Prerequisites

Concepts you must know before doing this course

- Automation Testing:
    - Automation life cycle
    - Test tools at various levels of testing and challenges in automation testing.

Working knowledge, you must have

- Write and execute test cases manually and report defects

Hardware requirement

- **Computer Processor:** Minimum: 1.6 Ghz or higher and recommended: 3Ghz or higher.
- **Operating System:** Windows 7 Service Pack 1(32-bit/64-bit).
- **Memory:** Minimum of 2GB when more than three add-ins are loaded simultaneously.

  Note: An additional 512MB of RAM are required when using virtual machine. Additional memory is required when loading more add-ins.
- **Hard-Disk Space:** 2GB of free disk space for application files and folders you must also have an additional 1GB of free disk space on the system disk (the disk on which operating system is installed)

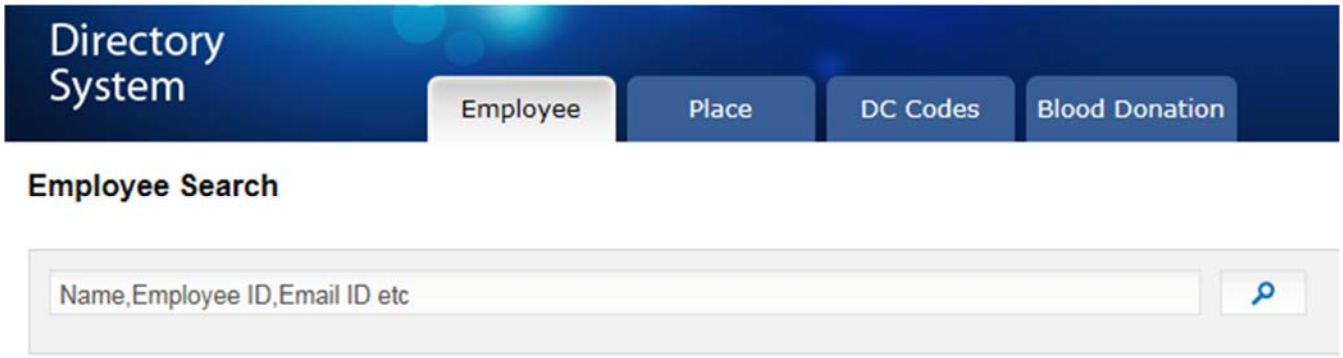
Introduction to Automation Testing

Why go for automation testing?

Let us examine a couple of test scenarios to understand how automation of tests can be beneficial.

Scenario 01

You have a functional test case to verify the search functionality of the Employee Directory System by entering an employee number.



To execute this test case manually, you will have to execute the following steps:

1. Open Employee Directory System web app
2. Enter the employee number to be searched
3. Check whether the details of the corresponding employee are displayed
4. Update the test case result

To execute the above test case, you might take approximately one minute.

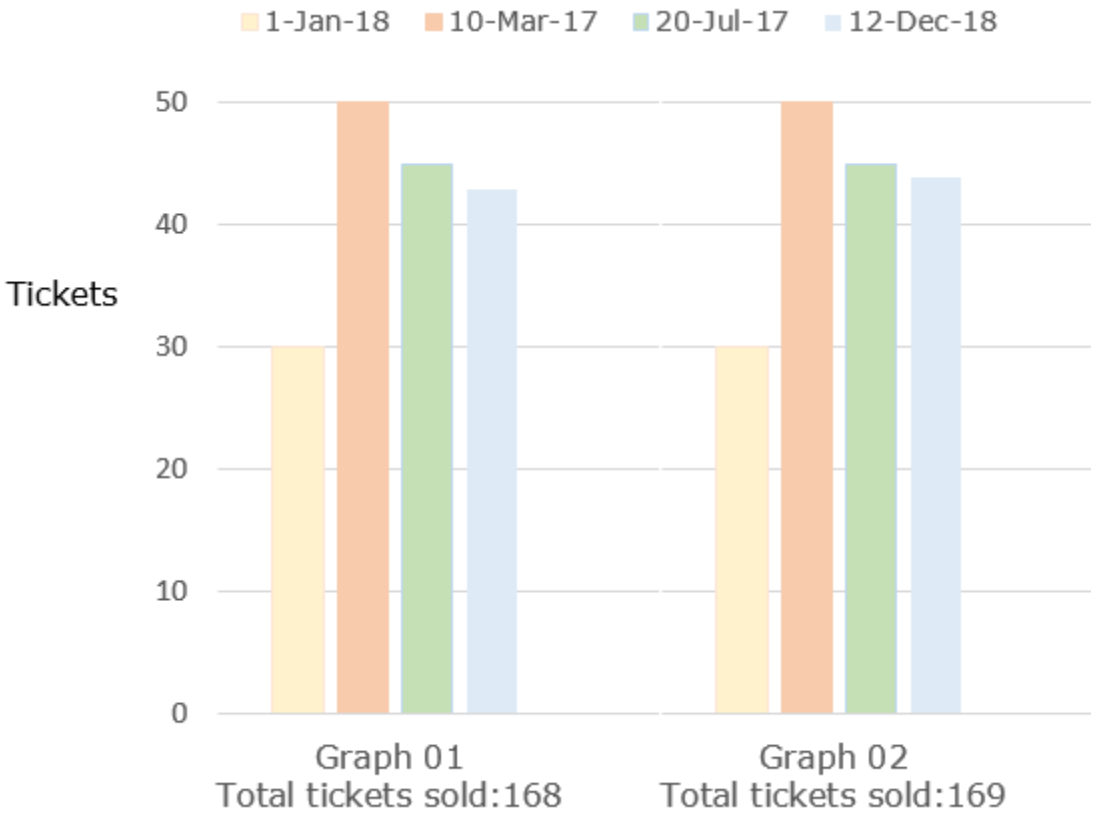
Now if you are asked to test the same functionality with 100 different employee numbers, you will have to repeat the same steps with different test data a 100 times and the process would take at least a 100 minutes.

But if you use an automation testing tool and write a script to execute all four test steps, all you have to do is start the script execution. Everything else, from launching the application to logging the test results, will be taken care of by the automation testing tool for all 100 employee numbers. The bonus is that the overall testing process would only take around ten minutes!

Scenario 02

A flight reservation application produces graphs which displays the daily ticket sales. The graph is to be updated after each successful ticket booking. You have a functional test case to verify whether the graph is updating for every successful ticket booking.

Given below are two graphs, the first was generated before inserting a ticket and the other after inserting a ticket, for the date 12-Dec-2018.



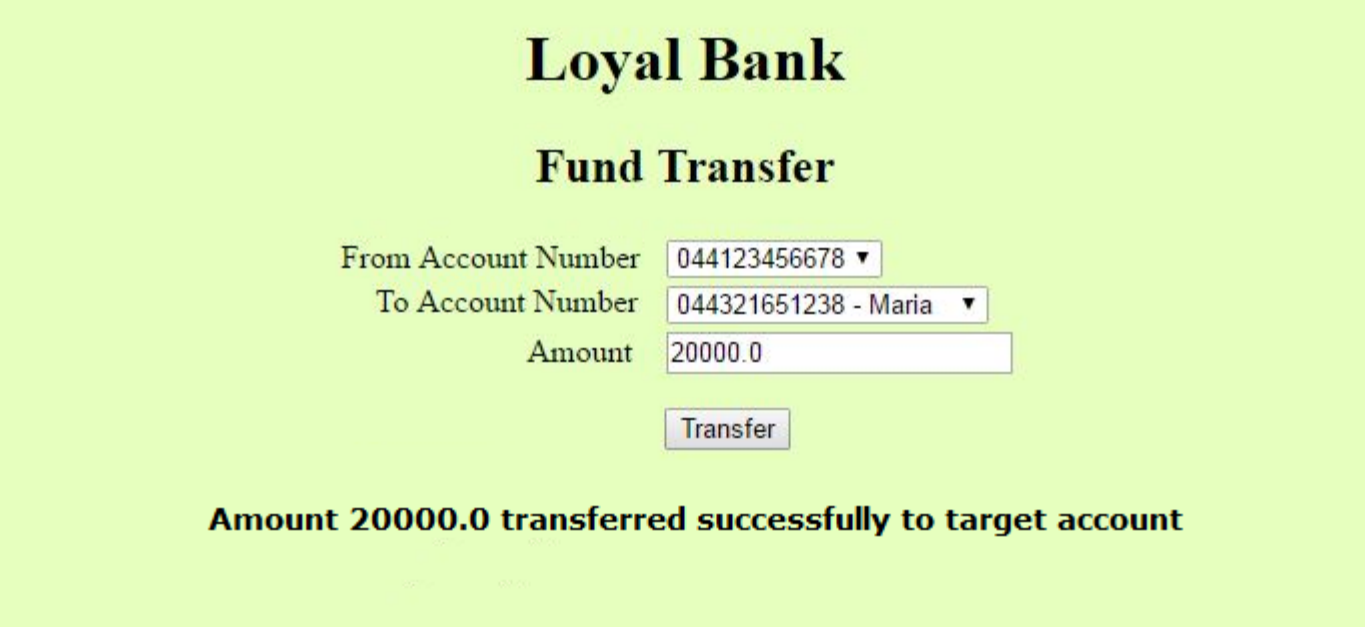
While verifying such a test case, you will have to compare the images manually and report whether the test case has passed or failed. Observing such minute changes can be challenging do through naked eyes.

Automation testing tool make it is very easy to execute such type of tests as these tools are capable of comparing the images thoroughly. It can identify even minor differences and will report the test case as a pass or fail on the basis of its comparison.

Scenario 03

'Loyal Bank' has an online banking application which allows its customers to perform banking transactions in an easy manner. Its fund transfer module (refer below image) has a performance requirement which states:

'If 1000 users are transferring funds simultaneously, the transaction must be successful and a success message should be displayed on the page within 3 seconds, for all 1000 users.'



It is not possible to test this requirement through manual testing as it is not practical to execute the scenario in 1000 machines simultaneously and to note the response time.

Through test automation, this requirement can be easily tested in a single machine by creating 1000 virtual users with the help of performance testing tools.

Note: Similar to performance testing, it is not practical to perform data warehouse testing (owing to large volume of data), web service testing (due to complex request formats) etc. manually.

All these scenarios would have given you an understanding of how test automation helps to conduct tests in a fast and efficient manner.

Automation testing - An overview

Test automation has become a vital part of the software development process as it enables you to test the functionality, performance and many other types of requirements in a fast and efficient manner.

Test automation

- process consists of converting manual tests into automated ones with the help of appropriate testing tools
- hence serves better if implemented on an existing manual process

- scripts are designed, coded and unit tested before the actual test execution
- is mainly used during regression testing phase, where the same test cases are repeatedly executed

Test automation involves repeatedly executing tasks/jobs with **high accuracy** in **minimal time** and almost **no human intervention**.

Owing to its high efficiency, test automation helps in achieving maximum test coverage and thereby enables you to deliver products to clients on time, without compromising on the quality.

Preliminary requirements for automation testing

Listed below are some of the key activities which are mandatory for achieving test automation.

1. Automation feasibility analysis
2. Identifying the right automation tool(s)
3. Test data with expected results
4. Setting up the test environment

Process of implementing these points are scattered across in automation life cycle process and we will discuss these in detail there.

1. Automation feasibility analysis

Although the objective is to achieve test automation, you will still need to have your manual test cases before starting with automation testing. This is a must as they will act as a base for creating the automation test scripts.

Now it's is not possible and/or beneficial to automate all manual test cases. This means that you must be able to analyze manual test cases and distinguish test cases which should be automated from those that cannot or need not be automated. This process is called **automation feasibility analysis**.

Below table gives you an overall idea about how scenarios can be identified whether it have to be automated or not.

Detailed questionnaire on automation feasibility analysis will be explained in automation testing life cycle.

Is this test scenario automatable?	How important is this test scenario?	Action Recommended
Yes, it is and the process is going to cost you little	I must absolutely test this scenario whenever possible	Definitely automate
Yes, it is and the process is going to cost you little	I need to test this scenario regularly	Should automate
Yes, it is and the process is going to cost you little	I need to test this scenario once in a while	Think before automation
No, it is not possible	NA	Not possible for automation

For example, consider the below scenario.

In a train ticket reservation portal there is a feature for senior citizens to get a discount of 10% on all the bookings they are doing.

