Department of CSE(AI&ML)

Vision

To develop competent and socially responsible engineers in the domain of Artificial Intelligence & Machine Learning for noteworthy contributions to the society.

Mission

- To educate the students in fundamental principles of computing and develop the skills needed to solve practical problems using contemporary computer-based technologies.
- To provide State-of-the-art computing laboratory facilities to promote industry-institute interaction to enhance student's practical knowledge.
- To Inculcate self-learning abilities, team spirit, and professional ethics among the students to serve society.

Program Educational Objectives (PEO) :

S.NO.	Program Educational Objectives (PEO)
PEO1	Graduates will have the ability to adapt, contribute and innovate new systems in the key domains of Artificial Intelligence and Machine Learning.
PEO2	Graduates will have the ability to pursue higher education in reputed institutions with AI Specialization.
PEO3	Graduates will be ethically and socially responsible solution providers and entrepreneurs in the field of Computer Science and Engineering with AI&ML Specialization.

Program Outcomes (PO) :

S.NO	Program Outcomes (PO):
P01	**Engineering Knowledge:** Apply the knowledge of Mathematics, Science, Engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
P02	**Problem Analysis:** Identify, formulate, research literature, and analyze complex engineering

	problems reaching substantiated conclusions using first principles of Mathematics, Natural Sciences, and Engineering Sciences.
PO3	**Design/Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct Investigations of Complex Problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and Sustainability:**Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development
PO8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
PO9	**Individual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSO) :

S.NO.	Program Specific Outcomes (PSO)
PSO1	Apply the fundamental knowledge for problem analysis and conduct investigations in Artificial Intelligence and Machine Learning for sustainable development

PSO2	Design and development of solutions by using modern software for the purpose of execution of the projects in specialized areas.
PSO3	Inculcate effective communication and ethics for lifelong learning with social awareness

Prerequisites:

1. A course on “Java programming”.

Course Objectives:

1. Understand knowledge of Software Testing Methods.
2. Develop skills in software test automation and management using latest tools.

Course Outcomes: At the end of the course, student will be able to

1. Develop the best test strategies in accordance to the development model.
2. Apply the test cases on test automation tools.

List of Experiments:

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 run time 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

INTRODUCTION TO SOFTWARE TESTING

Software Testing is the process of identifying the accuracy and quality of the software product and service under test. Apparently, it was born to validate whether the product fulfils the particular prerequisites, needs, and desires of the client. At the end of the

day, testing executes a framework or application with a specific end goal to point out bugs, errors or defects. The responsibility of testing is to point out the issues of bugs and give Dev (Developers) a clue to help them fix it right following the requirements.

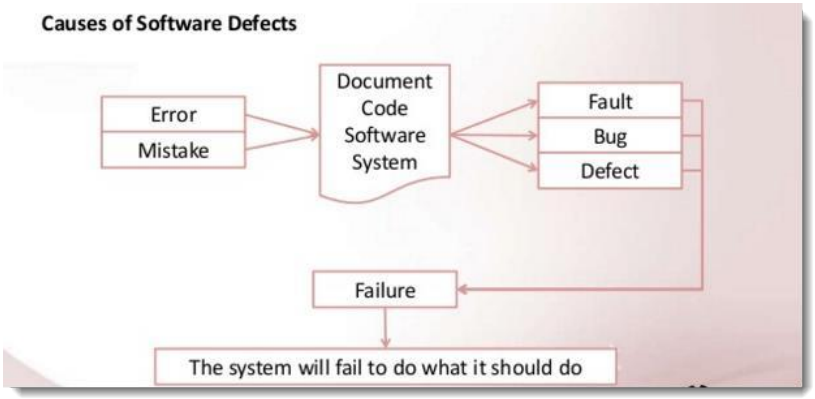
1. Software Testing Objectives:

- Uncover as many as errors (or bugs) as possible in a given product.
- Demonstrate a given software product matching its requirement specifications.
- Validate the quality of a software using the minimum cost and efforts.
- Generate high-quality test cases, perform effective tests, and issue correct and helpful problem reports.

Software testing is often divided into 2 main processes: Verification and Validation.

Verification in software testing is the process when your team just need to check whether the software, system or framework consistent, aligned with the requirements of a documentation. ***Validation*** is the process that your team needs to verify accuracy of the system. In this process, you will look back to product, system and think about what users actually want and what has been done.

In Software Testing, there is the difference between Errors, Defects, and Bugs that we should distinguish clearly to avoid misunderstanding problem.



Error is a deviation from the actual and the expected result. It represents the mistakes made by people.

