



SOFTWARE TESTING METHODOLOGIES LAB

LAB MANUAL

Subject Code : CS615PE
Regulation : R18/JNTUH
Academic Year : 2022-2023

III B. TECH II SEMESTER

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LIST OF EXPERIMENTS

S. No	List of Experiments	PAGE NO
1	Recording in context sensitive mode and analog mode	
2	GUI checkpoint for single property	
3	GUI checkpoint for single object/window	
4	GUI checkpoint for multiple objects	
5	a) Bitmap checkpoint for object/window a) Bitmap checkpoint for screen area	
6	Database checkpoint for Default check	
7	Database checkpoint for custom check	
8	Database checkpoint for runtime record check	
9	a) Data driven test for dynamic test data submission b) Data driven test through flat files c) Data driven test through front grids d) Data driven test through excel test	
10	a) Batch testing without parameter passing b) Batch testing with parameter passing	
11	Data driven batch	
12	Silent mode test execution without any interruption	
13	Test Case For Calculator In Windows Application	

Installation Procedure of Selenium IDE

Selenium IDE

Selenium IDE (Integrated Development Environment) is an open source web automation testing tool under the Selenium Suite. Unlike Selenium WebDriver and RC, it does not require any programming logic to write its test scripts rather you can simply record your interactions with the browser to create test cases. Subsequently, you can use the playback option to re-run the test cases.

Perhaps, creating test cases on Selenium IDE does not require any programming language but when you get to use selenese commands like **runScript**, a little knowledge prior to JavaScript would prove beneficial for you to understand the concepts more clearly

Installation

Install Selenium IDE from either the [Chrome](#) or [Firefox](#) web store.

Launch the IDE

Once installed, launch it by clicking its icon from the menu bar in your browser.

Troubleshooting

Don't see the icon for Selenium IDE in your menu-bar?

Option 1

Make sure the IDE is enabled in your browser's extension settings.

You can get there quickly by typing the following into your address bar and hitting Enter.

- Chrome: chrome://extensions
- Firefox: about:addons

Option 2

The extension might be enabled but the icon is hidden. Try resizing the menu bar to give it more space.

In Chrome, you can do this by clicking to the right of the address bar, holding the click, and dragging it left or right.

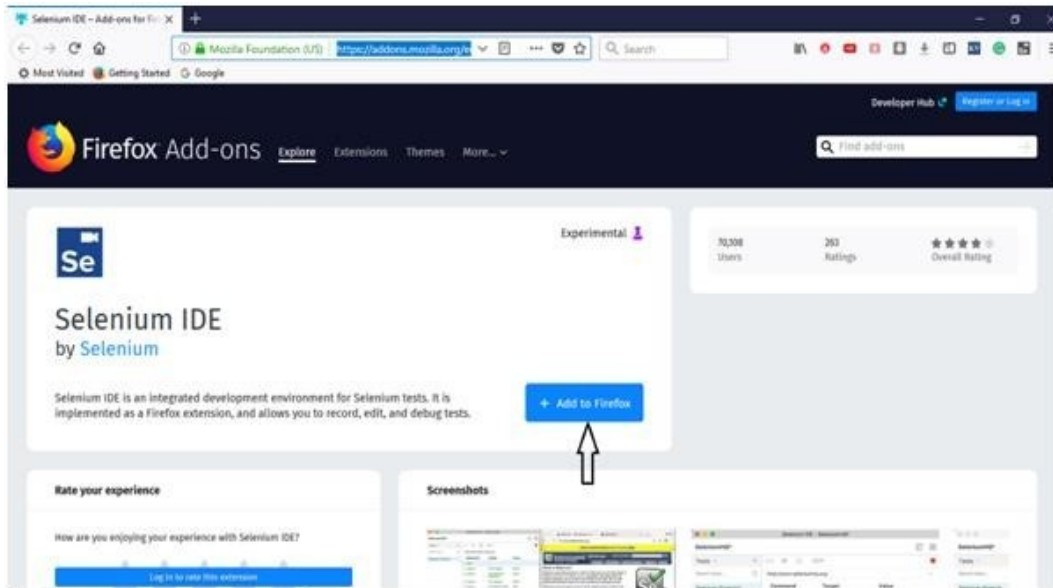
In Firefox you need to right-click, click Customize, make adjustments to the menu bar, and click Done.

What you need

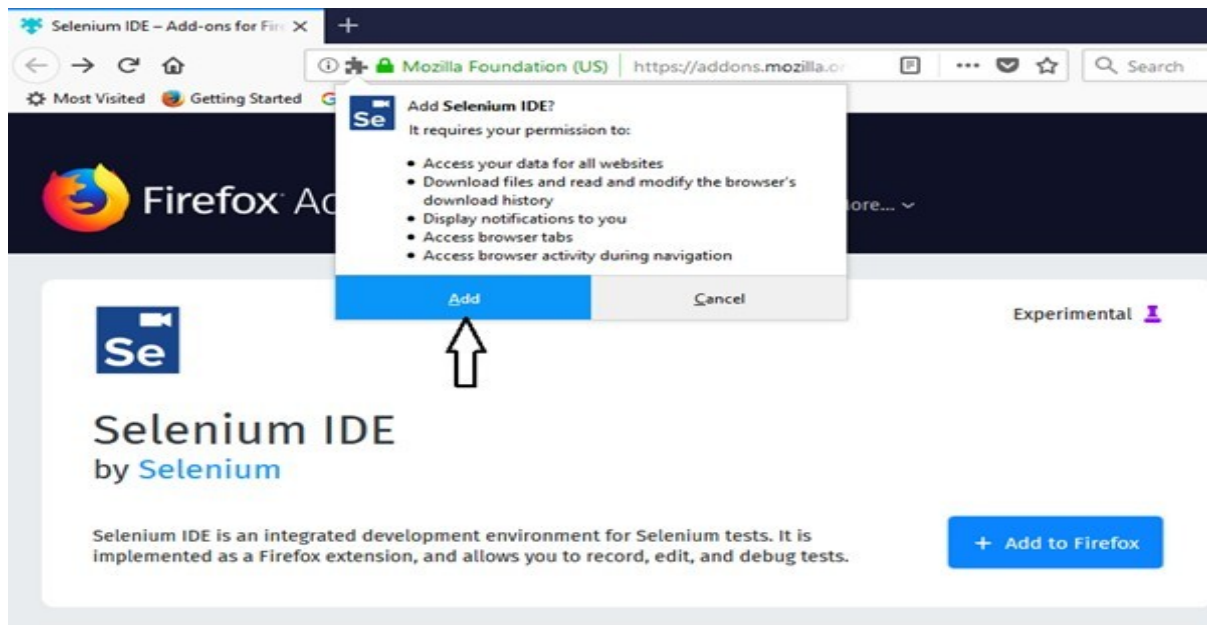
- Mozilla Firefox
- Active Internet Connection

If you do not have Mozilla Firefox yet, you can download it from <http://www.mozilla.org/en-US/firefox/new>.

Steps 1) Launch Firefox and navigate to <https://addons.mozilla.org/en-US/firefox/addon/selenium-ide/>.
Click on Add to Firefox

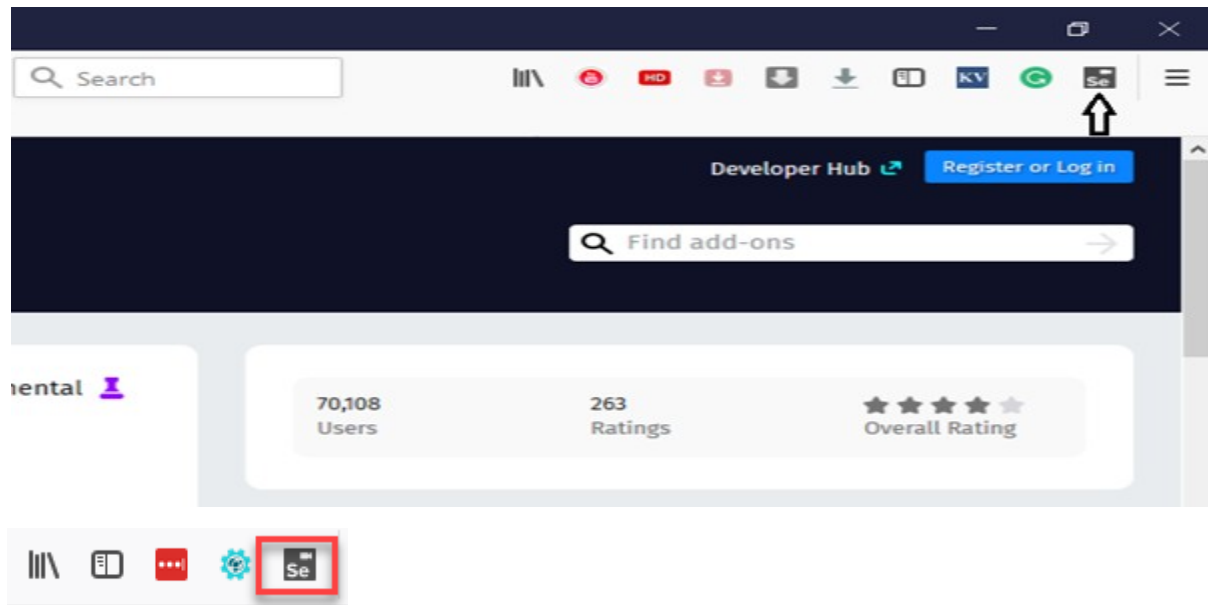


Steps 2) Wait until Firefox completes the download and then click "Add."



Steps 3) Once install is complete, you will get a confirmation message. Click "OK"

Steps 4) Click on the Selenium IDE icon



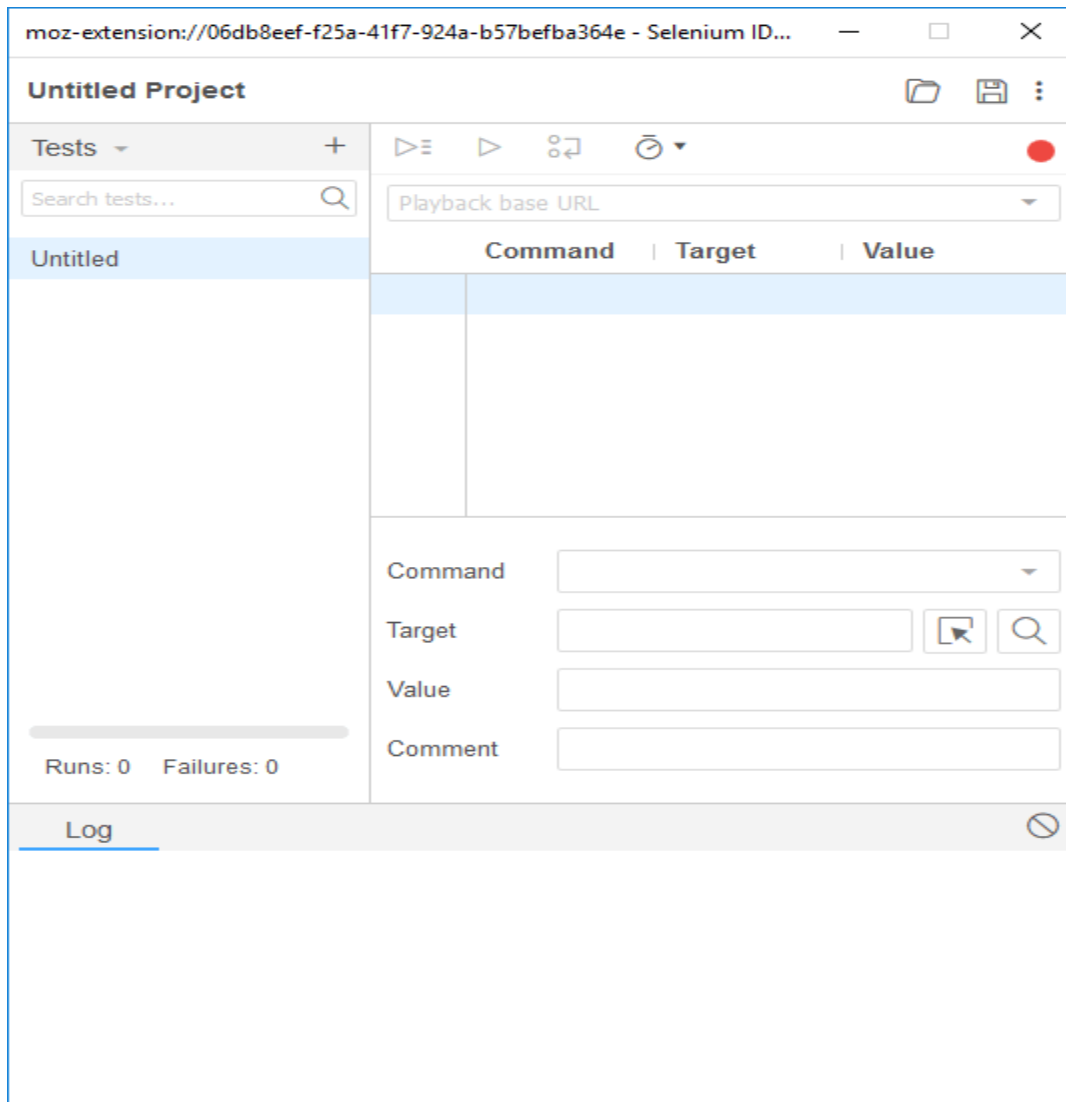
Welcome Screen

Upon launching the IDE you will be presented with a welcome dialog.