- Is this test scenario automatable? - Yes it is and the process is going to cost you little
- How important is this scenario? - I need to test this scenario regularly

The above scenario is a feature which is commonly used by normal users as many users belong to the category senior citizenship. Since this is a key feature of the application and since it is commonly used by the user, this scenario should be automated.

Note: As mentioned earlier, for specialized testing areas like performance, data warehouse etc. only automation testing is practical. Hence, the automation feasibility analysis stage will be skipped.

2. Identifying the right automation tool(s)

Post automation feasibility analysis, selecting the automation tool which best fits your project requirements is key. Given below are some of the major categories of test automation tools which are commonly used in projects.

a. Functional testing tools

These tools are used for automated verification of the functional requirements of the AUT. They typically compare the actual output of a test case to its expected output, and the result (pass or fail) will be logged and displayed accordingly. These tools help in checking the existence of a page, image comparison, object property comparison, text matching, database value comparison etc.

E.g. HP Unified Functional Testing, Selenium

b. Performance testing tools

These tools are used for assessing the performance of the AUT by running various types of performance tests such as load tests, stress tests, spike tests, endurance tests, volume tests etc. It is not possible to do any of these tests using manual testing methods and so, performance testing tools have a high demand and hence are very costly. Additionally, unlike functional testing tools, performance testing tools require high configuration servers to balance the load while conducting the testing process.

E.g. HP Performance Centre, Apache JMeter

c. Web service testing tools

Web services typically accept service requests only in a particular format like XML, JSON etc. So as a web service tester, you will have to create test data in these required formats, which might be difficult, time consuming and moreover, you might not be skilled in those technologies. This is where web service tools come in. They are able to generate the test data in the formats required by the web service. They are also capable of testing the request send, response received, evaluating the performance of the service etc.

E.g. CA DevTest, SmartBear SoapUI

d. Data warehouse testing tools

Data warehousing is the process of integrating the data from different data sources like MS SQL Server, Oracle, flat files etc. into a single target data source in enterprise applications. Often the data from the various sources are formatted (transformed) to meet the requirements of the target data source. To validate the quality of the extracted and transformed data, different types of data warehouse testing tools are used. These tools will help you compare target data to its various sources to identify issues such as data mismatch, integrity violation, statistical differences etc.

E.g. Informatica Data Validation

e. Test management tools

Test management tools help in managing test artifacts (requirements, test cases, defects etc.) in a centralized repository. These tools are used for storing requirements, creating test cases, executing test cases, logging the defects, creating the requirement traceability matrix etc. Detailed report generation and customization options are also a standard feature of all test management tools. You can also integrate certain test automation tools like HP UFT, HP PC etc with test management tools like HP ALM to provide an integrated automation experience.

E.g. HP Application Lifecycle Management (HP ALM), Atlassian JIRA, Mozilla Bugzilla

3. Test data with expected results

Before coding the automation test scripts, you should be ready with the test data and the corresponding expected results for all the automatable test scenarios. Test data is typically stored in Excel sheets, flat files or databases and your automation test scripts will pick the data from these sources during test execution iterations and comparisons.

4. Setting up the test environment

Ensuring the availability of the required hardware, software and application configurations helps in the smooth running of test automation tools.

Let us see how the above mentioned points are implemented with the help of automation life cycle.

Automation testing life cycle

The automation testing life cycle consists of four phases, namely,

Implementing these phases accurately will help you in defining and executing the test automation process, ultimately yielding a better product.

The first two phases are ideally performed by business analysts, project managers etc. and the last two phases are done by test engineers.

So let us take a closer look at each of these phases.

Requirement gathering and automation feasibility analysis

There are mainly fours steps involved in the process of requirement gathering and automation feasibility analysis.

1. Understand the project and technical architecture of the AUT
2. Baseline the requirements
3. Automation feasibility analysis
4. Zeroing down on the appropriate test automation tool(s)

All the below four steps will be performed by business analyst, project manager etc. after having proper discussion with the higher management team.

1. Understand the project and technical architecture of the AUT

Before starting the test automation process, project requirements and its technical architecture. It is necessary that you go through all the design level documents so that you have a clear picture regarding the scope of the project and what you are supposed to test.

2. Baseline the requirements

Similar to the regular STLC, it should be ensured that all the requirements are thorough, testable and non-ambiguous. Identifying issues pertaining to requirements early on will help in preventing the spreading of defects across the project. While baselining the requirements, additionally categorize them as functional, performance, security etc. After that automatable functional requirements should be identified through a feasibility analysis.

3. Automation feasibility analysis

For every requirement module, checking have to be done on whether its corresponding test cases can be automated. If yes, then weigh out the pros and cons of doing so and conclude whether the test cases can/should be automated.

Given below is some of the questions which needs to be answered or discussed while conducting the automation feasibility study. If any of

the answer matches with the below table, then automation of that scenario is not feasible.

Sl No	Questions	Answer	Remarks
1	Is the test case reusable across scenario / cycle / region / country	No	This scenario will not be considered for automation.
2	Is the business process addressed by the test case due for sun setting / change?	Yes	This scenario will not be considered for automation.
3	Is the manual execution effort for the test case nearly same (equal, less, less difference) as automation execution effort	Yes	This scenario will not be considered for automation.
4	Is there a report getting generated from this test case?	Yes	Can be automated unless question 7 is answered as no
5	Is the flow within the test case changing because of configuration settings / data	Yes	This scenario will not be considered for automation. In order to consider this for automation 1. All possible variants based on data and navigation should be considered. 2. If the flow is predictable due to configuration settings
6	Is the waiting time predictable between switching systems?	No	This scenario will not be considered for automation. Else a maximum wait time needs to be set for the script.
7	Is 'verification of the report content' an expected result from this test case?	Yes	This scenario will not be considered for automation. Report generating test case can be automated if it is found to be suitable for automation. However, Report verification should preferably be done manually.
8	Is there an external system like printer, fax, telephone etc. involved in the test case?	Yes	This scenario will not be considered for automation.
9	Is 'verification of a downloaded file' there a file getting downloaded after execution of the test case?	Yes	This scenario will not be considered for automation. Analysis after downloads will not be carried out by Automation scripts. If any external system windows are involved during download, such test cases will not be automatable.
10	Is there an involvement of non UI applications (like UNIX)?	Yes	This scenario will not be considered for automation.
11	Is there a legacy system involvement in the test case?	Yes	This scenario will not be considered for automation The test script should not involve any legacy systems. As the legacy systems will over a period of time be replaced by the new system, the scripts will not be valid after a period of time.

Example

Scenario name: ‘Transfer fund’ for XYZ bank application

Requirement: User should be able to transfer a maximum amount of 2 lakh rupees from his/her account to any other bank account

Let's cross check the scenario with all the eleven questions in the above feasibility analysis table.

Sl No	Questions	Answer	Scope for automation
1	Is the test case reusable across scenario / cycle / region / country	Yes	Yes
2	Is the business process addressed by the test case due for sun setting / change?	No	Yes
3	Is the manual execution effort for the test case nearly same (equal, less, less difference) as automation execution effort	No	Yes
4	Is there a report getting generated from this test case?	No	Yes
5	Is the flow within the test case changing because of configuration settings / data	No	Yes
6	Is the waiting time predictable between switching systems?	Yes	Yes
7	Is 'verification of the report content' an expected result from this test case?	No	Yes
8	Is there an external system like printer, fax, telephone etc. involved in the test case?	No	Yes
9	Is 'verification of a downloaded file' there a file getting downloaded after execution of the test case?	No	Yes
10	Is there an involvement of non UI applications (like UNIX)?	No	Yes
11	Is there a legacy system involvement in the test case?	No	Yes

The answers in the feasibility analysis table indicate that there **is scope for automation** in this scenario. If the answer to any question indicates that there is no scope for automation in the scenario, the analysis process can be stopped.

To finalize the automation feasibility analysis, we have to decide on two more factors.

1. Usage of the module – How often the module is used by the user?

   Transfer fund is a common transaction and hence is a highly used module by end users.

2. Object identification – Whether the tool is able to identify the objects in the module

   Let us assume that we have selected HP UFT tool for automation and the tool is identifying the objects in the module successfully.

   Note: Tool selection process will be explained in the next step. Ideally tool selection and automation feasibility analysis steps will be done in parallel.

Thus we arrive to the below points:

Scope for automation – **Yes**

Usage of module – **High**

Object identification – **Yes**

So with reference to the above points we can come to the conclusion that the scenario 'Transfer fund' should be automated.

The automation feasibility analysis report for three scenarios is created by considering the above facts.

Automation Feasibility Analysis Report								
Sl No	Functionality name	No. of test cases for particular functionality documented	Access/privilege required for automation granted	Tool used	Usage of module	Scope for automation	Possibility of object identification using selected tool	**Module should be automated?**
1	Login	7	Yes	HP UFT	High	Yes	Yes	Yes
2	Forget Password	3	Yes	HP UFT	Low	Yes	Yes	No
3	Transfer Fund	15	Yes	HP UFT	High	Yes	Yes	Yes

In the above table 'Forget Password' scenario is rarely being used by the end users. Even though scope for automation and object identification is there, this scenario is still not recommended for automation due to low usage.

4. Choosing appropriate test automation tool(s)

Primary considerations while choosing the appropriate test automation tool are

a. Type of testing tool

Depending upon the types of testing, appropriate tool need to be chosen.

For example, if it is functional testing to be done then functional test automation tools like UFT, Selenium etc. can be chosen and for data ware house testing, you can go for Perfaware, Informatica DVO etc.

b. Client recommendations

In certain scenarios client would have already decided the testing tool to be used by the testers depending upon their budget, previous experience, alliance with tool vendors etc.

c. Built-in features of the tool and enhancement options

Different tools are available in market for each type of testing. A small research on the tools must be done and should identify which tool is having the most number of features which is required for your project.

For example, if the project has more GUI (Graphical user interface) test cases, then UFT is preferred over Selenium in functional testing tools.

d. Vendor support (in case of licensed tools)

In case of licensed tools, vendor support is one of the most important factor companies are looking for. If it is a critical project and if something goes wrong with the tool, then immediate support should be available from the vendor side. Past experience, market ratings and recommendations helps in identifying the best vendor supported tools.

e. Reporting feature of the tool

The report generated by the tool after the execution should be clear and concise as the reports will be shared between different stakeholders like customers, developers etc.

Licensed tools are preferred over freeware tools as they are considered in reporting the details in the best possible manner.

f. Budget considerations

It is up to the client to decide how much they are ready to pay for the tool and which tool to be selected based upon budget considerations

g. Availability of skilled resources

Automation tool requires skilled people with enough knowledge in the tool. If not available considerable amount of cost and time have to be utilized for creating the skilled resources.

Example

Scenario: A banking application has to be functionally tested through test automation process. The client is new to automation and is ready to pay 50 lakhs per year for the tool license as they need their product to be flawless. The client need detailed report of test result after each cycle.

Let us see which tool is best for this scenario

Parameters	Remarks
Type of testing tool	Functional testing tool as you are supposed to do functional testing on the application
Client recommendations	Nil
Built-in features	Web object identification since it is a web application
Vendor support	Required since banking applications are high risk applications
Reporting feature	Need good reporting feature as client needs the report cycle wise
Budget constraints	50 lakhs per year for the license

From the above conditions we can narrow our search to functional testing tools like HP UFT, RFT, VSTS etc.

Assuming that you have enough resources for HP UFT among the selected tools we can finalize the automation tool as HP UFT.

Strategy and design

There are mainly fours steps involved in the process of strategy and design

1. Perform ROI analysis based on estimated values
2. Prepare the strategy document
3. Identify the most suitable test automation framework
4. Setting up test environment

1. Perform ROI analysis based on estimated values

Performing a Return of Investment (ROI) for your planned automation can help you estimate, right at the beginning, the actual returns that can be derived from the investments made in terms of time, resource and money. You can weigh those against the benefits you will gain from automation.

Scenario

After performing the automation feasibility analysis, the testing team came up with the conclusion that a total of 500 test cases can be automated. They performed an ROI calculation and came up with the conclusion that a total of 2191 Person Hours is needed for creating and executing automation scripts, for all the 500 test cases, for three cycles.