Bug is an error found BEFORE the application goes into production. A programming error that causes a program to work poorly, produce incorrect results, or crash. An error in software or hardware that causes a program to malfunction.

Defect happens once the error is identified during testing; it is logged as a 'Defect' in the tracking

syste
m.

Failure is the incapacity of a system to conduct its required functions within clarified performance requirements, literally a disappointment or a letdown. And no one wants to do business with a failure.

2. Why is Software Testing important?

One of the controversy topics that is mentioned in a long meeting before starting or after reviewing a project's sprint is about the associated costs of fixing bugs. The more extended a bug goes undetected, the more exorbitant it is to settle. Basic costs versus benefits investigation will indicate overwhelmingly that the advantages of utilizing a QA Testing Engineer to approve the code far exceed the expenses.

Besides the expensive cost needed to be paid for the delay of detecting bugs, if the testing team missed out on capturing or identifying the risks and software issues accurately or exhaustively it can lead to some disasters. For example: It took NASA 7 years to identify gigantic hole in the sky due to an error in the data analysis software, the huge hole in the ozone was flagged as a software issue in the system and was ignored because they did not expect such an extreme deviation in the results thrown up by the software.

When NASA engineers reviewed their raw data, they realized that their satellites had indeed detected the hole long ago, but they completely missed it. So that, here are 5 reasons why every member of the team should take an eye on Product Testing for the whole process.

To ensure there is no difference between the actual and the expectation

Software testing is created to detect any problems occurred during coding and developing a new function or feature of a software product before it is delivered to the end users. These upcoming features should be correctly matched with what it is supposed to

be. Also, Testing tool will be a

gadget checking whether your team decently refers to the requirement. Hence, it basically is a device helping to close the gap between actual and expectation in making a software product.

To make sure your product is powerful enough no matter how many people are using.

There is a big distinction when there is a person using your product referring to hundreds of people trying to do the same thing at the same time. Your software needs to be strong enough to guarantee that there will be no crashing down or loading annoying happened when a number of people are trying to run your products. Therefore, it should be smoothly and perfectly working with everyone.

To figure out as many as possible the chance of bugs generating.

It can't be denied to say that nothing is perfect. There is always some unseen problems which may generate during using your application. The responsibility of testing tool is to avoid bugs found by users. We, who develop this application/software product should take the duty of falling down as many as possible the number of bugs which may interrupt users in future, so deliver the best experience for our users whilst using our apps.

To offer a product that can perfectly work on different browsers and tech devices

At the booming moment of this technology era, we, fortunately, welcome the existence of a number of technology devices, browsers, and operating systems, giving us the chance to choose different technology instruments to greater our tech experience. Therefore, the stress of creating an application or product which could perfectly work on most of the technology instruments has never been so great before.

To deliver the best product that we can.

Again, testing tool is created to provide the most excellent software product and service to the end users. Of course, we couldn't bring the best (since there is nothing perfect, we all know) but we could minimize the chance of bugs occurred within our capability. Before releasing, we could be proud and confident enough on the product we bring to market. In any case, unseen bugs may have a real impact on a real person. Therefore, if we could have a chance to encounter the bug before the users find it out, nothing could be greater than this.

3. Type of Software Testing:

Testing is surely a fundamental part of software development. Poor testing methodologies cause the troublesome products and unsustainable development. Having a well-prepared testing plan makes a product be more competitive and assure the products coming in a predictable timeline associated with high quality.

Apparently, a product is usually tested from a very early stage when it is just a small code tested piece by piece then being tested at the final of development when it is under the shape of a full application or software product in general. Of course, there are a number of Software Testing types out there (more than 100 different types in general); however, at the beginning, we just need to adjust a few common types that every product usually goes through before going further.

Unit Test

It is not exaggerated saying that people usually hear about Unit Test before getting noted about the software testing industry since it is the most basic testing used at the developer level. We focus on the single piece of unit code whilst it is already isolated from any outside interaction or dependency on any module before. This test requires the developer checking the smallest units of codes they have written and prove that units can work independently.

Integration Test

Still, at the developer level, after Unit Test, the combination (or integration) of these smallest codes should also be checked carefully. Integration test provides the testing modules which access to network, databases and file system. They will indicate whether the database and the network are working well when they are combined into the whole system. Most importantly, the connection between small units of code tested in the previous stage will be proven at this stage.

Functional Testing

There is no doubt to claim that functional testing is the higher level of test type should be used after Integration Test. Functional tests check for the accuracy of the output with respect to the input defined in the specification. Not much emphasis is given to the intermediate values but more focus is given on the final output created.

