

ZUG User Manual		Page 1/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

ZUG User Manual (Version V6.2)

Whilst all reasonable care has been taken to ensure that the details are true and not misleading at the time of publication, no liability whatsoever is assumed by Automature LLC, or any supplier of Automature LLC, with respect to the accuracy or any use of the information provided herein.

Any license, delivery and support of software require entering into separate agreements with Automature LLC.

This document may contain confidential information and may not be modified or reproduced, in whole or in part, or transmitted in any form to any third party, without the written approval from Automature LLC.

Copyright © 2012 Automature LLC

All rights reserved.

ZUG User Manual		Page 2/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

Revision History

Version	Date	Name	Description of Changes
1.0	2010-09-23	Nitish Rawat	First Edition
1.1	2010-11-04	Nitish Rawat	Revised Edition with illustrative snapshot and new chapters
1.2	2010-11-09	Nitish Rawat	Changes in section ordering
2.2.2	2011-08-30	Sankha Sil	Changes in Atom path, new Option and Features
2.3	2011-09-16	Sankha Sil	Changes in Relative Path and Command Line Macro Value Passing
3.0	2012-01-09	Sankha Sil	New Feature added. Zug uses methods which are dynamically loaded
3.2	2012-03-25	Sankha Sil	New Feature added Zug have Configuration Management for MVM
4.0	2012-03-29	Sankha Sil	New Feature added Zug Reports Results using Davos
4.3	12-06-12	Dipayan Sengupta	New Features : BuildTag and TestPlan added
4.4	20-06-12	Dipayan Sengupta	New Feature :TopologySet
5.6	02-01-13	Md Sarfaraz Khan	Added a trouble shooting , JRE dependence
5.7	22-01-13	Sankha Sil	Added a new switch logfilename.
6.2	28-03-13	Md Sarfaraz Khan	Added new module Reporting in testlink
6.3	20-05-13	Md Sarfaraz Khan	Added command line options (-ignore,-atomexectime) and DB and script location configurations

ZUG User Manual		Page 3/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

Contents

Table of Contents

ZUG User Manual

(Version V6.2)	1
Revision History	2
Contents	3
1. Introduction	5
1.1 Document Purpose.....	5
1.2 Intended Audience.....	5
2. Concepts and Terminologies	6
2.1 Test Suite.....	6
2.2 Test Case.....	6
2.3 Atom.....	6
2.4 Molecule.....	6
2.5 Action.....	6
2.6 Test Plan.....	6
2.7 Test Step.....	6
2.8 Verification Step.....	7
2.9 MVM Configuration.....	7
3. ZUG Options	8
3.1 -TestCaseID=Test001,.....	8
3.2 -Repeat -NoRepeat.....	8
3.3 -Autorecover -NoAutorecover.....	8
3.4 -Verbose NoVerbose.....	8
3.5 -Debug NoDebug.....	9
3.6 -Verify -NoVerify.....	9
3.7 -AtomPath=<location>.....	9
3.8 -Include=<location>.....	9
3.9 -Execute -NoExecute.....	9
3.10 -\$<macroname>=<value>.....	9
3.11 -TestCycleID=<integer>.....	10
3.12 -TestPlan=<Product:Release:Sprint:TestPlan>.....	10
3.13 -TestPlanID=<integer>.....	10
3.14 -TopologySetID=<integer>.....	10
3.15 -TopologySet=<AlphaNumeric>.....	10
3.16 -BuildTag=<AlphaNumeric>.....	10
3.17 -LogFileName=<AlphaNumeric>.....	10
3.18 -macrocolumn=<file identifier:column value>.....	11
3.19 -macrofile=<file location>.....	11
3.20 -ignore.....	11
3.21 -atomexectime.....	11
4 ZUG Modes	12
4.1 Production Mode.....	12
4.2 Developer Mode.....	12
4.3 Performance Mode.....	12
5 Process Flow	14
5.1 Architecture overview.....	14
6 Running your first automated test	15

ZUG User Manual		Page 4/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

6.1 Preparing Zermatt for Zug.....	15
6.2 Managing Atoms.....	15
6.3 Preparing Zug for Zermatt.....	16
6.4 Executing the automated test.....	17
6.5 Viewing results in Zermatt.....	22
6.6 Interpreting the results.....	22
6.7 Archival the Log files.....	25
6.8 Reporting to TestLink.....	25
6.8.1 Basic Terminologies of TestLink.....	25
6.8.2 Preparing ZUG for TestLink.....	26
6.8.3 Preparing TestLink For ZUG:.....	27
6.8.4 Executing the automated tests.....	30
7. Implicit Molecule Calls.....	32
7.1 ZCase_Verify.....	32
7.2 ZStep_Verify.....	32
7.3 ZEnv_Sensor.....	32
8. Troubleshooting.....	34
8.1 Debug Log.....	34
8.2 Primitive log.....	34
8.3 Database connection problem.....	34
8.4 Unable to Archive Log Files.....	35
8.5 License Expired.....	35
9 ZugINI.xml	37
9.1 Inprocess atoms configuration.....	37
9.2 DB Configuration and Script Locations.....	38

ZUG User Manual		Page 5/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

1. Introduction

Zug is Automature's software test automation tool. Zug provides an execution environment for tests specified using a high level test specification language namely CHUR. Zug is platform independent, and can be used on any environment that supports the Java Runtime Environment.

1.1 Document Purpose

The user manual explains you operation of Zug options. You will be able to use several options for customizing the execution of test cases.

1.2 Intended Audience

The user manual is intended for users who want to learn or run execution of test suites. Test suites are written in Chur. The test suite file is given as input to Zug.

ZUG User Manual		Page 6/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

2. Concepts and Terminologies

2.1 Test Suite

As explained before Test Cases are individual programs designed to test a feature of a product. As there are usually hundreds or thousands of features in a single application, therefore the number of Test Cases would be equally high. To manage so many of them, Test Cases are grouped into *Test Suites*.

2.2 Test Case

A Test Case is a program whose purpose is to test a certain feature of a product. It tells the developer (or whoever else is trying to make the product work) whether the feature is working properly by verifying the results and if not, help them identify the cause of the malfunction.

The Automature Framework has a hierarchical structure in which *Test Cases* are at the core of it.

2.3 Atom

Atoms are the smallest unit of action in Chur. They are entities, such as programs or scripts that can be executed at the command line level in a shell (e.g. the Command Prompt in Windows). Atoms can be invoked on the Test Cases or Molecules worksheets in the Test Suite spread sheet on a single line. An example of an atom can be a program that enters text into a form field inside a web page, or can simulate a button click.

2.4 Molecule

Molecules are a collection of atoms in a sequence, with the added ability to express more complex logic. Molecules may call atoms directly, or through other nested molecules. Test Cases, themselves could be considered Molecules themselves, except that no other test case or molecule can call them. Example of a molecule can be to simulate a user login, by using the atom examples above.

2.5 Action

A test case is a sequence of one or more Actions. Each Action may take as many arguments as necessary. An Action atom is expected to return an exit status code that implicitly tells Zug if the action was successful. By convention a non-zero status is interpreted as a failure. When Zug encounters a failure status it automatically invokes the appropriate clean up steps.

2.6 Test Plan

Test plan is the comprehensive planning of how features can be tested with the respective test cases in certain topology sets. Each test plan may contain one or more test cycles which may be executed during the course of a certain phase of the product.

2.7 Test Step

A Test Step should be specified for each Action of a test case or a Molecule but not for a Verification Step. It should be monotonically increasing number for each test case action. If two steps of a test case have the same steps, then the two steps are executable concurrently. Some of these steps can be considered as initialization steps, few others the action steps and the rest as cleanup steps.

ZUG User Manual		Page 7/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

2.8 Verification Step

Each Action of a Test Case may consist of none, or several Verification methods. And each Verification method may take several arguments as needed. A verification atom is expected to return an exit status code that implicitly tells Zug if the verification was successful. By convention, a non-zero status code is interpreted as a failure. When Zug encounters a failure status in verification, it automatically invokes the corresponding clean up steps.

2.9 MVM Configuration

While running any testcase if it contains more than a certain level of multivalued macro it may fail due to java jvm max memory which is not sufficient for keeping that many Cartesian producted testcases. Due to low machine configuration it fails but it can run in high configuration machines. As so from Zug3.2 version this configuration dependency is introduced in ZugINI.xml as <configurations> tag. Under that tag there is another type tag <mvm-configurationjvm-max-memorysize="853">. This name attribute of this tag mainly identifies the machine JVM memory configuration, which helps to find out what is the JVM max memory it can avail. Under the tags the mvm cardinality is defined which have the number of cardinality of testcases. Cardinality: the total testcases generated by Zug doing Cartesian /indexed expansion over the number of MVMs in the testcase.

Below is a example of a configurations tags in ZugINI.xml.

```
<configurations>
  <mvm-configuration jvm-max-memorysize="853">4000</mvm-configuration>
  <mvm-configuration jvm-max-memorysize="455">3000</mvm-configuration>
  <mvm-configuration jvm-max-memorysize="247">2500</mvm-configuration>
  <mvm-configuration jvm-max-memorysize="122">1500</mvm-configuration>
</configurations>
```

ZUG User Manual		Page 8/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

3. ZUG Options

Zug has a set of standard options that are supported in the execution test environment. The syntax the launching Zug is as follows:

RunZUG.bat [options] FILE.xls

3.1 -TestCaseID=Test001,...

If -TestCaseID is specified, Zug will execute only specific test cases as listed and will ignore the rest. The value to be specified is comma separated list of Automated Test Case Ids. This option is not required. By default all test cases specified in the input file are executed.

3.2 -Repeat | -NoRepeat

The -Repeat option allows the user to run a selection of test cases repeatedly, for a specified duration, or a specified number of iterations. The Longevity tests run the test plan setup and cleanup only once, and record a single result as the outcome.

Note: Zug debug logs are turned OFF during LONGEVITY MODE to ensure Zug does not consume too much disk space. By default, -NoRepeat is selected.

-Count=*integer*

This option specifies number of times the test cases mentioned in the test plan will be executed in iteration. This should be a number.

Example: -Repeat -Count=5

-Duration=*time*

This option specifies how long the test cases mentioned in the test suite will be iterated through.

The *time* value has to specified in

$\frac{35}{17}$ days]d

$\frac{35}{17}$ hours]h

$\frac{35}{17}$ minutes]m

$\frac{35}{17}$ seconds]s

Example: -Repeat -Duration=3d

Note: -Duration and -Count are mutually exclusive. If both of them are specified on the command prompt, then -Count takes precedence over -Duration.

3.3 -Autorecover | -NoAutorecover

The option -Autorecover specifies Zug to run cleanup during test plan/test step timeout or failure. By default, -Autorecover will be selected unless specified otherwise. If -NoAutorecover is specified, then there are no cleanup steps to execute.

3.4 -Verbose | NoVerbose

The option -Verbose is used to display debug messages on the output console. By default -NoVerbose will be selected unless explicitly mentioned. The option -NoVerbose does not display any debug messages on the output console.