This will give you quick access to the following options:

- Record a new test in a new project
- Open an existing project
- Create a new project
- Close the IDE

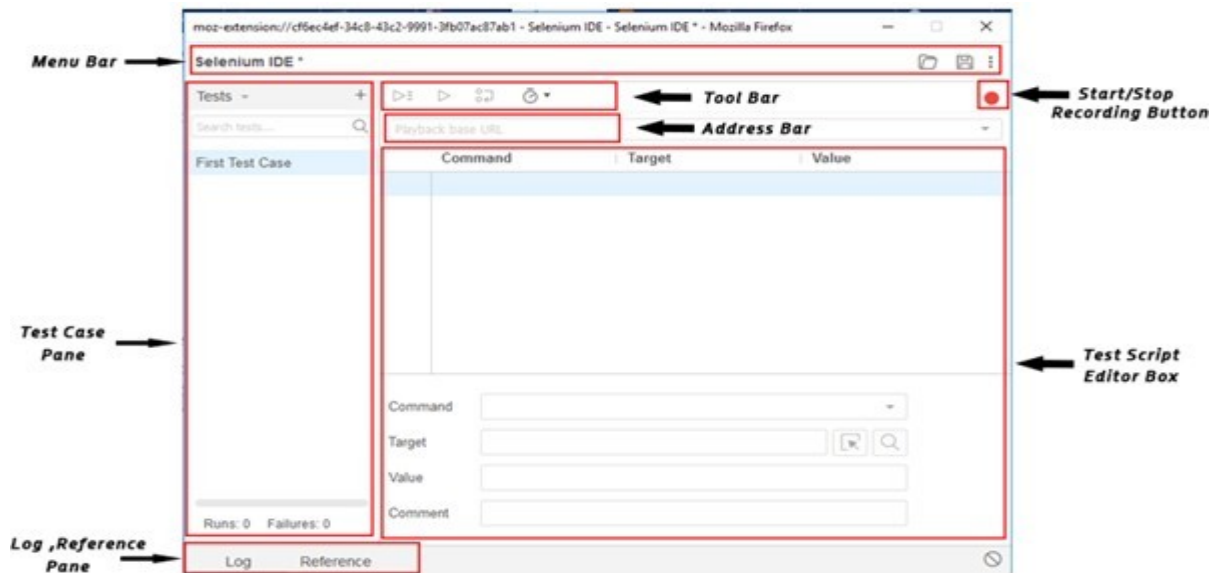
If this is your first time using the IDE (or you are starting a new project), then select the first option.



Selenium IDE-Features

Selenium IDE is divided into different components, each having their own features and functionalities. We have categorized seven different components of Selenium IDE, which includes:

1. Menu Bar
2. Tool Bar
3. Address Bar
4. Test Case Pane
5. Test Script Editor Box
6. Start/Stop Recording Button
7. Log, Reference Pane



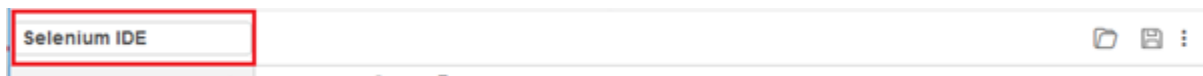
Now, we will look at the features and functionalities of each component in detail.

1. Menu Bar

Menu bar is positioned at the top most portion of the Selenium IDE interface. The most commonly used modules of menu bar include:

- Project Name

It allows you to rename your entire project.



- Open Project

It allows you to load any existing project from your personal drives.



- Save Project

It allows you to save the entire project you are currently working on.

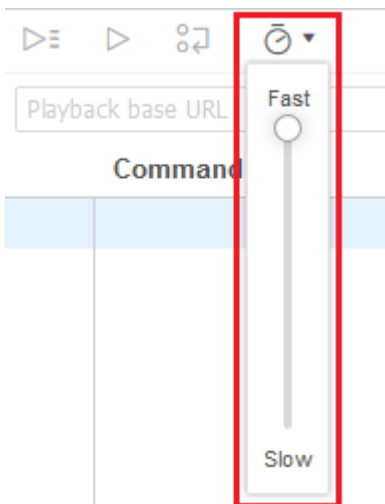


2. Tool Bar

The Tool bar contains modules for controlling the execution of your test cases. In addition, it gives you a step feature for debugging you test cases. The most commonly used modules of Tool Bar menu include:

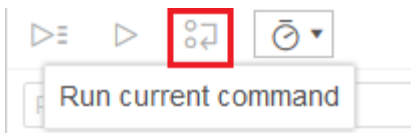
- Speed Control Option

It allows you to control the execution speed of your test cases.



- Step Feature

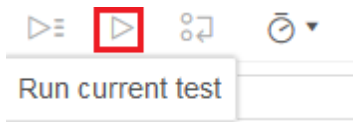
It allows you to "step" through a test case by running it one command at a time. Use for debugging test cases.



- Run Tests

It allows you to run the currently selected test. When only a single test is loaded "Run Test" button

and "Run all" button have the same effect.



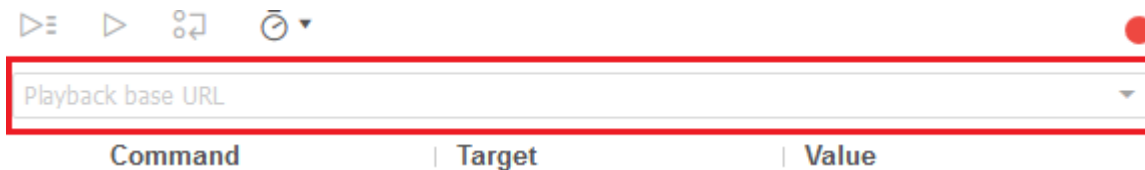
- Run All

It allows you to run the entire test suite when a test suite with multiple test cases is loaded.



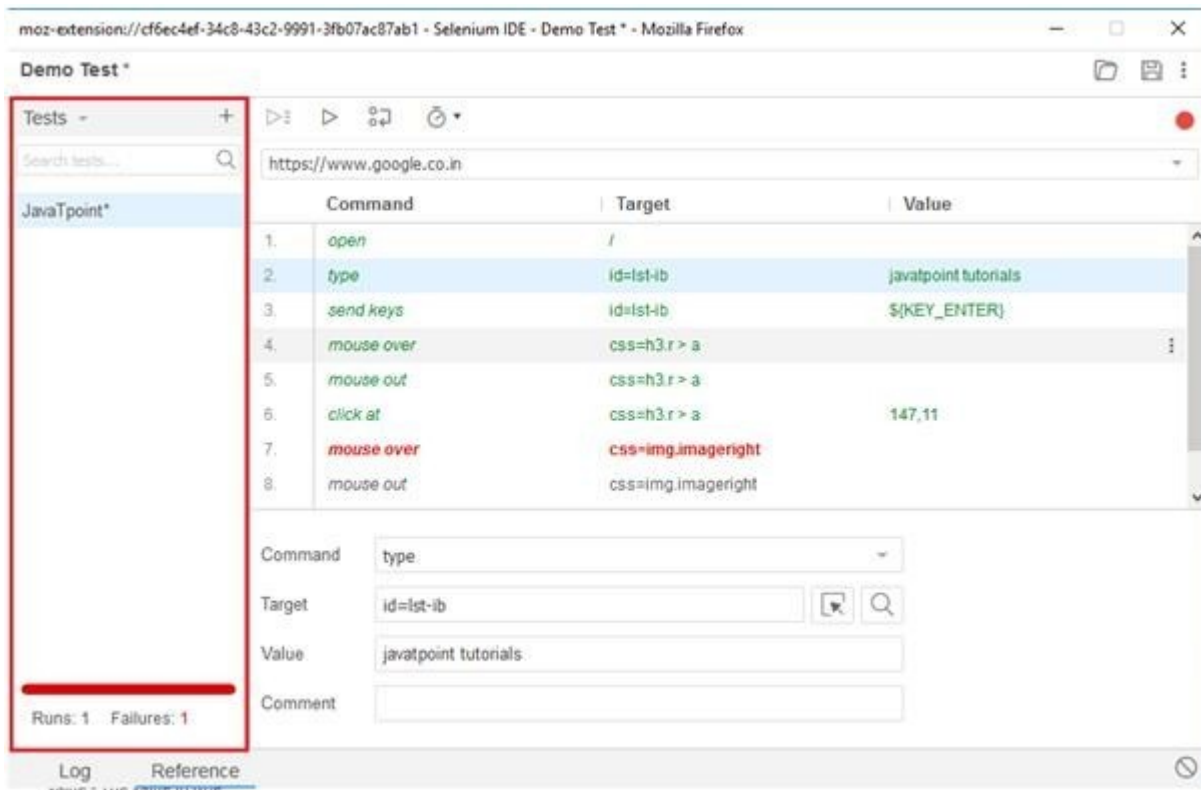
3. Address Bar

This module provides you a dropdown menu that remembers all previous values for base URL. In simple words, the base URL address bar remembers the previously visited websites so that the navigation becomes easy later on.



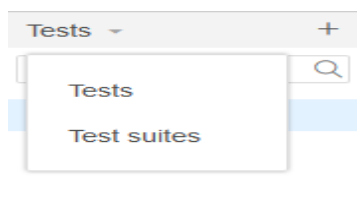
4. Test Case Pane

This module contains all the test cases that are recorded by IDE. In simple words, it provides the list of all recorded test cases at the same time under the test case pane so that user could easily shuffle between the test cases.



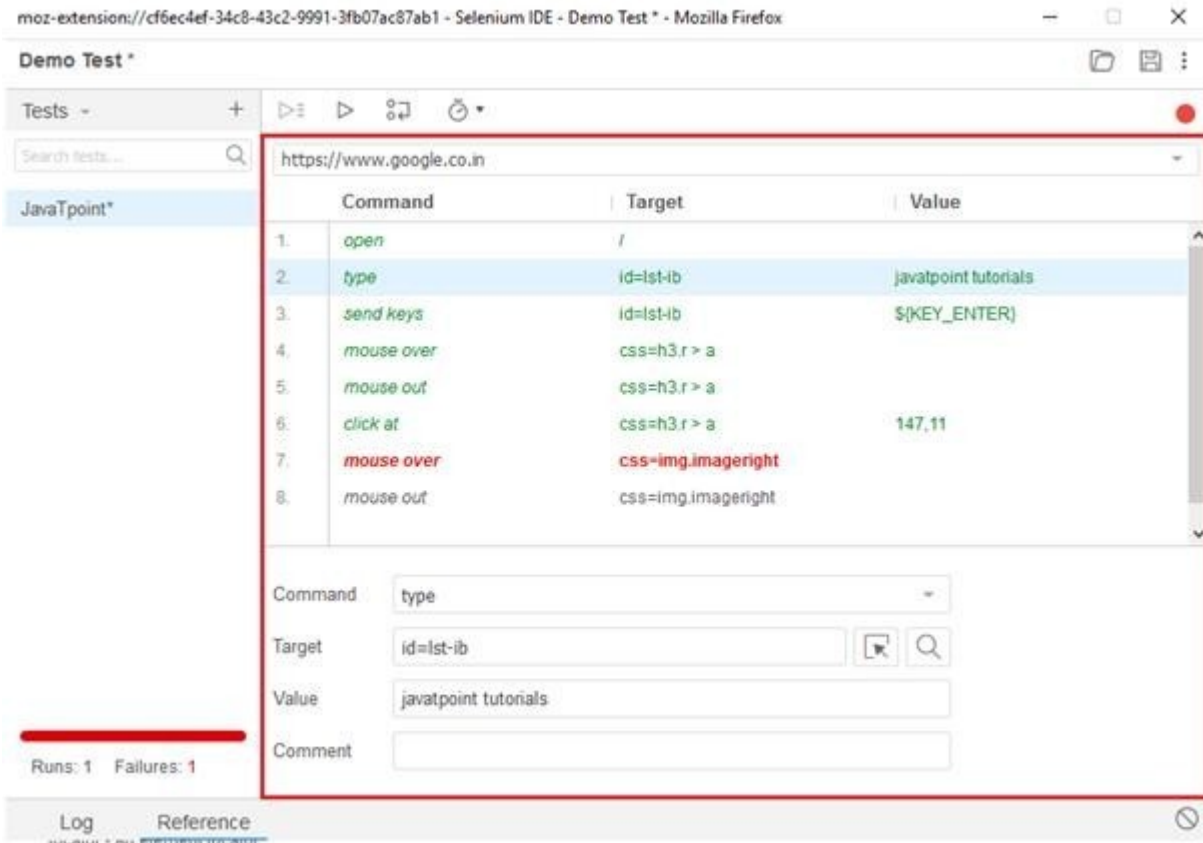
At the bottom portion of the Test Case Pane, you can see the test execution result summary which includes the pass/fail status of various test cases.

