ZUG User Manual (Version V5.0)

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 uisng Davos
4.3	12-06-12	Dipayan Sengupta	New Features : BuildTag and TestPlan added
4.4	20-06-12	Dipayan Sengupta	New Feature :TopologySet

Contents

Table of Contents

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.

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.

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 <configuations> 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>
```

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

- [days]d
- [hours]h
- [minutes]m
- [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.

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 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: -include=C:\Tests\Molecules

3.9 -Execute | -NoExecute

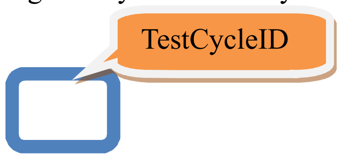
-NoExecute mode will verify if the Test Design Excel sheet prepared by the user is syntactically correct or not. By default -Execute mode is selected unless explicitly specified. If -NoExecute is selected, Zug will just validate the test suite design sheet, display a list of errors, and exit without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

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 -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:**

TOPOLOGYSET_ID

Example: -TopologySetId=26

3.14 -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"

3.15 -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

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:

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

- -Verbose
- -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.

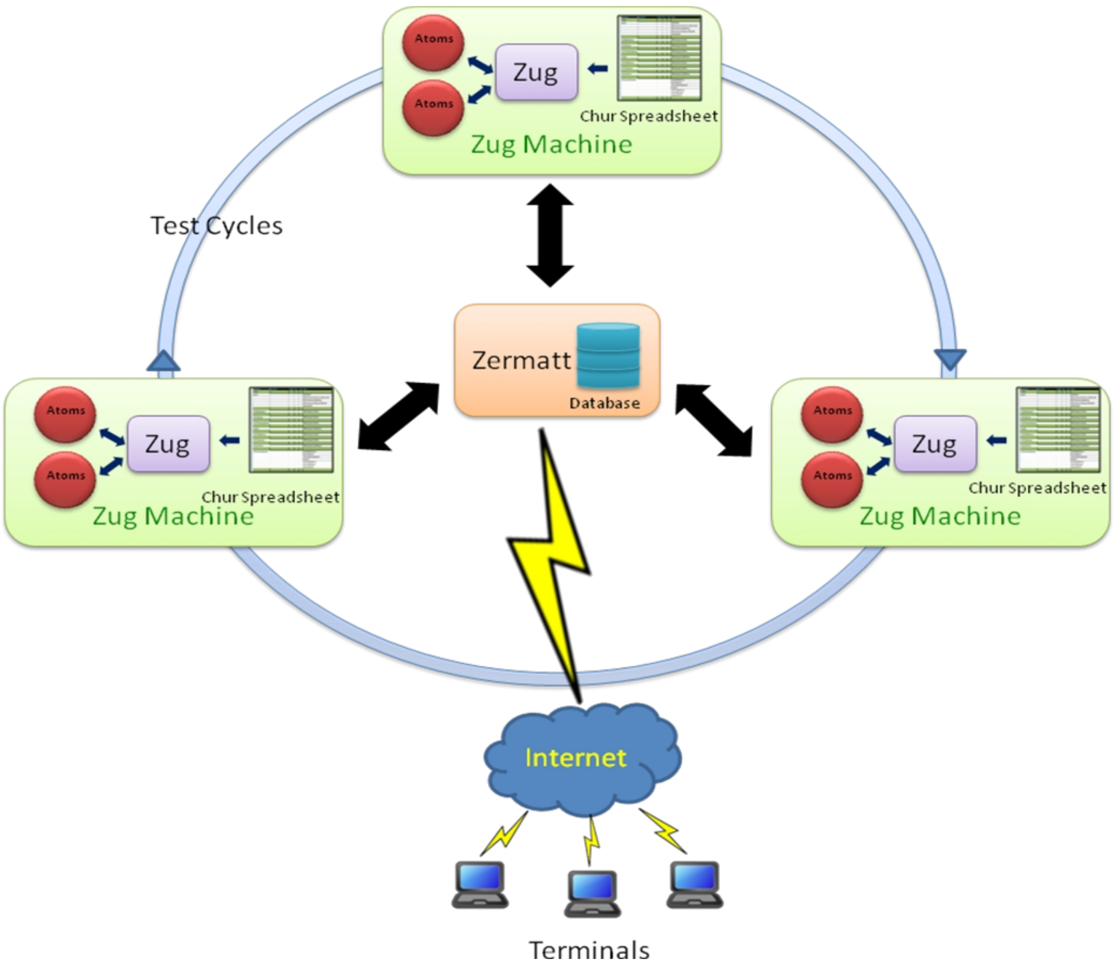
The following options are primarily used to design in this mode:

- -Repeat
- -NoVerify