ZUG User Manual		Page 9/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

3.5 -Debug | NoDebug

The -Debug option is specified to run the Automation in Debug Mode. In this case if any atom is not implemented then the Zug will prompt with a default atom. By default -NoDebug will be selected unless explicitly disabled.

3.6 -Verify | -NoVerify

The option -Verify specifies Zug to execute the testcase without verification. By default, -Verify is selected means ZUG will run verification actions for each testcase unless explicitly -NoVerify is specified.

3.7 -AtomPath=<location>

This option specifies the location from where ZUG will pick up atoms for Test Automation/Execution. This should be a fully qualified location and *should not be a relative path*. We can give multiple locations by ; separator.

Example: -Atompath=C:\Tests\Atoms

3.8 -Include=<location>

This option specifies the location from where ZUG will pick up molecules and macros for Test Automation/Execution. The location can be a file name residing in the same directory of that of the test suite or fully qualified location of the file but it should not be a relative path. We can give multiple locations by comma separator.

Example: -include=C:\Tests\Molecules

One more powerful feature of this option is to provide name spaces to the test suite included from command line. The syntax for giving name space in the -include option is

-include=namespace1#filename1,namespace2#filename2

In the test suite when we are using the macros and molecules of the included test suite then we should append the name space. For example if we are using a macro say \$test of filename1 then in the main test suite we should write \$namespace1.test.

3.9 -Execute | -NoExecute

-NoExecute mode will verify if the Test Design Excel sheet prepared by the user is syntactically correct or not. By default -Execute mode is selected unless explicitly specified. If -NoExecute is selected, Zug will just validate the test suite design sheet, display a list of errors, and exit without executing any test cases in the test suite.

3.10 -\$<macroname>=<value>

-\$<macroname>={<value1>,<value2>}

This option specifies the macro value of the macro in the chur sheet. If the macro name is not written in the macro sheet of the chur spread sheet then it will add one while it is passed in the command line. The macro name should not contain any blank space and the macro value should not contain any blanks also. The macro value should be normally passed without giving any ""(quotes). Multi valued macro is also can be passed by providing the braces({}). It is also possible in providing of macro value as {1..2} for extended macro values.

ZUG User Manual		Page 10/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

3.11 -TestCycleID=<integer>

Use Zermatt to look for TestCycleID. If it is not provided then Zug will generate a new ID and update the results under it. By default, Zug always generates a new testcycle in ZERMATT unless explicitly specified to use an existing TestCycleID. TestCycleID can be found in ZERMATT page navigating to **Test Cycle Report**.

Example: -TestCycleID=72



3.12 -TestPlan=<Product:Release:Sprint:TestPlan>

This is required to upload test case execution results in Zermatt under a particular Test Plan. Instead of using the testplanid, you can also report to Zermatt by naming the testplan in the following way:

Example: -TestPlan="ZUG:First Release:rc7 sprint:Smoke test plan"

3.13 -TestPlanID=<integer>

This is required to upload test case execution results in Zermatt under a particular Test Plan. Instead of using the testplan, you can also report to Zermatt by putting the testplanid in the following way:

Example: -TestPlanID=84

3.14 -TopologySetID=<integer>

This is required by Zug to register results in a testcycle for the specified Topology Set. The Topology Set has to be written in Zermatt and its id is to be specified. The Topology Set Id can be seen in the Zermatt page as shown below in the status bar of **List all Topology Sets for a chosen Test Plan:**

Example: -TopologySetId=26



3.15 -TopologySet=<AlphaNumeric>

Instead of using the topologysetid, Zug can register results in a testcycle by specifying the name of the topologyset. The topologyset set associated with the test cycle should be stated while running zug.

Example: -topologyset="First TopologySet"

3.16 -BuildTag=<AlphaNumeric>

This is required by Zug to upload the name of the Build in Zermatt for a particular sprint. You can report to Zermatt by naming the BuildTag in the following way:

Example: -BuildTag=Build074

3.17 -LogFileName=<AlphaNumeric>

This is required by Zug to change the logfile names where zug logs the execution messages(error,debug). The logfile name will be created as <logfile name>-Atom.log , <logfile name>-Debug.log.

3.18 -macrocolumn=<file identifier:column value>

We can have multiple value column in a test suite for a macro. This option let us select a particular macro value column for a test suite. The syntax for -macrocolumn is

-macrocolumn=file identifier:column number, file identifier:column number

The file identifier is the file name,default name space or the optional name space provided in the -include option. If we have provided any optional name space in the -include option then it should be the file identifier in the -macrocolumn. If any cell of the selected column is empty then it takes the value of the default column cell(1st column i.e, the "value" column).

3.19 -macrofile=<file location>

This option is used to include a macro text file from command line. The text file contains macro in the format \$macro=value. Each macro should begin in a new line. At run time ZUG treats these macros as it belongs to the main test suite. The syntax is -macrofile=filelocation .

Example: - macrofile=D:\Files\macroFile.txt

3.20 -ignore

This command line feature act as an overview mode. As the name suggests ZUG ignores all the errors that occur during the execution of test steps. If a test step of a Test Case or of a Molecule fails ZUG shows the error message but does not stop the execution and all the test steps of the test cases and that of the Molecules are executed.

3.21 -atomexectime

This command line feature is used to measure the performance of the atoms that are invoked from the test suite. After executing each atom ZUG appends the time taken to execute the atom in the end of the success message of the atom and at the end of the execution ZUG reports the min,avg, max execution time of the 10 most time consuming atoms in milli seconds. Given below is a sample output format for this command line switch.

[TC001] Execution Started Action Zbrowser.GoToURL with values [148283,http://www.google.com]
 [TC001] Action ZBROWSER.GOTOURL SUCCESSFULLY Executed in 3023 milli sec.

```

*****
Total time taken to execute all the test cases <End to End> is -> 185266 milli Seconds.
*****
10 Most time consuming Atom in the Test Suite along with thier execution time in milli seconds
Atom Name                               Min.   Avg.   Max.
zbrowser.gotourl                         4020   4020.00  4020
zbrowser.browsetopagebyurl               1907   3711.39  7026
zbrowser.clickbuttonbyvalue              3113   3113.00  3113
zbrowser.initialize                      2523   2523.00  2523
zbrowser.clicklinkintablebyrowcolumnandlinkindex  523   1641.10  2991
zbrowser.clickimagebytitle               1484   1590.50  1697
zbrowser.clickbuttonbytext                519   1392.00  2265
zbrowser.gettablecolumnrbycolumnname     633   666.80   731
zbrowser.gettablerownrbycolumnnameandtext 628   653.00   696
zbrowser.getrowcountintable              630   649.67   672
*****

Exiting ZUG
-----
Automation Finished.

```

ZUG User Manual		Page 12/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

4 ZUG Modes

4.1 Production Mode

In production mode the tests are run and result are registered in Zermatt. This is done for unattended testing.

The following options are primarily used to design in this mode:

- ³⁵/₁₇ -NoRepeat
- ³⁵/₁₇ -AutoRecover
- ³⁵/₁₇ -NoVerbose
- ³⁵/₁₇ -Verify
- ³⁵/₁₇ -Execute
- ³⁵/₁₇ -TestCycleID=[integer]
- ³⁵/₁₇ -TestPlanID=[integer]
- ³⁵/₁₇ -TestPlan=[String:String]
- ³⁵/₁₇ -TopologySetID=[integer]
- ³⁵/₁₇ -TopologySet=[String]
- ³⁵/₁₇ -BuildTag=[String]

The above options are discussed in detailed in **Chapter 3:** . ZUG Options.

4.2 Developer Mode

In this mode Test Automation Developer can debug the automated scripts

The following options are primarily used to design in this mode:

- ³⁵/₁₇ -Verbose
- ³⁵/₁₇ -Debug

The above options are discussed in detailed in **Chapter 3:** . ZUG Options.

4.3 Performance Mode

This mode is done to stress and measure performance parameters, values and attributes.

The following options are primarily used to design in this mode:

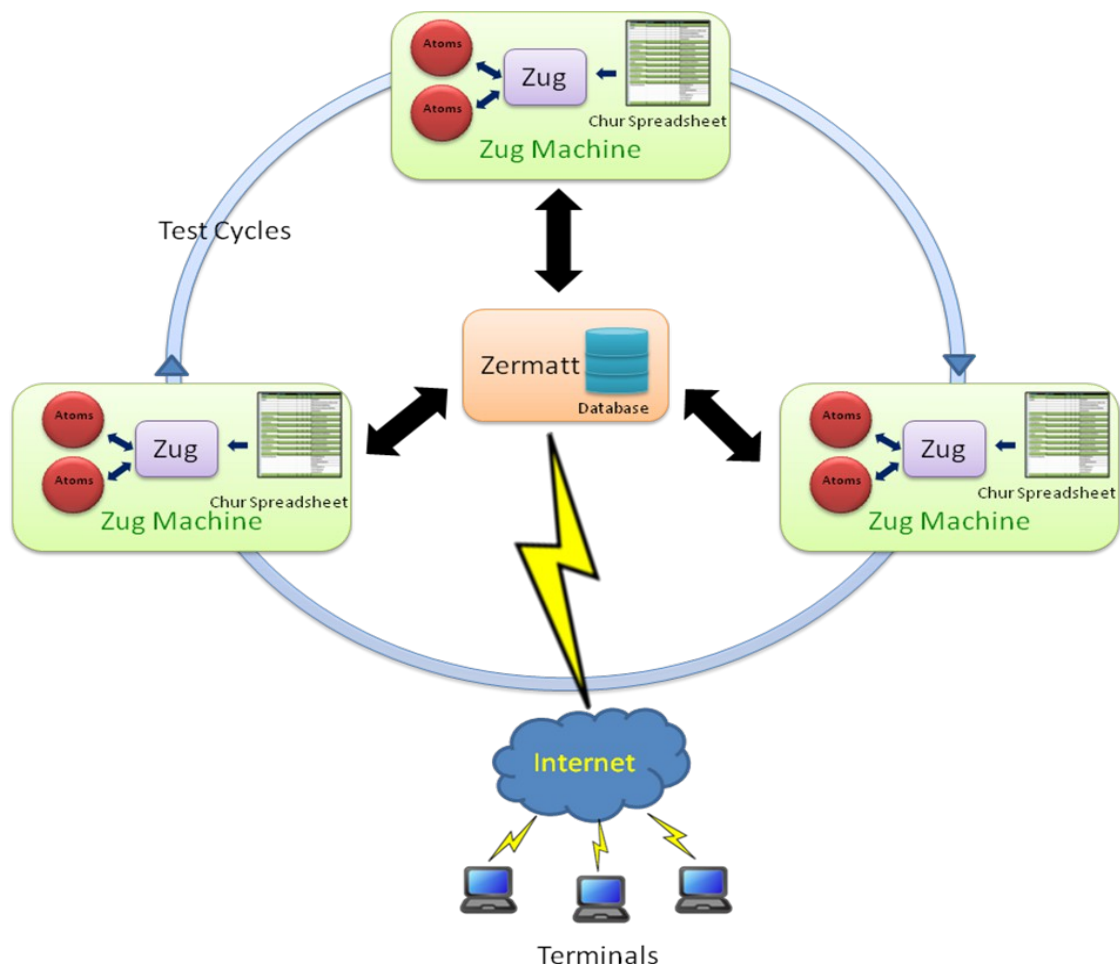
- ³⁵/₁₇ -Repeat
- ³⁵/₁₇ -NoVerify
- ³⁵/₁₇ -Execute
- ³⁵/₁₇ -TestPlanID=[integer]
- ³⁵/₁₇ -TopologySetID=[integer]

The above options are discussed in detailed in **Chapter 3:** . ZUG Options.