Smoke Test

Smoke Tests analogy comes from the electronics where an issue means the circuit board giving out smoke. After functional tests are done, a simple test will be executed at the starting point, after a fresh installation and newer input values.

Regression Test

Whenever complex bugs are stuck in a system, typically which affect the core areas of the system, regression tests are used to retest all the modules of the system.

UI Test

Well, apart from those core testing types above, GUI test is also a well-known and really popular in software engineering industry now. This graphic user interface testing ensures the specific application or products being friendly for all users. Principally, GUI Testing evaluates design components such as layout, colors, fonts, size, and so on. Also, GUI Testing can be executed both manually and automatically.

This is only a brief introduction to what is software testing. If you are interested in learning more about the discipline, you might want to start with webinars and eBooks on Huddle or try this post on Manual Testing & Automated Testing.

EXPERIMENT 1

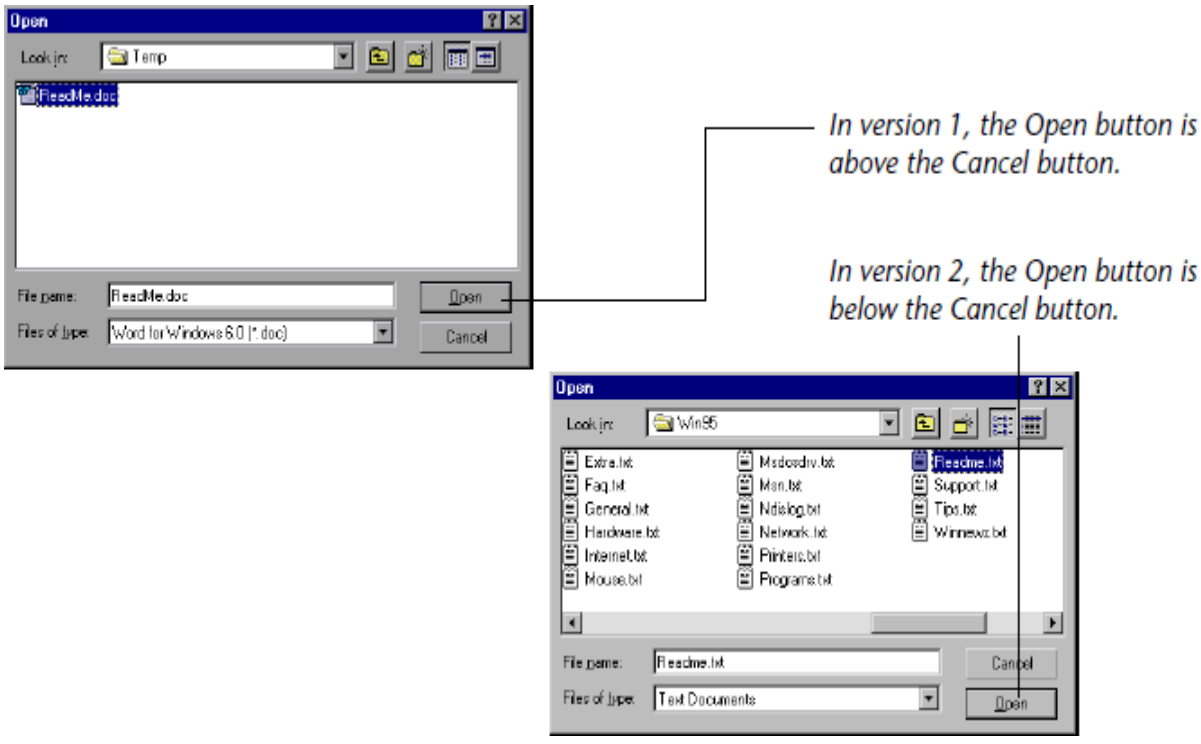
RECORDING IN CONTEXT SENSITIVE MODE AND ANALOG MODE Context

Sensitive mode records the operations you perform on your application in terms of its GUI objects. As you record, WinRunner identifies each GUI object you click (such as a window, button, or list), and the type of operation performed (such as drag, click, or select).

For example, if you click the **Open** button in an Open dialog box, WinRunner records the following:

```
button_press
("Open");
```

When it runs the test, WinRunner looks for the Open dialog box and the Open button represented in the test script. If, in subsequent runs of the test, the button is in a different location in the Open dialog box, WinRunner is still able to find it.



Use Context Sensitive mode to test your application by operating on its user interface.