Automation					
Process	UOM	Cycle 1	Cycle 2	Cycle 3	Total
Total number of test cases be automated	Test case	500	500	500	
Automation feasibility analysis	Hrs	100			100
Automation scripting	Hrs	2016			2016
Specific changes	Hrs	0			0
One time execution	Hrs	25	25	25	75
Total automation effort	Hrs				**2191**

UOM = Unit of measurement

For the same 500 test cases, the effort required for manual testing of all 500 test cases was identified as 250 Person Hours per cycle, i.e., a total of 750 hours for three cycles.

Manual					
Process	UOM	Cycle 1	Cycle 2	Cycle 3	Total
One time Execution	Hrs	250	250	250	750
Specific changes	Hrs	0	0	0	0
Total manual effort	Hrs				**750**

For three cycles, the total manual testing effort of 750 Person Hours way better as compared to the total automation testing effort of 2191 Person Hours. But let us see what happen when the project reaches its 25th cycle.

Without automation			With automation		
Cycles	Effort (P-Hrs)	Cumulative Effort (P-Hrs)	Initial Effort (P-Hrs)	Effort (P-Hrs)	Cumulative Effort (P-Hrs)
1	250	250	**2116**	25	2141
2	250	500		25	2166
3	250	750		25	2191
4	250	1000		25	2216
5	250	1250		25	2241
6	250	1500		25	2266
7	250	1750		25	2291
8	250	2000		25	2316
9	250	2250		25	2341
10	250	**2500**		25	**2366**
11	250	**2750**		25	**2391**
12	250	**3000**		25	**2416**
13	250	**3250**		25	**2441**
14	250	**3500**		25	**2466**
15	250	**3750**		25	**2491**
16	250	**4000**		25	**2516**
17	250	**4250**		25	**2541**
18	250	**4500**		25	**2566**
19	250	**4750**		25	**2591**
20	250	**5000**		25	**2616**
21	250	**5250**		25	**2641**
22	250	**5500**		25	**2666**
23	250	**5750**		25	**2691**
24	250	**6000**		25	**2716**
25	250	**6250**		25	**2741**

P-Hrs = Person Hours

From the above data, it is clear that from the 10th cycle onwards, we can see significant saving of effort with automation testing. The effort savings keeps on increasing as the number of cycles increase.

Hence ROI analysis serves its purpose by confirming the benefits of test automation and we can now proceed to create the automation test strategy document.

2. Prepare the strategy document

As you already know, a test strategy document is a guideline which helps in achieving the test objectives mentioned in the test plan document. In addition to the details captured in a regular STLC test strategy document, you also have to record test automation specific details such as

- **test automation objectives**

  This part will define the objectives which have to be achieved through test automation which includes the percentage of test cases to be passed, allowed rate of low priority bugs etc. It also includes the details about the final product to be delivered.

  Example:

  Total test cases to be automated : 500

  Required minimum pass percentage : 95% (475 test cases)

  Based on the objective, the tester has to stop testing once 500 test cases have been automated and 475 of them have passed

- **test environment setup requirements for automation**

  Hardware and software requirements needed for the servers in which the AUT and testing tools to be installed have to be detailed in this part. Referring to these data, test environment setup will be done.

- **test automation and test management tools required**

  Details of test automation tool and test management tool used for test automation has to be defined in this section. This includes the name of the tool, version of the tool and the vendor details.

- **tool license management plan**

  If the tool selected is a licensed one, then the details about the license including the cost, validity, server in which the license should be installed etc. have to be mentioned.

There is no standard template for test strategy document as each project or company has their own unique priority and set of rules for the software

testing approach. You will need to go through the test strategy documents from time to time to map your testing progress and to ensure that it is headed in the right direction.

3. Identify the most suitable test automation framework

A test automation framework is the scaffolding that provides an efficient execution environment for your automation test scripts. It is used to automate the test execution process by relating all the test scripts, test data and all other functions which are stored in separate files (XLS, XML, VBS etc.). The framework will call these files based on the conditions specified by you in the framework and execute the test cases on the AUT. It also combines coding standards, best practices, project hierarchies, modularity, reporting mechanisms etc.

While there are many advantages in using test automation frameworks, some of the most valuable ones are ease of scripting, scalability, modularity, process definition, reusability and low maintenance requirements.

Example:

- If the test cases have operations which is being used in lot of other applications (eg: open browser, click, enter text etc.) then the suitable frame work is **keyword driven**
- If the test cases have to be connected with test data saved in external sources like XLS, database etc. then the suitable frame work is **data driven**

Note: More details, including the creation of automation testing frameworks, can be learned along with different test automation tool courses.

4. Setting up test environment

Next, test environment settings needed for the installation of test automation tools need to be defined. Backup plan and restore strategy of test data also have to be defined during test strategy phase. Following are the key areas that has to be taken take care while setting up the automation test environment

- Systems have to be ready and the AUT, database etc. must be installed in the corresponding servers
- Test data should be ready in the required format (Eg: XLS, XML etc.)
- Corresponding operating system and the tool must be successfully installed with the license key in the testing machine.

- Desired browsers, extra hardware components, network connections etc. should be up in the system
- Documentation required like reference documents/configuration guides/installation guides/ user manuals must be ready for reference purpose

Scripting and dry run

There are mainly fours steps involved in the process of scripting and preparing for execution

1. Create test scripts and organize them into a library
2. Associate the test scripts with test data
3. Integrate the test with framework and perform a dry run
4. Prepare the execution plan

As a test engineer you have to take care of the following four steps in the process of scripting and dry run. Implementation of these steps will be learned in detail in the corresponding tool related courses.

1. Create test scripts and organize them into a library

You have to create test scripts for every identified automatable test scenario. The process of test script creation as well as the scripting language will differ from tool to tool. Once you have created the scripts, you should organize them into reusable components which, depending upon the automation tool used, are recognized with different terms such as functions, classes, actions etc. You can then save these reusable components to a centralized location called library. A library can be a centralized location external to the testing tool or it can be inside a test management tool, such as HP ALM.

Given below is a sample HP UFT test script written inside a function.

```
Function Booking()

    Browser("Home").Page("Home").Link("RequestASeed").Click
    Browser("Home").Page("Request a Seed").WebEdit("txtSeedName").Set "Guava"
    Browser("Home").Page("Request a Seed").WebEdit("txtQuantity").Set "3"
    Browser("Home").Page("Request a Seed").WebEdit("txtRequesterName").Set "Mark"
    Browser("Home").Page("Request a Seed").WebEdit("txtRequesterEmailId").Set "mark@mit.com"
    Browser("Home").Page("Request a Seed").WebEdit("txtRequesterMobileNumber").Set "7878909090"
    Browser("Home").Page("Request a Seed").WebList("ddlNotification").Select "Mobile"
    Browser("Home").Page("Request a Seed").WebButton("Submit Your Request!!!").Click
    Browser("Home").Page("Request a Seed").WebElement("Your request has been").Click
    phone_no=mid(Browser("Home").Page("Request").WebElement("request").getroproperty("innertext"),62,10)
        If phone_no="7878909090" Then
            reporter.ReportEvent micPass,"TC01","Passed"
        else
            reporter.ReportEvent micFail,"TC01","Failed"
        End If
            Browser("Home").Page("Request a Seed").Link("Home").Click

End Function
```

2. Associate the test scripts with test data

The scripts that you have created and organized have to be associated with its corresponding test data. As discussed earlier, test data will be available in Excel sheets, databases or flat files. You will have to write appropriate scripts to access data from the source and pass it to the application under test. Again, the process of implementing this will differ from tool to tool.

3. Integrate the test with framework and perform a dry run

Automation framework helps in executing all the needed test cases at a go and the results are reported accordingly. So once the scripts are ready, you need to integrate the scripts, test data sources, function libraries, recovery files etc. into the automation framework which was identified during the strategy and design phase.

However, before starting the actual test execution, you have to perform a dry run on the framework to ensure that no scripting errors have crept in during the framework integration process. Any errors identified during the dry run have to be corrected and you have to verify that the framework is working perfectly before proceeding for actual test execution.

4. Prepare the execution plan

Once the framework is ready, you have to decide on the test execution plan which basically involves planning the date and time for starting test

execution of each test script. You can initiate test script execution in two ways.

- Manually

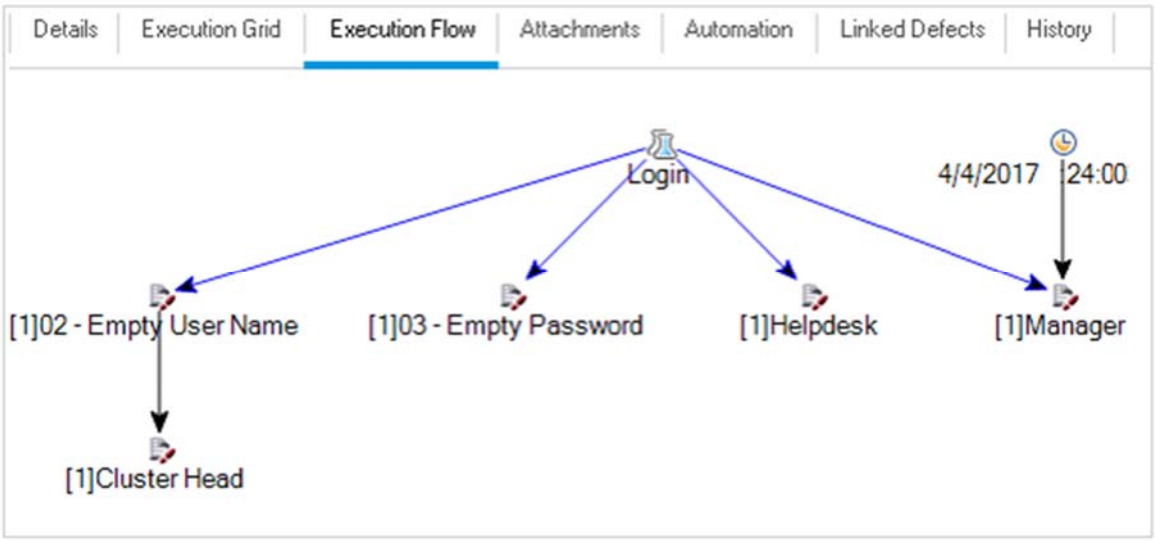
  You have to click the execution (run) button in the tool manually to start the execution process. The executions have to be started by you at the planned date and time for each cycle like the one give below.

Date	Module	No: of test cases	Time
17-Apr-2017	Login	4	10:00 AM
17-Apr-2017	Balance check	8	2: 00 PM
18-Apr-2017	Transfer fund	12	9:00 AM
19-Apr-2017	Bill Payment	20	10:00 AM

- Automated through test management tool

  You can automate the initiation of test execution process with the help of test management tools. You have to configure the test management tool so that it can connect with the test automation tool. Then, the date and time to start execution have to be fed inside the test management tool. At the mentioned start time, the test management tool will automatically invoke the test automation tool and trigger test execution.

  Below given is a sample execution plan in ALM



Execution and result analysis

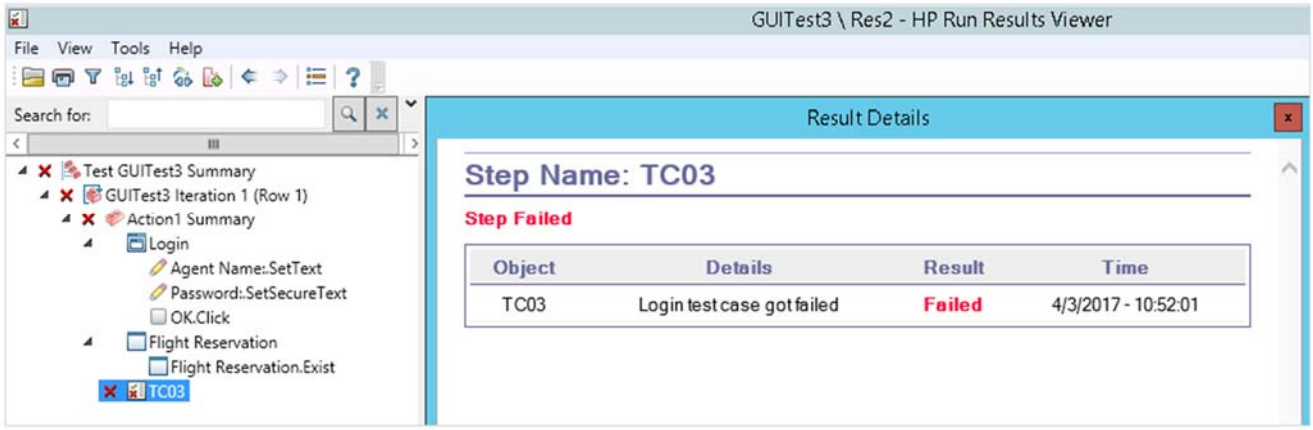
There are mainly fours steps involved in the process of execution and result analysis

1. Execute the scripts and verify the results
2. Log the defects for failed test cases
3. Rerun the scripts after fixing the defects
4. Share the results with necessary stakeholders

1. Execute the scripts and verify the results

Execute your scripts according to the execution plan and verify the results. The result of test case will either be a pass or a fail. This can be verified through the result analysis page of your test automation tool.