ZUG User Manual		Page 13/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

5 Process Flow

5.1 Architecture overview



The atoms written in any scripting language are executed in Zug platform with the help of Chur spread sheet in different Zug machines. After execution the results are stored in the data repository of Zermatt. The end-users can access the results in Zermatt through the internet at real time.

ZUG User Manual		Page 15/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

6 Running your first automated test

In this manual we are using an example for illustration where we open a document using an office application and close it automatically.

(If you are not using Zermatt, then kindly ignore the following sections 6.1)

6.1 Preparing Zermatt for Zug

³⁵₁₇ Once the test plan is created in Zermatt, it can be used to store automated test results. The testplan_id of the test plan is one of the inputs required by Zug to file the results. The testplan_id can be found in Zermatt.

³⁵₁₇ The topology sets are also needed to be written in Zermatt for integration with Zug. The topology sets has to be added to the list of topology sets of the test plan. The topology set written should have at least one topology. The topologysset_id is found in the status bar of **List all Topology Sets for a chosen Test Plan**.

³⁵₁₇ We can see the participating topologies of the topology set in the Zermatt page **List All Topology Sets**:

³⁵₁₇ The role of any one topology as shown in the above figure should match with the role stated in the Chur spreadsheet that is to be written later.

(For deeper understanding on CHUR, please refer to the Language Reference Manual of Chur)

6.2 Managing Atoms

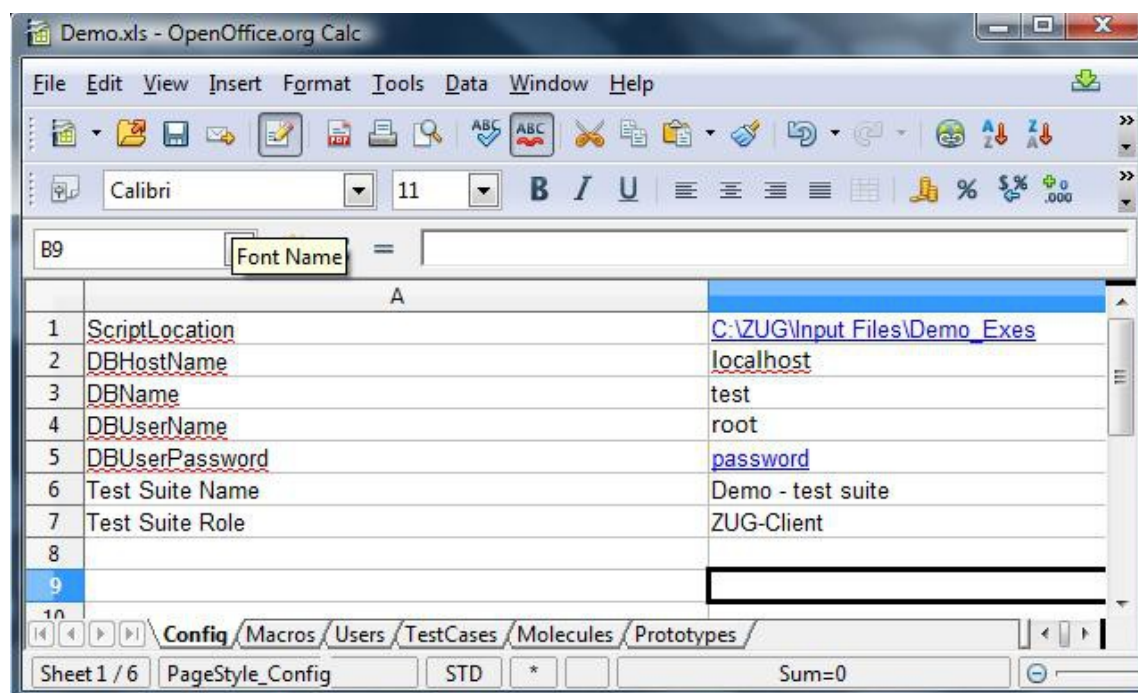
The atoms in Zug can be written in any scripting or programming language. In this manual we have created the atoms using AutoIT, a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting. For web based products, we have our own proprietary libraries named Zuoz. For more information regarding Zuoz kindly refer to www.automature.com.

The demo atoms are written in AutoIT and stored in a folder inside the Input files folder of the Zug Kit.

6.3 Preparing Zug for Zermatt

Once the atoms are written, the Chur spread sheet is prepared. For more information on how to write a Chur file please refer to the Language Reference Manual of Chur.

The figure below shows the configuration page of Chur file for this test.



The Config page states the environmental information which is used by Zug to locate supporting programs and uploading results into the database.

³⁵₁₇ *ScriptLocation* states location of the folder where the Zug atoms are written for this test.

³⁵₁₇ *DBHostName* states the hostname for the database server. To keep the data repository of Zermatt in the Zug machine, **localhost** is given as input. We can provide the exact port number also for reporting.

³⁵₁₇ *DBName* states the database name.

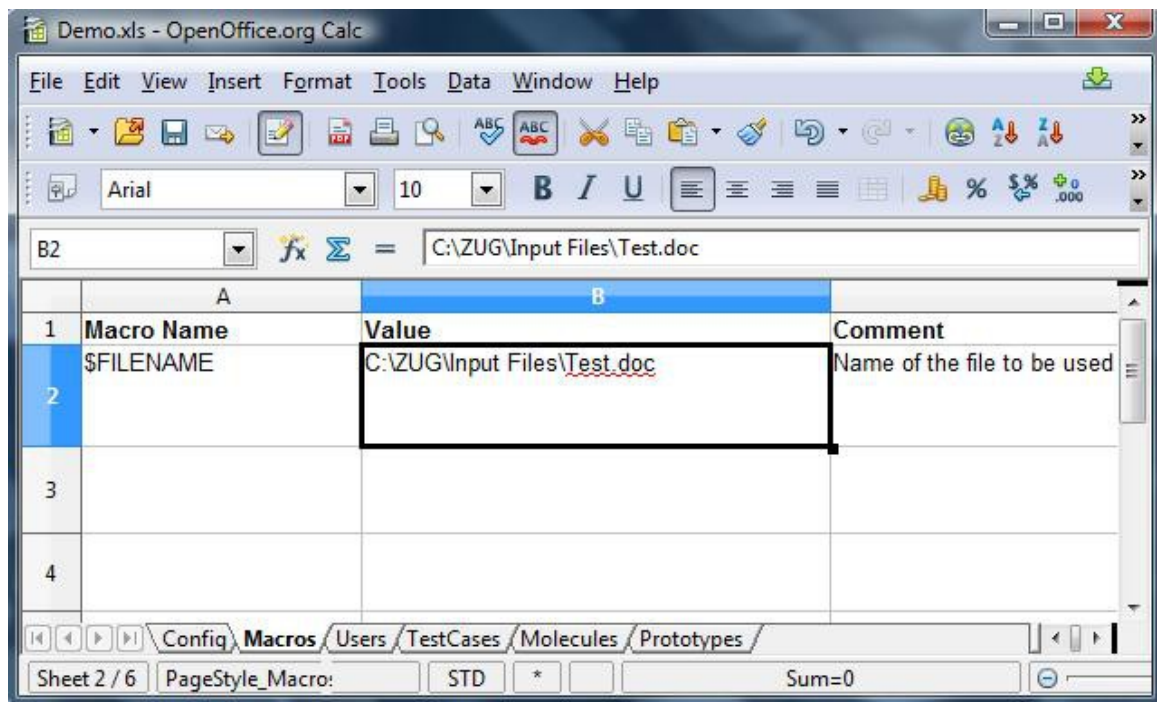
³⁵₁₇ *DBUserName* states the name of the user to do authentication to database.

³⁵₁₇ *DBUserPassword* states the password of the user required for SQL authentication.

³⁵₁₇ *Test Suite Name* states the name of the test suite which was stated in Zermatt.

³⁵₁₇ *Test Suite Role* states the role of the node where Test Suite will be executed.

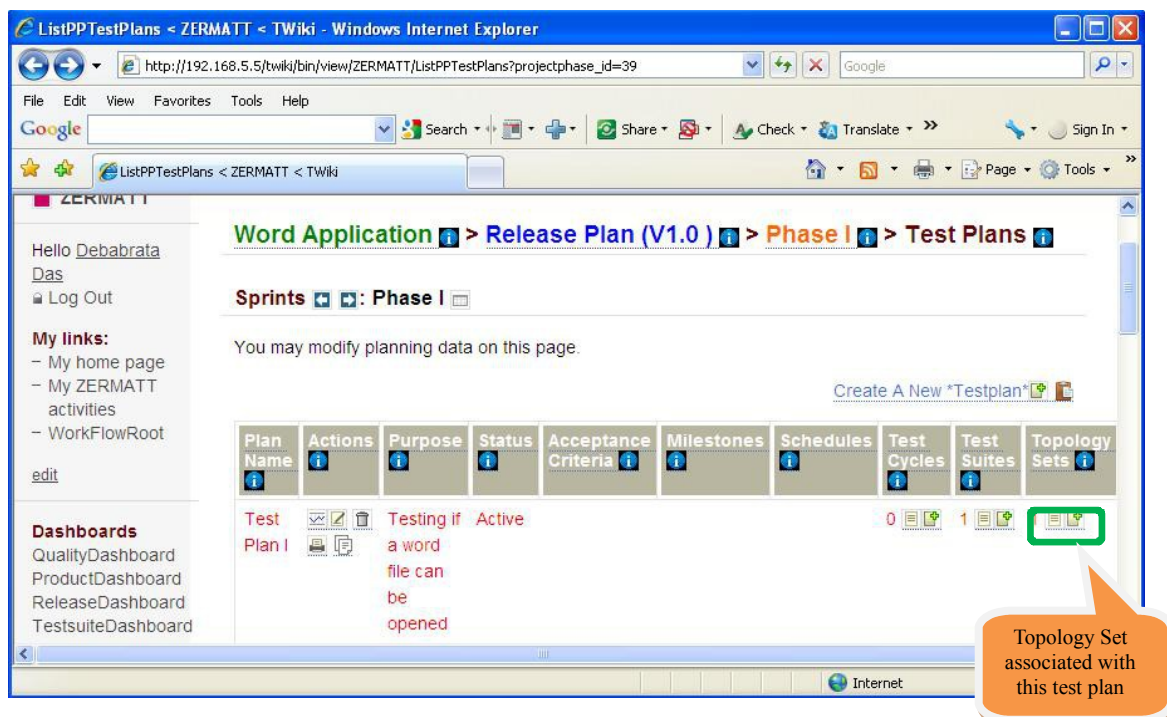
The figure below shows the macros page of Chur file for this test.