- -Execute
- -TestPlanID=[integer]
- -TopologySetID=[integer]

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

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.

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

- 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.

- 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 topologyset_id is found in the status bar of **List all Topology Sets for a chosen Test Plan**.

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

- 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

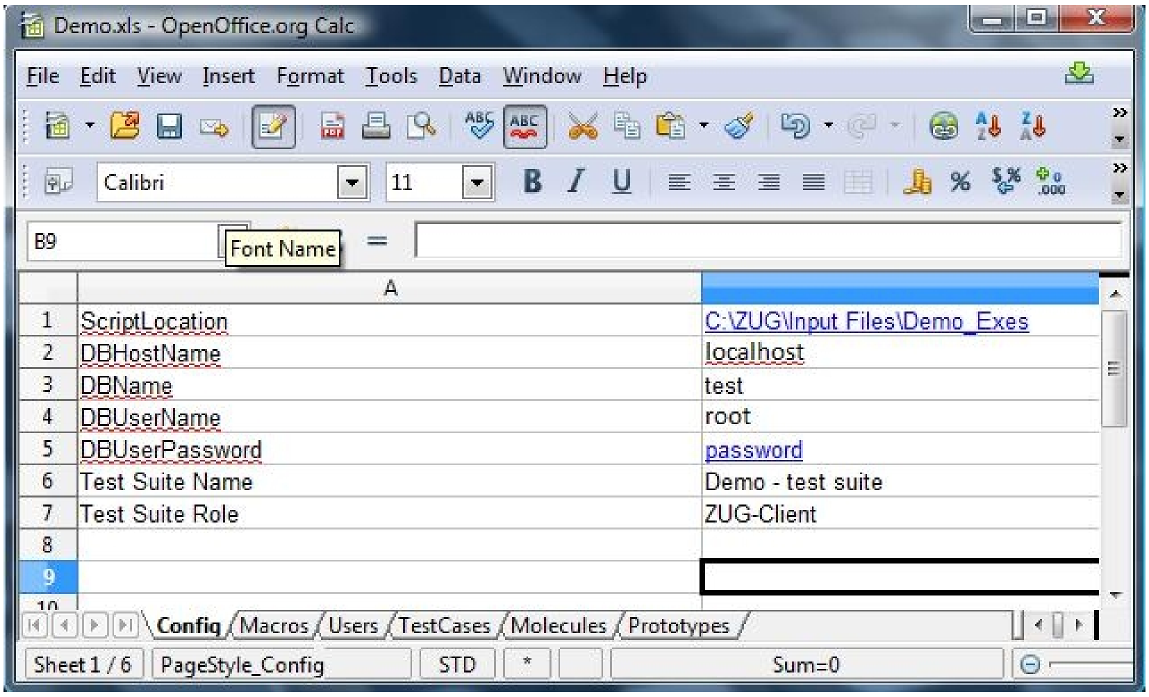
The atoms in Zug can be written in any scripting or programming language. In this manual we have created the atoms using AutoIT, a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting. For web based products, we have our own proprietary libraries named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

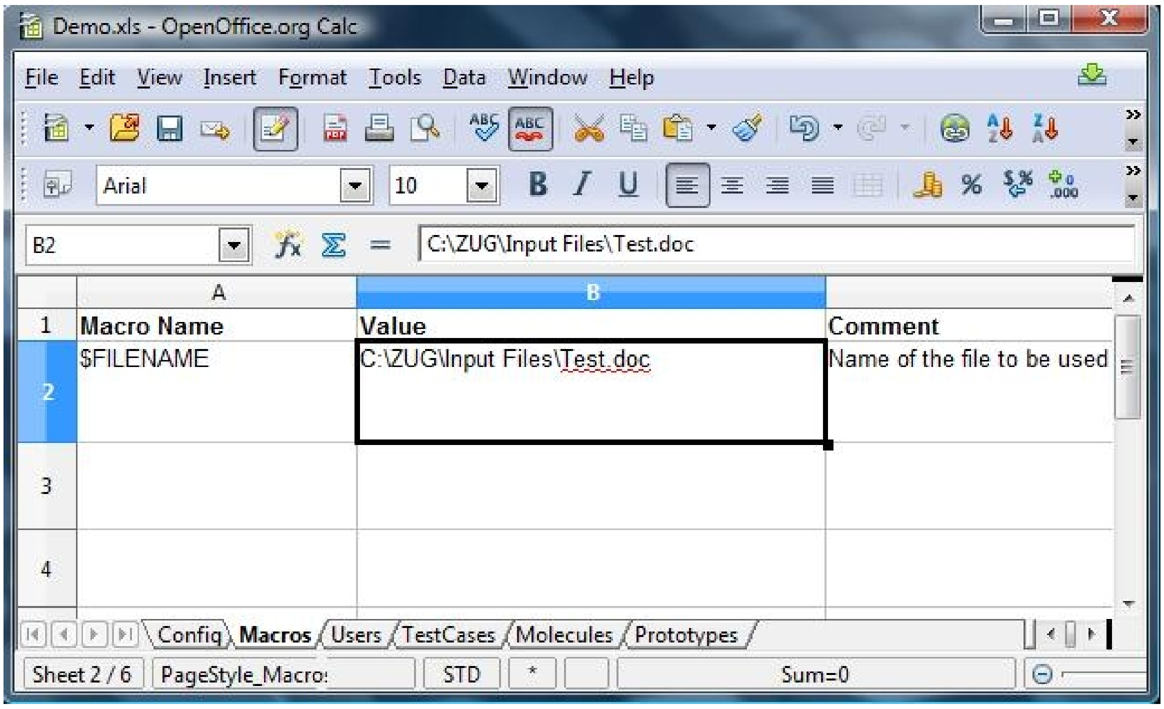

Demo.xls - OpenOffice.org Calc
File Edit View Insert Format Tools Data Window Help
Calibri
11
B9
Font Name
Config Macros Users TestCases Molecules Prototypes
Sheet 1 / 6
PageStyle_Config
STD
Sum=0

	A	
1	ScriptLocation	C:\ZUG\Input Files\Demo_Exes
2	DBHostName	localhost
3	DBName	test
4	DBUserName	root
5	DBUserPassword	password
6	Test Suite Name	Demo - test suite
7	Test Suite Role	ZUG-Client
8		
9		

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

- *ScriptLocation* states location of the folder where the Zug atoms are written for this test.
- *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.
- *DBName* states the database name.
- *DBUserName* states the name of the user to do authentication to database.
- *DBUserPassword* states the password of the user required for SQL authentication.
- *Test Suite Name* states the name of the test suite which was stated in Zermatt.