Test Case Pane also includes features like Navigation panel which allow users to navigate between test cases and test suites.



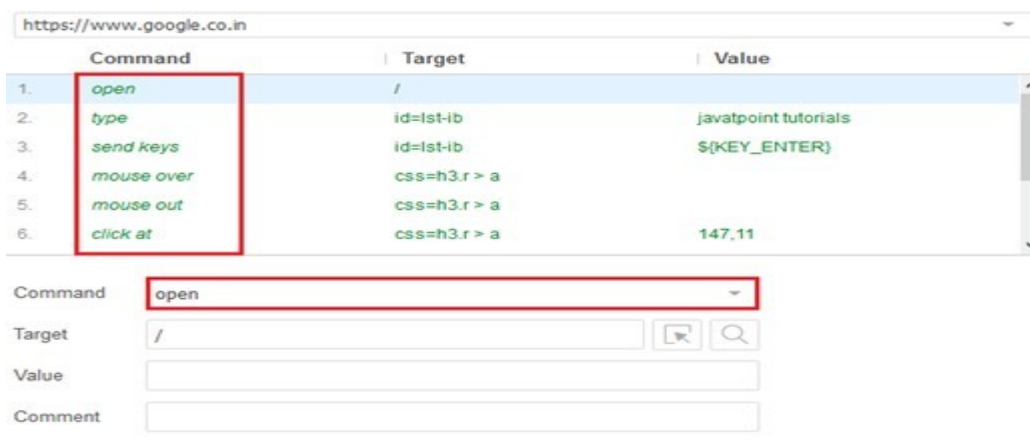
5. Test Script Editor Box

Test Script Editor Box displays all of the test scripts and user interactions that were recorded by the IDE. Each user interaction is displayed in the same order in which they are performed. The Editor box is divided into three columns: Command, Target and Value.



- Command:

Command can be considered as the actual operation/action that is performed on the browser elements. For instance, if you are opening a new URL, the command will be 'open'; if you are clicking on a link or a button on the web page, then the command will be 'clicked'.



- Target:

Target specifies the web element on which the operation has to be performed along with a locator attribute. For instance, if you are clicking on a button called javaTpoint, then the target link will be 'javaTpoint'.

https://www.google.co.in

	Command	Target	Value
1.	open	/	
2.	type	id=lst-ib	javatpoint tutorials
3.	send keys	id=lst-ib	\$(KEY_ENTER)
4.	mouse over	css=h3.r > a	
5.	mouse out	css=h3.r > a	
6.	click at	css=h3.r > a	147,11

Command	type
Target	id=lst-ib
Value	javatpoint tutorials
Comment	

- Value:

Value is treated as an optional field and can be used when we need to send some actual parameters. For instance, if you are entering the email address or password in a textbox, then the value will contain the actual credentials.

https://www.google.co.in

	Command	Target	Value
1.	open	/	
2.	type	id=lst-ib	javatpoint tutorials
3.	send keys	id=lst-ib	\$(KEY_ENTER)
4.	mouse over	css=h3.r > a	
5.	mouse out	css=h3.r > a	
6.	click at	css=h3.r > a	147,11

Command	type
Target	id=lst-ib
Value	javatpoint tutorials
Comment	

6. Start/Stop Recording Button

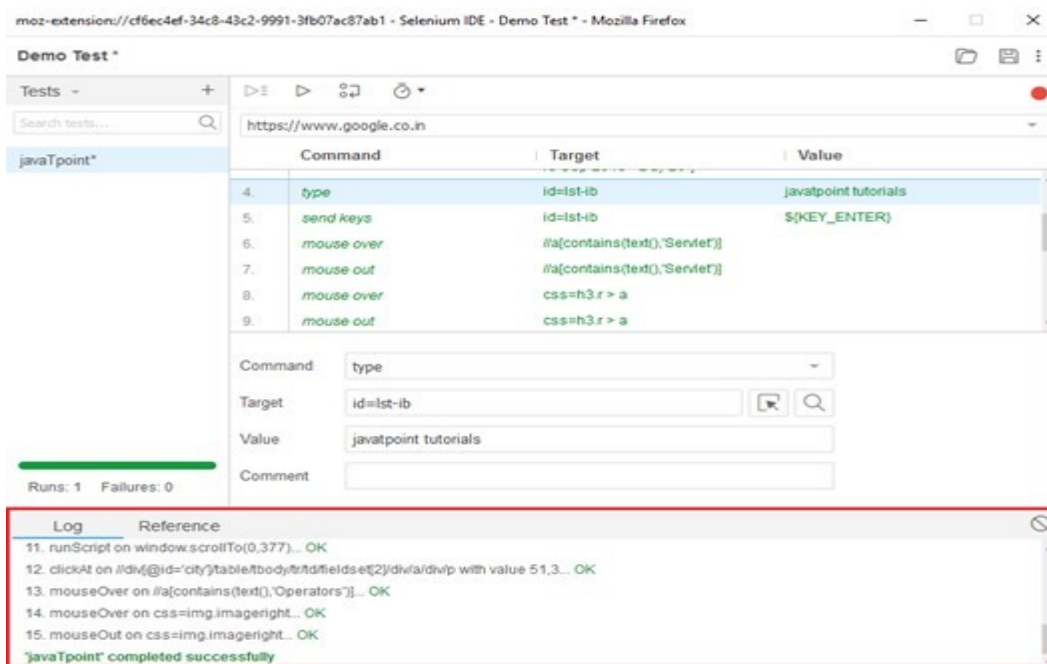
Record button records all of the user actions with the browser.



7. Log, Reference Pane

The Log Pane displays the runtime messages during execution. It provides real-time updates of the actions performed by the IDE. It can be categorized into four types: info, error, debug and warn.

The reference Pane displays the complete detail of the currently selected selenese command in the editor.



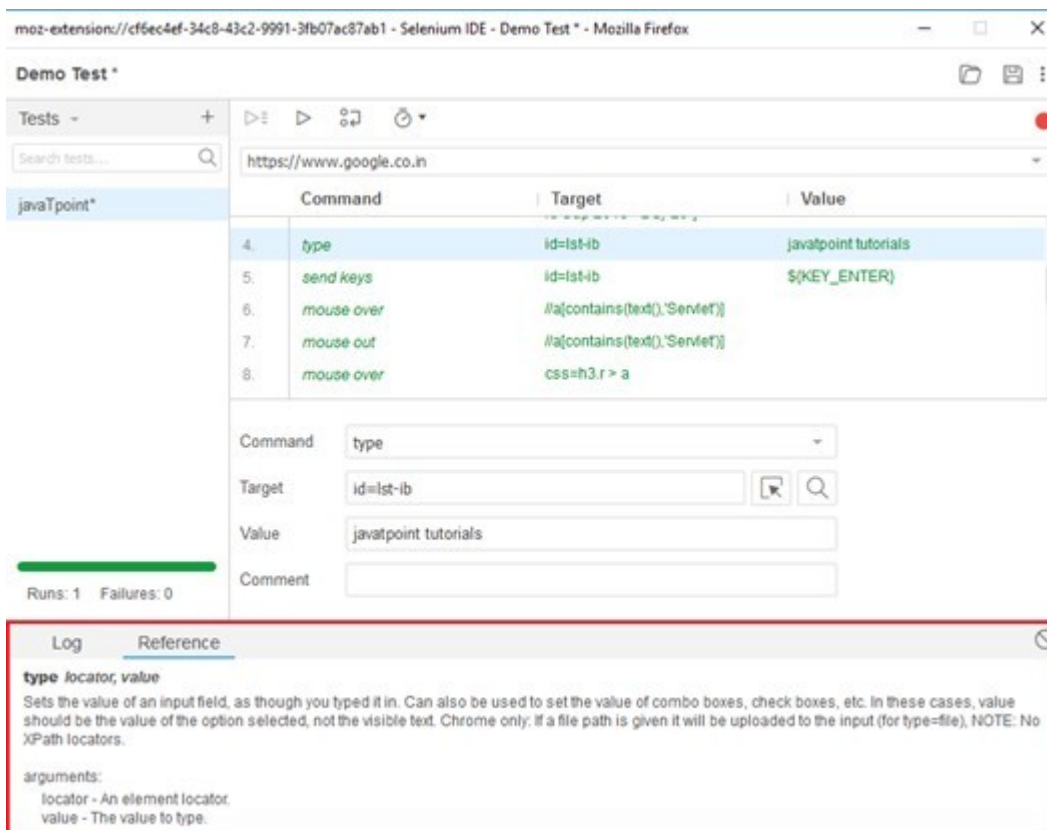
The screenshot shows the Selenium IDE interface with the 'Demo Test' suite selected. The 'Log' pane is active, displaying a list of executed commands and their status. The commands are:

Command	Target	Value
4. type	id=lst-lb	javatpoint tutorials
5. send keys	id=lst-lb	\$(KEY_ENTER)
6. mouse over	//a[contains(text(),'Servlet']	
7. mouse out	//a[contains(text(),'Servlet']	
8. mouse over	css=h3.r > a	
9. mouse out	css=h3.r > a	

The 'Log' pane also shows the following messages:

- 11. runScript on window.scrollTo(0,377)... OK
- 12. clickAt on //div[@id='city']/table/tbody/tr/td/fieldset[2]/div/a/div/p with value 51,3... OK
- 13. mouseOver on //a[contains(text(),'Operators')]... OK
- 14. mouseOver on css=img.imageright... OK
- 15. mouseOut on css=img.imageright... OK

The 'Log' pane is titled 'Log' and 'Reference'. The 'Reference' pane is currently empty.



The screenshot shows the Selenium IDE interface with the 'Demo Test' suite selected. The 'Reference' pane is active, displaying detailed information about the 'type' command. The commands in the table are:

Command	Target	Value
4. type	id=lst-lb	javatpoint tutorials
5. send keys	id=lst-lb	\$(KEY_ENTER)
6. mouse over	//a[contains(text(),'Servlet']	
7. mouse out	//a[contains(text(),'Servlet']	
8. mouse over	css=h3.r > a	

The 'Reference' pane is titled 'Log' and 'Reference'. The 'Log' pane is currently empty. The 'Reference' pane contains the following information:

type locator, value

Sets the value of an input field, as though you typed it in. Can also be used to set the value of combo boxes, check boxes, etc. In these cases, value should be the value of the option selected, not the visible text. Chrome only: If a file path is given it will be uploaded to the input (for type=file). NOTE: No XPath locators.

arguments:

- locator - An element locator.
- value - The value to type.

Program 1:

AIM: Recording in context sensitive mode and analog mode using Selenium IDE

Objective:

- ✓ Student should be able to
- ✓ Describes Context Sensitive mode
- ✓ Record a test script
- ✓ Read the test script
- ✓ Run the recorded test and analyze the results

Context Sensitive Recording mode

- It is the default mode of recording which takes full advantage of Quick Test Professional's test object model.
- It recognizes objects in the application regardless of their location on the screen.
- It records the operations that are performed in an application by identifying the GUI objects.