The macros allow the testcase designer to declare short names that can then be substituted in the testcase descriptions during execution. In this case, the path of the file Test.doc is given as input.

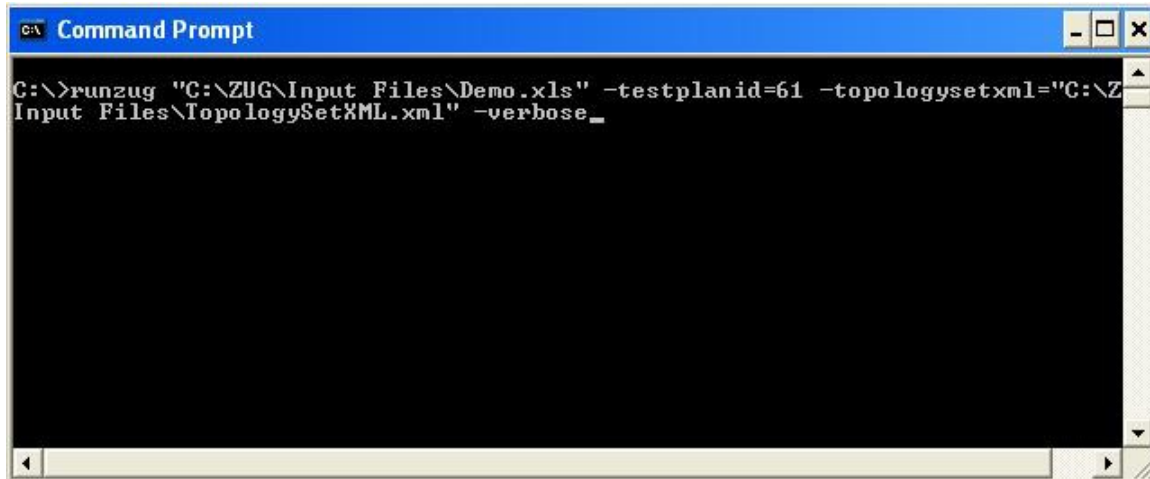
6.4 Executing the automated test

Time has come to execute the automated test in Zug. While executing, the test cycles will be incremented in Zermatt automatically. Before executing, the Zermatt page will look as shown in the figure below.



ZUG User Manual		Page 18/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

³⁵₁₇ Type `runzug "C:\ZUG\Input Files\Demo.xls" -testplanid=61 -topologysetid=1 -verbose`



³⁵₁₇ The following screens will appear automatically. The automation starts....

```
C:\>runzug "C:\ZUG\Input Files\Demo.xls" -testplanid=61 -topologysetxml="C:\ZUG\
Input Files\TopologySetXML.xml" -verbose
Automation Started
-----
Controller/Main : Validating Command Line Arguments

Controller/Main: Command Line Arguments Validated

Current date is : 2010-11-2
Expiry date is : 2011-5-1
Automation started for Automature Inc.
Reading the TestCases Input Sheet C:\ZUG\Input Files\Demo.xls.

SUCCESSFULLY Read the TestCases Input Sheet C:\ZUG\Input Files\Demo.xls
Connecting to the Database : framework of Host 192.168.5.5 with User zermattadmi
n
Connection to the Database is successful.
```

ZUG User Manual		Page 19/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

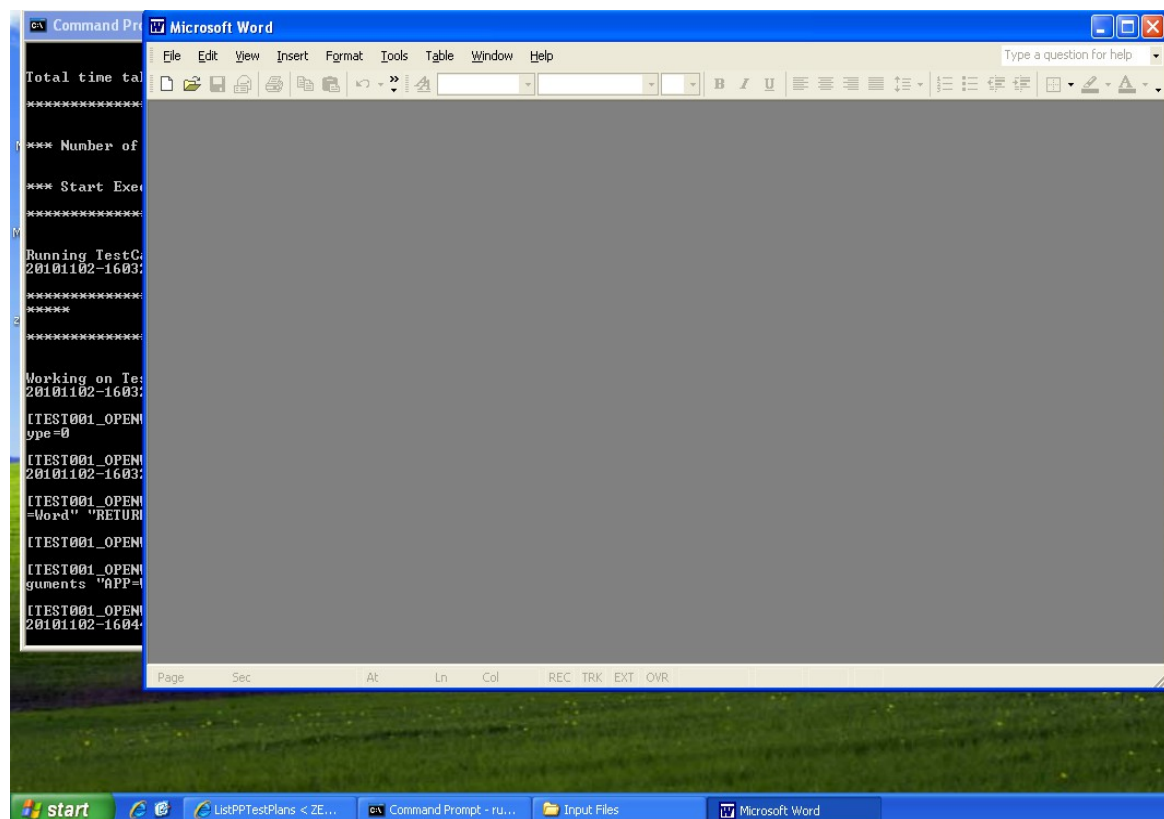
³⁵₁₇ The atom named STARTAPP.EXE is called to execute.

```

*****
Total time taken to initialize the Harness is -> 13000 milli Seconds.
*****
*** Number of TestCase to Execute is 2***
*** Start Executing the testcases ***
*****
Running TestCase ID Test001
20101102-160326
***** Running Molecule Test001_OpenWordApp *****
*****
Working on Test Case Variable Combination OpenWordApp
20101102-160326
[TEST001_OPENWORDAPP] Action SETCONTEXTUAR Execution STARTED With Arguments AppI
ype=0
[TEST001_OPENWORDAPP] Action SETCONTEXTUAR SUCCESSFULLY Executed
20101102-160326
[TEST001_OPENWORDAPP] Action @STARTAPP.EXE Execution STARTED With Arguments "APP
=Word" "RETURNWINDOW=AppIype"

```

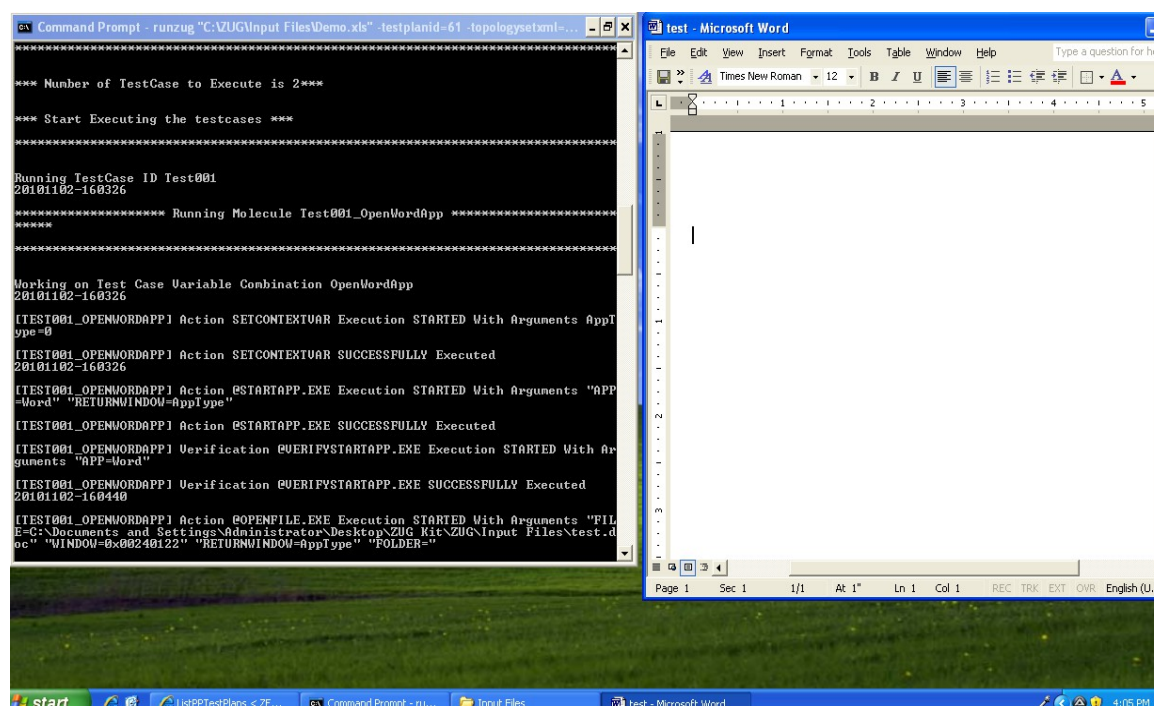
³⁵₁₇ The Word Application is opened.



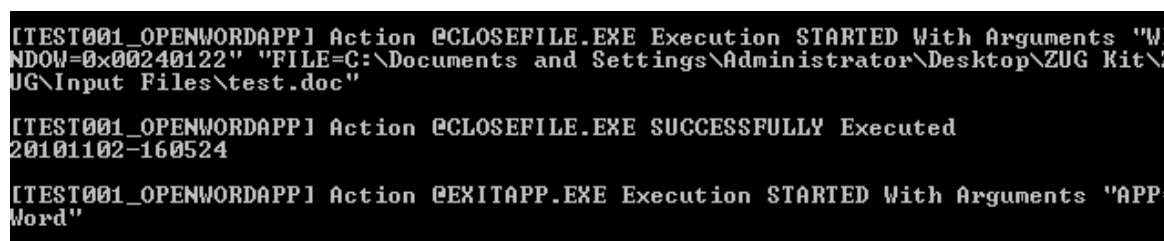
ZUG User Manual		Page 20/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

35
17 The Word file is opened by executing the OPENFILE.EXE.