- *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.

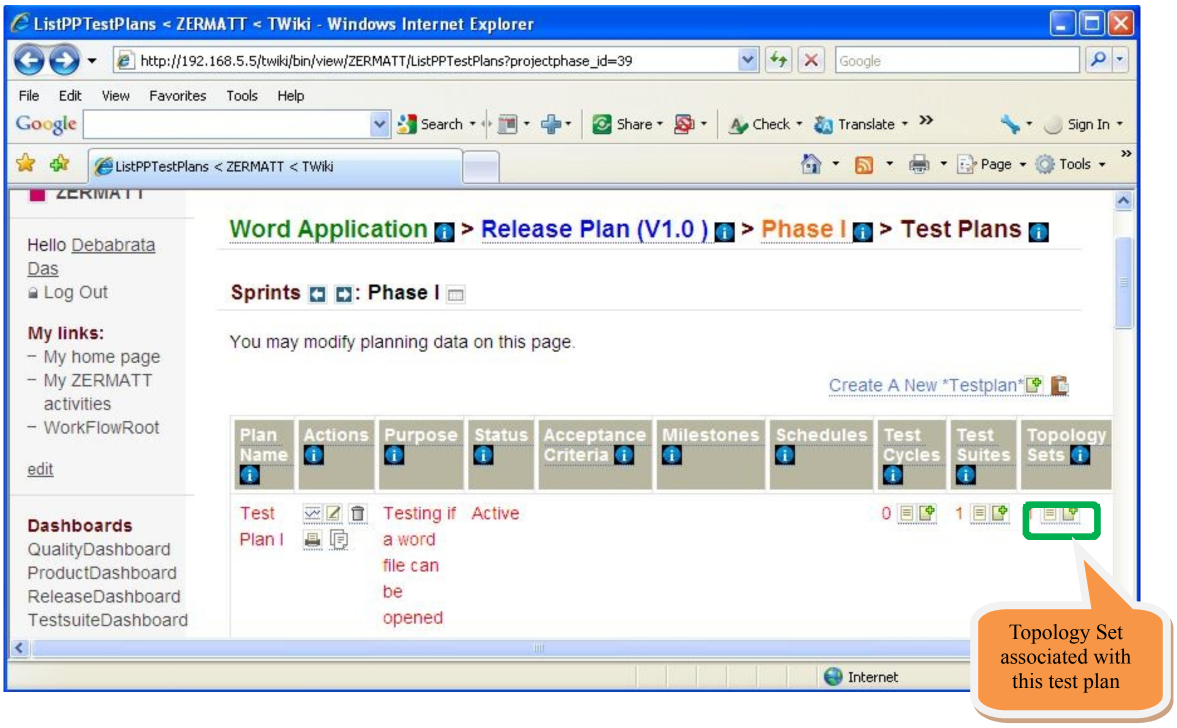

Demo.xls - OpenOffice.org Calc
File Edit View Insert Format Tools Data Window Help
Arial
10
B2
C:\ZUG\Input Files\Test.doc
Config
Macros
Users
TestCases
Molecules
Prototypes
Sheet 2 / 6
PageStyle_Macros
STD
Sum=0

	A	B	C
1	Macro Name	Value	Comment
2	$FILENAME	C:\ZUG\Input Files\Test.doc	Name of the file to be used
3			
4			

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.

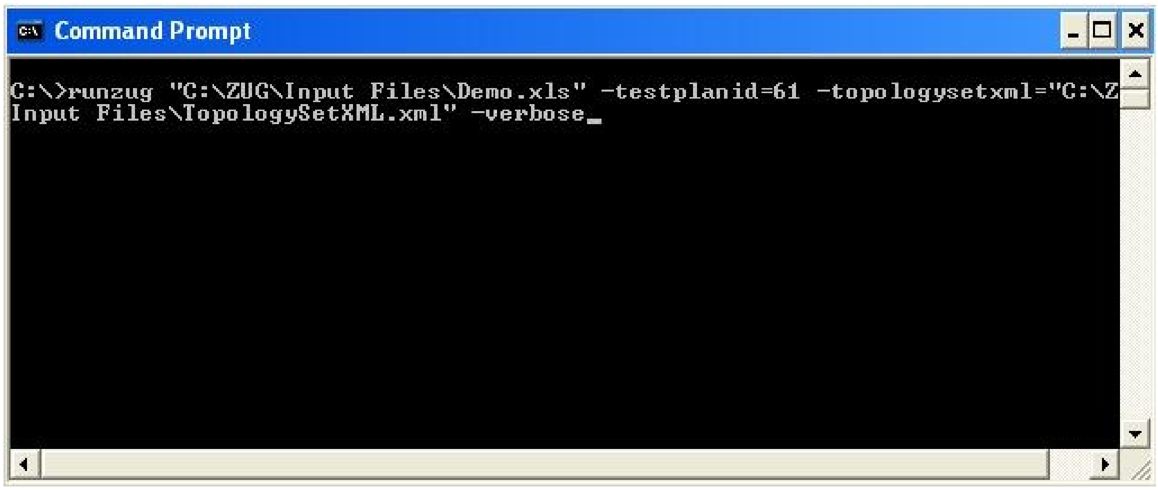

ListPPTestPlans < ZERMATT < TWiki - Windows Internet Explorer
http://192.168.5.5/twiki/bin/view/ZERMATT/ListPPTestPlans?projectphase_id=39
File Edit View Favorites Tools Help
Hello Debabrata Das
Log Out
My links:
– My home page
– My ZERMATT activities
– WorkFlowRoot
edit
Dashboards
QualityDashboard
ProductDashboard
ReleaseDashboard
TestsuiteDashboard
Word Application > Release Plan (V1.0) > Phase I > Test Plans
Sprints : Phase I
Create A New *Testplan*
Internet
Topology Set associated with this test plan

You may modify planning data on this page.

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				0	1	

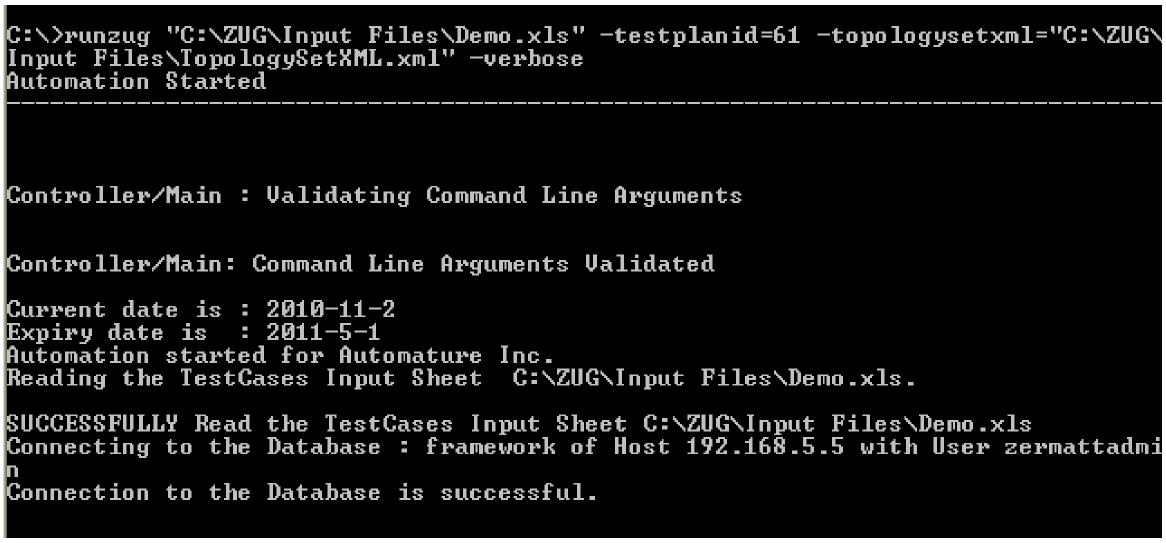
- Open the command prompt.

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



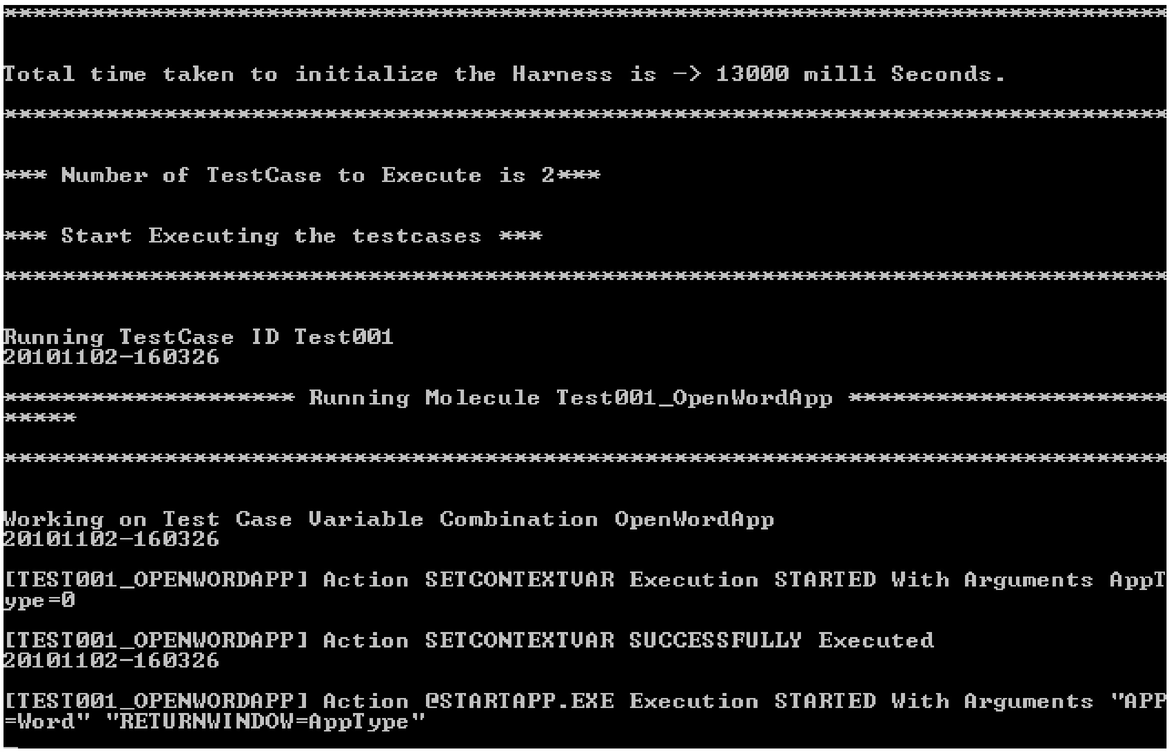
```
C:\>runzug "C:\ZUG\Input Files\Demo.xls" -testplanid=61 -topologysetxml="C:\Z
Input Files\TopologySetXML.xml" -verbose_
```

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



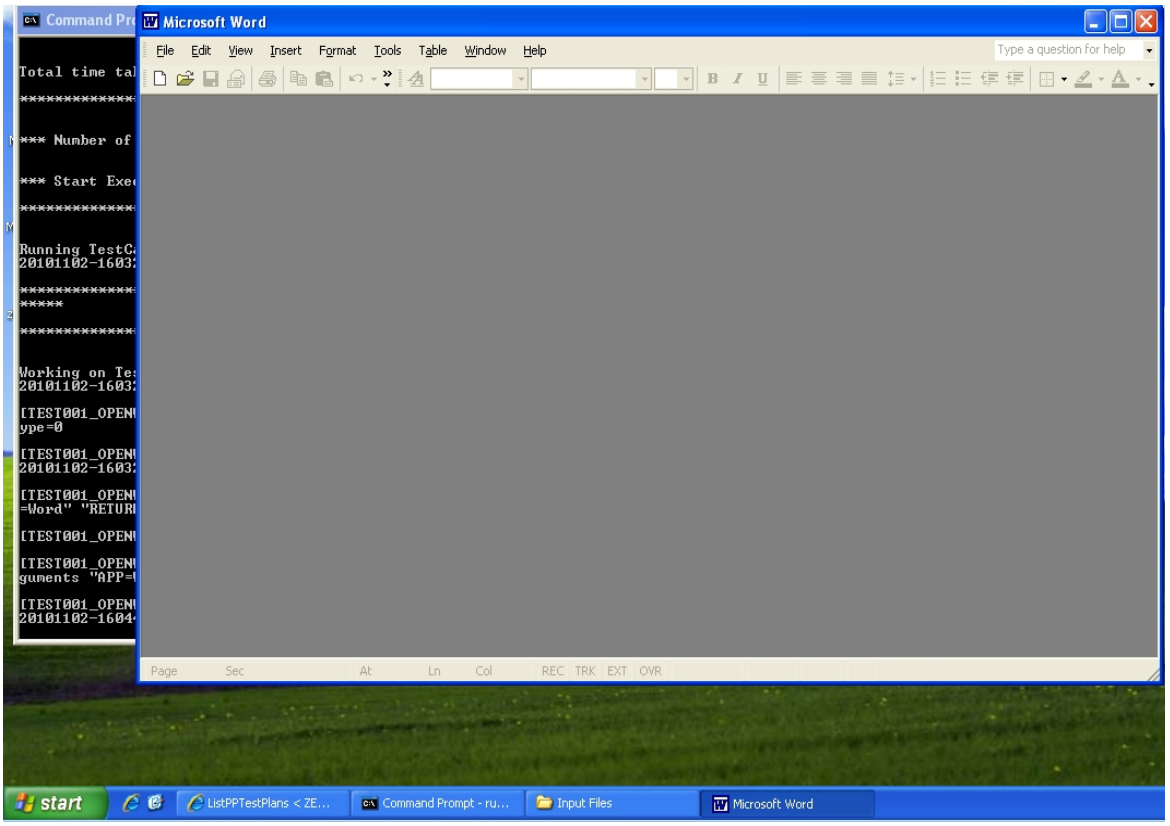
```
C:\>runzug "C:\ZUG\Input Files\Demo.xls" -testplanid=61 -topologysetxml="C:\ZUG\
Input Files\TopologySetXML.xml" -verbose
Automation Started
--------------------------------------------------------------------------------