To record a test in context sensitive mode:

1. Choose **Test** > **Record–Context Sensitive** or click the **Record–Context Sensitive** button.
   The letters Rec are displayed in dark blue text with a light blue background on the ***Record*** button to indicate that a context sensitive record session is active.
2. Perform the test as planned using the keyboard and mouse.
   Insert checkpoints and synchronization points as needed by choosing the appropriate commands from the User toolbar or from the **Insert** menu menu: GUI Checkpoint, Bitmap Checkpoint, Database Checkpoint, or Synchronization Point.
3. To stop recording, click **Test** > **Stop Recording** or click **Stop**.

EXPERIMENT 2

GUI CHECKPOINT FOR SINGLE PROPERTY

You can check a single property of a GUI object. For example, you can check whether a button is enabled or disabled or whether an item in a list is selected. To create a GUI checkpoint for a property value, use the Check Property dialog box to add one of the following functions to the test script: **button_check_info scroll_check_info**

edit_check_info

static_check_info

list_check_info win_check_info

obj_check_info

To create a GUI checkpoint for a property value:

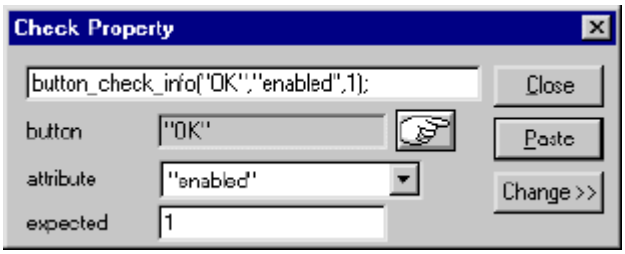
1. Choose **Insert > GUI Checkpoint > For Single Property**. If you are recording in Analog mode, press the

   CHECK GUI FOR SINGLE PROPERTY softkey in order to avoid extraneous mouse movements.

   The WinRunner window is minimized, the mouse pointer becomes a pointing hand, and a help window opens on the screen.

2. Click an object.

   The Check Property dialog box opens and shows the default function for the selected object. WinRunner automatically assigns argument values to the function.



1. You can modify the arguments for the property check.
   - To modify assigned argument values, choose a value from the **Attribute** list. The expected value is updated in the Expected text box.
   - To choose a different object, click the pointing hand and then click an object in your application. WinRunner automatically assigns new argument values to the function.

   Note: that if you click an object that is not compatible with the selected function, a message states that the current function cannot be applied to the selected object. Click OK to clear the message, and then click Close to close the Check Property dialog box. Repeat steps 1 and 2.

2. Click **Paste** to paste the statement into your test script.

EXPERIMENT 3

GUI CHECKPOINT FOR SINGLE OBJECT/WINDOW

You can create a GUI checkpoint to check a single object in the application being tested. You can either check the object with its default properties or you can specify which properties to check.

Each standard object class has a set of default checks. For a complete list of standard objects, the properties you can check, and default checks, see "Property Checks and Default Checks".

Creating a GUI Checkpoint using the Default Checks

You can create a GUI checkpoint that performs a default check on the property recommended by WinRunner. For example, if you create a GUI checkpoint that checks a push button, the default check verifies that the push button is enabled.

To create a GUI checkpoint using default checks:

1. Choose **Insert > GUI Checkpoint >for Object/Window**, or click the **GUI Checkpoint for Object/Window** button on the User toolbar. If you are recording in Analog mode, press the CHECK GUI FOR OBJECT/WINDOW soft key in order to avoid extraneous mouse movements. Note that you can press the CHECK GUI FOR OBJECT/WINDOW soft key in Context Sensitive mode as well.

   The WinRunner window is minimized, the mouse pointer becomes a pointing hand, and a help window opens on the screen.
2. Click an object.
3. WinRunner captures the current value of the property of the GUI object being checked and stores it in the test's expected results folder. The WinRunner window is restored and a GUI checkpoint is inserted in the test script as an **obj_check_gui** statement.

Creating a GUI Checkpoint by Specifying which Properties to Check

You can specify which properties to check for an object. For example, if you create a checkpoint that checks a push button, you can choose to verify that it is in focus, instead of enabled.