Analog recording

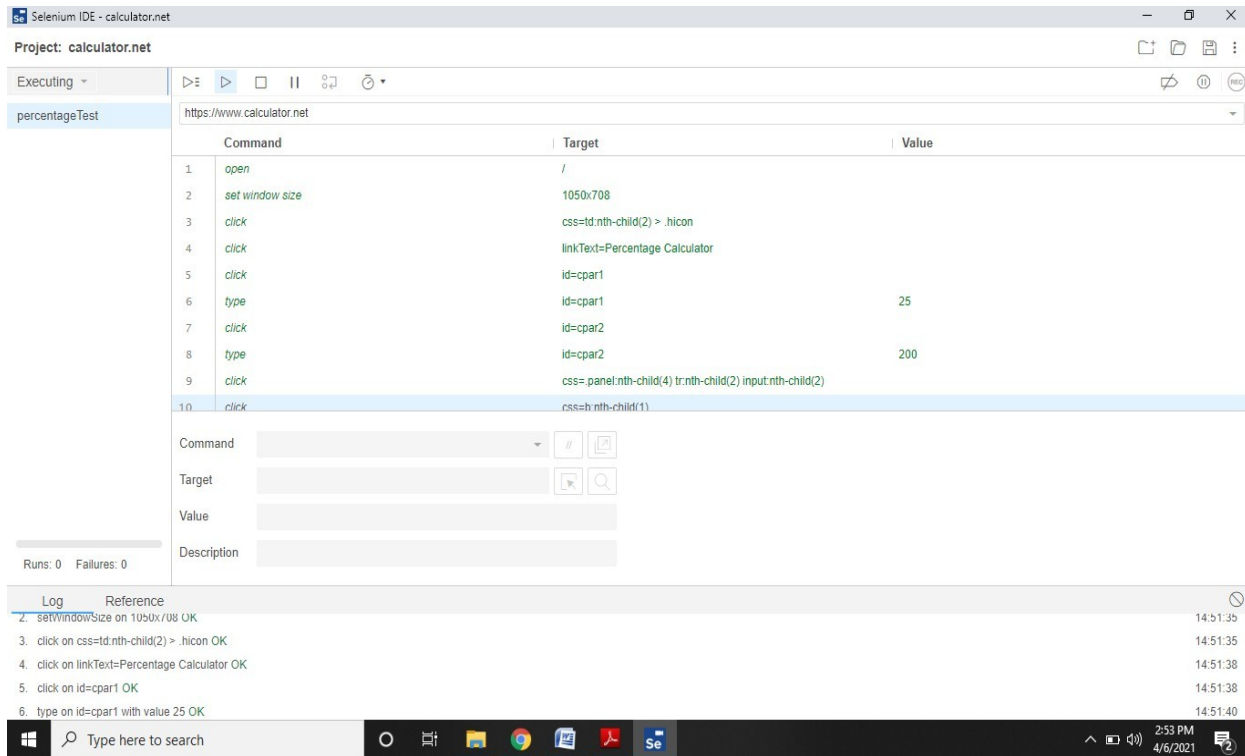
- records the inputs from keyboard, mouse clicks, the x and y coordinates that are travelled by the mouse pointer across the screen.
- It can't capture GUI windows and objects.

Procedure:

(I) Recording (recording user interactions with the browser)

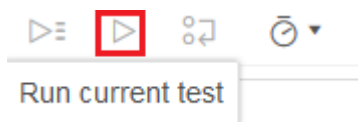
- Step1: Name the project as "Demo Test".
- Step2: Name the test case as "Demo1 test".
- Step 3: Click on the "Start Recording" Button present on the top right corner on the IDE to start recording the test case.
- Step 4: Type <https://avniet.ac.in> in the search box.
- Step 5: Hit enter to go to home page

- Step 6: It will redirect you to the that webpage and what operations you perform will be recorded. Meanwhile, you will get the notifications of the actions performed by the IDE at the extreme right corner of your web browser.
- Step 7: Now, go the IDE and click on the "Stop Recording" button to stop recording your actions further.



(II) Playing back (executing the recorded script)

- Click on the "Run Current Test" button present on the tool bar menu of the IDE. It will execute all of your interactions with the browser and gives you an overall summary of the executed test script.
- Run Tests
It allows you to run the currently selected test. When only a single test is loaded "Run Test" button and "Run all" button have the same effect.



(III) Saving the test suite

- Click on the save button present on the extreme right corner of the menu bar.
- save the entire test suite as "Demo Test".

Output:

The screenshot displays the Selenium IDE interface for a project named 'demo*'. The test suite 'demo2*' is shown with the following commands:

	Command	Target	Value
1	open	/	
2	set window size	1552x840	
3	run script	window.scrollTo(0,0)	

Below the commands, there are input fields for Command, Target, Value, and Description. A progress bar indicates 'Runs: 1 Failures: 0'. The log at the bottom shows the following entries:

Log	Reference
2. setWindowSize on 1552x840 OK	14:49:27
3. runScript on window.scrollTo(0,0) OK	14:49:27
'demo2' completed successfully	14:49:31

Program 2:

AIM: GUI checkpoint for single property

Objective: Student should be able to

- ✓ Explain how to check the behavior of GUI objects
- ✓ Create a test that checks GUI objects
- ✓ Run the test on different versions of an application and examine the results

Graphical User Interface:

- GUI testing evaluates design elements such as layout, colors, fonts, font sizes, labels, text boxes, text formatting, captions, buttons, lists, icons, links and content.
- GUI testing processes can be either done by the two categories are manual or automatic, and are often performed by third -party companies, rather than developers or end users.
- It can be require a lot of programming and is time consuming whether manual or automatic.
- Usually the software author writes out the intended function of a menu or graphical button.
- GUI testing also tends to test for certain program behaviors that users expect, like an hourglass when the program is busy.

Procedure:

- Step 1: open selenium IDE and open a new test
- Step 2: Name the project as "Demo Test".
- Step 3: Name the test case as "singleproperty_test".
- Step 4: Click on the "Start Recording" Button present on the top right corner on the IDE to start recording the test case.
- It will redirect you to the Google search engine page.
- Step 5: Type "calculator.com" in the Google search box.
- Step 6: Hit enter to get the search results. Click on the link provided under the URL <https://www.calculator.com>
- Step 7: Perform certain operation for single property

- Step 8: After recording the single click then stop the recording
- Step 9: Click on the "Run Current Test" button present on the tool bar menu of the IDE.
It will execute all of your interactions with the browser and gives you an overall summary of the executed test script.
- Step 10: give command and target to perform test and value as webpage address
- Step 11: create GUI checkpoint for the click /enter of value to perform operation.
- Step 12: will now check in log and reference about the GUI checkpoint and whether the test is successful or failure.

Output:

The screenshot displays a test script editor. At the top, a browser address bar shows 'https://www.google.co.in'. Below it is a table with three columns: 'Command', 'Target', and 'Value'. The table contains six rows of commands. A red box highlights the first three rows of the table. Below the table, there is a detailed view of the first command, 'open', with its target '/' and value field empty. The 'Comment' field is also empty.

	Command	Target	Value
1.	open	/	
2.	type	id=lst-ib	javatpoint tutorials
3.	send keys	id=lst-ib	\$(KEY_ENTER)
4.	mouse over	css=h3.r > a	
5.	mouse out	css=h3.r > a	
6.	click at	css=h3.r > a	147,11

Command	open
Target	/
Value	
Comment	

Program 3:

AIM: GUI checkpoint for single object/window

Objectives: Student should be able to

- ✓ Explain how to check the behavior of GUI objects
- ✓ Create a test that checks GUI objects/window
- ✓ Run the test on different versions of an application and examine the results

Procedure:

- Step 1: open selenium IDE and open a new test
- Step 2: Name the project as "Demo Test".
- Step 3: Name the test case as "singlewindow_test".
- Step 4: Click on the "Start Recording" Button present on the top right corner on the IDE to start recording the test case.
- It will redirect you to the Google search engine page.
- Step 5: Type "calculator.com" in the Google search box.
- Step 6: Hit enter to get the search results Click on the link provided under the URL <https://www.calculator.com>
- Step 7: Perform certain operation for window/objects
- Step 8: After recording the single click then stop the recording
- Step 9: Click on the "Run Current Test" button present on the tool bar menu of the IDE.
It will execute all of your interactions with the browser and gives you an overall summary of the executed test script.
- Step 10: give command and target to perform test and value as webpage address
- Step 11: will now check in log and reference about the GUI checkpoint and whether the test is successful or failure and GUI checkpoint of entry of value in the checkbox of object and a window.

Output:

The screenshot displays the Selenium IDE interface for a project named "division*". The test script "division1.test*" is loaded, showing a sequence of commands in a table. The URL bar at the top is set to "http://calculator.com".

	Command	Value
1	open	https://www.calculator.net/
2	set window size	1048x706
3	click	css=div:nth-child(3) > .scnm:nth-child(1) calculator.com
4	click	css=div:nth-child(1) > .scnm:nth-child(1)
5	click	css=div:nth-child(4) > .scnm:nth-child(1)
6	click	css=div:nth-child(4) > .scnm:nth-child(1)
7	double click	css=div:nth-child(4) > .scnm:nth-child(1)
8	click	css=div:nth-child(3) > .scnm:nth-child(3)
9	click	css=div:nth-child(4) > .scnm:nth-child(1) calculator.com

Below the table, the "set window size" command is selected, showing its configuration:

- Command: set window size
- Target: 1048x706
- Value: (empty field)
- Description: (empty field)

The "Reference" tab is active, showing the documentation for the `set window size resolution` command. It states: "Set the browser's window size, including the browser's interface." and provides the argument format: "resolution - Specify a window resolution using WidthxHeight. (e.g., 1280x800)."

The Windows taskbar at the bottom shows the time as 3:36 PM on 4/6/2021.

Program :4

AIM: GUI checkpoint for single multiple objects

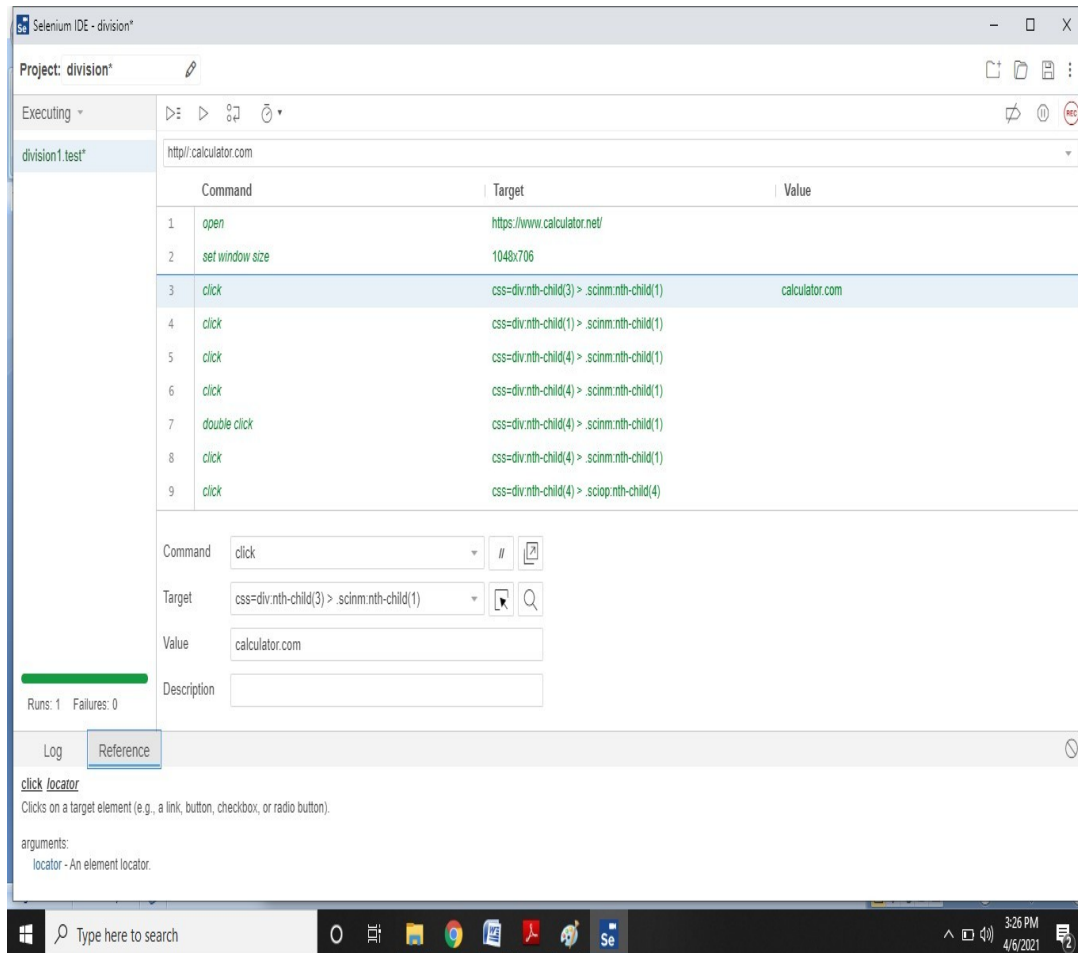
Objective: Student should be able to

- ✓ Explain how to check the behavior of GUI objects
- ✓ Create a test that checks GUI multiple objects
- ✓ Run the test on different versions of an application and examine the results

Procedure:

- Step 1: open selenium IDE and open a new test
- Step 2: Name the project as "Demo Test".
- Step 3: Name the test case as "multipleproperty_test".
- Step 4: Click on the "Start Recording" Button present on the top right corner on the IDE to start recording the test case.
- It will redirect you to the Google search engine page.
- Step 5: Type "calculator.com" in the Google search box.
- Step 6: Hit enter to get the search results. Click on the link provided under the URL <https://www.calculator.com>
- Step 7: Perform certain operation for Multiple property
- Step 8: After recording the single click then stop the recording
- Step 9: Click on the "Run Current Test" button present on the tool bar menu of the IDE.
It will execute all of your interactions with the browser and gives you an overall summary of the executed test script.
- Step 10: Give command and target to perform test and value as webpage address
- Step 11: if the click GUI checkpoint on multiple objects or navigating from one window to another is noted in reference block and in log section to check whether the test is successful or a failure.