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.
```

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

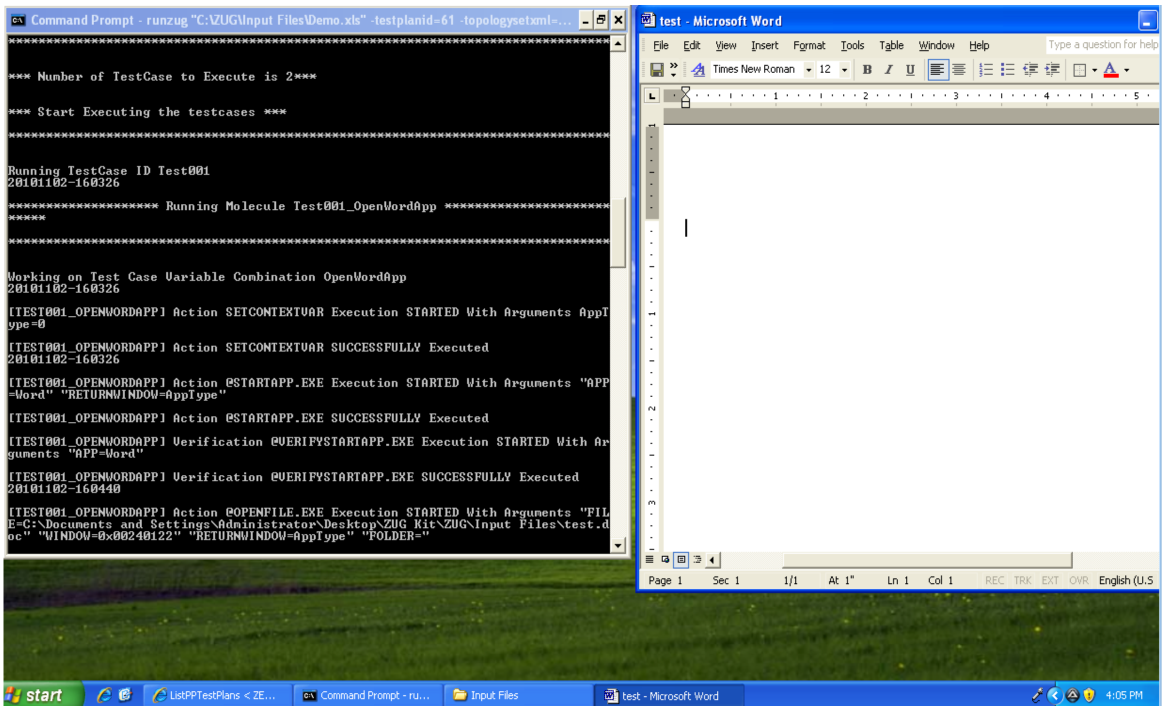