Given below is the result analysis data of a failed test case in HP UFT.



2. Log the defects for failed test cases

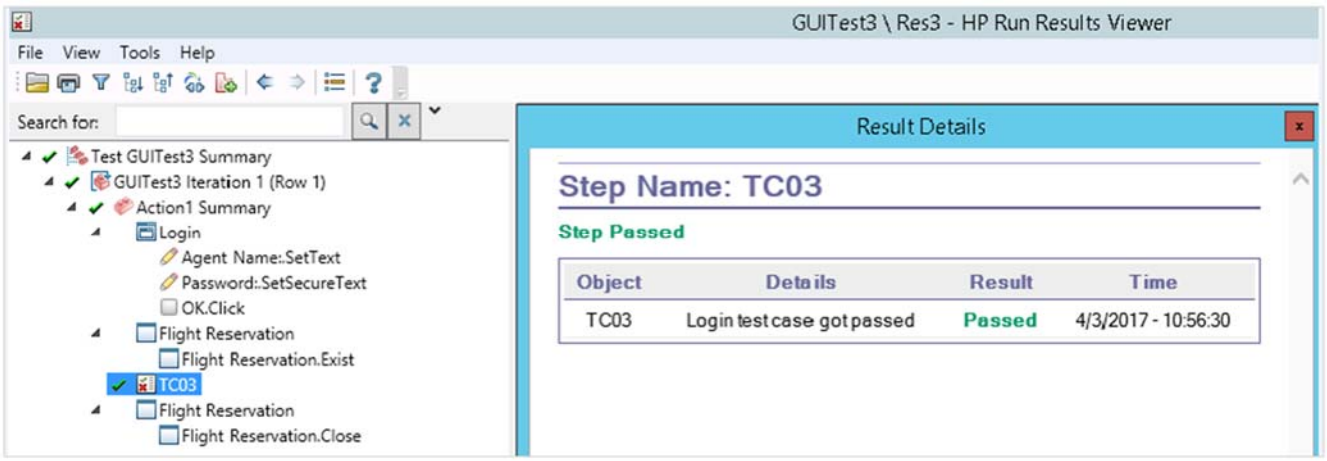
A failed test script points to defect(s) in the application. As soon as you have identified the cause of test script failure, you have to log the defect in the defect tracker.

If the initiation of execution is done through a test management tool, then the defects can be logged in the test management tool itself along with result analysis data from the test automation tool.

3. Rerun the scripts after fixing the defects

Once the logged defects are fixed by the developer you can rerun your scripts and verify the results. If the previously failed test cases are passed, you can close the logged defect. If not, you need to reopen the same.

Given below is the result analysis data of a passed test case in HP UFT.



4. Share the results with necessary stakeholders

Once the iterations have been completed, the test results can be shared with all stakeholders, primarily with the clients. Suggestions can be taken from the clients and once the changes are incorporated, you can get the sign off from them.

Advantages, disadvantages and challenges of automation testing

To summarize, the key advantages of automation testing are

- It is very helpful if a set of tests have to be executed repeatedly for a large number of users
- Automation facilitates to execute regression tests in mainstream scenarios on time. For example, execution can be set to happen overnight without any manual intervention and you will get the results the next morning
- It helps a lot in cross browser testing as the same test script with very little modification can be run on multiple browsers
- Unlike manual testing where the tests are run sequentially, automated tests can be run at the same time on different machines

Some of the common disadvantages of automation testing are

- The initial cost of creating test scripts and configuring automation framework is more than the cost of executing the tests manually
- Automating visual references is not possible. For example, a font or a color cannot be input via code or automation tools and thus it becomes more or less a manual testing process
- Licensed automation tools are expensive even though they provide good support in test automation. Since companies are investing a lot of money on the tool licenses, they have to use the tool where ever possible to get the maximum utilization.
- Proper training on automation tools and knowledge of various scripting languages are a must for working with automation testing tools.

The main challenges faced by companies trying to adopt test automation are

- Non-availability of a test tool which is a one-point solution for testing of applications developed in different technologies
- Huge upfront investment in automation tools and training and continuous maintenance costs
- Tool vendors downplay the limitations of test automation
- Inability of tool vendors to provide appropriate solutions/support for third party controls in the applications
- Non availability of personnel who are experienced in test automation concepts
- Non-availability of personnel who are experienced in testing as well as programming skills

Misconceptions about test automation

- Test automation is simply “record and playback”

  No, it’s not. You are supposed to do lot of enhancement in your test script to create the perfect script

- Test automation is the “silver bullet” to solve all testing problems

  No, it’s not. There are many limitations for test automation and it can’t reveal you all the bugs.

- ROI will be achieved immediately

  Never. For getting ROI you have to utilize the tool very efficiently and lot of cycles of iterations are needed to achieve ROI.

- Manual testing would be replaced by automation testing

  Never. Manual testing is the base of testing and it have to be done on all the applications to make sure the AUT is stable. Automation testing can be only implemented in stable products.

Case study - Effort saving through automation

Case study:

Below given is a case study of one of the client of Infosys who achieved significant reduction in the quality control cost and testing efforts with the implementation of test automation.

The Client is a leading provider of Insurance in United States. Their services include Group Life, Dental, Disability and Individual Disability Income Insurances. The project involves various testing of the following applications.

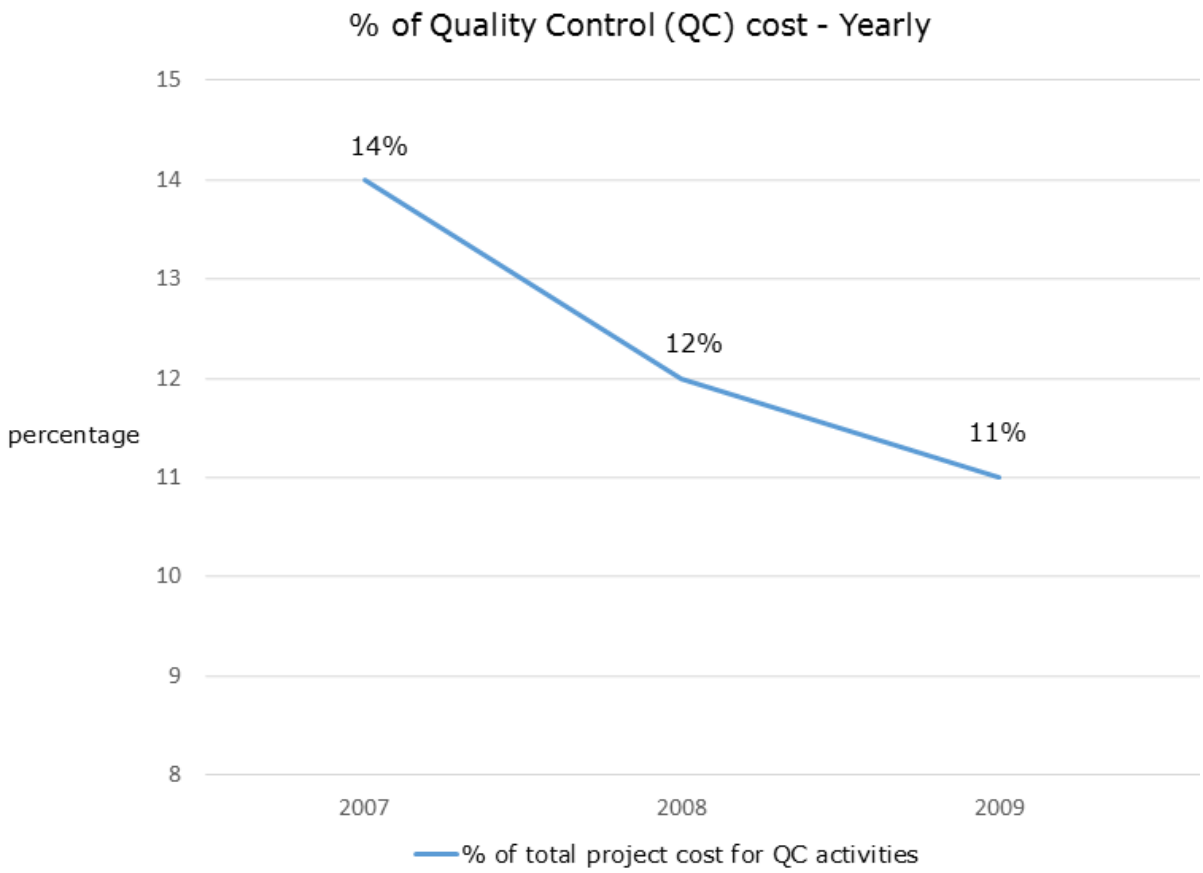
- Life Track
    - Web based Applications used by Group Life/CI/LTC Policy Administration, Claim generation and processing and Recordkeeping.
    - PowerBuilder based application for State Of Health (SOH) and Recordkeeping
- Dental Track
    - Mainframe Application for Dental claims filing, processing and verification
- Disability Track
    - Web based Applications used by Disability users for claim adjudication
    - Individual Disability Income Policy Administration
    - PowerBuilder based application for Claim generation and processing
    - Web Services based Testing for Web based Absence Management application.

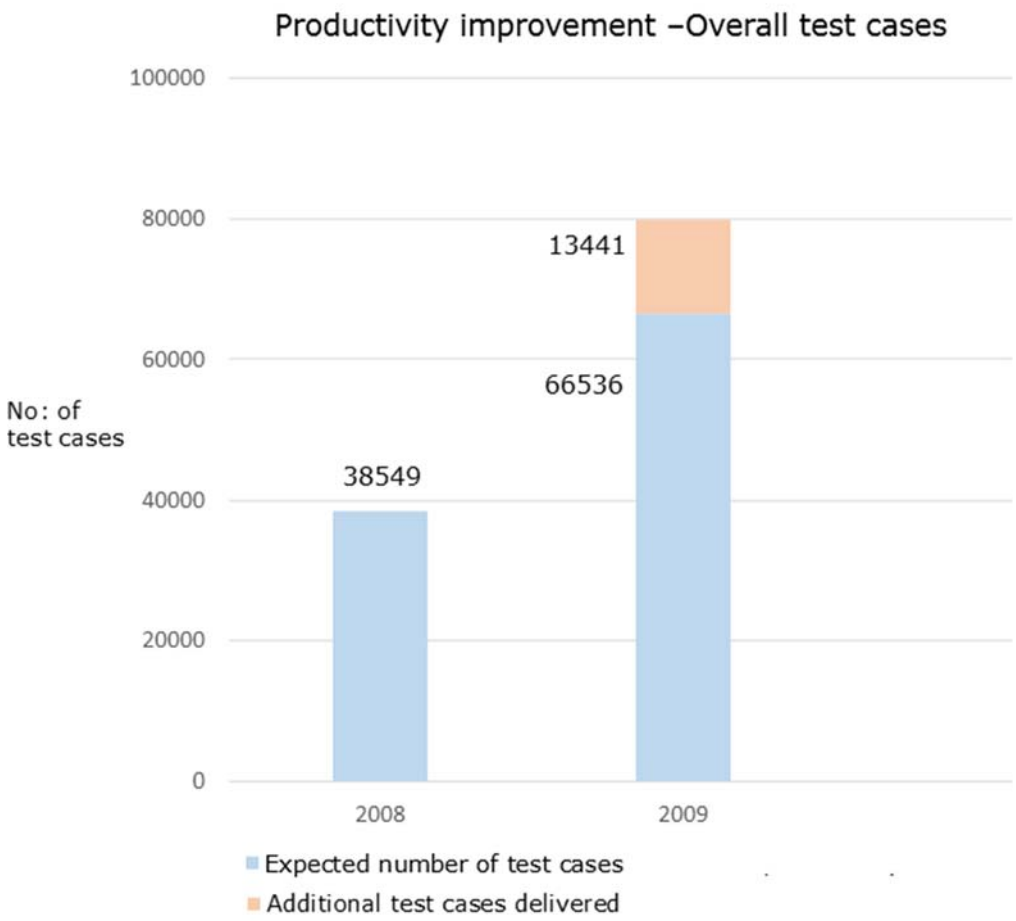
After one year of successful client engagement, the need to cut down on QC Costs and improve testing quality and productivity was sensed as an opportunity to build customer's trust.