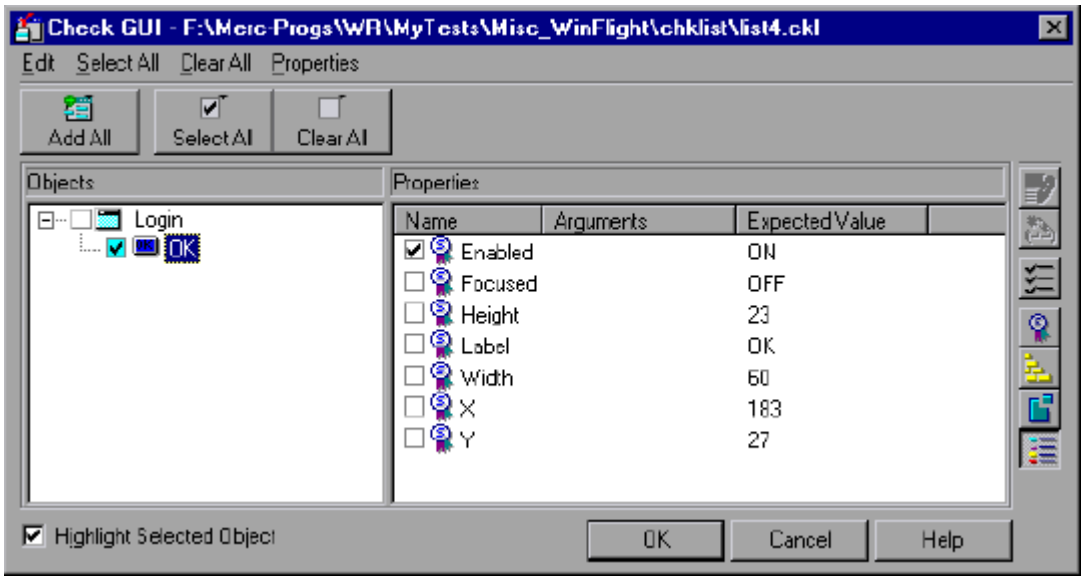
To create a GUI checkpoint by specifying which properties to check:

1. Choose **Insert > GUI Checkpoint >for Object/Window**, or click the **GUI Checkpoint for Object/Window** button on the User toolbar. If you are recording in Analog mode, press the CHECK GUI FOR OBJECT/WINDOW softkey in order to avoid extraneous mouse movements.

Note that you can press the CHECK GUI FOR OBJECT/WINDOW soft key in Context Sensitive mode as well.

The Win Runner window is minimized, the mouse pointer becomes a pointing hand, and a help window opens on the screen.

2. Double-click the object or window. The Check GUI dialog box opens.



1. Click an object name in the **Objects** pane. The **Properties** pane lists all the properties for the selected object.
2. Select the properties you want to check.

o 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 to edit it.

o To add a check in which you specify arguments, first select the property for which you want to specify arguments. Next, either click the **Specify Arguments** button, or double-click in the **Arguments** column. Note that if an ellipsis (three dots) appears in the Arguments column, then you must specify arguments for a check on this property. (You do not need to specify arguments if a default argument are specified.) When checking standard objects, you only specify arguments for certain properties of edit and static text objects. You also specify arguments for checks on certain properties of nonstandard objects.

o To change the viewing options for the properties of an object, use the **Show Properties** buttons.

3. Click **OK** to close the Check GUI dialog box.

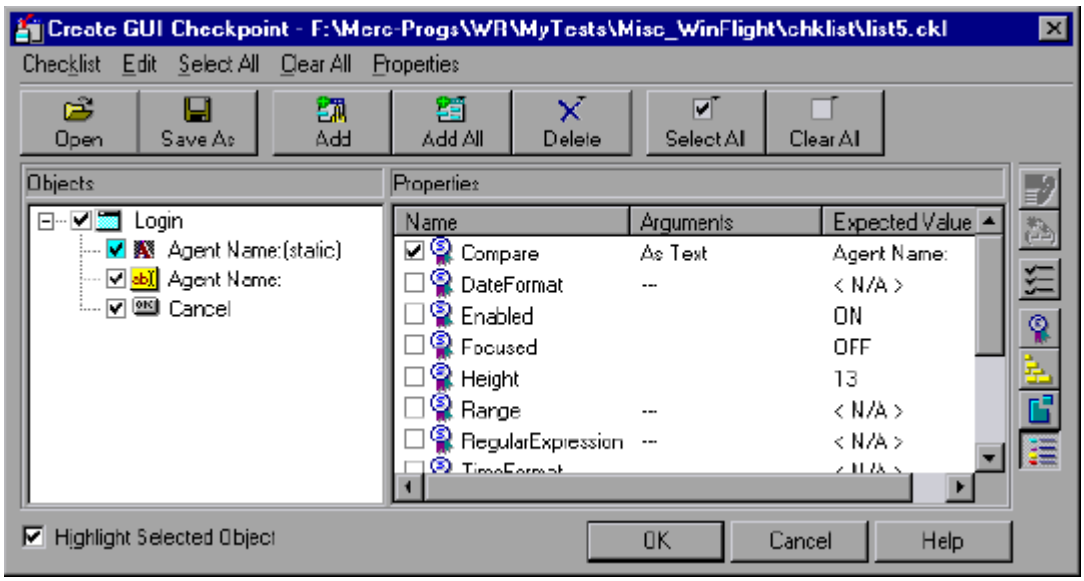
Win Runner captures the GUI information and stores it in the test's expected results folder. The Win Runner window is restored and a GUI checkpoint is inserted in the test script as an **obj_check_gui** or a **win_check_gui** statement. For more information, see "Understanding GUI Checkpoint Statements".