```
********************************************************************************

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 SETCONTEXTVAR Execution STARTED With Arguments AppT
ype=0

[TEST001_OPENWORDAPP] Action SETCONTEXTVAR SUCCESSFULLY Executed
20101102-160326

[TEST001_OPENWORDAPP] Action @STARTAPP.EXE Execution STARTED With Arguments "APP
=Word" "RETURNWINDOW=AppType"
```

- The Word Application is opened.

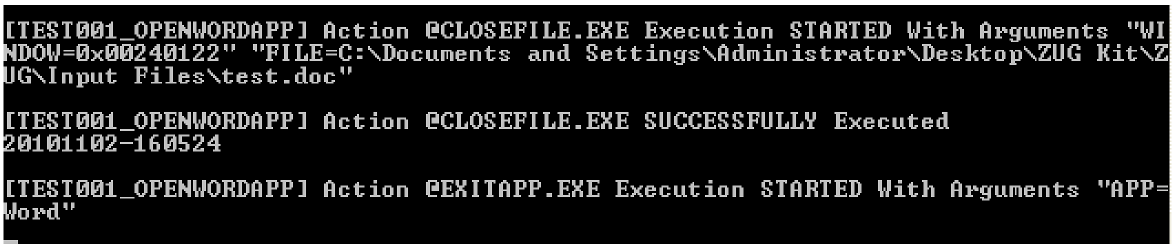


- The Word file is opened by executing the OPENFILE.EXE.
-


Command Prompt - runzug "C:\ZUG\Input Files\Demo.xls" -testplanid=61 -topologysetxml=...
test - Microsoft Word
File Edit View Insert Format Tools Table Window Help
Type a question for help
Times New Roman 12
Page 1 Sec 1 1/1 At 1" Ln 1 Col 1 REC TRK EXT OVR English (U.S
start
ListPPTestPlans < ZE...
Command Prompt - ru...
Input Files
test - Microsoft Word
4:05 PM

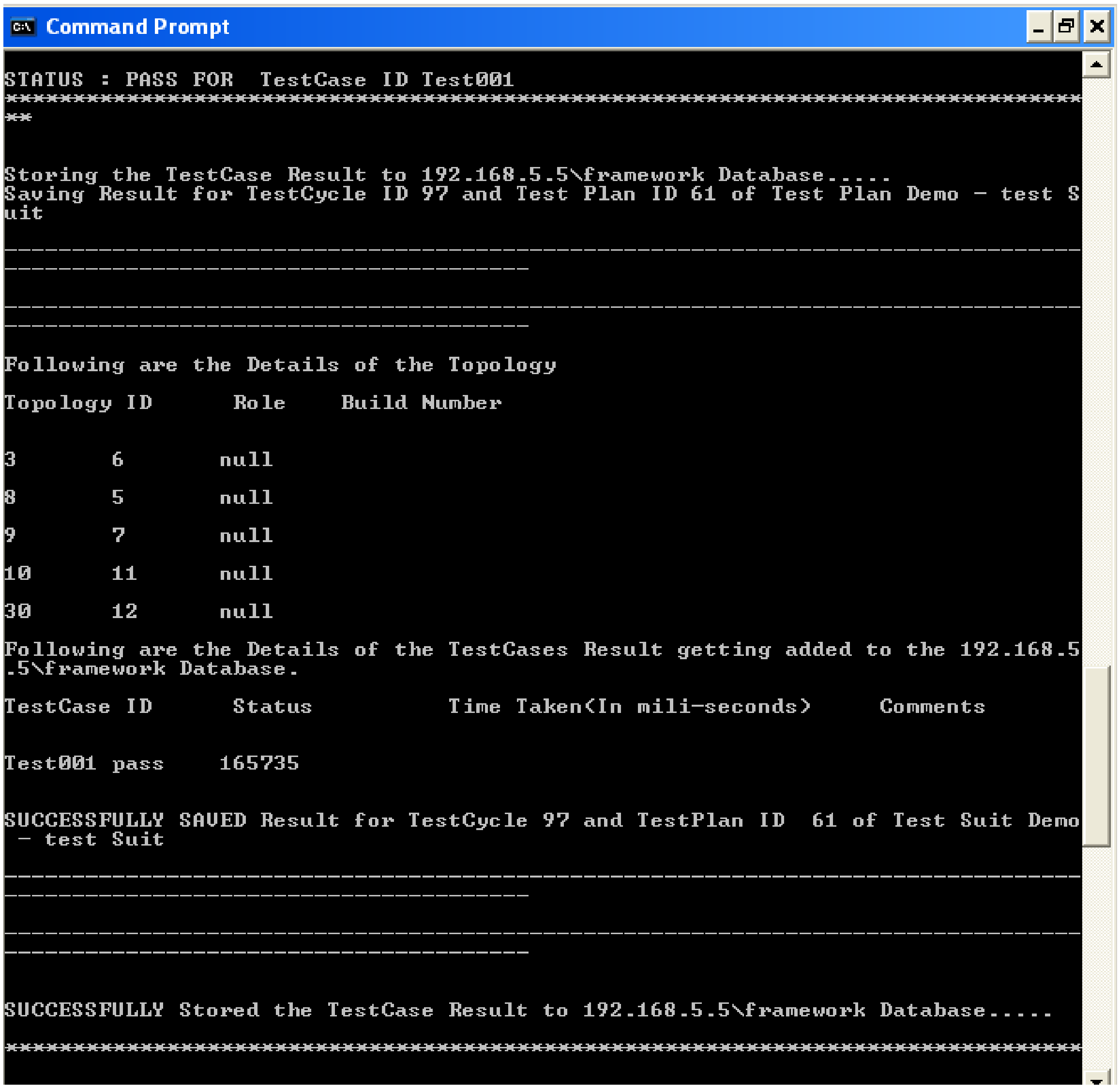
```
********************************************************************************

*** 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 SETCONTEXTVAR Execution STARTED With Arguments AppT
ype=0

[TEST001_OPENWORDAPP] Action SETCONTEXTVAR SUCCESSFULLY Executed
20101102-160326

[TEST001_OPENWORDAPP] Action @STARTAPP.EXE Execution STARTED With Arguments "APP
=Word" "RETURNWINDOW=AppType"

[TEST001_OPENWORDAPP] Action @STARTAPP.EXE SUCCESSFULLY Executed

[TEST001_OPENWORDAPP] Verification @VERIFYSTARTAPP.EXE Execution STARTED With Ar
guments "APP=Word"

[TEST001_OPENWORDAPP] Verification @VERIFYSTARTAPP.EXE SUCCESSFULLY Executed
20101102-160440

[TEST001_OPENWORDAPP] Action @OPENFILE.EXE Execution STARTED With Arguments "FIL
E=C:\Documents and Settings\Administrator\Desktop\ZUG Kit\ZUG\Input Files\test.d
oc" "WINDOW=0x00240122" "RETURNWINDOW=AppType" "FOLDER="
```

- 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.