Test automation in the below areas were conceptualized to decrease the effort and increase the efficiency

- Automation of regression test suite using the functional testing tool QTP 9.5
- Managing the testing activities using HP ALM standardization
- Use of tools like macros
- Expand the testing process like web service testing into automation

The below graphs display the quality control cost and productivity improvement that were achieved after implementing the strategies conceptualized.


% of Quality Control (QC) cost - Yearly
15
14
13
12
11
10
9
8
percentage
14%
12%
11%
2007
2008
2009
% of total project cost for QC activities



Also, after test automation, the company recorded:

- 100% defect removal efficiency
- 34% revenue growth in the project
- 26% increase in application coverage
- 14% decrease in effort

UFT Foundation

Why UFT

There are number of tools available in the market for performing functional testing like Selenium, RFT, VSTS CodedUI etc. Among these tools, UFT is preferred by the testing fraternity in the industry because of its rich set of features and support.

Among all the other tools, Selenium is a very popular tool in the open source test automation space as it supports various languages like Java, Python, etc. Let us try to look at the various differentiating features between UFT and Selenium

Feature	Selenium	UFT
Language Support	Java, C#, Ruby, Python, Perl, Javascript	VB Script
Windows Based Application Support	No	Yes
Browser Support	Internet Explorer, Firefox, Opera, Html Unit, Google chrome	Google Chrome, Internet Explorer, Firefox.
Environment Support	Windows, Linux, Solaris OS X, Others	Only Windows
Framework	Selenium + Eclipse + Maven / ANT + Jenkins / Hudson & its plugins / Cruise Control + TestNG + SVN	Easily Integrated with HP Quality Centre or HP ALM
Continuous Integration	Possible through Jenkins / Hudson / Cruise Control	Possible through Quality Center / ALM or Jenkins
Object Recognition/Storage	UI Maps and different object location strategy such as – XPath Element ID or attribute DOM	Inbuilt Object Repository (storing Element Id, multiple attributes) along with weightage that gives flexibility on deviation acceptance in control recognition
Image Based Tests	Possible but not easy	Easily possible
Reports	Integration with Jenkins can give good reporting & dashboard capability	Quality Center has in-built awesome dashboards
Software Cost	Zero	License & Annual maintenance fees
Coding Experience of Engineer	Should be very good along with technical capabilities of integrating different pieces of framework	Not Much
Script Creation Time	High	Less
Product Support	Open Source Community	Dedicate HP support along with support forums

Now that we have compared UFT with one of the other significant automation testing tools and seen how UFT is more capable in supporting more types of application and capability with testing with images. In the next page we will look at few of the salient features of UFT.

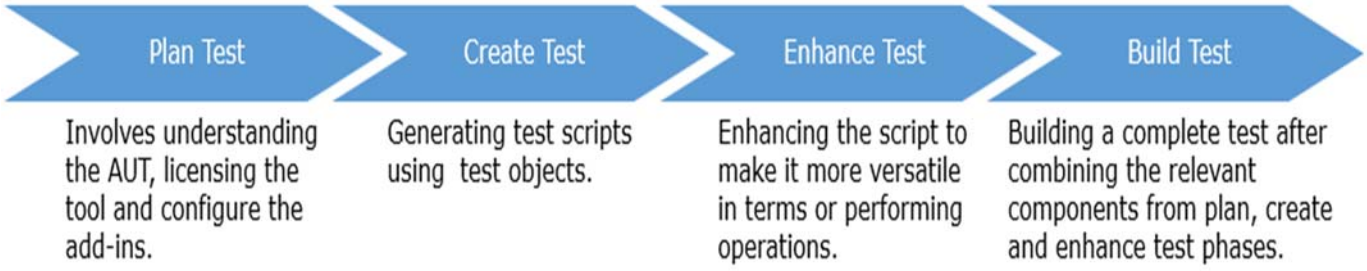
Important features of UFT

- Can automate both GUI and API based applications
- HP Unified Functional Testing (UFT) software is HPE's automated functional testing tool which is having all the features of various important tools such as Quick Test Professional, Win Runner and HP Service Test. It automates functional and regression test suites
- Supports testing of various applications like applications on Citrix server, services running on middle tier, mainframe applications, ERP applications etc. It automatically captures, verifies and replays user interactions
- Compatible with various technologies like .Net, Stingray 1, Terminal Emulator, Oracle, Siebel, PeopleSoft, Delphi, Flex, Power Builder, Windows Mobile, Web Services, WPF, SAP, Web, Java (Core and Advanced), VisualAge Smalltalk, Silverlight and mainframe terminal emulators
- Can generate reports after the test execution which can be used for analysis purposes by the testing team
- UFT has various features which gives it capability to test web services and service virtualization.
- It has the facility to execute scripts within specific interval of time using task scheduler/crone job.

Conclusion: After looking at the set of features provided by UFT we can easily conclude that it is one of the preferred automation testing tool in the market.

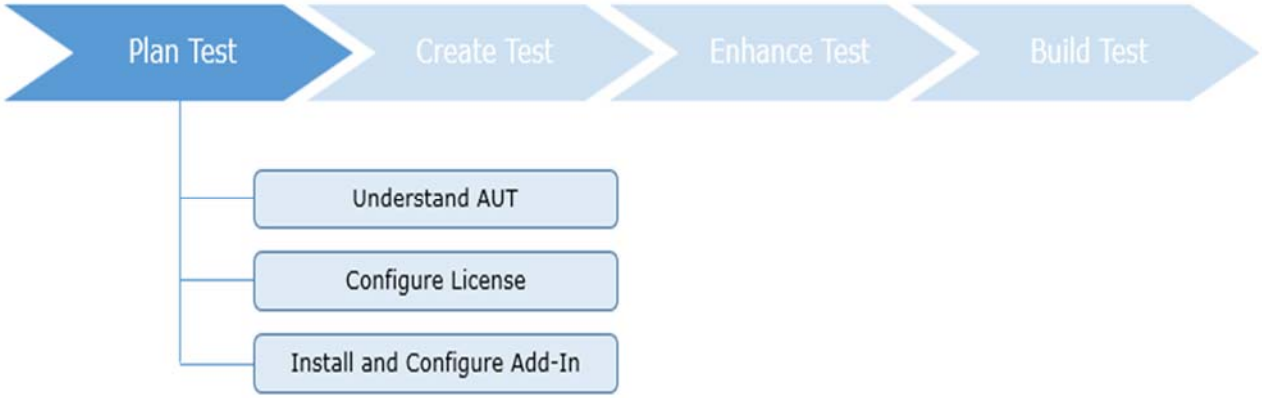
How to work with UFT

Below are the 4 major set of activities, with their brief set of operations, performed while using UFT:



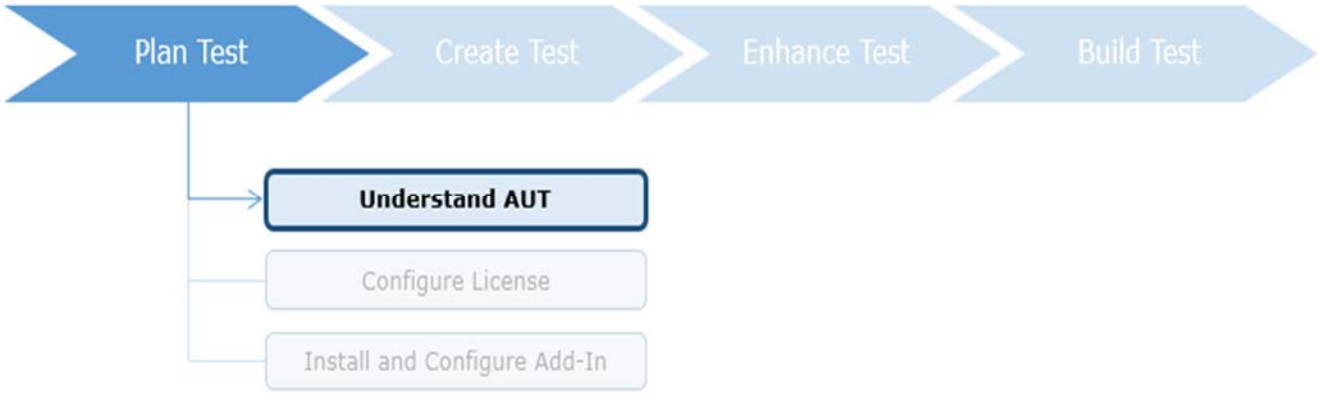
Plan Test

Planning test is one of the integral aspect followed while working with UFT. Below mentioned are the set of activities performed under planning



Let us see in detail what all activities are performed in each of the above steps.

Understand AUT



Understanding the AUT before testing plays a very critical role. This helps us to identify the various environmental components which would be required and the business critical scenarios which needs to be tested.

In this course we would be using two different applications.

- For demonstration purposes: Flight reservation application

In the subsequent sections we will be looking at the brief description of these applications.

Understand AUT - Flight reservation

For the demonstration purposes we will be using ‘Flight Reservation (FR)’ application which comes pre-installed with UFT.

Understanding the 'Flight reservation' application

Access path: C:\Program Files\HP\Unified Functional Testing\samples\flight\app\flight4a.exe

Login credentials:

Username	Password
Mercury	Mercury

Note: You can use any user name (characters>=4 in length) and password mercury (fixed)

You can explore the application and try performing below operations.

- Book the flight between two cities
- Update the order placed
- Fax the order placed
- Delete the booking

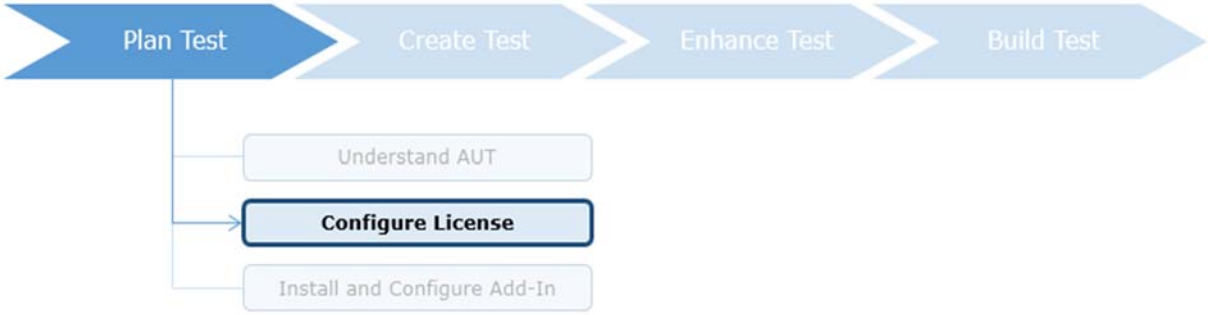
Environment of AUT

Flight Reservation is a window based application. Having it's database created in Microsoft Access on the same machine where the application is installed.

Various business scenarios in the application to be tested

- Login - Logout
- Login – Booking - Logout
- Login – Open Order - Update Order – Logout
- Login – Delete Order - Logout

Configure License - Types of UFT License



Before learning how to configure a license, let us understand what are the various types of licenses we have in UFT.