Output:



Selenium IDE - division*

Project: division*

Executing ▾

division1.test* http://calculator.com

	Command	Target	Value
1	open	https://www.calculator.net/	
2	set window size	1048x706	
3	click	css=div:nth-child(3) > .scinn:nth-child(1)	calculator.com
4	click	css=div:nth-child(1) > .scinn:nth-child(1)	
5	click	css=div:nth-child(4) > .scinn:nth-child(1)	
6	click	css=div:nth-child(4) > .scinn:nth-child(1)	
7	double click	css=div:nth-child(4) > .scinn:nth-child(1)	
8	click	css=div:nth-child(4) > .scinn:nth-child(1)	
9	click	css=div:nth-child(4) > .scinn:nth-child(4)	

Command: click // [?]

Target: css=div:nth-child(3) > .scinn:nth-child(1) [?] [Q]

Value: calculator.com

Description:

Runs: 1 Failures: 0

Log Reference

click locator
Clicks on a target element (e.g., a link, button, checkbox, or radio button).

arguments:
locator - An element locator.

Windows taskbar: Type here to search, 3:26 PM, 4/6/2021

Program 5:

AIM: Test case for calculator in windows application

Objectives: Students should be able to

- ✓ Know about what is Manual testing
- ✓ Able to write Test cases.

- **Manual Testing:** Manual testing is a testing process that is carried out manually in order to find defects without the usage of tools or automation scripting. A test plan document is prepared that acts as a guide to the testing process in order to have the complete test coverage.

Procedure:

Write the test cases based on the following functions and scenarios.

- Check the calculator if it starts by on button. If it is a software-based calculator, then check if it begins via specific means like from searching for a calculator in a search bar and then executing an application. Or by accessing menu items in Windows.
- Check if the calculator window maximizes to the specified window size.
- Check if the calculator closes when the close button is pressed or if the exit menu is clicked from the file > exit option.
- Check if the help document is accessed from Help > Documentation.
- Check if the calculator allows copy and paste functionality.
- Check if the calculator has any specific preferences.
- Check if all the numbers are working (0 to 9)
- Check if the arithmetic keys (+, -, *, %, /) are working.
- Check if the clear key is working.
- Check if the brackets keys are working.
- Check if the sum or corresponding key is working.
- Check if the square and square root keys are working.

Output:

Test Case:

Test scenario ID	Login-1		Test case ID	Login-1A		
Test case description	User login – Positive case		Test case priority	High		
Pre-requisite	User email id		Post-requisite	Username, password		
S.no	Action	Inputs	Expected Output	Actual Output	Test Result	Test Comments
1.	Launch Calculator App	Search in search bar and click on Calculator icon	Calculator homepage	Calculator homepage	Pass	Launch Successful
2.	Testing Maximize button	Click on maximize button()	Maximized window size	Maximized window size	Pass	Worked Successfully
3.	Close Application	Click on close(X)	Calculator closed	Calculator closed	Pass	Close Successful
4.	Testing Help option	Click on “help”	Help documentation	Help documentation	Pass	Worked Successfully
5.	Copy & Paste Functionality	Copy Math expression paste in app	Expression copied	Expression copied	Pass	Worked Successfully
6.	Working of (0-9) keys	Click on any number say 8	Number in result area	Number in result area	Pass	Worked Successfully
7.	Working of arithmetic keys	Click on any arithmetic key say ‘+’ As 2+3	2+3=5	2+3=5	Pass	Worked Successfully
8.	Testing “Clear” key	Click on c or clear key	Result area 0	Result area 0	Pass	Worked Successfully
9.	Testing Bracket keys	Type any expression which includes brackets say 2*(3+2)	2*(3+2)=10	2*(3+2)=10	Pass	Worked Successfully
10.	Testing square & square Root keys	Click on any number then click Square/SquareRoot	Sqr(9)=81	Sqr(9)=81	Pass	Worked Successfully

EXPERIMENT 6

Database checkpoint for Default check

When you create a default check on a database, you create a standard database checkpoint that checks the entire result set using the following criteria:

- The default check for a multiple-column query on a database is a case sensitive check on the entire result set by column name and row index.
- The default check for a single-column query on a database is a case sensitive check on the entire result set by row position.

If you want to check only part of the contents of a result set, edit the expected value of the contents, or count the number of rows or columns, you should create a custom check instead of a default check. For information on creating a custom check on a database, see “Creating a Custom Check on a Database,”

Creating a Default Check on a Database Using ODBC or Microsoft Query

You can create a default check on a database using ODBC or Microsoft Query. **To create a default check on a database using ODBC or Microsoft Query:**

1. Choose **Insert > Database Checkpoint > Default Check** or click the **Default Database Checkpoint** button on the User toolbar. If you are recording in Analog mode, press the CHECK DATABASE (DEFAULT) softkey in order to avoid extraneous mouse movements. Note that you can press the CHECK DATABASE (DEFAULT) softkey in Context Sensitive mode as well.
2. If Microsoft Query is installed and you are creating a new query, an instruction screen opens for creating a query.

If you do not want to see this message next time you create a default database checkpoint, clear the **Show this message next time** check box.

Click **OK** to close the instruction screen.

If Microsoft Query is not installed, the Database Checkpoint wizard opens to a screen where you can define the ODBC query manually. For additional information, see “Setting ODBC (Microsoft Query) Options”

3. Define a query, copy a query, or specify an SQL statement. For additional information, see “Creating a Query in ODBC/Microsoft Query” or “Specifying an SQL Statement”

4. WinRunner takes several seconds to capture the database query and restore the WinRunner window.

WinRunner captures the data specified by the query and stores it in the test's exp folder. WinRunner creates the msqr*.sql query file and stores it and the database checklist in the test's chklist folder. A database checkpoint is inserted in the test script as a **db_check** statement.

Creating a Default Check on a Database Using Data Junction

You can create a default check on a database using Data Junction. **To create a default check on a database:**

1. Choose **Insert > Database Checkpoint > Default Check** or click the **Default Database Checkpoint** button on the User toolbar.

If you are recording in Analog mode, press the CHECK DATABASE (DEFAULT) softkey in order to avoid extraneous mouse movements. Note that you can press the CHECK DATABASE (DEFAULT) softkey in Context Sensitive mode as well.

For information on working with the Database Checkpoint wizard, see “Working with the Database Checkpoint Wizard”

2. An instruction screen opens for creating a query.

If you do not want to see this message next time you create a default database checkpoint, clear the **Show this message next time** check box.

Click **OK** to close the instruction screen.

3. Create a new conversion file or use an existing one. For additional information, see “Creating a Conversion File in Data Junction”
4. WinRunner takes several seconds to capture the database query and restore the WinRunner window.

WinRunner captures the data specified by the query and stores it in the test's exp folder. WinRunner creates the *.djs conversion file and stores it in the checklist in the test's chklist folder. A database checkpoint is inserted in the test script as a **db_check** statement.

EXPERIMENT 7

Database checkpoint for custom check

When you create a custom check on a database, you create a standard database checkpoint in which you can specify which properties to check on a result set.

You can create a custom check on a database in order to:

- check the contents of part or the entire result set
- edit the expected results of the contents of the result set
- count the rows in the result set
- count the columns in the result set

You can create a custom check on a database using ODBC, Microsoft Query or Data Junction.

To create a custom check on a database:

1. Choose **Insert > Database Checkpoint > Custom Check**. If you are recording in Analog mode, press the CHECK DATABASE (CUSTOM) softkey in order to avoid extraneous mouse movements. Note that you can press the CHECK DATABASE (CUSTOM) softkey in Context Sensitive mode as well.
2. Follow the instructions on working with the Database Checkpoint wizard, as described in “Working with the Database Checkpoint Wizard”
3. If you are creating a new query, an instruction screen opens for creating a query.
If you do not want to see this message next time you create a default database checkpoint, clear the **Show this message next time** check box.
4. If you are using ODBC or Microsoft Query, define a query, copy a query, or specify an SQL statement.
If you are using Data Junction, create a new conversion file or use an existing one.
5. If you are using Microsoft Query and you want to be able to parameterize the SQL statement in the **db_check** statement which will be generated, then in the last wizard screen in Microsoft Query, click **View data or edit query in Microsoft Query**. Follow the instructions in “Parameterizing Standard Database Checkpoints”
6. WinRunner takes several seconds to capture the database query and restore the WinRunner window. *The Check Database dialog box opens*

EXPERIMENT 8

Database checkpoint for runtime record check

You can add a runtime database record checkpoint to your test in order to compare information displayed in your application during a test run with the current value(s) in the corresponding record(s) in your database. You add runtime database record checkpoints by running the Runtime Record Checkpoint wizard. When you are finished, the wizard inserts the appropriate **db_record_check** statement into your script.

Note that when you create a runtime database record checkpoint, the data in the application and in the database are generally in the same format. If the data is in different formats, you can follow the instructions in “Comparing Data in Different Formats” to create a runtime database record checkpoint. Note that this feature is for advanced WinRunner users only.

Using the Runtime Record Checkpoint Wizard

The Runtime Record Checkpoint wizard guides you through the steps of defining your query, identifying the application controls that contain the information corresponding to the records in your query, and defining the success criteria for your checkpoint.

To open the wizard, select **Insert > Database Checkpoint > Runtime Record Check**.

Define Query Screen

The Define Query screen enables you to select a database and define a query for your checkpoint. You can create a new query from your database using Microsoft Query, or manually define an SQL statement



You can choose from the following options:

- **Create new query:** Opens Microsoft Query, enabling you to create a new query. Once you finish defining your query, you return to WinRunner.
- **Specify SQL statement:** Opens the Specify SQL Statement screen in the wizard, enabling you to specify the connection string and an SQL statement.

Specify SQL Statement Screen

The Specify SQL Statement screen enables you to manually specify the database connection string and the SQL statement.



Enter the required information:

- **Connection String:** Enter the connection string, or click the **Create** button
- **Create:** Opens the ODBC Select Data Source dialog box. You can select a *.dsn file in the Select Data Source dialog box to have it insert the connection string in the box for you.
- **SQL:** Enter the SQL statement.

The Match Database Field screen enables you to identify the application control or text in your application that matches the displayed database field. You repeat this step for each field included in your query. This screen includes the following options:

- **Database field:** Displays a database field from your query. Use the pointing hand to identify the control or text that matches the displayed field name.

- **Logical name:** Displays the logical name of the control you select on your application.

(Displayed only when the **Select text from a Web page** check box is cleared.)

- **Text before:** Displays the text that appears immediately before the text to check. (Displayed only when the Select text from a Web page check box is checked.)
- **Text after:** Displays the text that appears immediately after the text to check. (Displayed only when the **Select text from a Web page** check box is selected.)

- **Select text from a Web page:** Enables you to indicate the text on your Web page containing the value to be verified.

The Matching Record Criteria screen enables you to specify the number of matching database records required for a successful checkpoint.



- **Exactly one matching record:** Sets the checkpoint to succeed if exactly one matching database record is found.
- **One or more matching records:** Sets the checkpoint to succeed if one or more matching database records are found.
- **No matching records:** Sets the checkpoint to succeed if no matching database records are found.

When you click **Finish** on the Runtime Record Checkpoint wizard, a **db_record_check** statement is inserted into your script.

Comparing Data in Different Formats

Suppose you want to compare the data in your application to data in the database, but the data is in different formats. You can follow the instructions below to create a runtime database record checkpoint without using the Runtime Record Checkpoint Wizard. Note that this feature is for advanced WinRunner users only.

For example, in the sample Flight Reservation application, there are three radio buttons in the Class box. When this box is enabled, one of the radio buttons is always selected. In the database of the sample Flight Reservation application, there is one field with the values 1, 2, or 3 for the matching class.

To check that data in the application and the database have the same value, you must perform the following steps:

1. Record on your application up to the point where you want to verify the data on the screen. Stop your test. In your test, manually extract the values from your application.
2. Based on the values extracted from your application, calculate the expected values for the database. Note that in order to perform this step, you must know the mapping relationship between both sets of values. See the example below.
3. Add these calculated values to any edit field or editor (e.g. Notepad). You need to have one edit field for each calculated value. For example, you can use multiple Notepad windows, or another application that has multiple edit fields.
4. Use the GUI Map Editor to teach WinRunner:
 - the controls in your application that contain the values to check
 - the edit fields that will be used for the calculated values
5. Add TSL statements to your test script to perform the following operations:
 - extract the values from your application
 - calculate the expected database values based on the values extracted from your application
 - write these expected values to the edit fields
6. Use the Runtime Record Checkpoint wizard, described in “Using the Runtime Record Checkpoint Wizard,” to create a **db_record_check** statement.

When prompted, instead of pointing to your application control with the desired value, point to the edit field where you entered the desired calculated value.