GUI checkpoint for multiple objects

You can use a GUI checkpoint to check two or more objects in a window. For a complete list of standard objects and the properties you can check, see “Property Checks and Default Checks”.

To create a GUI checkpoint for two or more objects:

1. Choose **Insert > GUI Checkpoint > For Multiple Objects** or click the **GUI Checkpoint for Multiple Objects** button on the User toolbar. If you are recording in Analog mode, press the CHECK GUI FOR MULTIPLE OBJECTS softkey in order to avoid extraneous mouse movements. The Create GUI Checkpoint dialog box opens.
2. Click the **Add** button. The mouse pointer becomes a pointing hand and a help window opens.
3. To add an object, click it once. If you click a window title bar or menu bar, a help window prompts you to check all the objects in the window.
4. The pointing hand remains active. You can continue to choose objects by repeating step 3 above for each object you want to check.
5. Click the right mouse button to stop the selection process and to restore the mouse pointer to its original shape. The Create GUI Checkpoint dialog box reopens.



6. The Objects pane contains the name of the window and objects included in the GUI checkpoint.

   To specify which objects to check, click an object name in the **Objects** pane.

   The Properties pane lists all the properties of the object. The default properties are selected.

- 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 to edit it.
- To add a check in which you specify arguments, first select the property for which you want to specify arguments. Next, either click the **Specify Arguments** button, or double-click in the **Arguments** column. Note that if an ellipsis appears in the Arguments column, then you mustspecify arguments for a check on this property. (You do not need to specify arguments if a default argument is specified.) When checking standard objects, you only specify arguments for certain properties of edit and static text objects.You also specify arguments for checks on certain properties of nonstandard objects.
- To change the viewing options for the properties of an object, use the Show Properties buttons.

7. To save the checklist and close the Create GUI Checkpoint dialog box, click **OK**.
   WinRunner captures the current property values of the selected GUI objects and stores it in the expected results folder. A **win_check_gui** statement is inserted in the test script.

EXPERIMENT 5

A. Bitmap checkpoint for object/window

You can capture a bitmap of any window or object in your application by pointing to it. The method for capturing objects and for capturing windows is identical. You start by choosing **Insert > Bitmap Checkpoint > For Object/Window**. As you pass the mouse pointer over the windows of your application, objects and windows flash. To capture a window bitmap, you click the window's title bar. To capture an object within a window as a bitmap, you click the object itself.

Note that during recording, when you capture an object in a window that is not the active window, WinRunner automatically generates a **set_window** statement.

To capture a window or object as a bitmap:

- Choose **Insert > Bitmap Checkpoint > For Object/Window** or click the **Bitmap Checkpoint for Object/Window** button on the User toolbar. Alternatively, if you are recording in Analog mode, press the CHECK BITMAP OF OBJECT/WINDOW softkey.

The WinRunner window is minimized, the mouse pointer becomes a pointing hand, and a help window opens.

- Point to the object or window and click it. WinRunner captures the bitmap and generates a **win_check_bitmap** or **obj_check_bitmap** statement in the script.

The TSL statement generated for a window bitmap has the following syntax:

win_check_bitmap (*object*, *bitmap*,
time **);** For an object bitmap, the syntax
is: **obj_check_bitmap (** *object*, *bitmap*,
time **);**

For example, when you click the title bar of the main window of the Flight Reservation application, the resulting statement might be:

win_check_bitmap ("Flight Reservation", "Img2", 1);

However, if you click the Date of Flight box in the same window, the statement might be:

obj_check_bitmap ("Date of Flight:", "Img1", 1);

B. Bitmap checkpoint for screen area

When working in Context Sensitive mode, you can capture a bitmap of a window, object, or of a specified area of a screen. WinRunner inserts a checkpoint in the test script in the form of either a **win_check_bitmap** or **obj_check_bitmap** statement.

To check a bitmap, you start by choosing **Insert > Bitmap Checkpoint**. To capture a window or another GUI object, you click it with the mouse. To capture an area bitmap, you mark the area to be checked using a crosshairs mouse pointer.