UFT is a proprietary tool, whose licenses are sold by Hewlett Packard Enterprise (HPE). There are two types of licenses that can be purchased for UFT:

- Seat License
- Concurrent License- Further divided into below sub types:
    - Site Concurrent
    - Area Concurrent
    - Global Concurrent

License Type	Description
Seat License	• Purchased as a single unit • The license key is valid only on one specific computer where UFT is installed
Concurrent License	• Purchased as a set of fixed number of units • Network based license authentication • Multiple machines in the network can share the same licenses, however the number of concurrent users are restricted to number of units purchased

Types of concurrent license

Concurrent License Type	Description
Site Concurrent	Usage restricted to single LAN only
Area Concurrent	Usage restricted to single country only
Global Concurrent	No restriction based on location. Can be used anywhere in the world

Demo 1 : Configure License

Highlights:

- Configure UFT License
- Concurrent Licenses

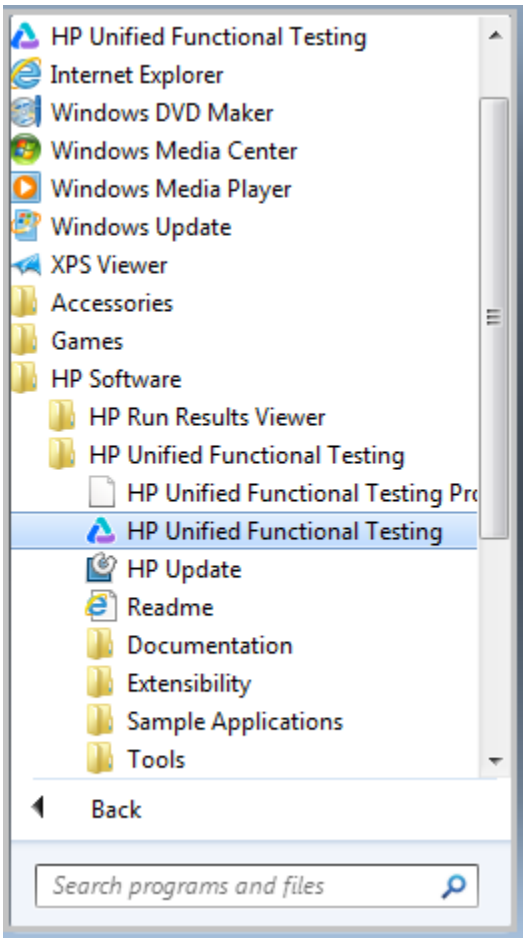
Demo Steps:

UFT License - Configuration

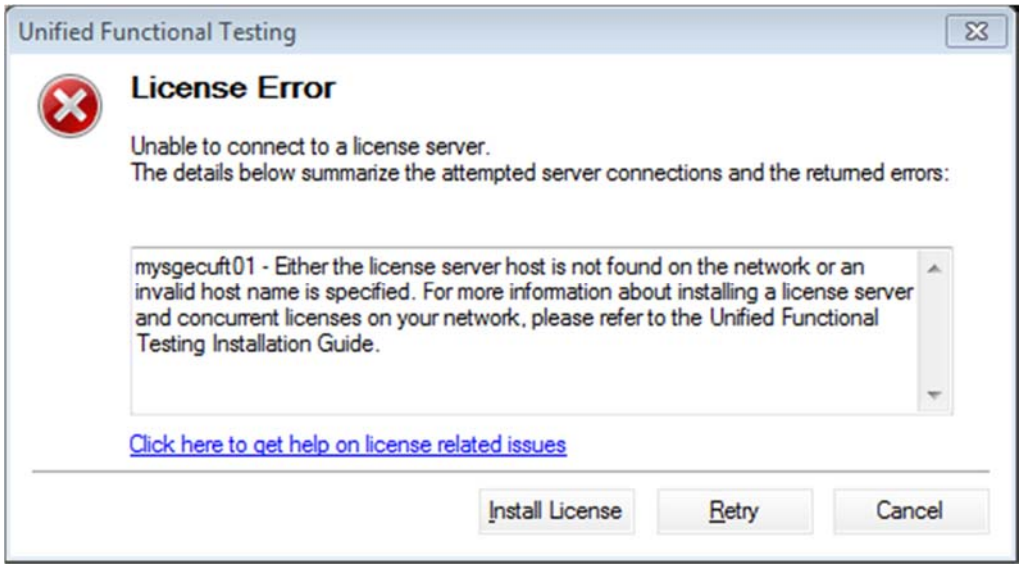
For practical purposes we would be using concurrent license.

Below mentioned steps are performed to configure the license in UFT:

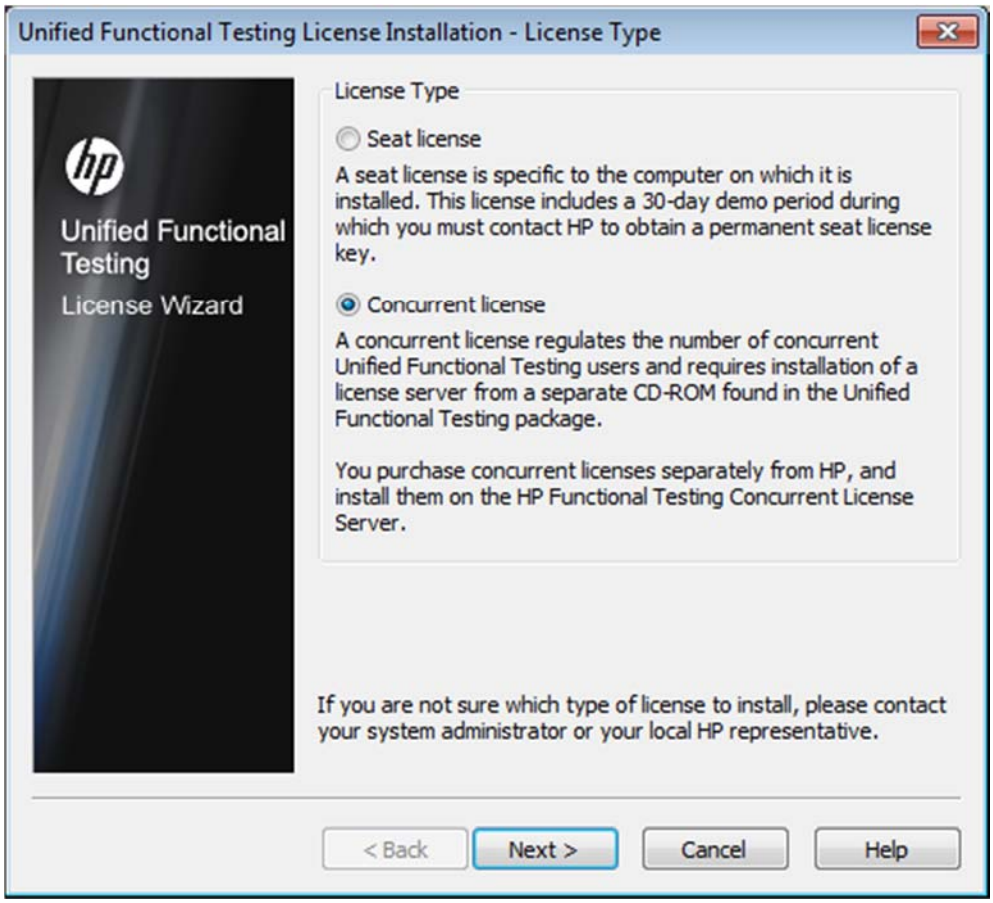
Step 1: Open HP UFT through Start >All Programs>HP Software>HP Unified Functional Testing



Step 2: If the ‘License Error’ pop-up window appear, then click on ‘Install License’ button.

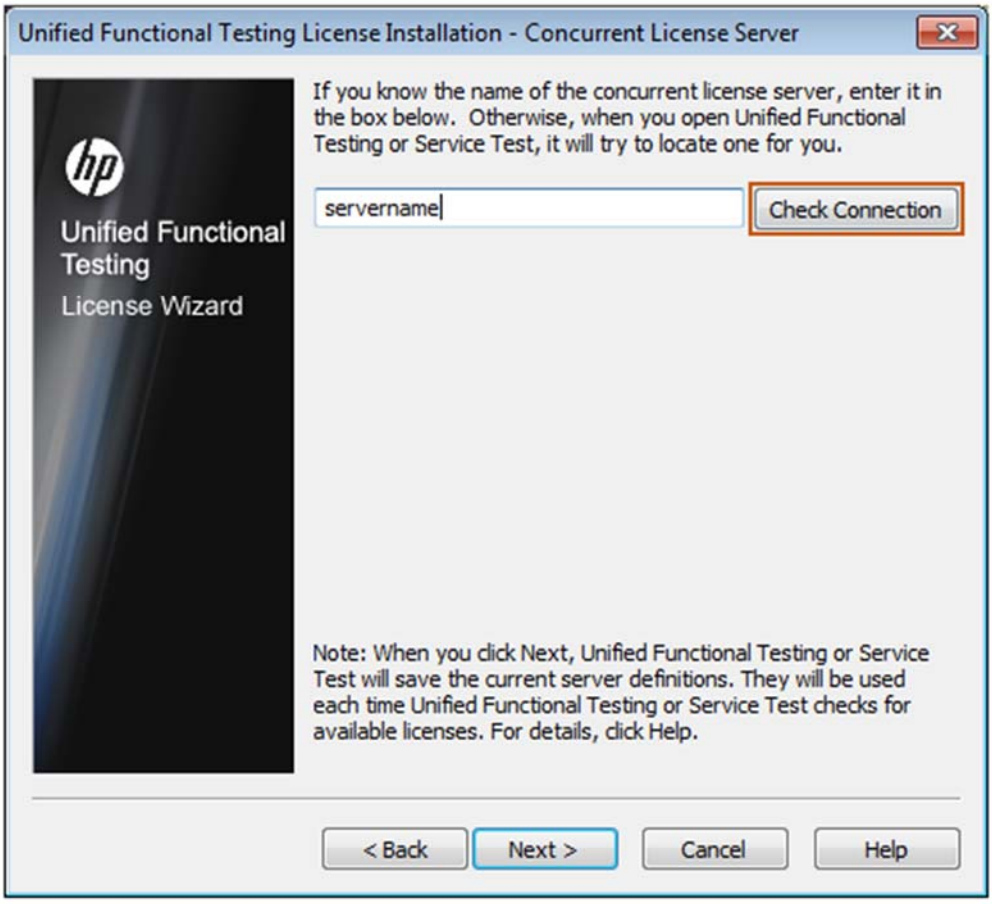


Step 3: Click on ‘Concurrent License’ as the license type and click on ‘Next’



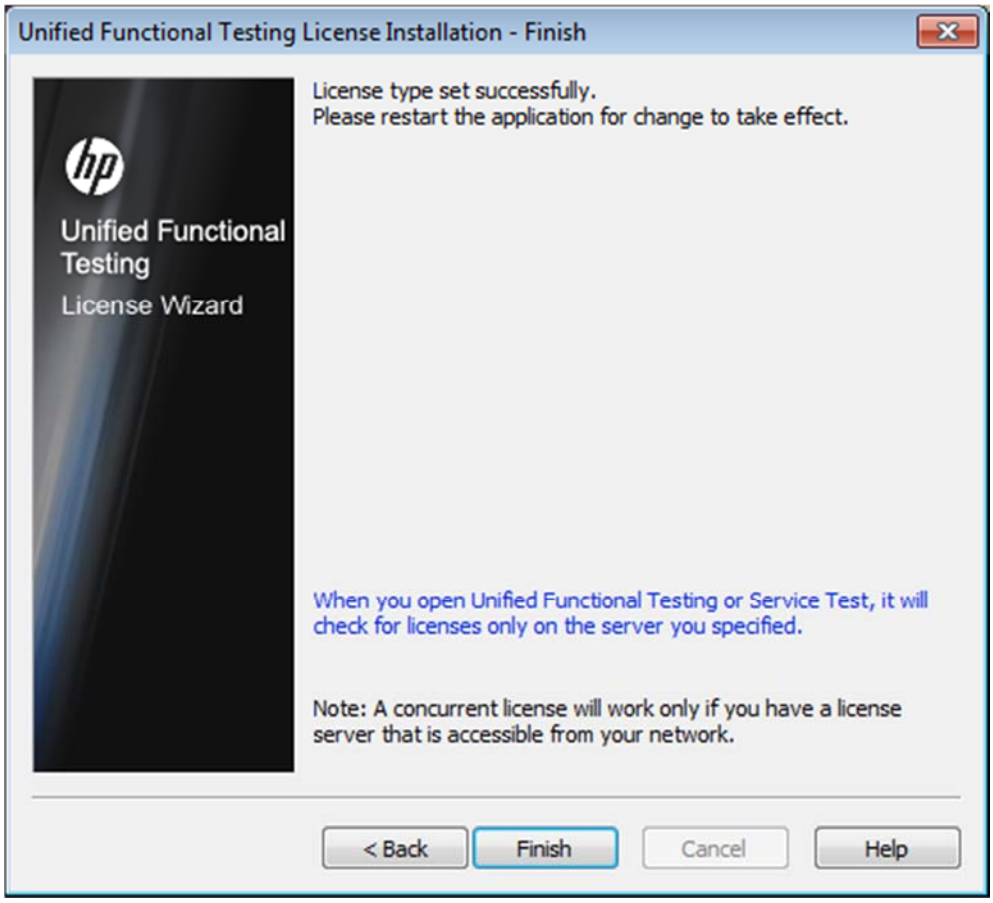
Step 4: In the Concurrent License Server window enter the license server name, Click on ‘Check Connection’.

