to it is a warning message: "WARNING! If there is an open command prompt window that executed a testcase, you'll have to close that window so that the next execution can proceed. Do not close the command prompt which is running the katana server!". At the bottom right is an "Execution Result" panel.

You will be able to select the ‘Mode’ by clicking on the dropdown:

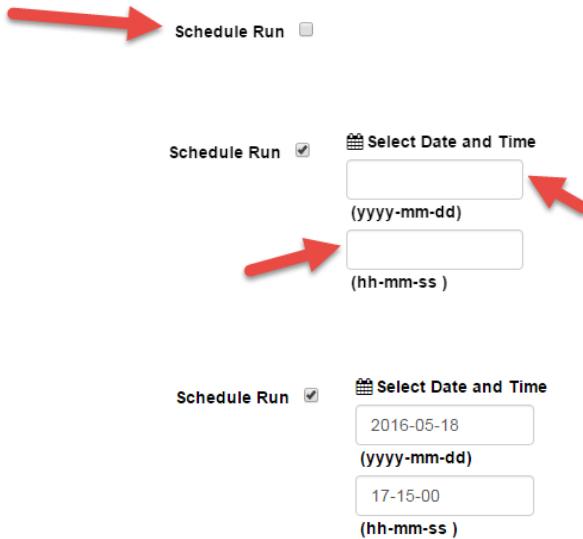




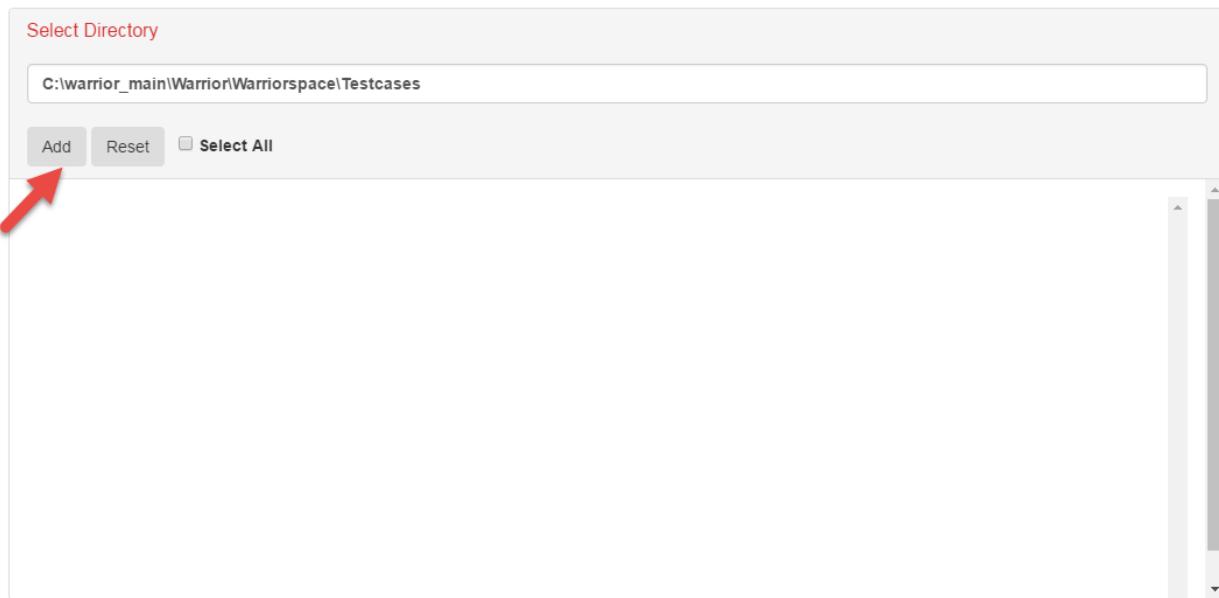
For JIRA auto defect creation, you can select your project from the 'AutoDefect' dropdown:



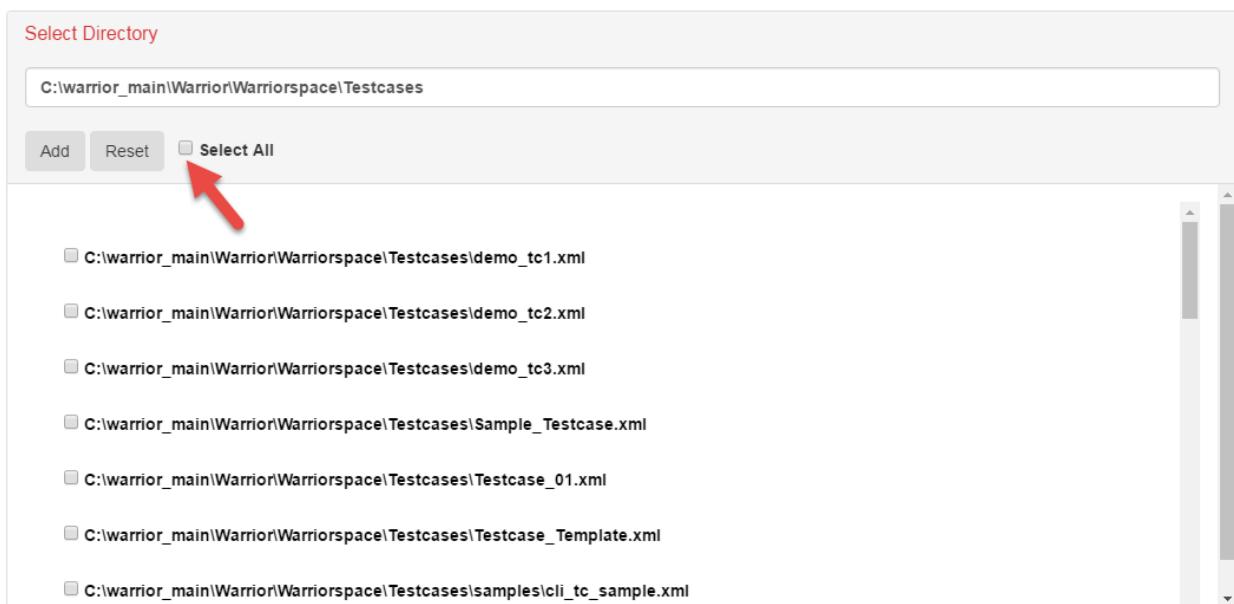
You can schedule the Case run by checking the 'Schedule Run' dropdown and then adding the date and time