```
[TEST001_OPENWORDAPP] Action @CLOSEFILE.EXE Execution STARTED With Arguments "WI
NDOW=0x00240122" "FILE=C:\Documents and Settings\Administrator\Desktop\ZUG Kit\Z
UG\Input Files\test.doc"

[TEST001_OPENWORDAPP] Action @CLOSEFILE.EXE SUCCESSFULLY Executed
20101102-160524

[TEST001_OPENWORDAPP] Action @EXITAPP.EXE Execution STARTED With Arguments "APP=
Word"
```

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

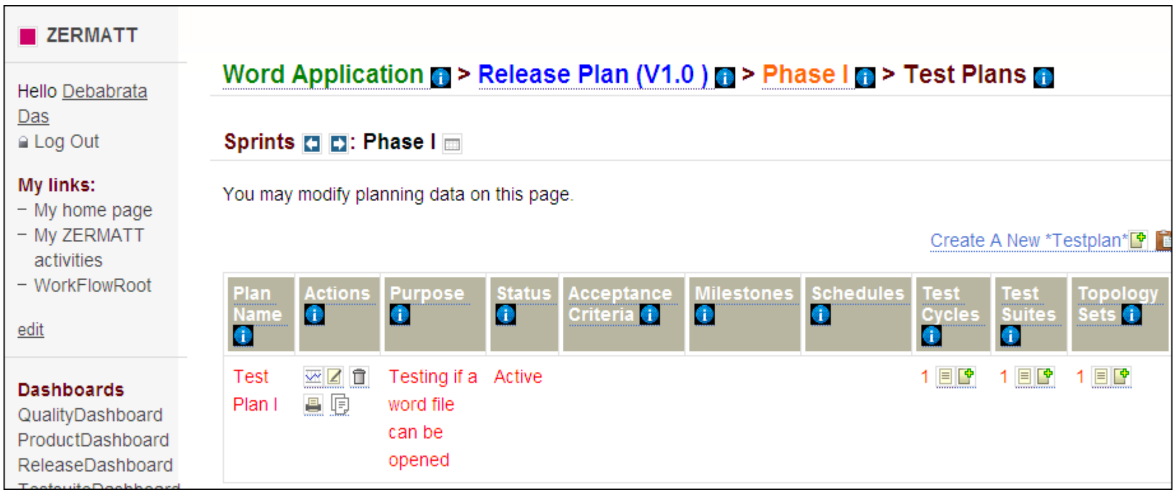

Command Prompt

```
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

- 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.



Word Application > Release Plan (V1.0) > Phase I > Test Plans

Sprints : Phase I

You may modify planning data on this page.

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

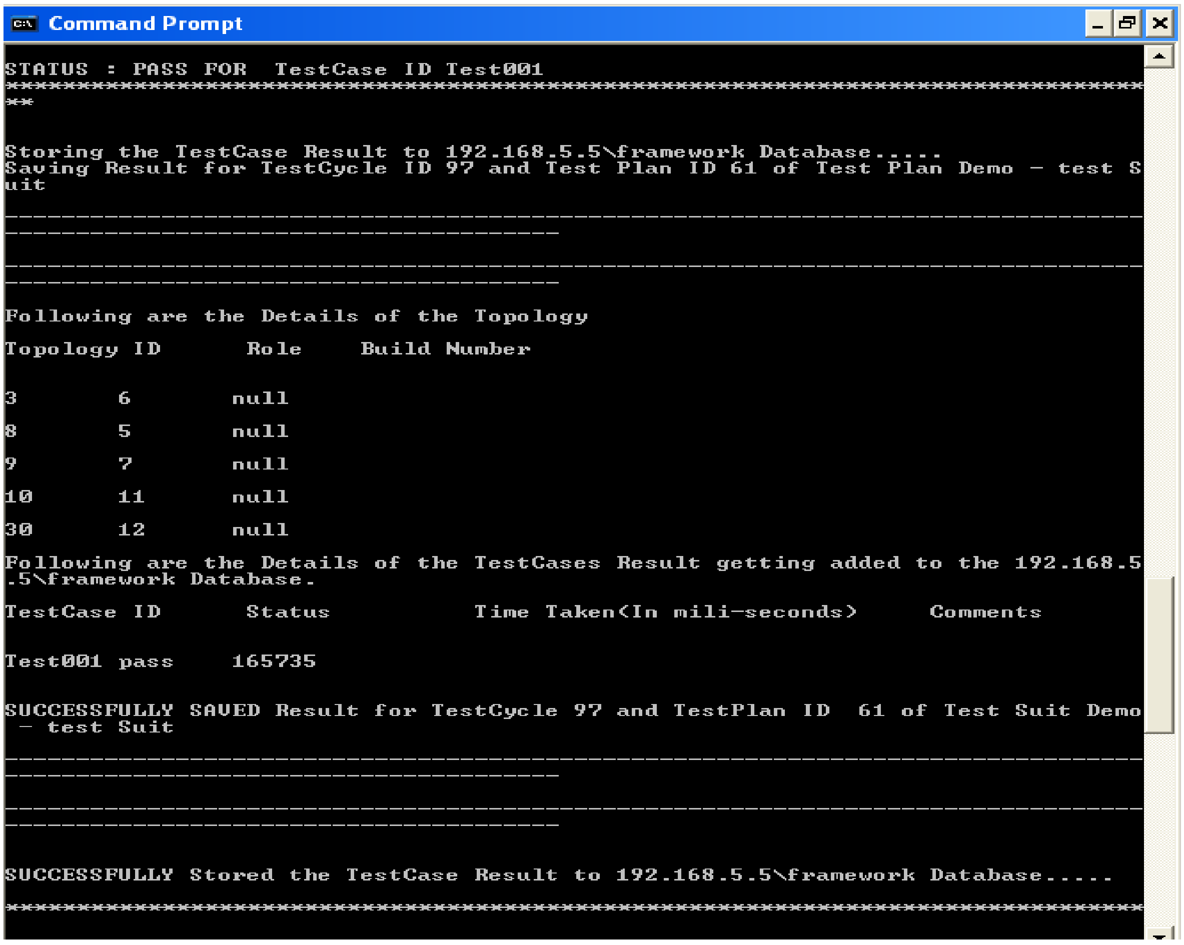
6.6 Interpreting the results

➢ <u>Results in Command Prompt</u>

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



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

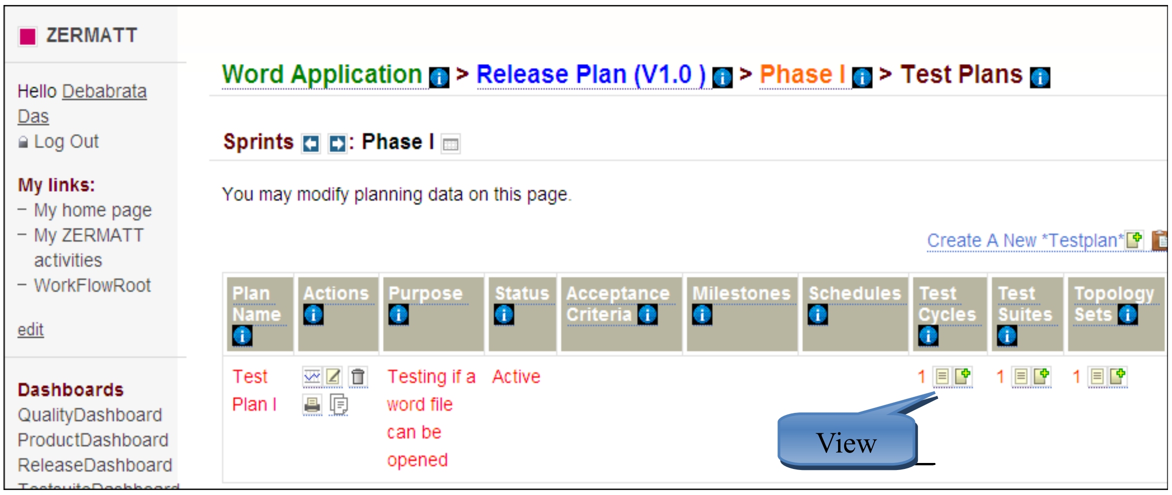
Zug will generate four types of Logs. These logs will be created inside the AppData Folder (Application Data folder of windows OS) of logged in user.

