Zug

Day 4

automature

Topics

- Exception Handling
- Test suite Initialization & Cleanup
- Writing Negative Test Cases
- Building up your molecule library

automature

Exception Handling

(because things do go wrong!)

automature

Exception Handling

- **Limitations of Test cases:**
  - If test cases fail in way that it affects the next test cases, it becomes difficult to run the next test case
- **Purpose:**
  - System must be restored to a “safe state” after a failure occurs, e.g. open files must be closed.
  - Cleanup is a good idea, so that subsequent tests can execute from a clean state, and not be adversely affected by previous errors.

automature

Exception Handling

- Initialization & Cleanup for a Test Case/Molecule
- Initialization & Cleanup for a Test Suite

automature

Introducing Step Sequence Index

A Step Sequence Index provides Zug with ordering information regarding the execution sequence of a group of steps

Step sequence Index

TestCase ID	Description	property	User	Step	Action	ActionArg_1	ActionArg_2	ActionArg_3
Login					SetContextVar	Handle		
				1i	@OpenBrowserWithUrl.rb	$URL	Handle	
					@SetTextByName.rb	%Handle%	$USERNAME_FIELDNAME	$MYUSERNAME
					@SetTextByName.rb	%Handle%	$PASSWORD_FIELDNAME	$MYPASSWORD
					@ClickButtonByID.BAT	%Handle%	$LOGIN_BUTTON_ID	
				1c	@CloseBrowser.BAT	%Handle%		

automature

How does a test case fail?

- All out-of-process atoms are expected to return a status code indicating their outcome
  - 0 = OK
  - All other values indicate failure
- All in-process atoms are expected to throw an exception, when it fails
- Unless the test step has indicated otherwise, a failure status code, or an exception, will immediately stop the executing the test case, and report the outcome as a failure.

automature

Exception Handling in a Test Case

- Initialization and cleanup steps are designated in pairs, both having the same step sequence index
- Cleanup actions are matched with their corresponding "initialization" actions using the step sequence index.
- For a given pair, the initialization action is a numeral, post fixed by an "i" and the cleanup action is a numeral, post fixed with a "c".

Initialization & Cleanup steps

TestCase ID	Description	property	User	Step	Action	ActionArg_1	ActionArg_2	ActionArg_3
Login					SetContextVar	Handle		
				1i	@OpenBrowserWithUrl.rb	$URL	Handle	
					@SetTextByName.rb	%Handle%	$USERNAME_FIELDNAME	$MYUSERNAME
					@SetTextByName.rb	%Handle%	$PASSWORD_FIELDNAME	$MYPASSWORD
					@ClickButtonByID.BAT	%Handle%	$LOGIN_BUTTON_ID	
				1c	@CloseBrowser.BAT	%Handle%		

Exception Handling : Test Case

- Whenever an initialization step fails the corresponding cleanup action (bearing the same index) is executed.

1	TestCase ID	Description	property	Step	Action	ActionArg_1	ActionArg_2	Verify	VerifyArg_1	VerifyArg_2
2										
5	TC-002	Create a file		1i	@CreateFile.bat	$file_location	$file_name_1	@VerifyExistence.bat	$file_location	$file_name_1
6					@Wait.bat	$timeout				
7				2i	@CreateFile.bat	$file_location	$file_name_2	@VerifyExistence.bat	$file_location	$file_name_2
8					@Wait.bat	$timeout				
9				3i	@CreateFile.bat	$file_location	$file_name_3	@VerifyExistence.bat	$file_location	$file_name_3
10					@Wait.bat	$timeout				
11				3c	@DeleteFile.bat	$file_location	$file_name_3			
12					@Wait.bat	$timeout				
13				2c	@DeleteFile.bat	$file_location	$file_name_2			
14					@Wait.bat	$timeout				
15				1c	@DeleteFile.bat	$file_location	$file_name_1			
16										

When to use Exception Handling?

Example:

- Create an order just for the test case
  - Initialization: Create the Order
  - Cleanup: Cancel/Delete the Order
  - Exception Handling: If test case fails after order was created, we must remove the order before testcase completes

automature

Test Suite Setup & Teardown

automature

Test Suite Setup & Teardown

- Ability to initialize the test suite, prior to executing any test case.
- Cleanup the environment after all test cases have been executed.
- Initialization and clean up are not reported to the database.