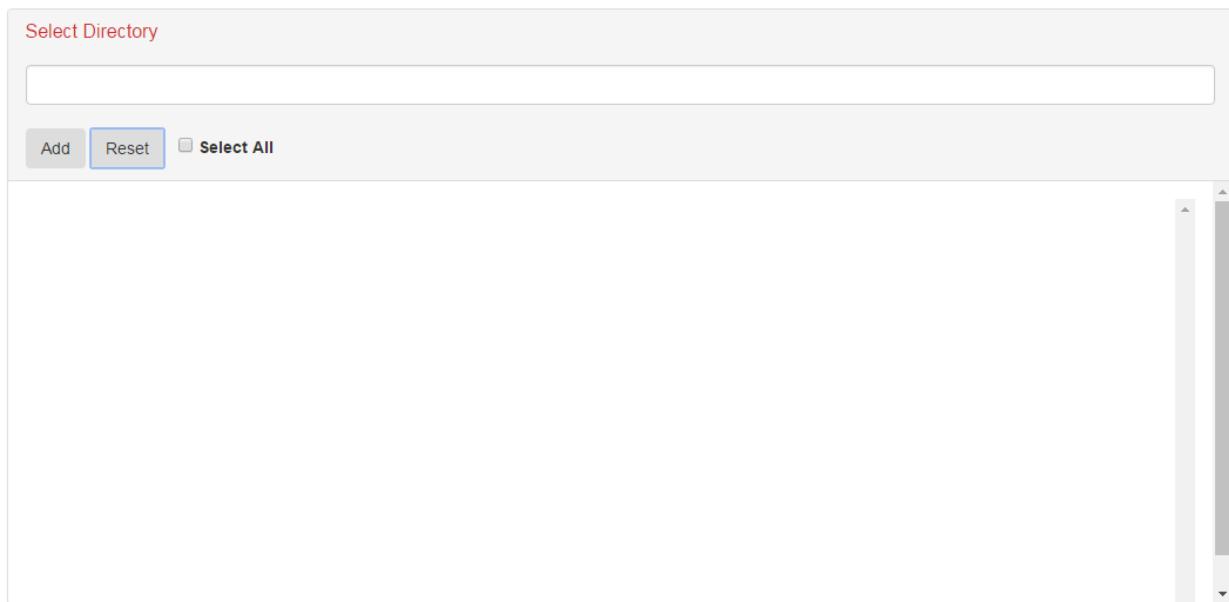
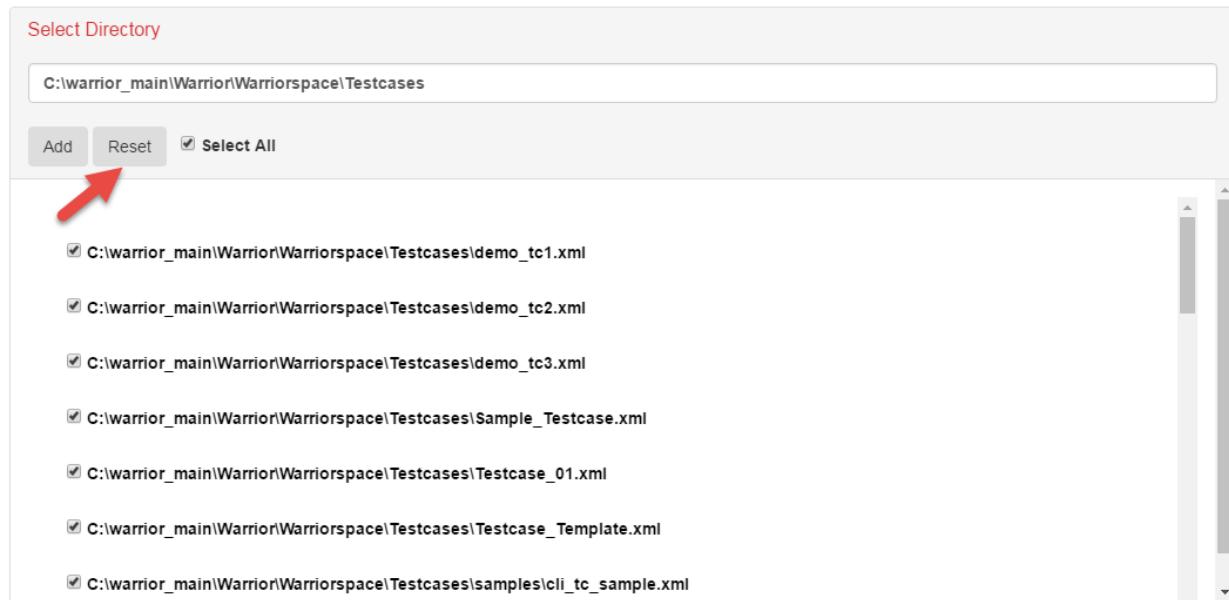
Then, You can click on add, to add all the cases available in the directory showing up in the input box beneath the 'Select Directory', You can also edit that directory if you want to run Cases from a different directory.



Checking 'Select All' lets you select all cases.



Clicking 'Reset', resets the entire page.



If you want to run a few, particular cases, you can check the checkboxes against those cases. The cases would run in the order in which to check the checkboxes.

Select Directory

C:\warrior_main\Warrior\Warriorspace\Testcases

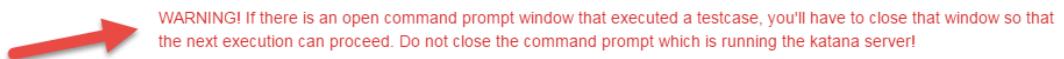
Add Reset Select All

- C:\warrior_main\Warrior\Warriorspace\Testcases\demo_tc1.xml
- C:\warrior_main\Warrior\Warriorspace\Testcases\demo_tc2.xml
- C:\warrior_main\Warrior\Warriorspace\Testcases\demo_tc3.xml
- C:\warrior_main\Warrior\Warriorspace\Testcases\Sample_Testcase.xml
- C:\warrior_main\Warrior\Warriorspace\Testcases\Testcase_01.xml
- C:\warrior_main\Warrior\Warriorspace\Testcases\Testcase_Template.xml
- C:\warrior_main\Warrior\Warriorspace\Testcases\samples\cli_tc_sample.xml

Clicking ‘Run’ would run the command in xterm – if you running Katana on Linux, or the Command Prompt – if you are running Katana on Windows.