Note that when you record a test in Analog mode, you should press the CHECK BITMAP OF WINDOW softkey or the CHECK BITMAP OF SCREEN AREA softkey to create a bitmap checkpoint. This prevents WinRunner from recording extraneous mouse movements. If you are programming a test, you can also use the Analog function **check_window** to check a bitmap.

If the name of a window or object varies each time you run a test, you can define a regular expression in the GUI Map Editor. This instructs WinRunner to ignore all or part of the name. For more information on using regular expressions in the GUI Map Editor, see "Editing the GUI Map."

Your can include your bitmap checkpoint in a loop. If you run your bitmap checkpoint in a loop, the results for each iteration of the checkpoint are displayed in the test results as separate entries. The results of the checkpoint can be viewed in the Test Results window. For more information, see "Analyzing Test Results."

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.

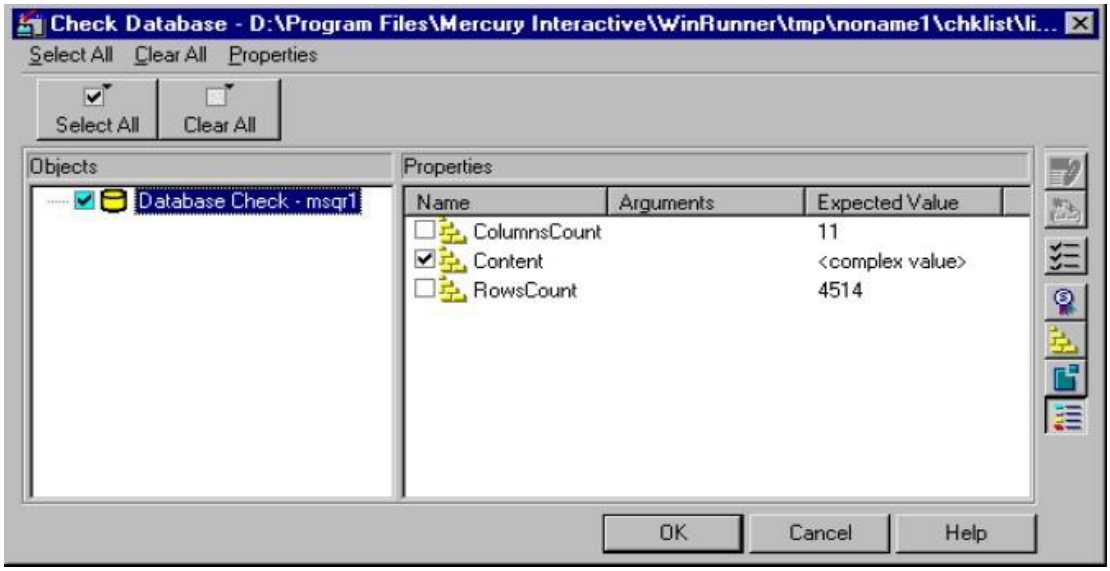
When prompted, instead of pointing to your application control with the desired value, point to the edit field where you entered the desired calculated value.

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,"

RowsCount: 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

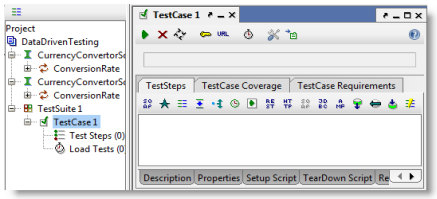
script as a **db_check** statement.

EXP

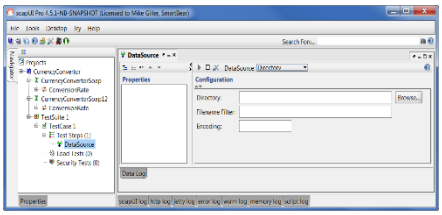
ERIMENT 9

B. Data driven test through flat files

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

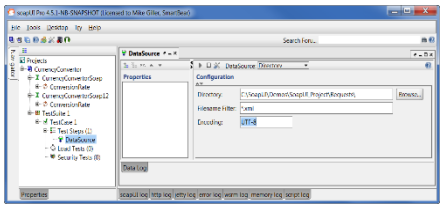


Now add a DataSource TestStep and select the DataSource type “Directory” from the dropdown in the toolbar. You should now have:

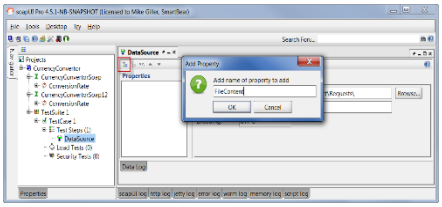


Now, select the directory where your input files are stored, add an applicable filter (e.g. “*.txt” or