Initialization & Cleanup Test Cases

TestCase ID	Description	property	User	Step	Action	ActionArg_1	ActionArg_2	ActionArg_3
Init					SetContextVar	Handle		
					@OpenBrowserWithUrl.rb	$URL	Handle	
					@SetTextByName.rb	%Handle%	$USERNAME_FIELDNAME	$MYUSERNAME
					@SetTextByName.rb	%Handle%	$PASSWORD_FIELDNAME	$MYPASSWORD
					@ClickButtonByID.BAT	%Handle%	$LOGIN_BUTTON_ID	
Test001					&SearchGoogle	Handle	$$QUERY	
Cleanup					@ClickLinkByText.BAT	%Handle%	$LOGOUT_TEXT	
					@CloseBrowser.BAT	%Handle%		
					UnSetContextVar	Handle		

automature

When to use Test Suite Initialization & Cleanup?

Examples:

- Database connection pooling
  - Initialization:
    - Set up a database connection, that can be subsequently reused by multiple test cases
  - Cleanup:
    - Close the database connection
- Reuse same login session on a browser
  - Initialization:
    - Open a browser, navigate to site, login
  - Cleanup:
    - Logout and close the browser(s)

automature

Negative Testing

automature

What is Negative Testing?

Test cases that verify if the application can handle error conditions gracefully

The test case outcome should be reported as passed when (and only when) a certain test step fails in an expected manner.

automature

Negative Testing

- Negative Test Step
- Negative Action
- Negative Verification

automature

Negative Testing

- **Negative Test Step**
  - To use negative testing feature on any test step put the word "**negative**" in property column
    - Both action and verification must fail in order to pass the test step.
  - **ZUG_EXCEPTION** context variable will be set with the Exception message and the test case will continue to execute

comment	Invalid number of arguments					
ZS001NG3	**Compare** atom- negative testing			SetContextVar	string	
				AppendToContextVar	contextvar=string	Value=10
		negative		Zstring.compare	%string%	
				Zstring.matchSubstring	%ZUG_EXCEPTION%	Argument mismatch
				UnSetContextVar	string	

the error messages by Zug is stored in this Context Variable

blank field

Negative Testing

- **Negative Action**
  - To use negative testing feature on any action place the word **neg-action** or **Negative-Action** in property column
  - Action must fail and verification must pass in order to pass the test step
  - **ZUG_ACTION_EXCEPTION** context variable will be set with the Exception message and the test step will pass

TestCase ID	Description	property	Step	Action	ActionArg_1	ActionArg_2	ActionArg_3	Verify	VerifyArg_1	VerifyArg_2
comment	making the action step fail and verify step pass and the test step outcome should pass									
ZS001NGC5	**Compare** atom- negative testing			SetContextVar	incorrecttext					
				AppendToContextVar	contextvar=incorrecttext	text1=Comparing	text2= string			
		Negative-Action		Zstring.compare	$INCORRECT_TEXT	%incorrecttext%		Zstring.matchSubstring	%ZUG_ACTION_EXCEPTION%	String do not match
				UnSetContextVar	incorrecttext					

Negative Testing

- **Negative Verification**
  - To use negative testing feature on any verification put the word **neg-verify** or **Negative -Verify** in property column
  - Action must pass and verification must fail in order to pass the test step
  - **ZUG_VERIFY_EXCEPTION** context variable will be set with the Exception message and the test step will pass

TestCase ID	Description	property	Step	Action	ActionArg_1	ActionArg_2	ActionArg_3	Verify	VerifyArg_1	VerifyArg_2
comment	making the action step fail and verify step pass and the test step outcome should pass									
ZS001NGC5	**Compare** atom- negative testing			SetContextVar	incorrecttext					
				AppendToContextVar	contextvar=incorrecttext	text1=Comparing	text2= string			
		Negative-Action		Zstring.compare	$INCORRECT_TEXT	%incorrecttext%		Zstring.matchSubstring	%ZUG_ACTION_EXCEPTION%	String do not match
				UnSetContextVar	incorrecttext					

automature

Negative Testing

- Figure shows the various outcomes of a test step for different properties of Zug.