35
17



35
17 After the Word file is opened, the CloseFile and the ExitFile executable files are called which closes the Word file and then exits from the Word application.



35
17 The following screens appear on the command prompt which gives the result of the automated test.

```

C:\ Command Prompt
STATUS : PASS FOR TestCase ID Test001
*****
**
Storing the TestCase Result to 192.168.5.5\framework Database.....
Saving Result for TestCycle ID 97 and Test Plan ID 61 of Test Plan Demo - test S
uit
-----
-----
Following are the Details of the Topology
Topology ID      Role      Build Number
3              6        null
8              5        null
9              7        null
10             11       null
30             12       null
Following are the Details of the TestCases Result getting added to the 192.168.5
.5\framework Database.
TestCase ID      Status      Time Taken(In mili-seconds)      Comments
Test001 pass      165735
SUCCESSFULLY SAVED Result for TestCycle 97 and TestPlan ID 61 of Test Suit Demo
- test Suit
-----
-----
SUCCESSFULLY Stored the TestCase Result to 192.168.5.5\framework Database.....
*****

```

The detailed explanation will be stated in the following section 6.5-6.5 Viewing results in Zermatt.

6.5 Viewing results in Zermatt

³⁵₁₇ You will see that the test cycle is automatically increased to one. Refer to section Executing the automated test to see the previous value of test cycle. Now every time you run Zug, the value increments by itself.

The screenshot shows the ZERMATT web application interface. On the left is a sidebar with user information (Hello Debabrata Das, Log Out) and navigation links. The main content area displays a breadcrumb trail: **Word Application > Release Plan (V1.0) > Phase I > Test Plans**. Below the breadcrumb, it shows **Sprints: Phase I** and a message: "You may modify planning data on this page." There is a link "Create A New *Testplan*". A table lists test plans with columns: Plan Name, Actions, Purpose, Status, Acceptance Criteria, Milestones, Schedules, Test Cycles, Test Suites, and Topology Sets. The first row shows "Test Plan I" with a status of "Active".

6.6 Interpreting the results

➤ Results in Command Prompt

The progress of the test-cases are displayed one after another. The specified Action and Verification Steps inside the test cases are executed and are shown at the same time in the command prompt. If any step fails then the exception message is displayed explaining the cause of failure.

At the end the brief result are shown in tabular form. The table has columns TestCase ID, status, Time Taken, Comments.

TestCase ID	The identifier of the test-cases which ran inside test suite
status	Pass/Fail
Time Taken (milli seconds)	Milliseconds taken to execute the test case
Comments	On failing, this states the exception message with the reason of failing

```

C:\> Command Prompt

STATUS : PASS FOR TestCase ID Test001
*****
Storing the TestCase Result to 192.168.5.5\framework Database....
Saving Result for TestCycle ID 97 and Test Plan ID 61 of Test Plan Demo - test S
uit

-----

Following are the Details of the Topology
Topology ID      Role      Build Number
3              6        null
8              5        null
9              7        null
10             11       null
30             12       null

Following are the Details of the TestCases Result getting added to the 192.168.5
.5\framework Database.
TestCase ID      Status      Time Taken<In mili-seconds>      Comments
Test001 pass      165735

SUCCESSFULLY SAVED Result for TestCycle 97 and TestPlan ID 61 of Test Suit Demo
- test Suit

-----

SUCCESSFULLY Stored the TestCase Result to 192.168.5.5\framework Database....
*****

```

➤Results in Log Files

Zug will generate four types of Logs. These logs will be created inside the AppData Folder (Application Data folder of windows OS) of logged in user.

All the Log File Name will be appended with a Date Time. For example Debug log will be named as 2008128-163637543-Debug.log (format is yyyyymmdd-hhmmssmillisecond-debug) which means that this Debug Log is created on 8th December 2008 at 16 hours 36 minutes (last 5 are seconds and milli-seconds). Following is the description of the log files -

- 1."Result.log" - This log will contain the result of the test case, It will contain the status of the action(s) specified in the TestCase worksheet and also the appropriate error result if any error thrown during the test case series execution.
- 2."Debug.log"- This log will contain all the debug related messages.
- 3."Error.log"- This log will contain the error and warning messages if any error occurred during the execution.
- 4."Primitives.log" - This log will contain Logs from the Atoms during atom execution.

Viewing the test results from Zermatt:

To view the results in Zermatt, navigate to the page where the test plans are written. As said earlier, the test cycle is incremented from 0 to 1.

ZERMATT

Hello [Debabrata Das](#)
[Log Out](#)

My links:

- My home page
- My ZERMATT activities
- WorkFlowRoot

[edit](#)

Dashboards

- QualityDashboard
- ProductDashboard
- ReleaseDashboard
- TestSuiteDashboard

[Word Application](#) > [Release Plan \(V1.0\)](#) > [Phase I](#) > [Test Plans](#)

Sprints : Phase I

You may modify planning data on this page.

[Create A New *Testplan*](#)

Plan Name	Actions	Purpose	Status	Acceptance Criteria	Milestones	Schedules	Test Cycles	Test Suites	Topology Sets
Test Plan I		Testing if a word file can be opened	Active				1	1	1

[View](#)

Click on View and you will be able to see the following page. It states the time and the date of the initialization and completion of the test cycle.

Hello [Debabrata Das](#)
[Log Out](#)

My links:

- My home page
- My ZERMATT activities
- WorkFlowRoot

[edit](#)

Dashboards

- QualityDashboard

[Word Application](#) > [Release Plan \(V1.0\)](#) > [Phase I](#) > [Test Plan I](#) > [Test Cycles](#)

Test Plans :

You may modify test execution data on this page.

[Add New Test Cycle](#)

Cycle Description	Actions	Started	Completed	ms to Initialize	ms to Complete	Execution Topologysets
NULL		2010-11-02 16:03:21	2010-11-02 16:06:10	13000	183843	1

If you navigate into the topology set of that test plan, the test execution results are shown. It stated that the test has passed and also the duration and the date of the test execution.

Execution Topology Sets : [First Topology Set](#) [First Topology Set](#) [First Topology Set](#) [First Topology Set](#)

You may modify test execution data on this page.

[Add New *Manual Test Execution Result*](#)

Test Case	Action	Test Suite	Execution Date	Test Engineer	Status	Comment	Build Tag	Duration	Performance Details	Bug
Test001		Demo - test Suit	2010-11-02	Automation	pass			165735		

If you navigate to the Dashboards, the various kinds of results that might be possible is shown in various formats. For further information regarding dashboards, kindly refer to Zermatt User Manual.

ZUG User Manual		Page 25/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

6.7 Archival the Log files

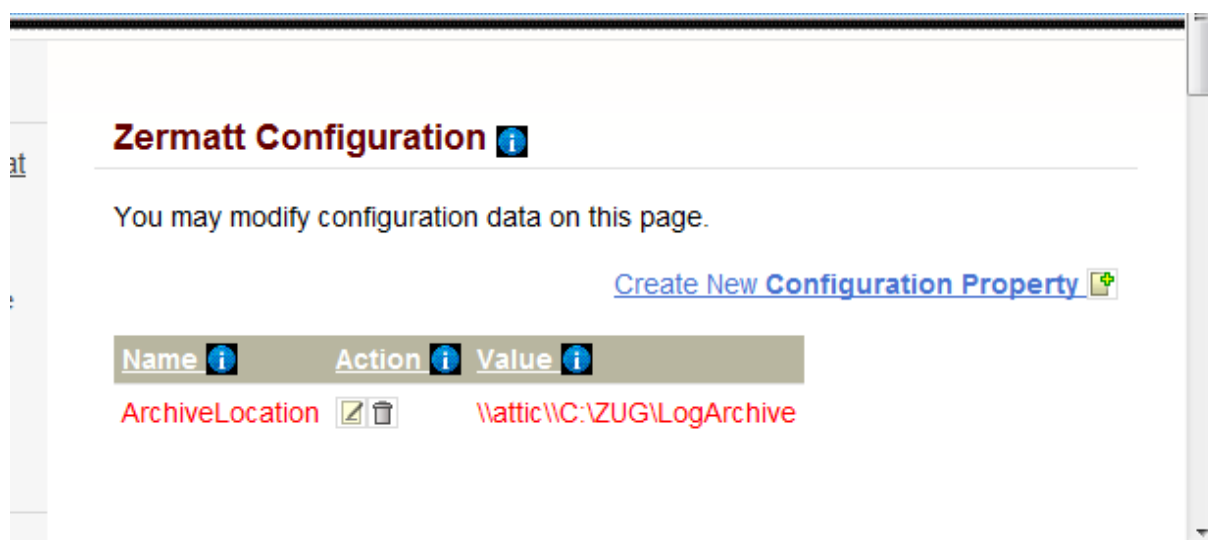
Zug supports log archiving. If Zermatt integration is used then Zug will archive the log. Zug will archive two types of logs:

1. Tested product (like LMDC Client, Server) log files. Log file specified in Config sheet of Test design sheet. This is done by specifying [ProductLogLocations](#) in the Config sheet of Chur Test Suite.

2. Zug created log files (Debug, Error, Result). These are the log files which are generated by the Zug during test plan execution.

Log Archival Setup

You can specify Archive Location in the Zermatt Configuration Page:



6.8 Reporting to TestLink

6.8.1 Basic Terminologies of TestLink

Test Projects are the basic organizational unit of Test Link. Test Projects could be products or solutions of your company that may change their features and functionality over time but for the most part remain the same. A Test Project includes Test Specifications with Test Cases, Requirements and Keywords. Users within the project have defined roles. Test Projects are independent and do not share data.

Test Case describes a testing task via steps (actions, scenario) and expected results. The Test Case is the fundamental element of Test Link.

Test Suite organizes Test Cases into units. Each Test Suite consists of a title, a formatted description, Test Cases and possibly other Test Suites. TestLink uses a tree structure for Test Suites.

Test Plans are the basis for test execution activity. A test plan is created when you wish to execute Test Cases. A Test Plan contains name, description, collection of chosen Test Cases, Builds and some optional attributes (Test Results, milestones, tester assignment and priority definition). Each Test Plan is related to the current Test Project.

ZUG User Manual		Page 26/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

6.8.2 Preparing ZUG for TestLink

We need to provide some information in the configuration page of the Chur spread sheet in order to report the execution results of test cases in Test link.

The figure below shows the configuration page of Chur file for this test.

	A	B
1	ScriptLocation	C:\Program Files\Automature\Zuoz\GeneralPurpose\Ruby;
2	DBHostName	http://localhost:8098/testlink/lib/api/xmlrpc.php
3	TestFramework	testlink
4	DBUserName	admin
5	DBUserPassword	1234
6	Test Suite Name	CV
7	Test Suite Role	ZUG
8	Include	
9		

The Config page states the environmental information which is used by Zug to locate supporting programs and uploading results into the Test Frame Work.