This Warning appears only if Katana is being run on Windows



You would be able to see the Command that was sent to the xterm or Command Prompt here.

Execution Result

```
Command to Run: C:\warrior_main\Warrior -ad -jiraproj warrior -schedule 2016-05-18-17-15-00
C:\warrior_main\Warrior\Warriorspace\Testcases\demo_tc2.xml C:\warrior_main\Warrior\Warriorspace\Testcases\Sample_Testcase.xml
C:\warrior_main\Warrior\Warriorspace\Testcases\Testcase_01.xml

Execution Result: Command has been sent to Windows command prompt for execution
```

You would be able to view the execution on the console

WARHORN

USER GUIDE

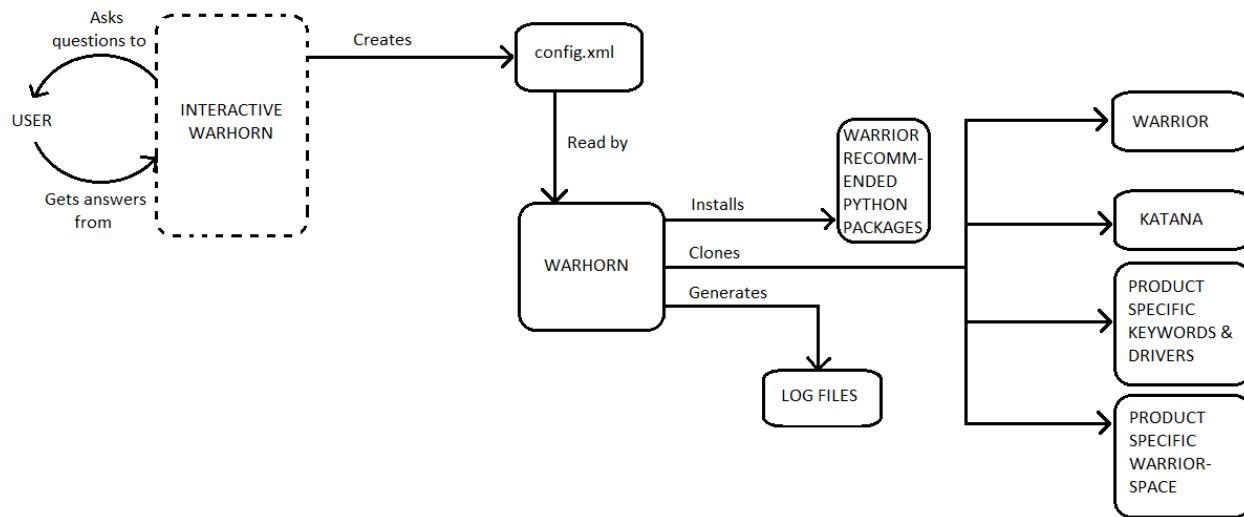
1. WHAT IS WARHORN?

Warhorn is Warrior Framework's installation tool. Warhorn has the capability to install the python packages that are recommended by Warrior Framework, Warrior Framework, Katana – another Warrior Framework tool, Product Specific Keywords and Drivers, and Warriorspace for you in your system.

With Warhorn, you can customize your Warrior Framework environment to suit your needs and specifications (This is explained in more detail in [Warhorn User Guide: Section 6](#).

2. HOW DOES WARHORN WORK?

Warhorn's premise is quite simple. Diagrammatically, it can be represented this way:



There are basically just three main components in Warhorn – the Interactive Warhorn, the Configuration file in XML format, and Warhorn.