Note: - License server name will be provided by the respective lab anchors.

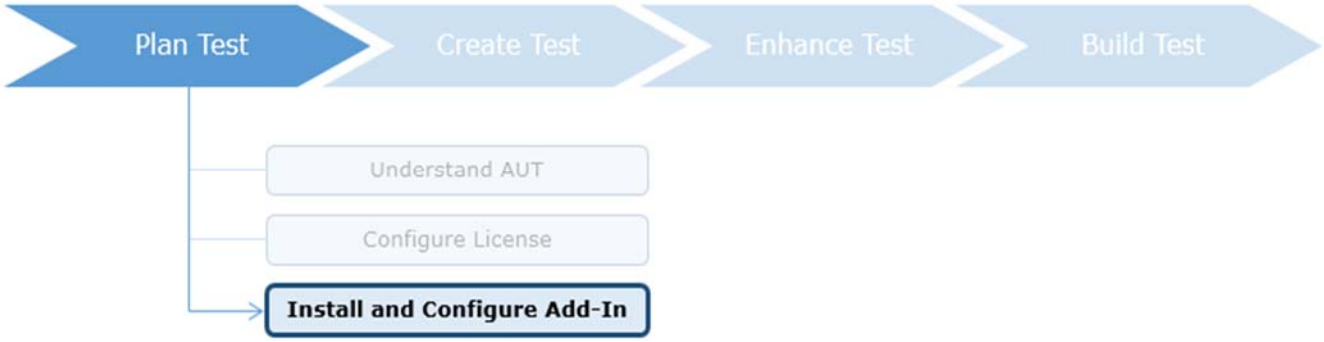


Note:- You will get a success message once the connection check pass. Click on ‘Next’.

Step 5: In the ‘License Installation - Finish’ window click on the ‘Finish’ button



Install and Configure Add-In



Add-In is a utility which will help UFT understand the objects present in the AUT created using multiple technologies.

To automate any application in UFT, the Add-in which supports the particular application technology needs to be loaded in the Add-in Manager. Below are the steps

- Install the required Add-in

- Configure the Add-in

UFT is supplied with the following add-ins by HPE.

Windows Environment	Web Environment	Windows and Web
• Standard Windows • ActiveX • Delphi • PowerBuilder • Qt • Stingray • Terminal Emulator • Visual Basic • VisualAge Smalltalk	• Web • Oracle • PeopleSoft • Siebel	• SAP • Java • Flex • .Net

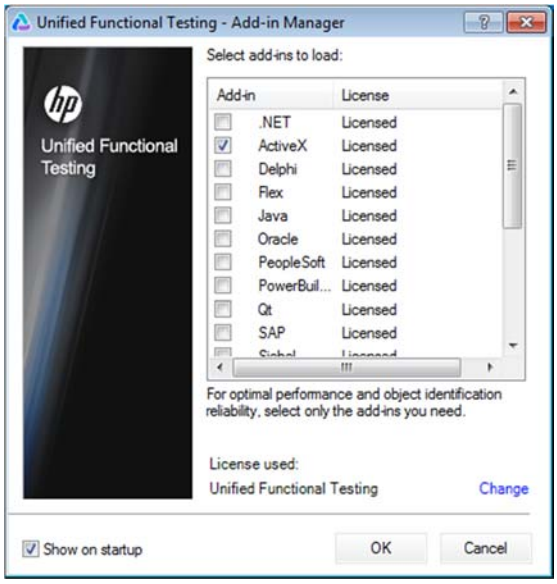
Install Add-in

We can install Add-ins while installing UFT for the first time or modify the installation, by using re-install option.

Configure Add-in

UFT Add-in Manager facility helps us to select the technology / environment, from variety of environments options available, suitable for the AUT (Application under Test).

- Add only the required add-ins as it affects the performance of UFT, both in terms of memory and reorganization of AUT objects.
- Only those add-ins will be available on launch which we have selected during installation of UFT
- We can check and modify the add-in's for opened test "File-Settings-Properties"



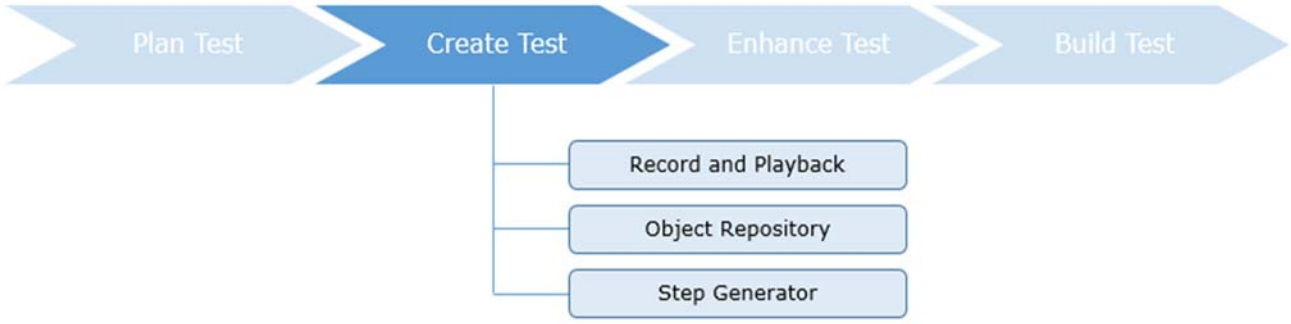
Plan Test – Summary

In this phase, we learnt about the below mentioned points:

- Analyzing the application under test
    - Understand the environment of the AUT
    - Gather business critical scenarios of AUT
- Configure of licenses in UFT
    - Types of licenses
    - Configure concurrent license
- Install and configure Add-in
    - What are Add-in and their types
    - Install Add-ins in the system
    - Configure the required Add-in

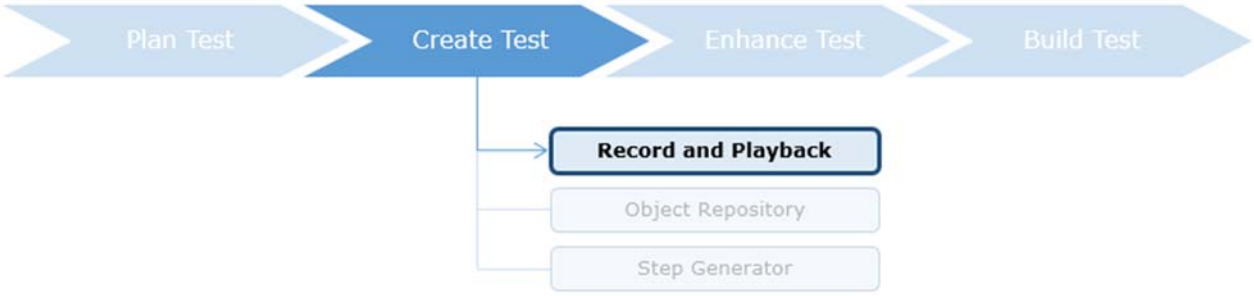
Create Test

After understanding the application and the environmental components required, next step is to create the test script in UFT. There are various ways of creating a test script in UFT. In this course we will learn about below three methods of creating the test scripts.



Before actually starting with the test script creation, let us look at the various GUI components of UFT, which will help us to identify the relevant components to be used while working on the tool.

Record and Playback



Recording is one of the best options that UFT provides for creating the test scripts easily.

Why Record and Playback?

- Using record option UFT users can build automation scripts automatically without having any knowledge of scripting
- Starting from the moment when Record button is clicked till the moment Stop button is clicked UFT records all the user actions as you navigate through your website or application and generates the script accordingly

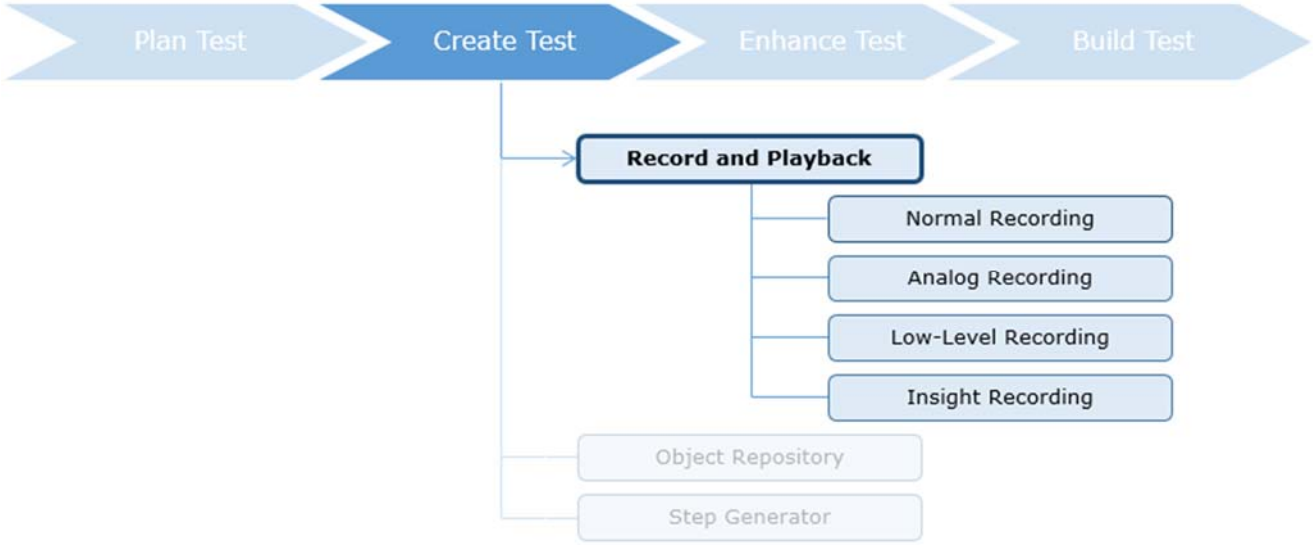
Note: Recording in real time situation is majorly used on the applications for verifying if UFT supports the technology in which the application is built by checking if the objects present on the AUT are being identified by UFT or not.

What is Recording?

- Recording is one of the functionalities available in UFT which will help the end user to capture the sequence of events performed on the application and generate a script for those user actions automatically
- User actions can be seen in the script as **steps**
- A step in the script is generated by anything that user does on the application, that changes the content of a page or behavior of an object in your application. For example, clicking a link or typing data in an edit box

In the next few pages we will look at the various modes of recording supported by UFT and how they work.

Recording Modes – Types



Below mentioned are the various modes of recording supported by UFT. They differ in the type of events that are captured while recording.

Normal Recording Mode

- Records user action performed on the application.
- It is the default recording mode in UFT.

Analog Recording Mode

- It records mouse movements and keyboard inputs.