- **ScriptLocation** states location of the folder where the Zug atoms are written for this test.
- **DBHostName** states the host name for the Test framework server. Here we have to provide the the IP address along with the port number in which the server is listening and the rest of the part is the location of a PHP(xmlrpc.php) file in the Test Link framework.
- **TestFramework** states the name of the Test Frame Work in which we want to report the results of the test case execution. Since we want to report in TestLink the value of the field is testlink.
- **DBUserName** states the name of the user to do authentication to the Test Frame Work. In this case we are using the admin.
- **DBUserPassword** states the dev key ,also named as **Personal API access key or script key** for the user.
- **Test Suite Name** states the name of the test suite which was stated in Test Link.
- **Test Suite Role** states the name of the Test Project to which the Test Suite belong.

Note:The dev key,also called **Personal API access key** can be found in the MySetting link in TestLink. If the key is not generated then click on the button “Generate a new key”.A key will be generated. Below is the snapshot of the page.

ZUG User Manual	Page 27/ 38
Author: Md Sarfaraz Khan	Version: 6.5
	Date 20/05/13

localhost:8098/testlink/index.php

TestLink Prague 1.9.5 : admin [admin] [My Settings | Logout]

Project | Requirement Specification | Test Specification | Test Execution | Test Reports | User Management | Events |

Account Settings

Personal data

Login	admin
First Name	<input type="text" value="Testlink"/>
Last Name	<input type="text" value="Administrator"/>
Email	<input type="text"/>
Locale	English (wide/UK) <input type="button" value="v"/> English is the default development language and is always up to date

Personal password

Old Password	<input type="text"/>
New Password	<input type="text"/>
Confirm New Password	<input type="text"/>

API interface

Personal API access key = 1234

However if the last section of the page (API interface) is not there and there is no button for generating a new key then follow the subtopic “Enable API interface” in our next topic.

6.8.3 Preparing TestLink For ZUG:

- Create a Test Project and while creating a test project make sure that **Enable Test Automation (API keys)** box is checked. If the project is already created then click on the project and see if the check box is selected or not. If not then select the check option and save it.

ZUG User Manual		Page 28/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

TestLink Prague 1.9.5 : admin [admin] [My Settings | Logout]

Project | Requirement Specification | Test Specification | Test Execution | Test Reports | User Management | Events |

Test Project Management : Create a new project

Create from existing Test Project?

Name *

Prefix (used for Test case ID) *

Project description

Enhanced features

- Enable Requirements feature
- Enable Testing Priority
- Enable Test Automation (API keys)
- Enable Inventory

Issue Tracker Integration

- Active
- Issue Tracker

Availability

- Active
- Public

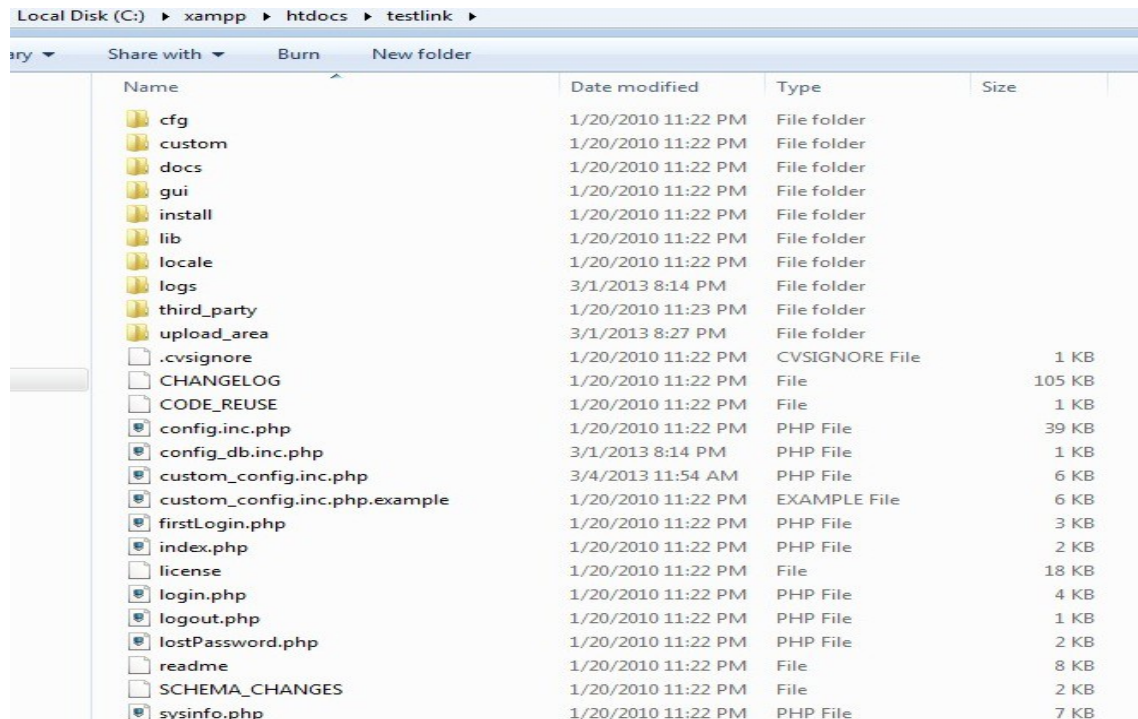
- Create a new Test Suite in the Test Project.
- Once the Test Suite is created we can add test cases in the Test Suite.
- We need to create a Test Plan and a build for that test plan.
- Finally once the Test Plan and the build is created the test cases need to be associated with the test plans.

Note: For reporting from ZUG in Test Link Test Project and Test Suite only needs to be created before running the tests. Last 3 steps mentioned above can also be performed by ZUG. However for that the user whose credentials is mentioned in the chur's configuration sheet must be authorized in test link to do the mentioned operations. For more information on TestLink read the user manual for test link(http://www.teamst.org/_tldoc/1.9/testlink_user_manual.pdf).

Enabling the API Interface:

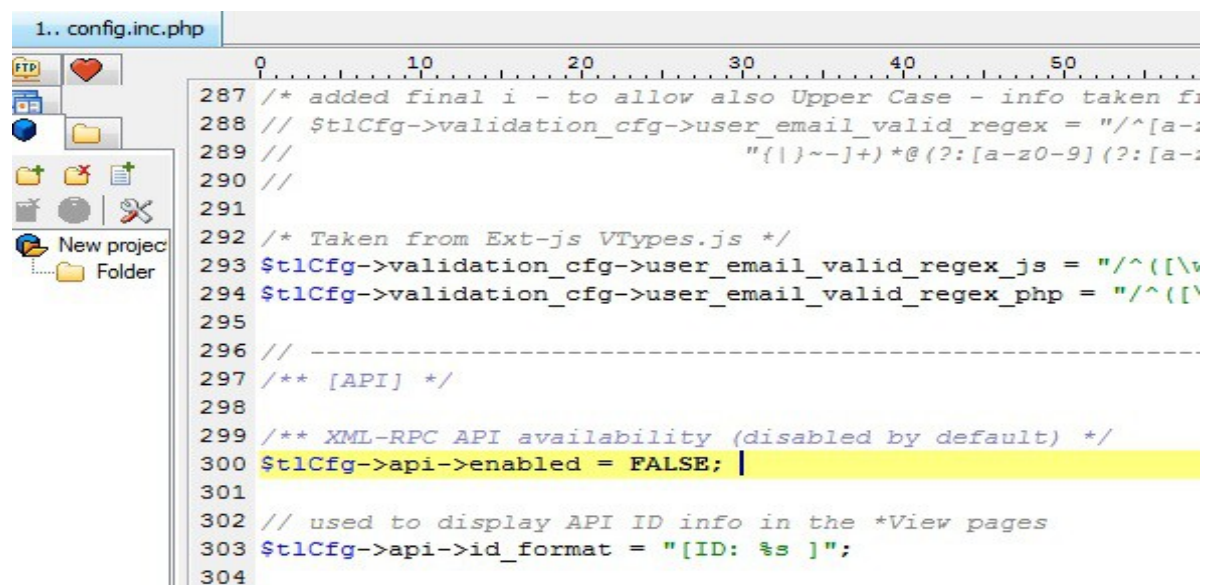
(This section if needed only if the API interface is disable and the user is not bale to generate API key)

By default the API Interface is **not** enabled in the TestLink. In the TestLink folder there is file config.inc.php.



Open this file with any text editor and search for the line :

`$tlCfg->api->enabled .If $tlCfg->api->enabled = FALSE;`



then set it to true by replacing the line with :

`$tlCfg->api->enabled = TRUE;`

```

1.. config.inc.php
293 $tlCfg->validation_cfg->user_email_valid_regex_js = "/^([\w]+) (.{1"
294 $tlCfg->validation_cfg->user_email_valid_regex_php = "/^([\w]+) (.{1"
295
296 // -----
297 /** [API] */
298
299 /** XML-RPC API availability (disabled by default) */
300 $tlCfg->api->enabled = TRUE;
301
302 // used to display API ID info in the *View pages
303 $tlCfg->api->id_format = "[ID: %s ]";
304
305
306 // -----

```

Now save the file and upload the folder to the server and start the server (If the server was already running then it should be stopped first and then needs to be restarted after the changes have been made).

6.8.4 Executing the automated tests

- Open a command prompt.
- Type `runzug "file path\filename.xls" -verbose -testplanname= mytestplanname -buildname=mybuildname` . For example

```

C:\Windows\system32\cmd.exe
E:\>runzug "C:\Users\skhan\Desktop\U6.0\TestSuites\contextvar.xls" -verbose -testplanname=TP1 -buildname=B-20130328-140
Automation Started

[Warning] ZugINI.xml contains blank inprocess package definition. Please refer to the readme.txt or Zug User Manual

```

If the test plan name doesn't exist then it creates one with the mentioned test plan name and also creates a build with the mentioned build name. The build name is optional if we don't give a build name it will create a new build with a new system generated build name.