The [Interactive Warhorn is an automated way to generate a configuration file](#). Warhorn can be run in the interactive mode by this command:

```
python warhorn.py -interactive
```

The interactive mode in the diagram above has been drawn with a dotted line – this indicates that it is not necessary to use the interactive mode to create the configuration file. You can create the configuration file (see [Warhorn User Guide: Section 6](#) for additional information) manually, in an editor - Gedit, Notepad++, or if you are feeling adventurous, vim - but the only caveat is that the configuration file has to be in a specific format as defined by Warhorn. Hence, it is recommended that you use the interactive mode.

The next component is the [configuration file](#) itself. It is essentially an [XML file that acts as a data store for Warhorn](#). All the information provided by you in the configuration file is used by Warrior Framework to install Warrior Framework recommended pre-requisites and repositories in your system.

The [third component is Warhorn](#) – it holds all the intelligence necessary for cloning and installing stuff on your system. Warhorn produces log files ([Warhorn User Guide: Section 8](#)) which stores information and details about what happened during the Warhorn execution.

3. PREREQUISITES

Warhorn requires the following to run successfully:

1. A Linux system
2. Python – version 2.7.0+ till the latest in the 2.7 family
3. Git

Warhorn by default would install the pre-requisites recommended by Warrior Framework. For this, warhorn may require you to have privileges to install packages on your system.

4. HOW TO INSTALL WARHORN?

The Warhorn Installation Tool is located at:

```
http://rtx-swtl-git.fnc.net.local/scm/wartools/warhorn.git
```

If you want to clone Warhorn into a particular location, type in:

```
cd path_to_the_desired_directory
```

Now that the system has git installed on it, open the terminal and type in to clone Warhorn:

```
git clone http://rtx-swtl-git.fnc.net.local/scm/wartools/warhorn.git
```

If you want to checkout a particular, branch, commit-id, or tag, type in:

```
cd warhorn
```

This command gets you into the newly cloned warhorn directory. Then type in:

```
git checkout branch_name/tag_name/commit_id
```

5. THE DIRECTORY STRUCTURE

The current Warhorn directory structure looks something like this:

warhorn			
Name	Date modified	Type	Size
docs	6/7/2016 1:52 PM	File folder	
source	6/7/2016 1:52 PM	File folder	
user_generated	6/7/2016 1:52 PM	File folder	
	6/7/2016 1:52 PM	Text Document	1 KB
default_config	6/7/2016 1:52 PM	XML Document	6 KB
readme	6/7/2016 1:52 PM	Text Document	4 KB
warhorn	6/7/2016 1:52 PM	Python File	51 KB

There are three folders and four files inside. The first is the Docs directory which stores the Warhorn User Guide.

docs			
Name	Date modified	Type	Size
Warhorn User Guide	6/7/2016 1:52 PM	Adobe Acrobat D...	503 KB

The second directory – source – contains all the files that Warhorn uses to work. You should not change any contents of this directory.

Then comes the user_generated directory. This is the default directory for storing files created via the interactive mode.

user_generated			
Name	Date modified	Type	Size
temp	6/7/2016 1:52 PM	Text Document	1 KB

The four files in the directory are – the .gitignore, which is just a file specifies intentionally untracked files that Git should ignore. The readme.txt – this file details the brief introduction to Warhorn, the default_config.xml which you are encouraged to open, read through, and edit, and finally the warhorn.py is the Warhorn executable.

6. HOW TO CONFIGURE THE DEFAULT CONFIGURATION FILE?

The configuration files are an integral part of running the Warhorn tool. Warhorn, when freshly cloned, comes with a default_config.xml file with it. The configuration file can be created using Katana or via an xml editor.

This configuration file feeds data to warhorn.py and tells it what to install and clone on your machine. The default_config.xml can be edited but it is strongly recommended that it not be moved from its original location and that its name not be changed as then, the command for running Warhorn would change ([Warhorn User Guide: Section 7](#))

Any configuration file, including the default_config.xml file, should contain these five tags - <warhorn>, <Warrior Framework>, <katana>, <drivers>, and <Warrior Frameworkspace>. Out of these tags, only the <Warrior Framework> tag is the mandatory one.