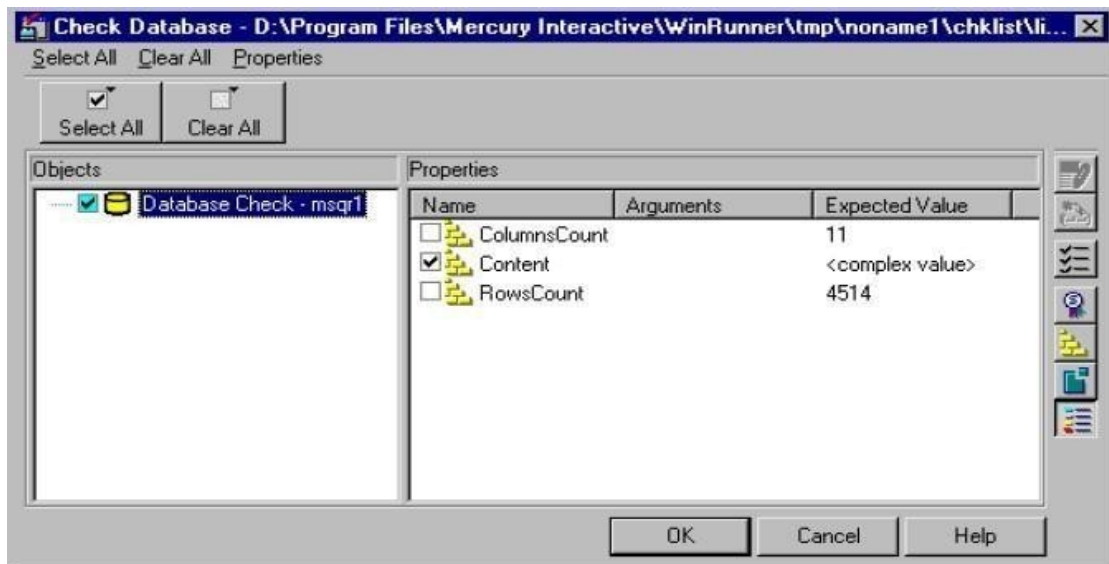
Example of Comparing Different Data Formats in a Runtime Database Record Checkpoint

The following excerpts from a script are used to check the Class field in the database against the radio buttons in the sample Flights application. The steps refer to the instructions.

step 1




```
# Extract values from GUI objects in application. button_get_state("First",vFirst);
button_get_state("Business",vBusiness); button_get_state("Economy",vEconomy);
```



The **Objects** pane contains “Database check” and the name of the *.sql query file or *.djs conversion file included in the database checkpoint. The **Properties** pane lists the different types of checks that can be performed on the result set. A check mark indicates that the item is selected and is included in the checkpoint.

7. Select the types of checks to perform on the database. You can perform the following checks:

ColumnsCount: Counts the number of columns in the result set.

Content: Checks the content of the result set, as described in “Creating a Default Check on a Database,”

RowCount: Counts the number of rows in the result set.

If you want to edit the expected value of a property, first select it. Next, either click the **Edit Expected Value** button, or double-click the value in the Expected Value column.

- For **ColumnsCount** or **RowCount** checks on a result set, the expected value is displayed in the **Expected Value** field corresponding to the property check. When you edit the expected value for these property checks, a spin box opens. Modify the number that appears in the spin box.

- For a **Content** check on a result set, <complex value> appears in the **Expected Value** field corresponding to the check, since the content of the result set is too complex to be displayed in this column. When you edit the expected value, the **Edit Check** dialog box opens. In the **Select Checks** tab, you can select which checks to perform on the result set, based on the data captured in the query. In the **Edit Expected Data** tab, you can modify the expected results of the data in the result set.
8. Click **OK** to close the Check Database dialog box.
- WinRunner captures the current property values and stores them in the test's exp folder. WinRunner stores the database query in the test's chklist folder. A database checkpoint is inserted in the test script as a **db_check** statement.

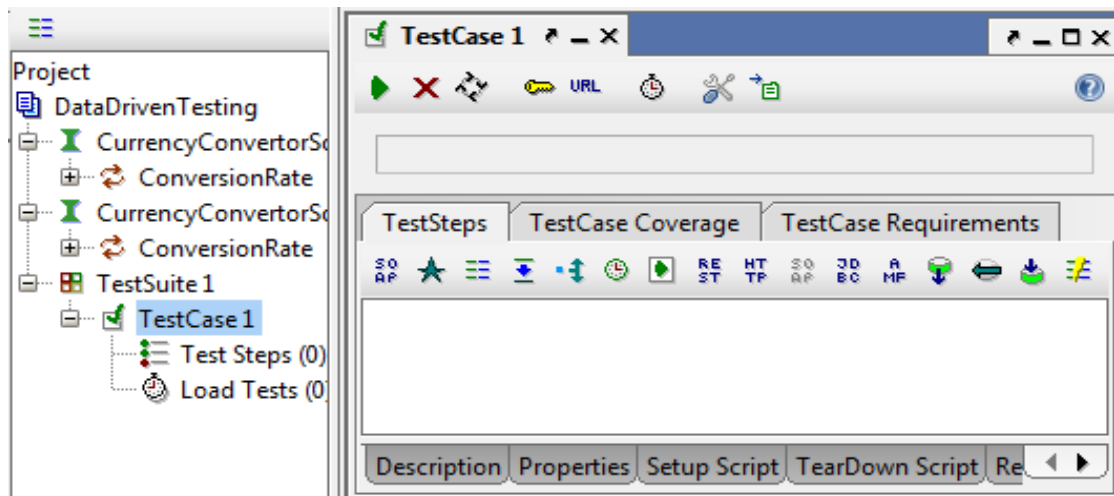
EXPERIMENT 9

B. Data driven test through flat files

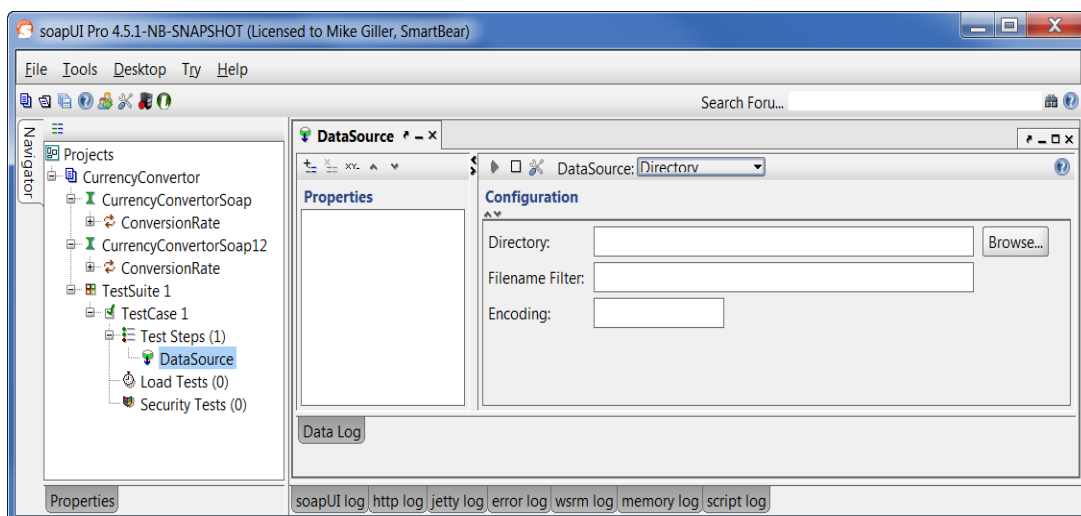
Steps:

Create DataSource

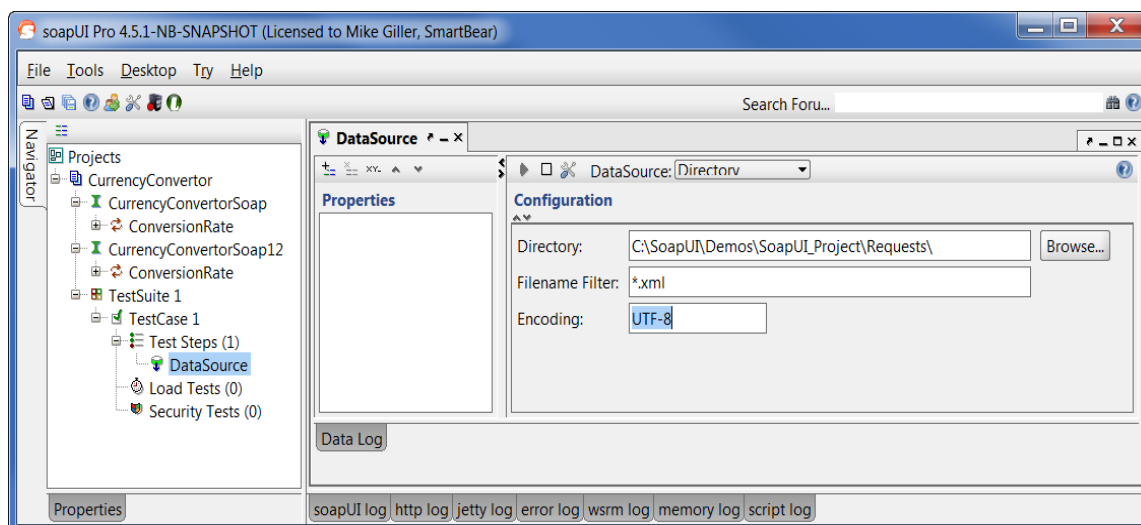
As in the Data Driven Testing guide, create a SoapUI Project from the publicly available CurrencyConverter WSDL (<http://www.websvcx.com/CurrencyConverter.asmx?wsdl>), then add a TestSuite and a TestCase and open its editor:




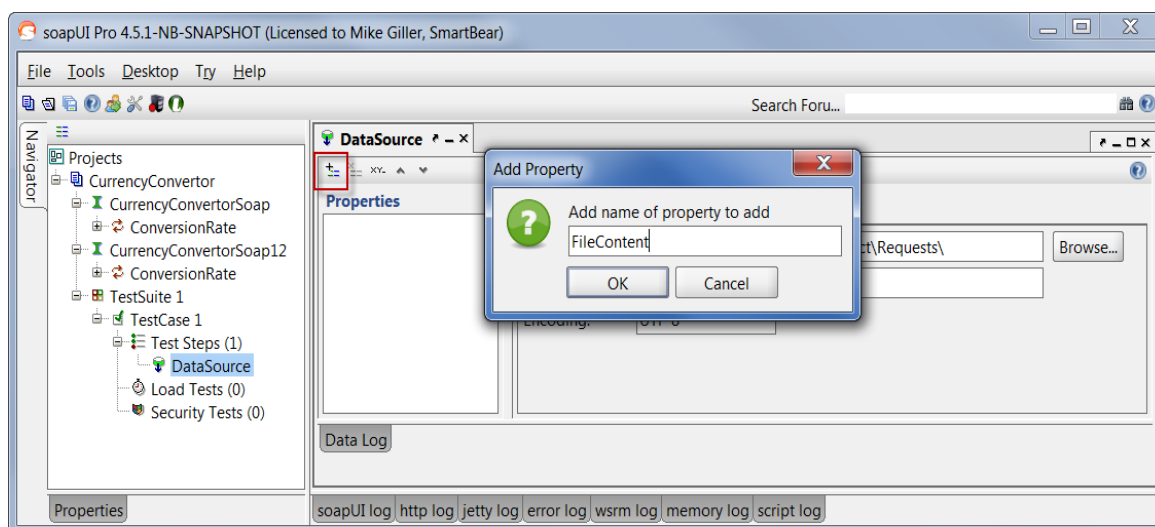
Now add a DataSource TestStep and select the DataSource type “Directory” from the dropdown in the toolbar. You should now have:



Now, select the directory where your input files are stored, add an applicable filter (e.g. “*.txt” or “*.xml” for text or XML files respectively), and potentially encoding.