“*.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

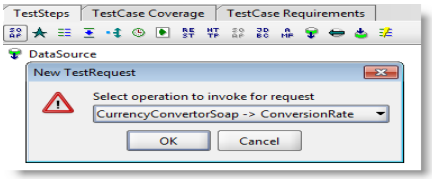


Quick tip: If your property is named “Filename” it will contain the name of the file instead of file’s

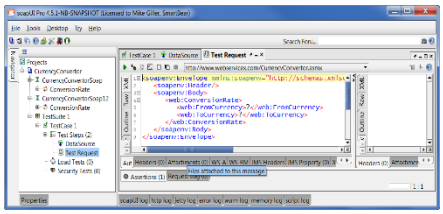
content
s.

2. 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):

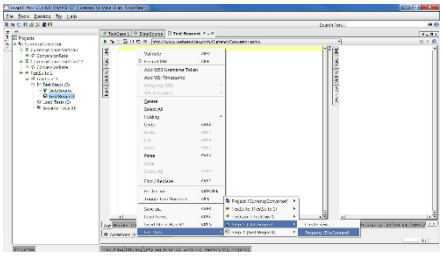


Now, I'm operating under the assumption that you have a fully built request in each of the files in your directory.

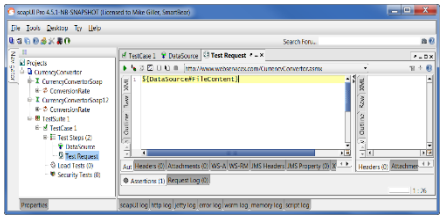
An example of an input file would be:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://www.webserviceX.NET/">
<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:

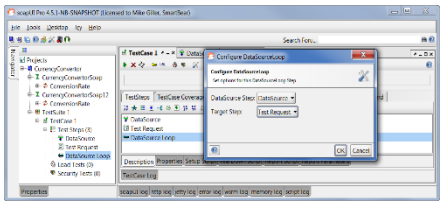


Note: If an XPATH window comes up, just click OK without selecting anything. Now your request should look like this:



3. Add DataSource Loop

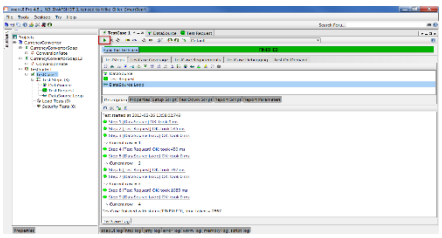
As a final step, we just need to iterate through all the files in our DataSource. So in your TestCase, add a DataSource loop step, and double click it to configure as in the picture below:



Click OK.

4. 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:

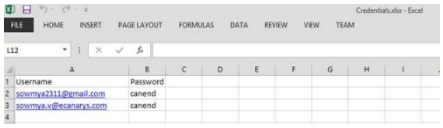
1. Java JDK 1.5 or above
2. Apache POI library v3.8 or above
3. Eclipse 3.2 above
4. Selenium server-standalone-2.47.x.jar
5. TestNG-6.9

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



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 **texit** statements, WinRunner interprets these statements differently for a batch test run than for a regular test run. During a regular test run, **texit** terminates test execution. During a batch test run, **texit** halts execution of the current test only and control is returned to the batch test.

EXPERIMENT 11

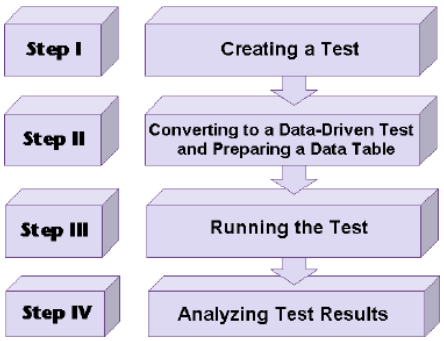
Data driven batch

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

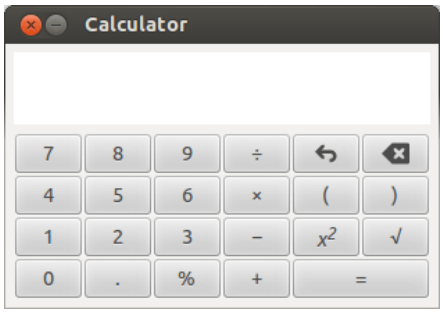
1 **Silent Mode** to continue **test execution without any interruption**, we can use this run setting.

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 t0 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