6.1 THE <WARHORN> TAG

The <warhorn> tag is the first tag in the configuration file.

```
▼<warhorn name="Warhorn">
```

This tag contains information about the pre-requisites and their version that Warhorn would install by default. The comment below notes all that information

```
▼<!--
    The required versions for each of the dependencies given below are:
        jira: 1.0.3 - Not needed for Warrior version 2.1.1 and above
        lxml: 3.5
        ncclient: 0.4.6 - Not needed for Warrior version 1.9 and above
        paramiko: 1.16.0
        pexpect: 3.1
        pysnmp: 4.3.1
        requests: 2.9.1
        selenium: 2.48.0
    Your system may or may not have the correct version installed.
    ** Set the 'install' attribute to yes if you want to install the
    dependency in your system.
    ** Set the 'correct_version' attribute to yes if you want to upgrade the
    existing package to the required version
-->
```

Since, Warhorn by default, clones the latest version of Warrior Framework, unless specified otherwise, Jira and ncclient, have been turned “off” in the default_config.xml. So pre-requisites section in the default_config.xml looks something like:

```

▼<!--
    Necessary for Jira module to log defects to Jira automatically
-->
<dependency name="jira" install="no"/>
<!-- Used by IronClaw tool -->
<dependency name="lxml" install="yes"/>
<!-- Used for netconf operations -->
<dependency name="ncclient" install="no"/>
<!-- Used by ncclient -->
<dependency name="paramiko" install="yes"/>
<!-- Used by cli utilities -->
<dependency name="pexpect" install="yes"/>
<!-- Used for snmp operations -->
<dependency name="pysnmp" install="yes"/>
<!-- Used for rest operations -->
<dependency name="requests" install="yes"/>
<!-- Used for web based testing -->
<dependency name="selenium" install="yes"/>

```

Here all the pre-requisites recommended by Warrior Framework have their own <dependency></dependency> tags. The pre-requisites name is given as value to the name attribute and whether you want to install that pre-requisite— ‘yes’ or ‘no’ is given as value to the install attribute. The pre-requisite corresponding to the install attribute marked as ‘yes’ or ‘no’ will only get installed if the install attribute is set to ‘yes’.

The </warhorn> closing tag marks the end of this section

```
</warhorn>
```

6.2 THE <WARRIOR FRAMEWORK> TAG

This tag carries with it information regarding the cloning of Warrior Framework.

```

<warrior url="http://rtx-swtl-git.fnc.net.local/scm/war/warrior_main.git" destination=""
          label="" clean_install_warrior="">
</warrior>

```

This is a mandatory tag. So, please make sure every configuration file contains this tag and that it has been filled out with the correct information.

The attributes in the <Warrior Framework></Warrior Framework> tag:

Attribute	Description
-----------	-------------

url	Holds the URL of the Warrior Framework repository. The url tag has already been prepopulated in the default_config.xml. You can change that if you want to, but make sure that it is a valid url.
destination	Contains the path to the directory in which you want to clone Warrior Framework. If it is left empty, then Warrior Framework would be cloned in the same directory as Warhorn.
label	Indicates the branch name, tag name, or commit-id that you may want to checkout. If it is left empty, then the latest version of Warrior Framework will be cloned.
clean_install	This tag lets you indicate whether you want to delete the existing Warrior Framework in your system or not. If the tag value is set to 'yes', the existing Warrior Framework will get deleted. If the tag is set to 'no', or is left empty, or is taken out altogether, existing Warrior Framework will not get deleted.

6.3 THE <KATANA> TAG

This tag carries with it information regarding the cloning of the Warrior Framework Tool - Katana.

```
<katana url="http://rtx-swtl-git.fnc.net.local/scm/war/katana.git"
         destination="" label="" clean_install="" clone="yes"></katana>
```

The attributes in the <katana></katana> tag:

Attribute	Description
url	Holds the URL of the Katana repository. The url tag has already been prepopulated in the default_config.xml. You can change that if you want to, but make sure that it is a valid url.
destination	Contains the path to the directory in which you want to clone Katana. If it is left empty, then Katana would be cloned in the same directory as Warhorn.
label	Indicates the branch name, tag name, or commit-id that you may want to checkout. If it is left empty, then the latest version of Katana will be cloned.
clean_install	This tag lets you indicate whether you want to delete the existing Katana in your system or not before cloning a fresh one. If the tag value is set to 'yes', the existing Katana will get deleted. If the tag is set to 'no', or is left empty, or is taken out altogether, existing Katana will not get deleted.

clone Katana would be cloned only if this ta is set to “yes”. If it is left empty, or set to “no”, Katana will not be cloned.