Property	Action	Verification	Test Step Outcome
Negative	Pass	not-executed	FAIL
Negative	Fail	Pass	FAIL
Negative	Fail	Fail	Pass
Negative-Action	Pass	not-executed	FAIL
Negative-Action	Fail	Pass	PASS
Negative-Action	Fail	Fail	FAIL
Negative-Verify	Pass	Pass	FAIL
Negative-Verify	Pass	Fail	Pass
Negative-Verify	Fail	not-executed	FAIL

automature

Conditional Execution

- **Executing a set of test steps based on some condition (success or failure of an atom)**

- **ROS** (Return on Success)
- **ROF** (Return on Failure)

automature

Conditional Execution

- **ROS** (Return on Success)
  - Used in the property of a test step in a molecule.
  - If that test step executes successfully then ZUG skips the rest of the test steps of that molecule
  - Control is returned to the caller of the molecule.

TestCase ID	Description	property	Step	Action	ActionArg_1	ActionArg_2	Verify	VerifyArg_1	VerifyArg_2
TC001				Setcontextvar	String=1				
				&MOL1	String		Zstring.compare	%String%	11
				UnSetcontextvar	String				

Molecule ID	Description	Property	Step	Action	ActionArg_1	ActionArg_2	Verify	VerifyArg_1	VerifyArg_2
MOL1				#define_args	String				
		ROS		AppendtoContextvar	#String	1	Zstring.compare	%#String%	11
				print	%#String%				
				AppendtoContextvar	#String	1	Zstring.compare	%#String%	111

Conditional Execution

- **ROF** (Return on Failure)
  - Used in the property of a test step in a molecule.
  - If that test step fails it gives a exception message where the failure occurs stating the reason of failure of the test step.
  - ZUG skips the rest of the test steps of that molecule
  - The failure of that test step does not affect the outcome of the test case
  - Execution is returned to the caller of the molecule.

Molecule ID	Description	Property	Step	Action	ActionArg_1	ActionArg_2	Verify	VerifyArg_1	VerifyArg_2
MOL2				#define_args	String				
				AppendtoContextvar	#String	1	Zstring.compare	%#String%	11
		ROF		prin	%#String%				
				AppendtoContextvar	#String	1	Zstring.compare	%#String%	11

automature

Conditional Execution Example

- A form submit evokes different reactions depending on the browser being used. Some browsers may display an alert, that needs to be clicked. Other browsers may not show any alerts.
  - Problem: How to handle the different flows?
  - Solution: Use conditional execution
- Exercise: Write a test case which uses conditional execution to handle a sporadic login dialogue

automature

Building Molecule Libraries

automature

External WorkSheets

- **Limitation of single Test Suite Files:**
  - Cannot use molecules that are defined in some other test suite
  - Maintaining multiple copies is error-prone
- **Purpose:**
  - Reuse the existing molecules/Macros by maintaining a single copy.
  - Common testing logic can be leveraged by many test suites.

automature

Example : External Molecules

These molecules are in a file named Example.xls

Molecule ID	Description	Property	User	Step	Action	ActionArg_1	ActionArg_2	ActionArg_3	ActionArg_	ActionArg_5	ActionArg_	ActionArg_7
Login					@OpenBrowserWithUrl.BAT	$input_arg2	$input_arg1					
					@SetTextByName.BAT	%input_arg1%	$input_arg3	$input_arg4				
					@SetTextByName.BAT	%input_arg1%	$input_arg5	$input_arg6				
					@ClickButtonByName.BAT	%input_arg1%	$input_arg7					
YahooLogin					&Login	$input_arg1	$YAHOO_URL	$YAHOO_USER_FIELDNAME	$input_arg2	$YAHOO_PASSWD_FIELDNAME	$input_arg3	$YAHOO_LOGIN_BUTTON_NAME

Filename - Namespace

TestCase ID	Description	property	User	Step	Action	ActionArg	ActionArg_2	ActionArg_3
Test005					SetContextVar	Handle		
					&Example.YahooLogin	Handle	$YAHOO_USER	$YAHOO_PASSWD

File path of Example.xls needs to be mentioned in the config sheet

Test Suite Role	ZUG-Client
Include	C:\ZUG\Input Files\TestSuites\Example.xls

automature

Example : External Macros

Macros may also be defined in an external spreadsheet file

Externally defined macros can be referenced using mechanism
Similar to externally defined molecules

Filename - Namespace

$Example.MyMacro

File path of Example.xls needs to be mentioned in the config sheet

Test Suite Role	ZUG-Client
Include	C:\ZUG\Input Files\TestSuites\Example.xls

automature