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 yyyymmdd-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 to1.


ZERMATT

Hello Debabrata Das
Log Out

My links:
- My home page
- My ZERMATT activities
- WorkFlowRoot

edit

Dashboards
QualityDashboard
ProductDashboard
ReleaseDashboard

Word Application > Release Plan (V1.0) > Phase I > Test Plans

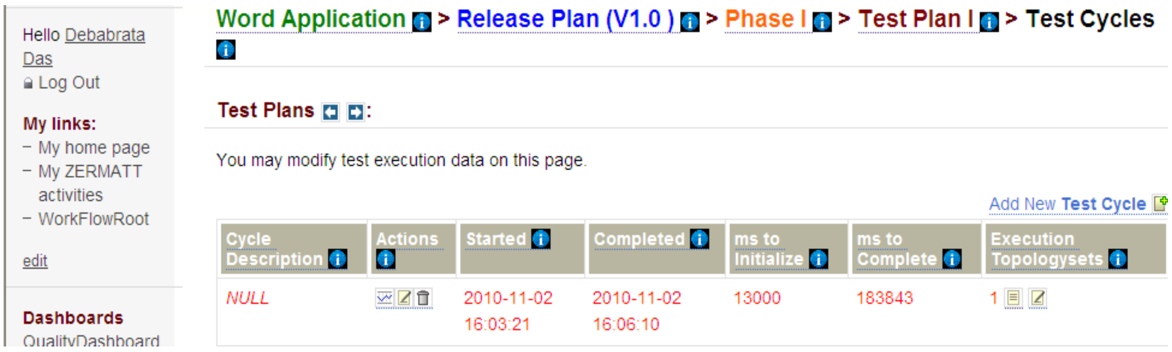
Sprints : Phase I

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

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

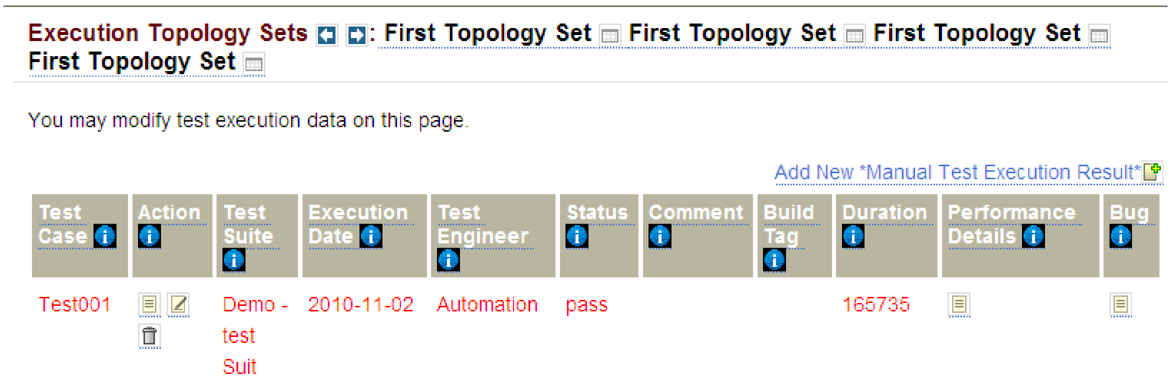
Test Plans :

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.

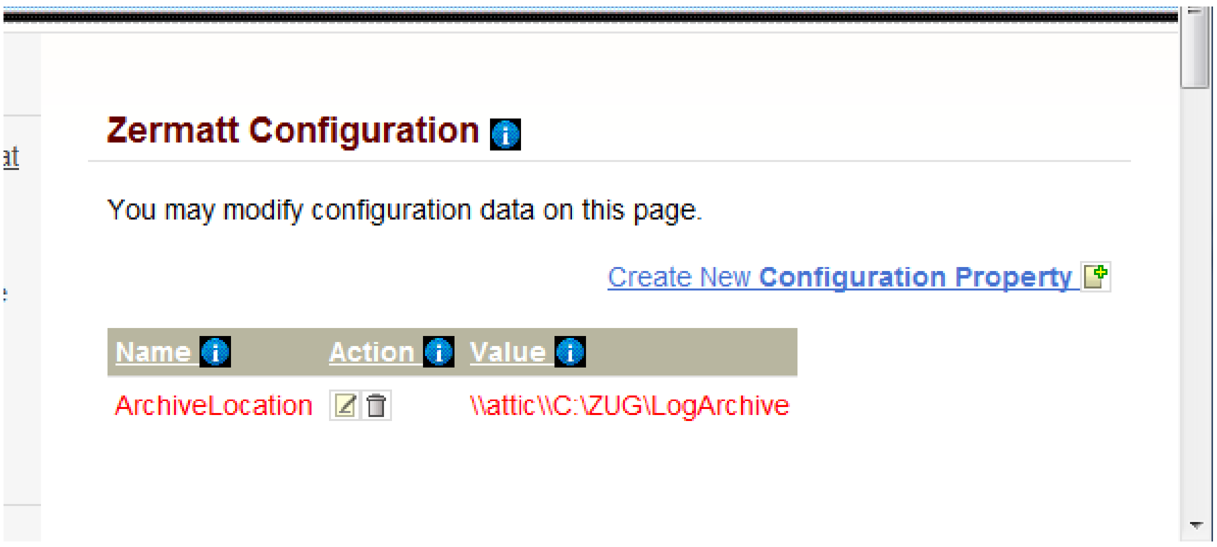
6.7 Archival the Log files

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

- 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.
- 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:



Zermatt Configuration

You may modify configuration data on this page.