- The following lines will appear automatically. The automation starts....

```

C:\Windows\system32\cmd.exe
E:\>runzug "C:\Users\skhan\Desktop\U6.0\TestSuites\contextvar.xls" -verbose -testplanname=TP1 -buildname=B-20130328-140
Automation Started

[Warning] ZugINI.xml contains blank inprocess package definition. Please refer to the readme.txt or Zug User Manual

Command Line Arguments Validated
Current date is : 2013-3-28
Expiry date is : 2014-1-1
Zug is Valid Automature LLC. This Generic License is issued to Automature and Manित्रा for usage of Zug.
Reading the TestCases Input Sheet C:\Users\skhan\Desktop\U6.0\TestSuites\contextvar.xls.
SUCCESSFULLY Read the TestCases Input Sheet C:\Users\skhan\Desktop\U6.0\TestSuites\contextvar.xls

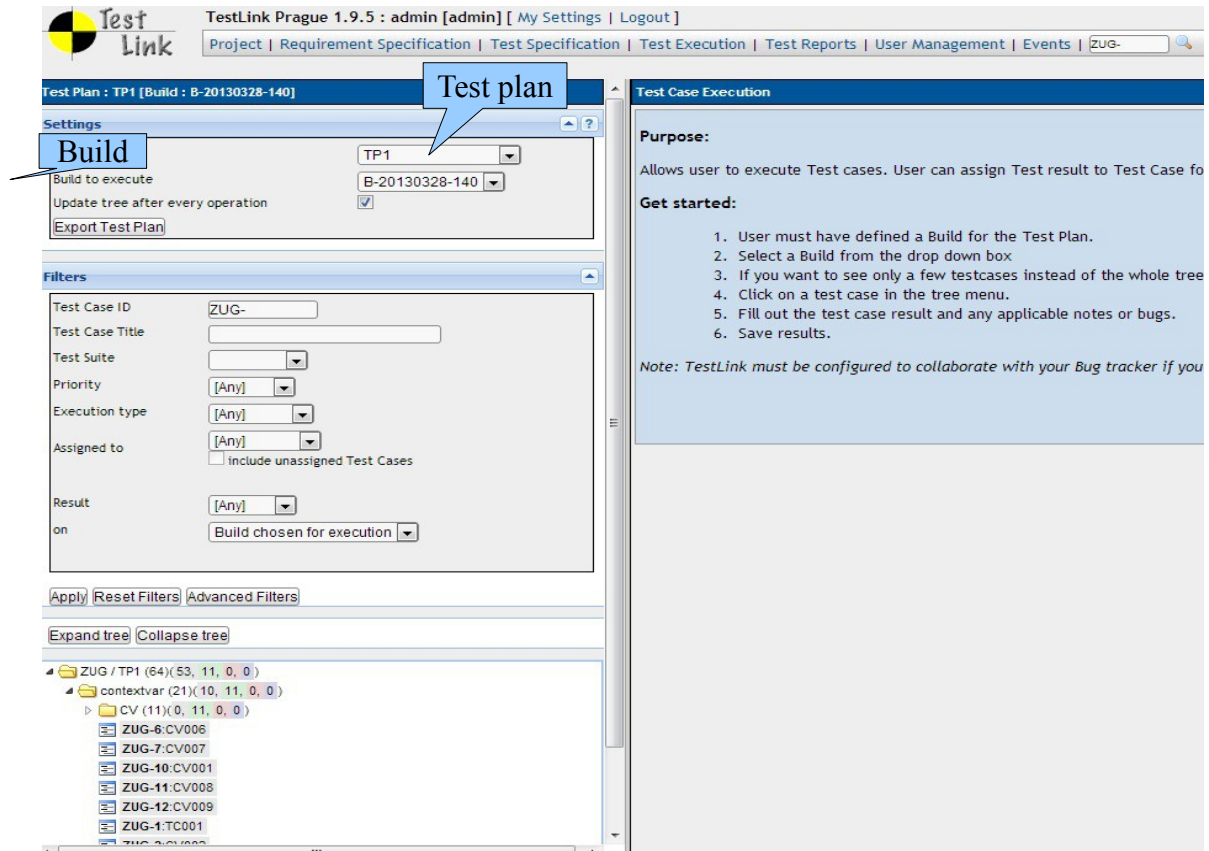
Connecting to Test Link http://localhost:8098/testlink/lib/api/xmlrpc.php
Connection to testlink is successful.

TestLinkReporter:Build name was not provided or the build name doesn't exist System has created a new build: B-20130328-140

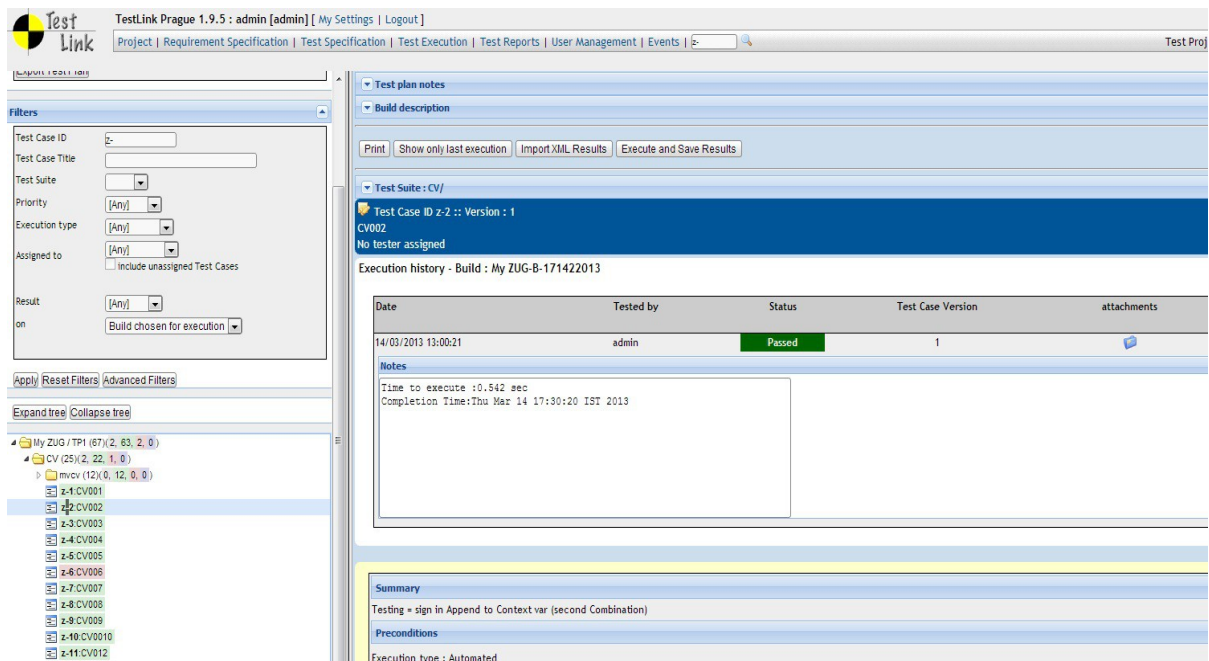
*****
Total time taken to initialize the Harness is -> 13764 milli Seconds.
*****

```

- Once the execution is finished we can go to the Execute Test cases link in the Test Link and select the test plan name and the build name that we mentioned in the command prompt in the runzug command.



- Once we choose the test plan and the build we'll be able to see the executed test cases as shown in the above pic. To see the execution result of a test case click just click on it as shown below.



ZUG User Manual		Page 32/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

7. Implicit Molecule Calls

Often we would like to run some operations or some verification that are implicit in nature. For example sensing the environment on which the tests are being run or checking after each test case that the application that is being tested is reachable or running. For this purpose there are some molecule names that have been reserved. When a molecule is defined in the test suite with the reserved molecule name that molecule is called implicitly by ZUG.

7.1 ZCase_Verify

If a molecule of this name exists in the Molecule sheet then that molecule is implicitly invoked after the execution of each Test Case. We don't have to call this molecule explicitly ZUG Implicitly calls this molecule after running each of the Test Case. However this Molecule is not invoked for the Init and CleanUp test case.

7.2 ZStep_Verify

If a molecule of this name exists in the Molecule sheet then that molecule is implicitly invoked after the execution of each test step. The test step can be of a Molecule or that of a Test Case. ZUG Implicitly calls this molecule after running each of the test step. However this Molecule is not invoked for the test step of Init and CleanUp test case and also for the test steps of ZCase_Verify molecule.

7.3 ZEnv_Sensor

A molecule is defined with this name to sense the environment in which the tests are being run. This molecule is invoked implicitly after running the Init test case or after running the first test case of the test suite.

TestCase ID	Description	property	Step	Action	ActionArg_1	ActionArg_2	ActionArg_3	Verify
Init				SetContextvar	envValues			
TC001				Zstring.compare	%envValues%	%ZEnv_Values%		
cleanup				UnSetContextvar	envValues			

Molecule ID	Description	Property	Step	Action	ActionArg_1	ActionArg_2	Verify
zenv_sensor				appendtoContextvar print	envValues	%ZEnv_Values%	
					inside molecule		

In the above pic we can see there is a test case and a molecule **Zenv_Sensor**. Also we have used a context variable **ZEnv_Values**. This context variable is defined by ZUG to store some environment related information. Initially it holds some values like OS Name=Windows 7,Java Version=1.6,User Name=Ellora ,etc as a comma separated list. We can further add some more details in the ZEnv_Values context variable in this molecule which may be needed. For example the name and the version of the browser that will be used for the browser operations.

ZUG User Manual		Page 33/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

```

*** Start Executing the testcases ***
*****
Running TestCase ID Init On(Current Date): 2013:05:13-16:46:38
[INIT] Action SETCONTEXTUAR Execution STARTED With Arguments envValues
[INIT] Action SETCONTEXTUAR SUCCESSFULLY Executed

STATUS : PASS FOR TestCase ID Init On(Current Date): 2013:05:13-16:46:38
*****
***** Running Molecule Init_zenv_sensor *****
*****

[INIT_ZENV_SENSOR] Action APPENDTOCONTEXTUAR Execution STARTED With Arguments contextvar = envValues valueToAppend = 0
S Name=Windows 7,Java version=1.7,LOGON SERVER=\\ELLORA,COMPUTER NAME=ELLORA,APPDATA=C:\Users\s Khan\AppData\Roaming,USERDOMAIN=ELLORA,PROCESSOR_ARCHITECTURE=x86,NUMBER_OF_PROCESSORS=2,RAM=3999 MB
[INIT_ZENV_SENSOR] Action APPENDTOCONTEXTUAR SUCCESSFULLY Executed
[INIT_ZENV_SENSOR] Executed Action print with values linside molecule]

***** Molecule Init_zenv_sensor Execution Finished *****
*****

Running TestCase ID TC001 On(Current Date): 2013:05:13-16:46:39
[TC001] Execution Started Action Zstring.compare with values lOS Name=Windows 7,Java version=1.7,LOGON SERVER=\\ELLORA,COMPUTER NAME=ELLORA,USER NAME=s Khan,APPDATA=C:\Users\s Khan\AppData\Roaming,USERDOMAIN=ELLORA,PROCESSOR_ARCHITECTURE=x86,NUMBER_OF_PROCESSORS=2,RAM=3999 MB, OS Name=Windows 7,Java version=1.7,LOGON SERVER=\\ELLORA,COMPUTER NAME=ELLORA,USER NAME=s Khan,APPDATA=C:\Users\s Khan\AppData\Roaming,USERDOMAIN=ELLORA,PROCESSOR_ARCHITECTURE=x86,NUMBER_OF_PROCESSORS=2,RAM=3999 MB]
[TC001] Action ZSTRING.COMPARE SUCCESSFULLY Executed

STATUS : PASS FOR TestCase ID TC001 On(Current Date): 2013:05:13-16:46:39
*****

Running TestCase ID TC002 On(Current Date): 2013:05:13-16:46:40
[TC002] Execution Started Action Zstring.compare with values lOS Name=Windows 7,Java version=1.7,LOGON SERVER=\\ELLORA,COMPUTER NAME=ELLORA,USER NAME=s Khan,APPDATA=C:\Users\s Khan\AppData\Roaming,USERDOMAIN=ELLORA,PROCESSOR_ARCHITECTURE=x86,NUMBER_OF_PROCESSORS=2,RAM=3999 MB, OS Name=Windows 7,Java version=1.7,LOGON SERVER=\\ELLORA,COMPUTER NAME=ELLORA,USER NAME=s Khan,APPDATA=C:\Users\s Khan\AppData\Roaming,USERDOMAIN=ELLORA,PROCESSOR_ARCHITECTURE=x86,NUMBER_OF_PROCESSORS=2,RAM=3999 MB]
[TC002] Action ZSTRING.COMPARE SUCCESSFULLY Executed

STATUS : PASS FOR TestCase ID TC002 On(Current Date): 2013:05:13-16:46:40
*****

Running TestCase ID cleanup On(Current Date): 2013:05:13-16:46:40
[CLEANUP] Action UNSETCONTEXTUAR Executed With NO Arguments

STATUS : PASS FOR TestCase ID cleanup On(Current Date): 2013:05:13-16:46:41
*****

Following are the Details of the TestCases Result Executed by ZUG Version -> ZUG Premium 6.2.20130507.148
Testcase ID      Status      Time Taken(In milli-seconds)  Comments
TC001    pass      242
TC002    pass      254

*****
Total time taken to execute all the test cases <End to End> is -> 5954 milli Seconds.
*****

```

Above is the console output of the test case. We can see that in the console output that the **Zenv_Sensor** molecule have been implicitly invoked in the end of the **Init** test case.