6.4 THE <DRIVERS> TAG

Warhorn provides you with the ability to clone entire repositories containing all drivers and actions packages or you can select only the drivers that you need and the corresponding actions packages with it will be cloned for you.

This is the format for adding specific drivers that you want to clone from a particular Keyword repository:

```
<repository url="http://repository/one/url.git" clone="yes" label=""  
all_drivers="no">  
  <driver name="driver_one_name" clone="yes"></driver>  
    <driver name="driver_two_name" clone=""></driver>  
    <driver name="driver_three_name" clone="no"></driver>  
    <driver name="driver_four_name"></driver>  
</repository>
```

This is the format for cloning the entire Keyword repository:

```
<repository url="http://repository/two/url.git" clone="yes"  
label="feature/xyz" all_drivers="yes">  
</repository>
```

Attributes in a drivers tag:

Attribute	Description
url	Holds the URL of the repository that you want to clone.
all_drivers	This attribute lets you clone all the drivers and all the actions packages when set to yes. On the chance that you do not want to clone all the drivers, but only want to clone specific drivers, the following steps need to be taken: <ol style="list-style-type: none">1. Set the all_drivers attribute to ‘no’2. Create a driver tag under the repository tag as mentioned in the sample3. Fill out the attribute of that driver tag. The driver tag has one attribute – name that takes in the name of the driver that you want and another attribute “clone” that lets you turn “off” a driver cloning.

4. Repeat steps 2 and 3 till all the needed driver tags are created.

label	Indicates the branch name, tag name, or commit-id that you may want to checkout.
clone	Gives you the ability to “turn off” the cloning of that particular repository. If this attribute is set explicitly to ‘no’, only then the repository would not be cloned. If it is set to ‘yes’, or is left blank, or is removed altogether, the repository will get cloned.

You can add as many repository tags as you want and Warhorn would clone all of them for you as long as all of them carry valid information. If you do not want to clone any Keyword repositories, the main drivers tag can be left empty.

6.5 THE <WARRIOR FRAMEWORKSPACE> TAG

The Warriorspacetag holds information regarding the product specific Warriorspacerepositories that you may want to clone. The process is similar to the drivers tag - you can add as many repository tags as you want as long as all of them carry valid information. The sample below represents the way in which the Warriorspacetag should be filled out.

```
<warriorspace>
  <!-- Sample
  <repository url="http://warriorspace/repository/url.git" label="f455scd00i">
    </repository>
  -->
</warriorspace>
```

Attribute	Description
url	Holds the URL of the repository that you want to clone.
label	Indicates the branch name, tag name, or commit-id that you may want to checkout.

7. HOW TO RUN WARHORN?

Warhorn can be run by going the command line and running the warhorn.py file using the command:

```
python warhorn.py
```

This command supplies no arguments and therefore, Warhorn uses the default_config.xml to get the data needed to perform its tasks. It is not recommended that default_config.xml be moved from its default location and the name of the file be changed to anything else, since, the command above will not work if that is done. Warhorn would still run but only if the location of the xml file is passed as an argument.

If you need to run a particular xml file, the command should be:

```
python warhorn.py path_to_directory/file_name.xml
```

This command lets Warhorn use a particular configuration file to get its data from.

The Warhorn Interactive mode can create the configuration file in the correct XML format. Running the following command creates that XML file, and gives you the option of saving and then running that newly created file. You can also directly run the configuration file without saving it. To enter the Warhorn Interactive mode, type in this command:

```
python warhorn.py -interactive
```

8. THE LOG FILES

Log files are generated each time warhorn.py is run. Log files are not generated when the Warhorn Interactive mode is used to generate the configuration file only.

There are two kinds of log files: console_log.txt and print_log.txt. The console_log.txt logs all the console output of warhorn.py, including the print statements and errors. The print_log.txt logs just the print statements.

Both these files are stored in a time stamped directory underneath the logs folder - which gets generated when warhorn.py is run for the first time – along with the configuration file that was used by warhorn.py during that particular run.