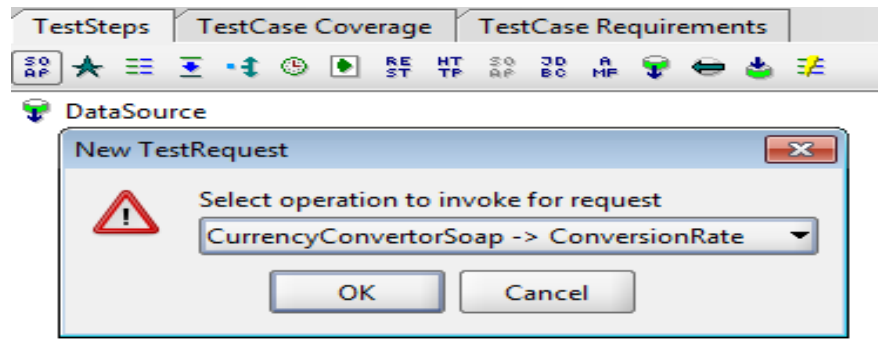
Now click on the  icon from the screen below and enter a property that will contain the content of each file



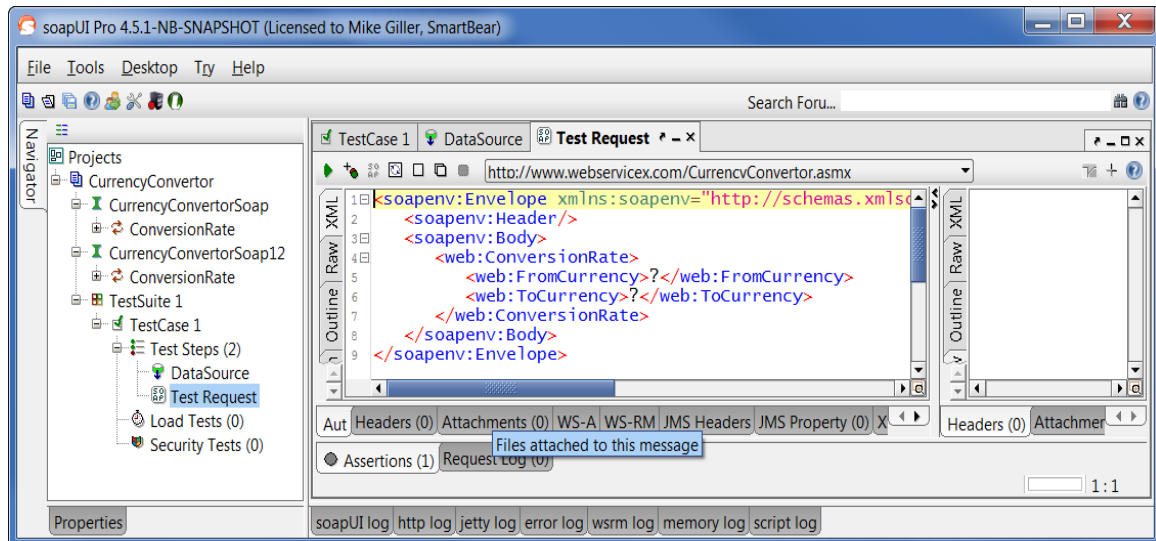
Quick tip: If your property is named “Filename” it will contain the name of the file instead of file’s contents.

1. Create Test Steps

Now you need to add a Test Request to your TestCase which you will use to test the Web Service. Press the SOAP Request button in the TestCase editor and select the ConversionRate operation in the CurrencyConverterSoap Interface.



Press OK in all dialogs. A SOAP Request Step will be added to the TestCase and the editor for the request is opened. Switch to the XML editor (if not already there):

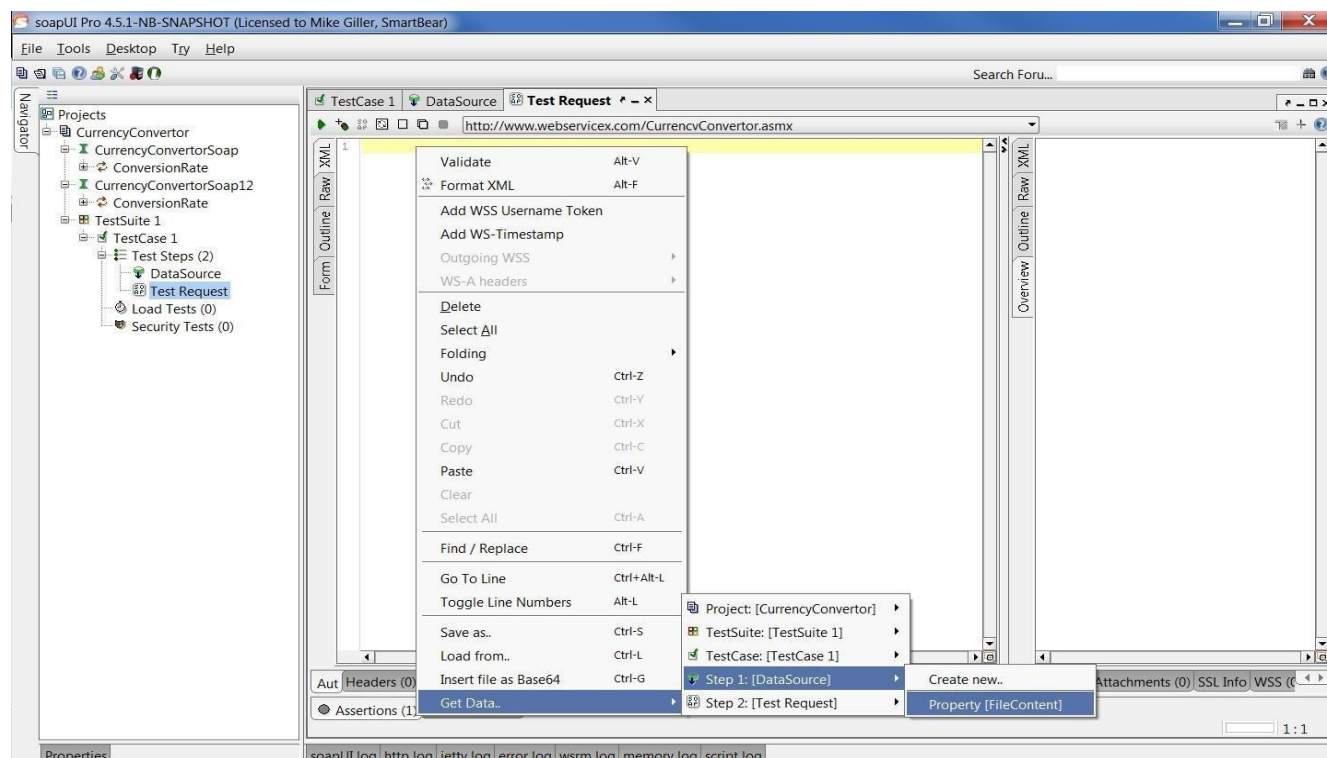


Now, I'm operating under the assumption that you have a fully built request in each of the files in your directory.

An example of an input file would be:

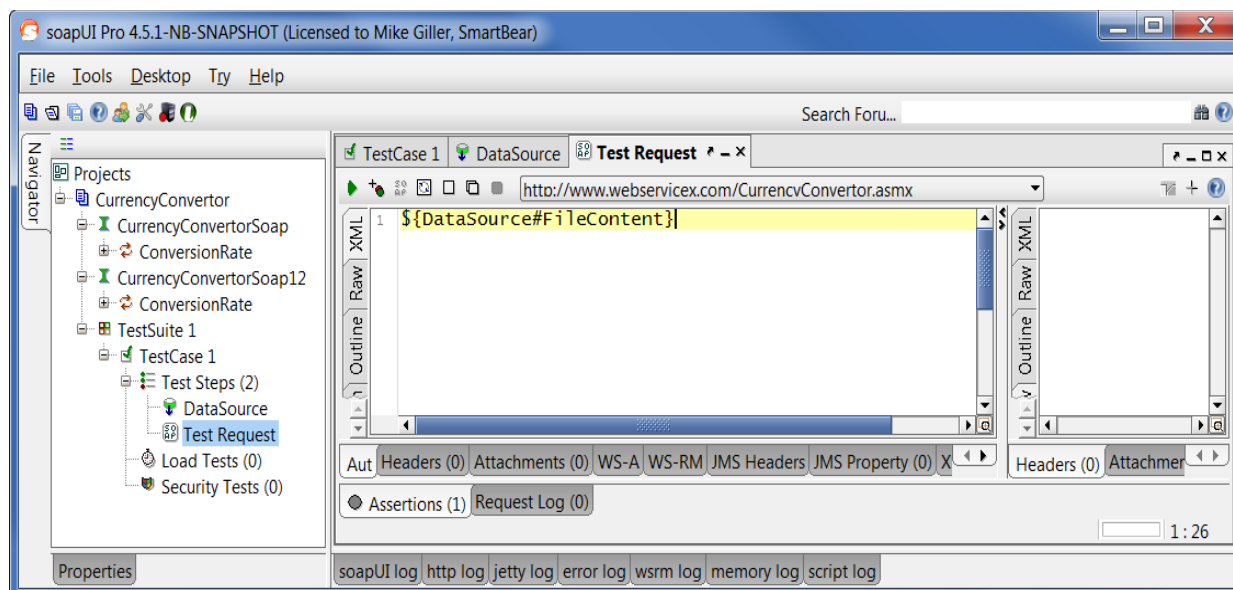
```
<soapenv:Envelope xmlns:soapenv="
xmlns:web=">
<soapenv:Header/>
<soapenv:Body>
<web:ConversionRate>
<web:FromCurrency>SEK</web:FromCurrency>
<web:ToCurrency>USD</web:ToCurrency>
</web:ConversionRate>
</soapenv:Body>
</soapenv:Envelope>
```

So, based on that, remove all the content in the XML tab, right-click and select the path to your DataSource property:



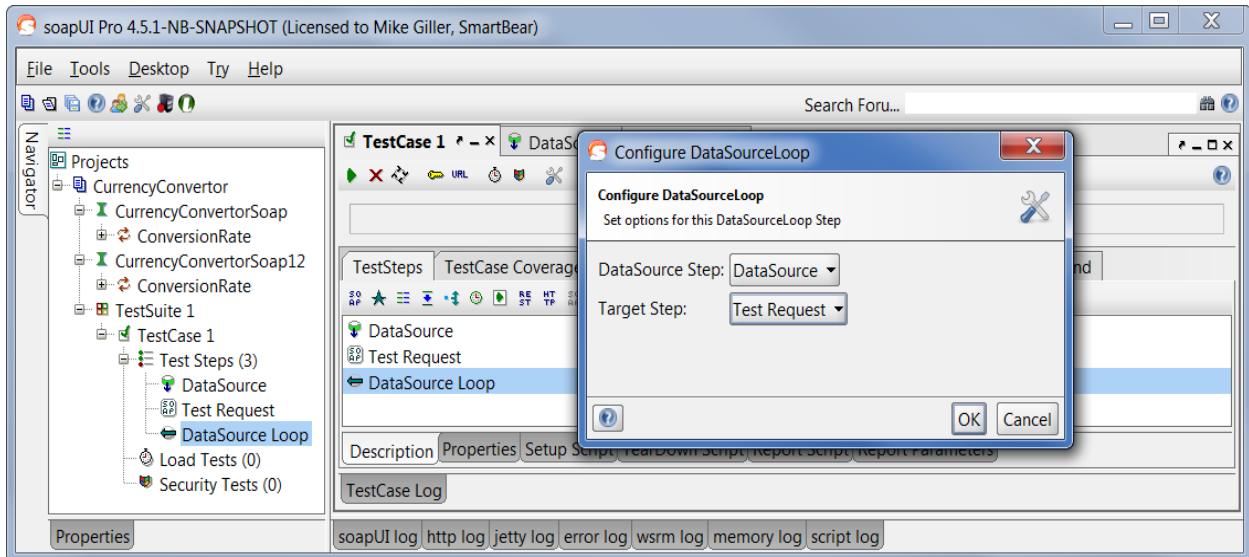
Note: If an XPATH window comes up, just click OK without selecting anything.

Now your request should look like this:




2. Add DataSource Loop

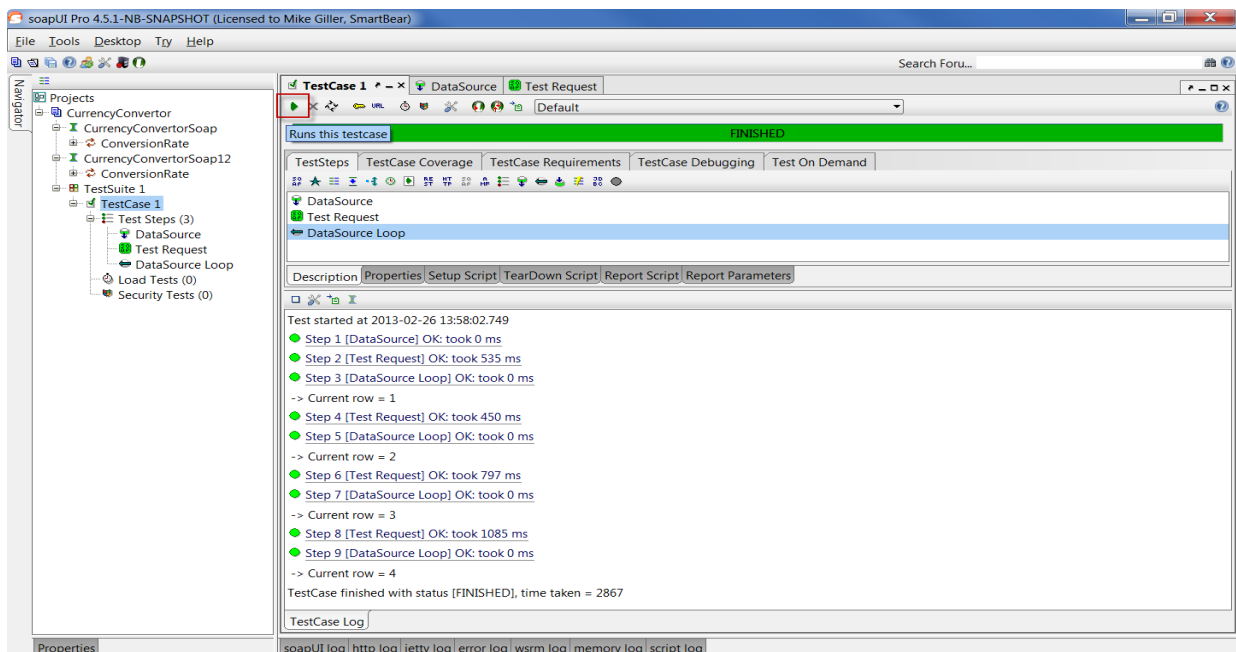
As a final step, we just need to iterate through all the files in our DataSource. So in your TestCase, add a DataSource loop step, and double click it to configure as in the picture below:



Click OK.