ZUG User Manual		Page 34/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

8. Troubleshooting

8.1 Debug Log

Zug generates different log file. Debug file stores messages which contain extensive contextual information. They are mostly used for problem diagnosis. In failures this file can be referred back to diagnose and investigate problems, exception. The step-by-step execution messages are appended during Zug automation. The file can be browsed through and

8.2 Primitive log

Zug executes Atoms. To learn how to write Atom please refer to Automature's Zuoz Programmer's Guide. Inside Atoms some logging command mechanism can be used to help future diagnosis of Atom execution. The Atom Logs are stored in Primitive Log File. In failures this file can be referred back to diagnose and correct Atoms

8.3 Database connection problem

In order to integrate test results with Zermatt there should be establishment of proper database connection. You can allow access to certain machines for the specified user connection. Do database user administration to add and modify users permission. Put proper connection details in preparing Zug for Zermatt. Refer Page 16 6.3 Preparing Zug for Zermatt

```

ca. Command Prompt - runZUG.Bat "Input Files\Demo - Copy.xls" -testplanid=1 -topologysetid=1 -v...
C:\ZUG>runZUG.Bat "Input Files\Demo - Copy.xls" -testplanid=1 -topologysetid=1 -
verbose
Automation Started

-----

Controller/Main : Validating Command Line Arguments

Controller/Main: Command Line Arguments Validated

Current date is : 2010-11-8
Expiry date is : 2011-5-1
Automation started for Automature Inc.
Reading the TestCases Input Sheet Input Files\Demo - Copy.xls.

SUCCESSFULLY Read the TestCases Input Sheet Input Files\Demo - Copy.xls
Connecting to the Database : test of Host 192.168.5.5 with User zermattadmin
Connection to the Database is successful.

org.hibernate.exception.JDBCConnectionException: Cannot open connection
    at org.hibernate.exception.sqlstateconverter.convert<sqlstateconverter.j
ava:72>
    at org.hibernate.exception.JDBCExceptionHelper.convert<JDBCExceptionHelp
er.java:43>
    at org.hibernate.exception.JDBCExceptionHelper.convert<JDBCExceptionHelp
er.java:29>
    at org.hibernate.jdbc.ConnectionManager.openConnection<ConnectionManager
.java:327>
    at org.hibernate.jdbc.ConnectionManager.getConnection<ConnectionManager.
java:118>
    at org.hibernate.jdbc.JDBCContext.connection<JDBCContext.java:127>
    at org.hibernate.transaction.JDBCTransaction.begin<JDBCTransaction.java:
57>
    at org.hibernate.impl.SessionImpl.beginTransaction<SessionImpl.java:1307>

```

ZUG User Manual		Page 35/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

8.4 Unable to Archive Log Files

Archiving of Log files are done on location specified in Configuration (Refer page 25 6.7 Archival the Log files) Zug machine should be given proper write-permission for the location specified. An exception can occur as following:

```

C:\> Command Prompt

SUCCESSFULLY Stored the TestCase Result to 192.168.5.5\framework Database....
*****
Total time taken to execute all the test cases (End to End) is -> 168313 milliseconds.
*****
Utility/copyFile():Unable to copy file. Error message is : \\Automature-Serv\kup\61\98\Demo - test Suit\1\20101108-202042\Zug\20101108-201744-Debug.log <Logon failure: account currently disabled>
Exception occurred while archiving log on \\Automature-Serv\Backup\61\98\Demo - test Suit\1\20101108-202042: \\Automature-Serv\Backup\61\98\Demo - test Suit\1\20101108-202042\Zug\20101108-201744-Debug.log <Logon failure: account currently disabled>
Cleanup starting... Closing Log
Exiting ZUG
-----
Automation Finished.
C:\>

```

8.5 License Expired

License can expire under the two following cases

- 1.You are using an expired license. Each license has a certain validity period and it expires on a particular date as mentioned.
- 2.You are using an improper license. The license may be incorrect or intended for other use.

```

Controller/Main : Validating Command Line Arguments

Controller/Main: Command Line Arguments Validated

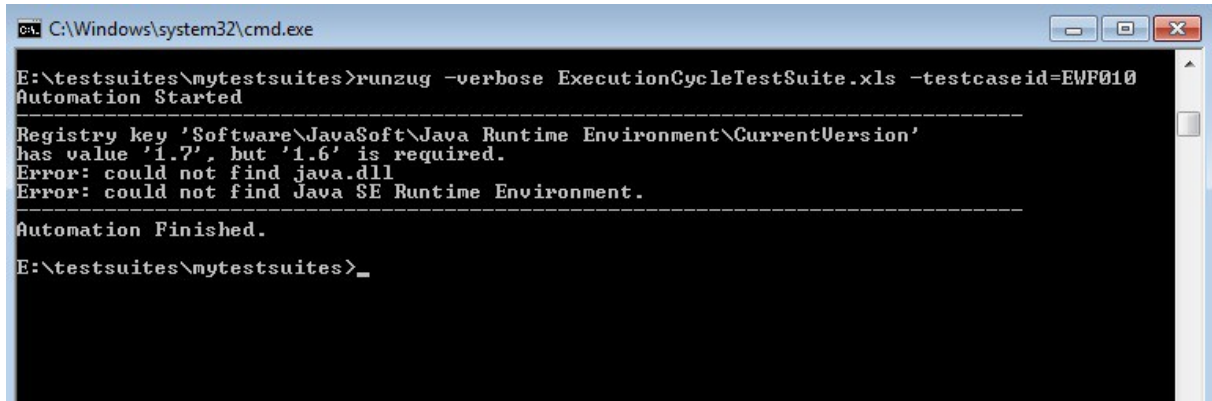
Current date is : 2010-11-1
Expiry date is : 2010-11-1
Controller/Main: The License of ZUG has expired. Please renew.
Visit www.automature.com
-----
Automation Finished.

```

ZUG User Manual		Page 36/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

8.6 JRE Dependency

If the JRE is updated from java 1.6 to java 1.7 after installing ZUG then ZUG will stop working and it will give an error message similar to the message given below



```
C:\Windows\system32\cmd.exe
E:\testsuites\mytestsuites>runzug -verbose ExecutionCycleTestSuite.xls -testcaseid=EF010
Automation Started
-----
Registry key 'Software\JavaSoft\Java Runtime Environment\CurrentVersion'
has value '1.7', but '1.6' is required.
Error: could not find java.dll
Error: could not find Java SE Runtime Environment.
-----
Automation Finished.
E:\testsuites\mytestsuites>
```

Solution: Reinstalling ZUG will fix the problem

ZUG User Manual		Page 37/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

9 ZugINI.xml

9.1 Inprocess atoms configuration

Zug3.0 version can call the external java jar files as built-in atoms through chur spreadsheet. To make this happen the ZugINI.xml parsing is needed. Inside the Zug installation directory, the ZugINI.xml file can be edited to by putting the values in the tags of

```
<inprocesspackages>
  <inprocesspackage name="" language="">
    <file-path></file-path>
    <jar-package></jar-package>
    <class-name></class-name>
  </inprocesspackage>
</inprocesspackages>
```

This xml file (ZugINI.xml) is in ZUG installation folder.

For any Java inprocess atoms, the inprocesspackage tag contains file-path,jar-package and class-name tags.

Example:

```
<inprocesspackages>
  <inprocesspackage name="Zbrowser" language="Java">
    <file-path>C:\Programfiles\Automature\ZUOZ\Bultins</file-path>
    <jar-package>com.automature.zuoz.builtins.zbrowser</jar-package>
    <class-name>BrowserOperations</class-name>
  </inprocesspackage>
</inprocesspackages>
```

For any C++ dll atoms the same inprocesspackage tag will contain different tag values example - the language tag will be changed to "JNI". The file-path tag will specifies the dll. The dll-name tag specifies the name of DLL file.

Example:

```
<inprocesspackages>
  <inprocesspackage name="JNIDLL" language="JNI">
    <file-path>C:\Programfiles\Automature\ZUOZ\Bultins</file-path>
    <dll-name>TestDLL</dll-name>
  </inprocesspackage>
</inprocesspackages>
```

For any COM dll atoms the Program ID is needed.

ZUG User Manual		Page 38/ 38
Author: Md Sarfaraz Khan	Version: 6.5	Date 20/05/13

For the same inprocesspackage tag, language attribute value will be “COM” and there will be one additional tag as prog-id.

Example:

```
<inprocesspackages>
  <inprocesspackage name="ZCOM" language="COM">
    <prog-id>Automature.Pioneer</prog-id>
  </inprocesspackage>
</inprocesspackages>
```

9.2 DB Configuration and Script Locations

In the config sheet of the test suite we have to mention the script location and also some information of the Test Management Framework (credentials, host name, etc) for reporting the execution result of the test cases. However Test suites running on the same machine may report to the same Test Management Framework with a default data base credentials and also we may want to have a default repository for the out process atoms. Thus we can mention default script locations and default configuration for the Test Management Framework in ZugINI.xml file. In the ZugINI.xml file inside the **configurations** we can mention the default script locations under the script location tag and the Test Management Framework information in their respective tags as shown below

```
<configurations>
  <scriptlocation>C:\ProgramFiles;C:\ProgramFiles (x86)</scriptlocation>
  <dbhostname>http://localhost:4567</dbhostname>
  <dbname>Framework</dbname>
  <dbusername>davosuser</dbusername>
  <dbuserpassword>user</dbuserpassword>
</configurations>
```

For further problems visit [ZUG Forum](#).