Create New Configuration Property

Name	Action	Value
ArchiveLocation		\\attic\\C:\ZUG\LogArchive

7. Troubleshooting

7.1 Debug Log

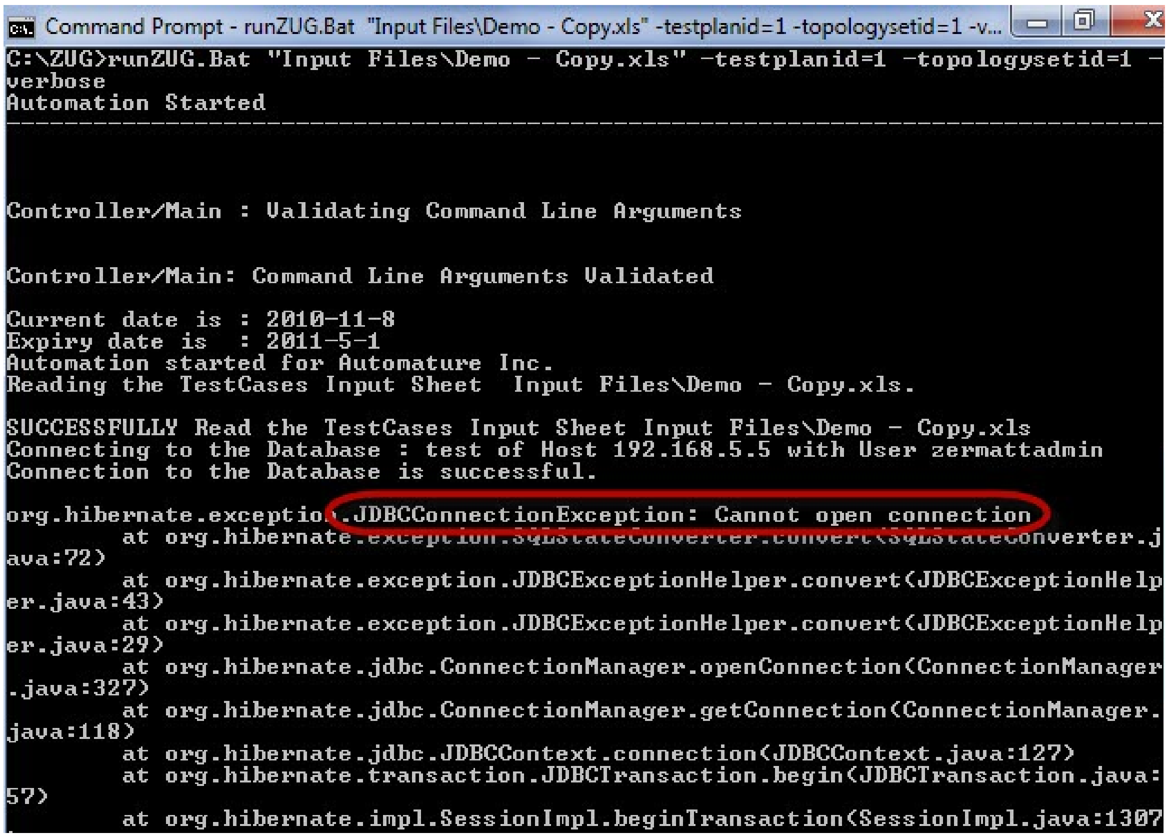
Zug generates different log file. Debug file stores messages which contain extensive contextual information. They are mostly used for problem diagnosis. In failures this file can be referred back to diagnose and investigate problems, exception. The step-by-step execution messages are appended during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

7.2 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

7.3 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 15 6.3 Preparing Zug for Zermatt

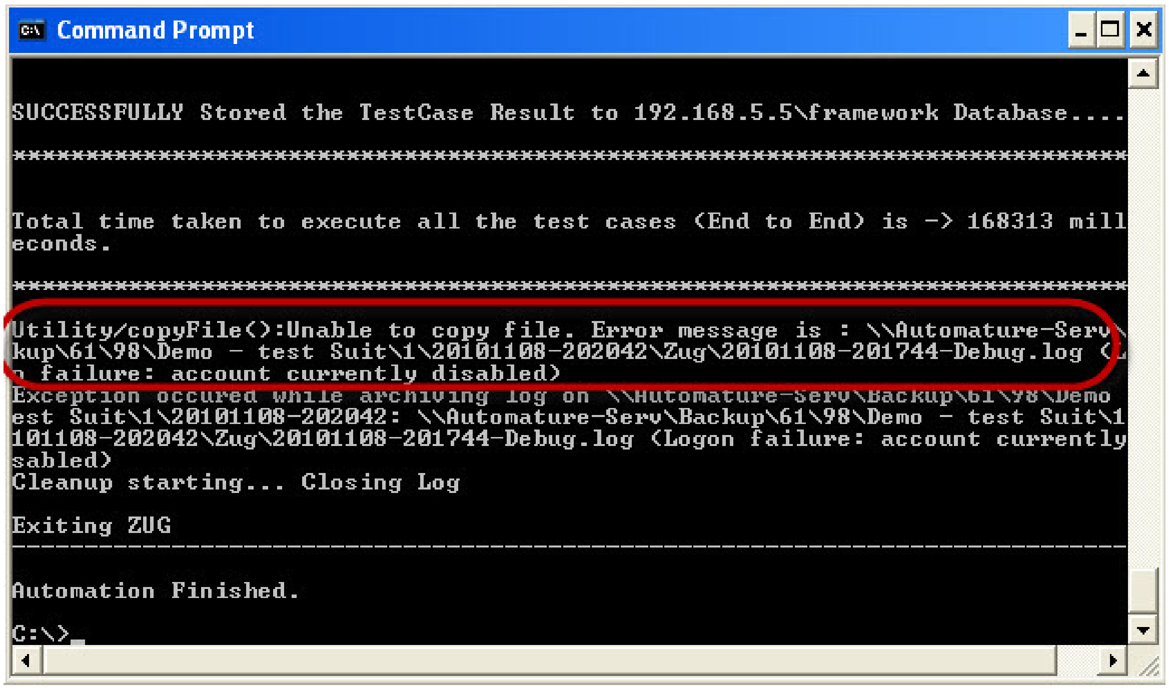


Command Prompt - runZUG.Bat "Input Files\Demo - Copy.xls" -testplanid=1 -topologysetid=1 -v...

```
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
```

7.4 Unable to Archive Log Files

Archiving of Log files are done on location specified in Configuration (Refer page 24 6.7 Archival the Log files) Zug machine should be given proper write-permission for the location specified. An exception can occur as following:


Command Prompt

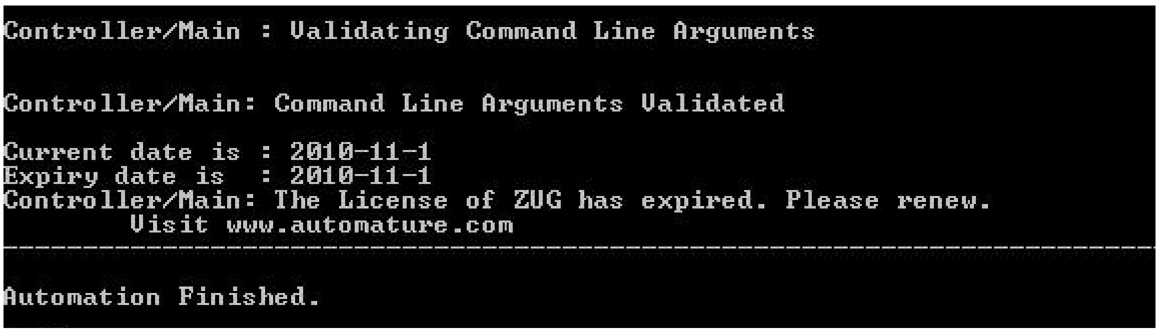
```
SUCCESSFULLY Stored the TestCase Result to 192.168.5.5\framework Database....
****************************************************************************

Total time taken to execute all the test cases (End to End) is -> 168313 mill
econds.
****************************************************************************
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 (L
n failure: account currently disabled)
Exception occured while archiving log on \\Automature-Serv\Backup\61\98\Demo
est Suit\1\20101108-202042: \\Automature-Serv\Backup\61\98\Demo - test Suit\1
101108-202042\Zug\20101108-201744-Debug.log (Logon failure: account currently
sabled)
Cleanup starting... Closing Log

Exiting ZUG
----------------------------------------------------------------------------

Automation Finished.

C:\>_
```

7.5 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.
```

8 ZugINI.xml 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\Builtins</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\Builtins</file-path>
           <dll-name>TestDLL</dll-name>
       </inprocesspackage>
</inprocesspackages>
```

For any COM dll atoms the Program ID is needed.

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>
```

For further problems visit ZUG Forum.