3. That's it

Now if you click on the  icon in the test case window, you can see the whole test run through each file:



C. Data driven test through excel test

Data-driven testing (DDT) is taking a test, parameterizing it and then running that test with varying data. This allows you to run the same test case with many varying inputs, therefore increasing coverage from a single test. In addition to increasing test coverage, data driven testing allows the ability to build both positive and negative test cases into a single test. Data-driven testing allows you to test the form with a different set of input values to be sure that the application works as expected.

It is convenient to keep data for automated tests in special storages that support sequential access to a set of data, for example, Excel sheets, database tables, arrays, and so on. Often data is stored either in a text file and are separated by commas or in Excel files and are presented as a table. If you need to add more data, you simply modify the file either in any text editor or in Microsoft Excel (in case of hard-coded values, you should modify both data and code).

Data-driven test includes the following operations performed in a loop:

- Retrieving input data from storage
- Entering data in an application form
- Verifying the results
- Continuing with the next set of input

data Pre-requisites:

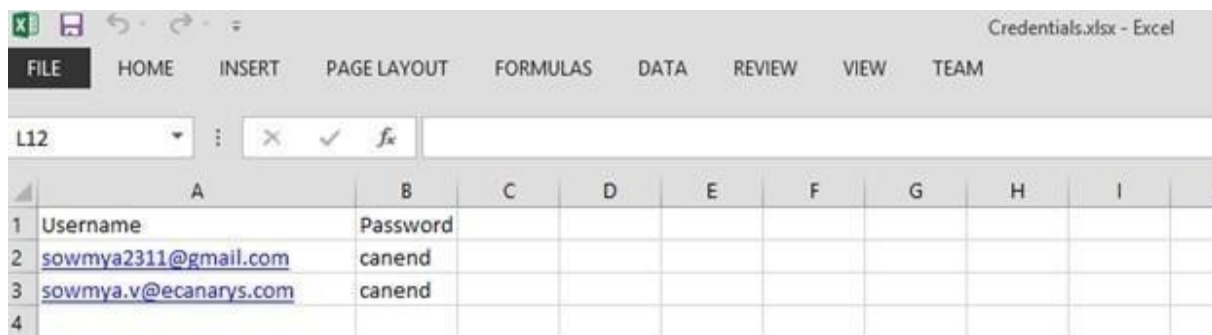
1. Java JDK 1.5 or above
2. Apache POI library v3.8 or above
3. Eclipse 3.2 above
4. Selenium server-standalone-2.47.x.jar
5. TestNG-6.9

Data Excel

Scenario -Open the application and login with different username and password. This data will be coming from excel sheet

Step 1: The first and the foremost step is to create the test data with which we would be executing the test scripts. Download JAR files of Apache POI and Add Jars to your project library. Let us create an excel file and save it as “Credentials.xlsx” and place it in the created package location.

We will use the data excel file and the file looks as below



	A	B	C	D	E	F	G	H	I
1	Username	Password							
2	sowmya2311@gmail.com	canend							
3	sowmya.v@ecanarys.com	canend							
4									

Step 2: Create a POM class file under com.coe.pom name it as “Loginpage.java”. Inside a login page we write code to identify the webelements of login page using `@FindBy` annotation. To initialize the web elements we use initelements of page factory class. We utilize the elements by writing a method to it.

Step 3: Create a ‘New Class’ file, by right click on the package com.coe.script and select New > Class and name it as “SuperClass.java”, and create a new class file with a name “ValidLoginLogout.java”.

Step 4: Create some test data in excel that we will pass to script. For demo purpose I have taken username and password in excel.

Step 5: Copy and paste the below mentioned code under com.coe.pom package class.

EXPERIMENT 10

A. Batch testing without parameter passing

A batch test is a test script that calls other tests. You program a batch test by typing call statements directly into the test window and selecting the **Run in batch mode** option in the **Run** category of the General Options dialog box before you execute the test.

A batch test may include programming elements such as loops and decisionmaking statements.

Loops enable a batch test to run called tests a specified number of times. Decision-making statements such as if/else and switch condition test execution on the results of a test called previously by the same batch script. See “Enhancing Your Test Scripts with Programming,” for more information.

For example, the following batch test executes three tests in succession, then loops back and calls the tests again. The loop specifies that the batch test should call the tests ten times.

```
for (i=0; i<10; i++)  
{  
call "c:\\pbtests\\open" ();  
call "c:\\pbtests\\setup" ();  
call "c:\\pbtests\\save" ();  
}
```

To enable a batch test:

1. Choose **Tools > General Options**.

The General Options dialog box
opens.

2. Click the **Run** category.
3. Select the **Run in batch mode** check box.

Running a Batch Test: You execute a batch test in the same way that you execute a regular test. Choose a mode (Verify, Update, or Debug) from the list on the toolbar and choose **Test > Run from Top**. See “Understanding Test Runs,” for more information.

When you run a batch test, WinRunner opens and executes each called test. All messages are suppressed so that the tests are run without interruption. If you run the batch test in **Verify** mode, the current test results are compared to the expected test results saved earlier. If you are running the batch

test in order to update expected results, new expected results are created in the expected results folder

for each test. See “Storing Batch Test Results” below for more information. When the batch test run is completed, you can view the test results in the Test Results window.

Note that if your tests contain TSL **test** statements, WinRunner interprets these statements differently for a batch test run than for a regular test run. During a regular test run, **test** terminates test execution. During a batch test run, **test** halts execution of the current test only and control is returned to the batch test.

EXPERIMENT 11

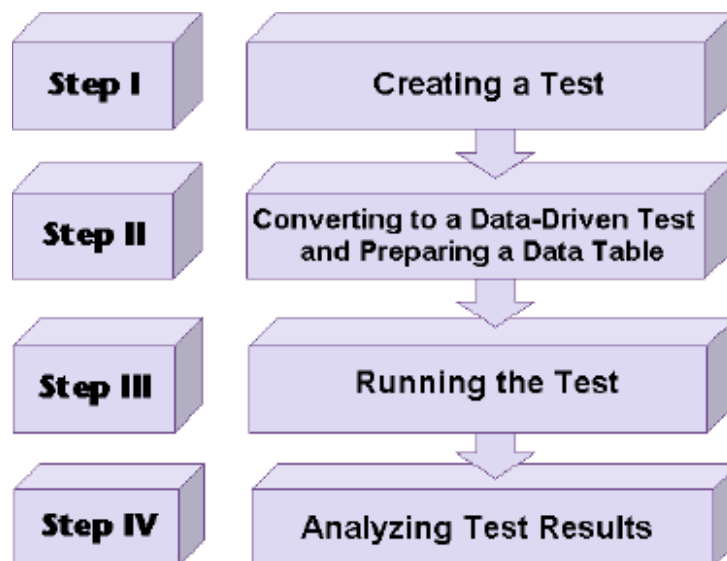
Data driven batch

When you test your application, you may want to check how it performs the same operations with multiple sets of data. For example, suppose you want to check how your application responds to ten separate sets of data. You could record ten separate tests, each with its own set of data.

Alternatively, you could create a data-driven test with a loop that runs ten times. In each of the ten iterations, the test is driven by a different set of data. In order for WinRunner to use data to drive the test, you must substitute fixed values in the test with parameters. The parameters in the test are linked with data stored in a data table. You can create data-driven tests using the DataDriver wizard or by manually adding data-driven statements to your test scripts.

For non-data-driven tests, the testing process is performed in three steps: creating a test; running the test; analyzing test results. When you create a data-driven test, you perform an extra two-part step between creating the test and running it: converting the test to a data-driven test and creating a corresponding data table.

The following diagram outlines the stages of the data-driven testing process in WinRunner:



EXPERIMENT 12

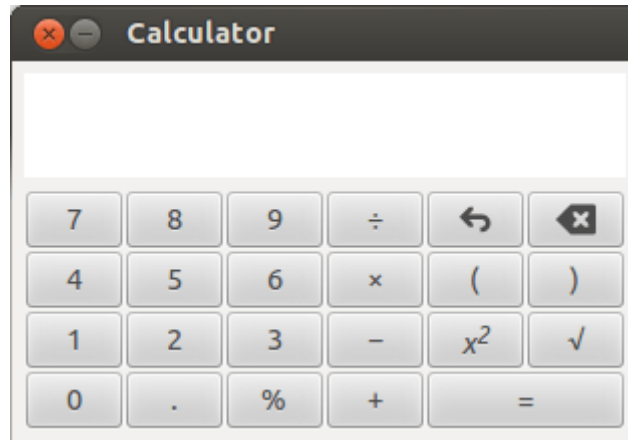
Silent mode test execution without any interruption

1 **Silent Mode** to continue **test execution without any interruption**, we can use this run setting.

1 Navigation Click Tools menu → Choose General options→ Select Run Tab → Select Run in batch mode (Figure III . 18 . 1) – Click ... NOTE In **silent mode** , **WinRunner** is **not** able to **execute** tester interactive statements .

EXPERIMENT 13

Test case for calculator in windows application



Basic Operational Tests

Write the test cases based on the following functions and scenarios.

- Check the calculator if it starts by on button. If it is software based calculator, then check if it starts via specific means like from searching for calculator in search bar and then executing application. Or by accessing menu item in the Windows.
- Check if the calculator window maximizes to certain window size.
- Check the if the calculator closes when the close button is pressed or if the exit menu is clicked from file > exit option.
- Check if the help document is accessed from Help > Documentation.
- Check if the calculator allows copy and paste functionality.
- Check if the calculator has any specific preferences.
- Check if all the numbers are working (0 to 9)
- Check if the arithmetic keys (+, -, *, %, /) are working.
- Check if the clear key is working.
- Check if the brackets keys are working.
- Check if the sum or equal key is working.
- Check if the square and square root key is working.

Functionality Test Cases

- Check the addition of two integer numbers.
- Check the addition of two negative numbers.
- Check the addition of one positive and one negative number.
- Check the subtraction of two integer numbers.
- Check the subtraction of two negative numbers.
- Check the subtraction of one negative and one positive number.
- Check the multiplication of two integer numbers.
- Check the multiplication of two negative numbers.
- Check the multiplication of one negative and one positive number.
- Check the division of two integer numbers.
- Check the division of two negative numbers.
- Check the division of one positive number and one integer number.
- Check the division of a number by zero.
- Check the division of a number by negative number.
- Check the division of zero by any number.
- Check if the functionality using BODMAS/BIDMAS works as expected.

Test Step	Calc 1
Requirement	1
Input	Start the computer, click on start button, select 'programs' go to 'Accessories' and then click on 'Calculator'
Expected Output	System should display 'Calculator' window
Test Step	Calc 2
Requirement	1.1
Input	Click on start button and then click on 'Run' enter 'calc.exe' and click on OK button
Expected Output	System should display 'Calculator' window
Test Step	Calc 3
Requirement	1.2
Input	Click on start button and then click on 'Run' enter 'xyz.exe' and click on OK button
Expected Output	System should display an error message 'Can't find the xyz .exe, make sure the path and file name are correct and that all required libraries are available'
Test Step	Calc 4
Requirement	1.3
Input	On 'Calculator' check for all buttons either its working properly or not [Ex: buttons: 1 to 9, +, -, *, /, etc]
Expected Output	All buttons should work fine according to requirements.
Test Step	Calc 5
Requirement	1.4
Input	Check for following conditions Additions, subtractions, Multiplications and division.
Expected Output	Calculator should work fine according to stated requirements
Test Step	Calc 6
Requirement	1.5
Input	Check for following condition 1. Zero divided by some number [Ex: 1,20,30,4,5,67...etc] and then press equal '=' button
Expected Output	Calculator should display value zero