Low-level Recording Mode

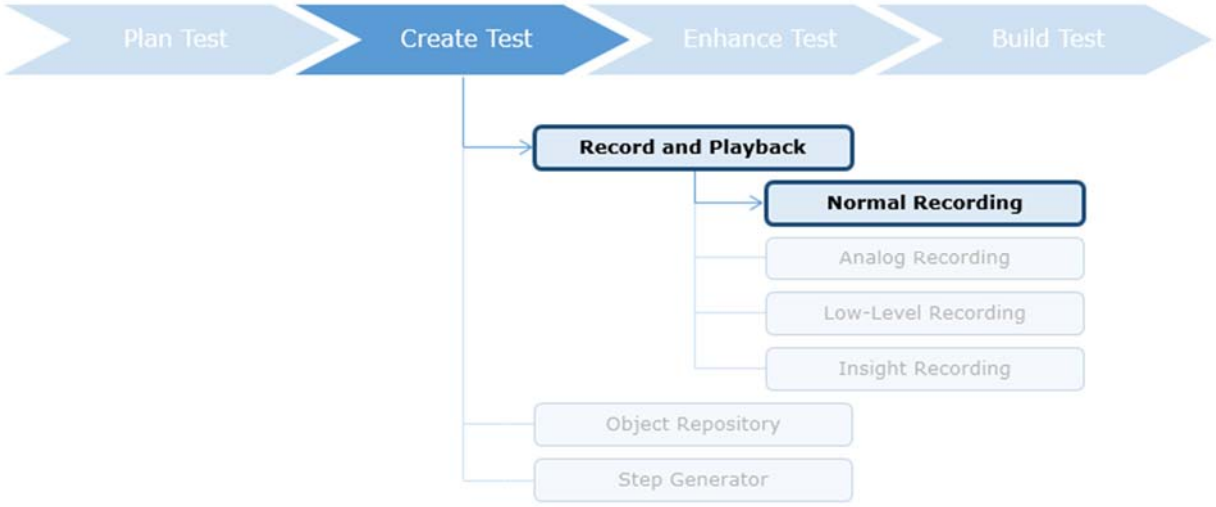
- Records the x-y coordinates of the GUI objects whenever a mouse click performed on them.

Insight Recording Mode

- It records any object displayed on AUT along with the screenshot of the object irrespective of what type of object it is.

We will be learning about above mentioned recording modes in following pages.

Recording Modes - Normal Recording

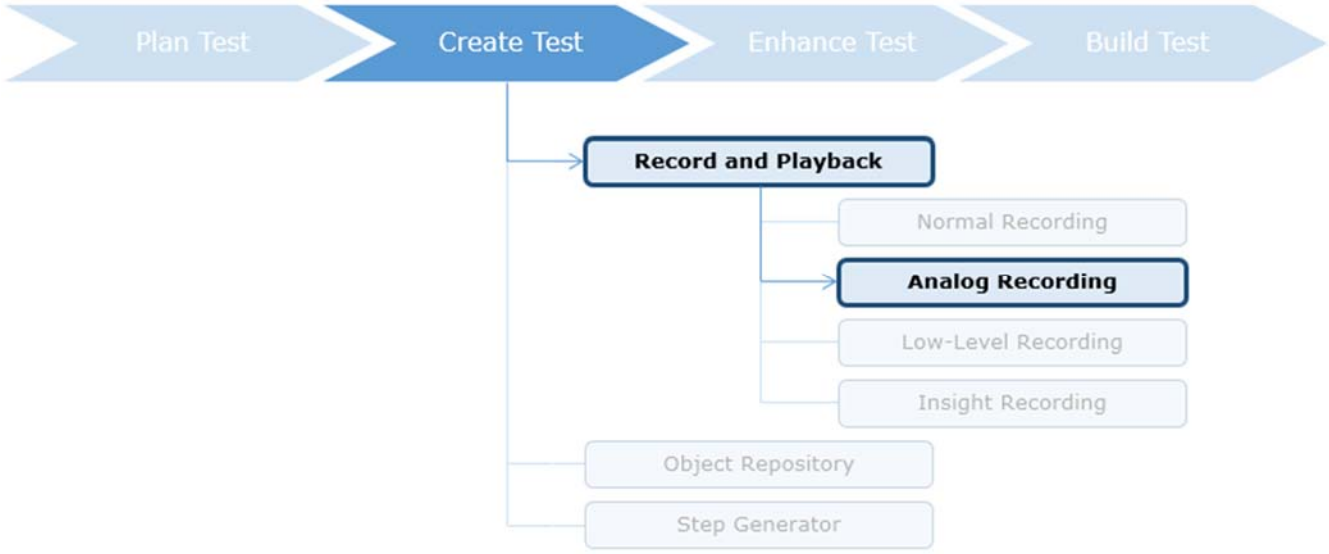


Normal Recording

- Normal Recording mode is also called as Context Sensitive Mode /Default recording/ Standard recording
- This is the default recording mode that records the objects and the operations performed on the application under test.
- In this mode while recording the application UFT generates VB script statements in TestPane and at the same time it adds corresponding objects to Object Repository

Lets see how to work with the normal recording with the help of a video in next section.

Recording Modes - Analog Recording



Analog Recording

Let us understand it's significance by considering a business critical situation.

The automation team was trying to automate the digital signature field with the help of Normal/Default recording mode. After the recording is done, they tried playing back the code that got generated but the script was unable to perform the signature activity on the signature field on the application.

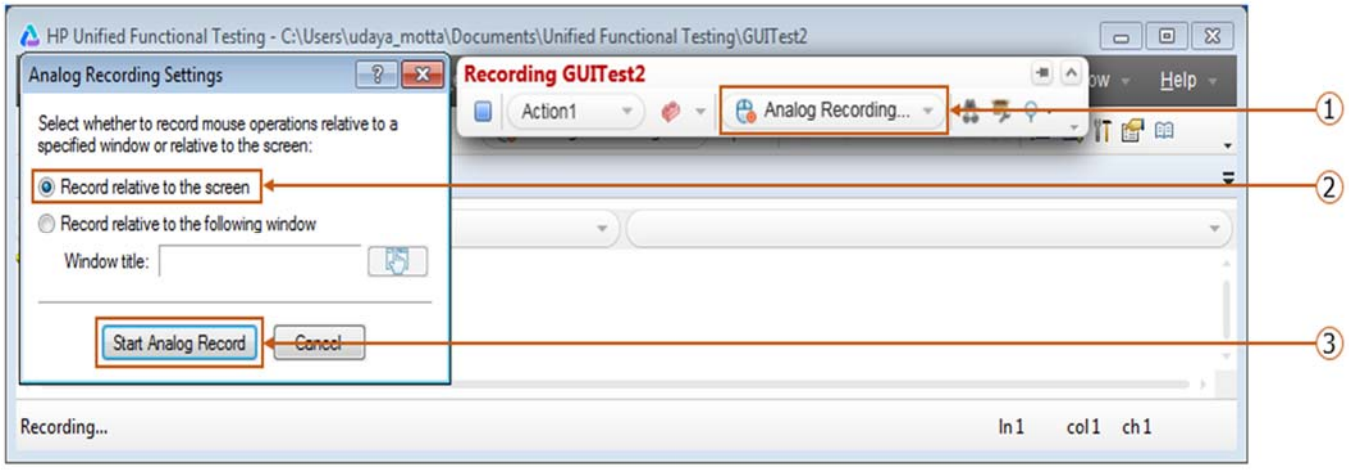
- The exception was due to the incompetence of normal/default recording mode to capture the continuous mouse movements.
- So in order to automate the signature type of activities where continuous mouse movements are needed analog recording is the best fit and can be implemented.

Analog recording can be done using either of the following options:-

- Relative to screen - Records with respect to desktop screen position
- Relative to window - Record with respect to a particular window

Relative to screen

Below are the steps to activate "Analog recording - Relative to screen".



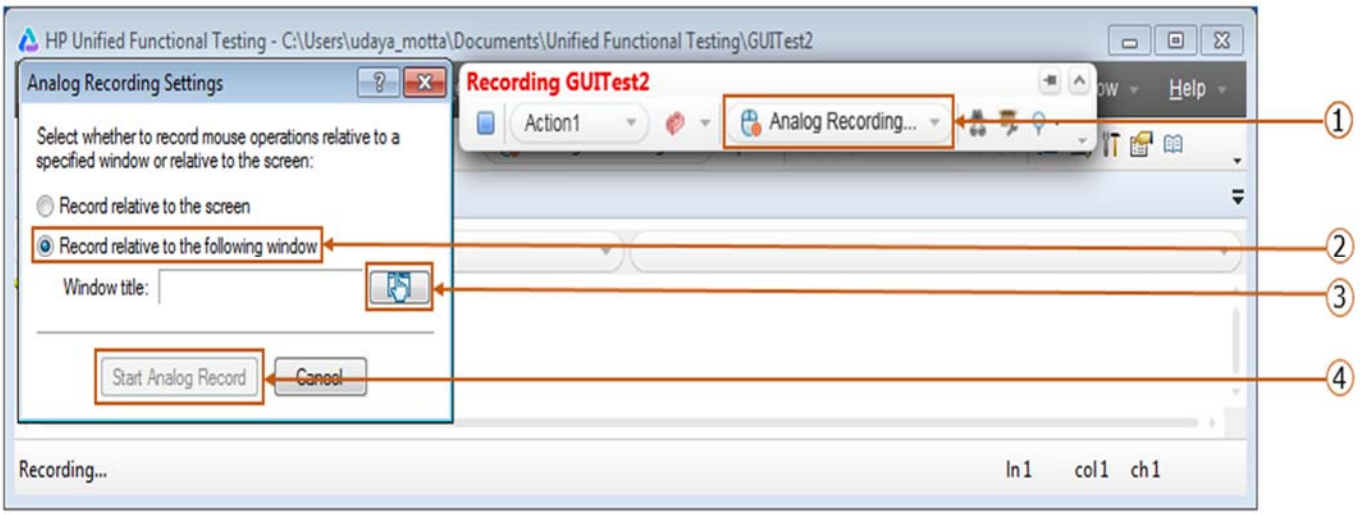
Step 1 - Enable Analog recording, whenever you start recording the script.

Step 2 - Select "Record relative to screen" from the "Analog Record Settings" window that pops up.

Step 3 - Click "Start Analog Record" button.

Relative to window

Below are the steps to activate "Analog recording - Relative to window".



Step 1 - Enable Analog recording, whenever you start recording the script.

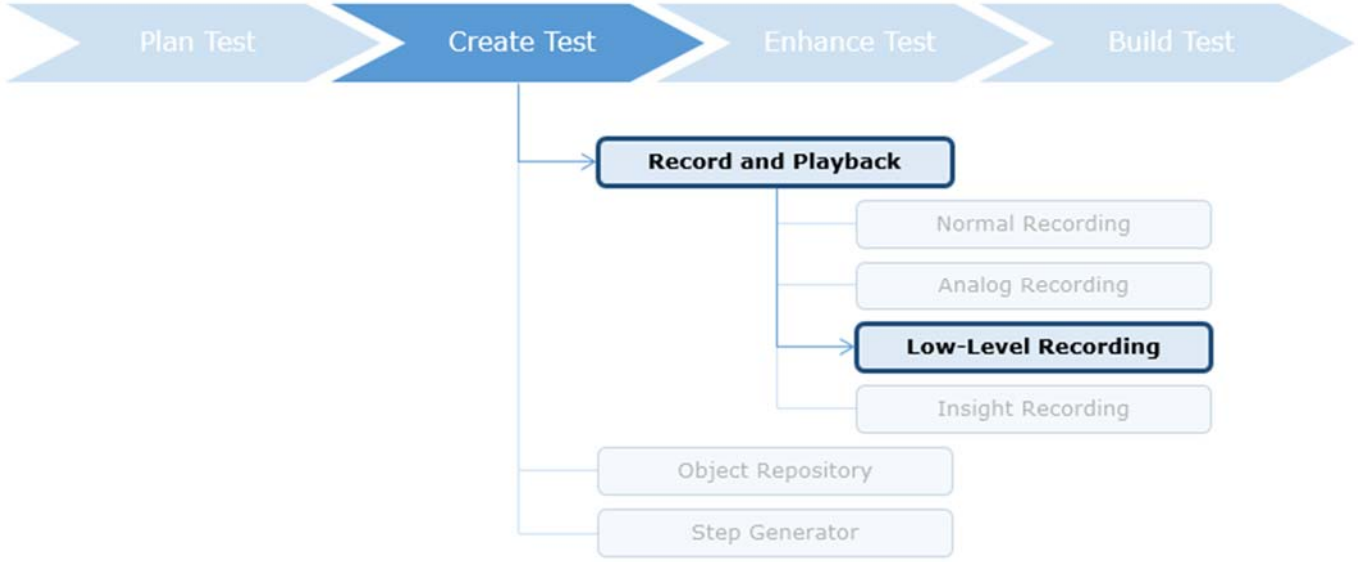
Step 2 - Select "Record relative to screen" from the "Analog Record Settings" window that pops up.

Step 3 - Click the pointing finger to select the window w.r.t which the analog recording has to be

done.

Step 4 - Click "Start Analog Record" button.

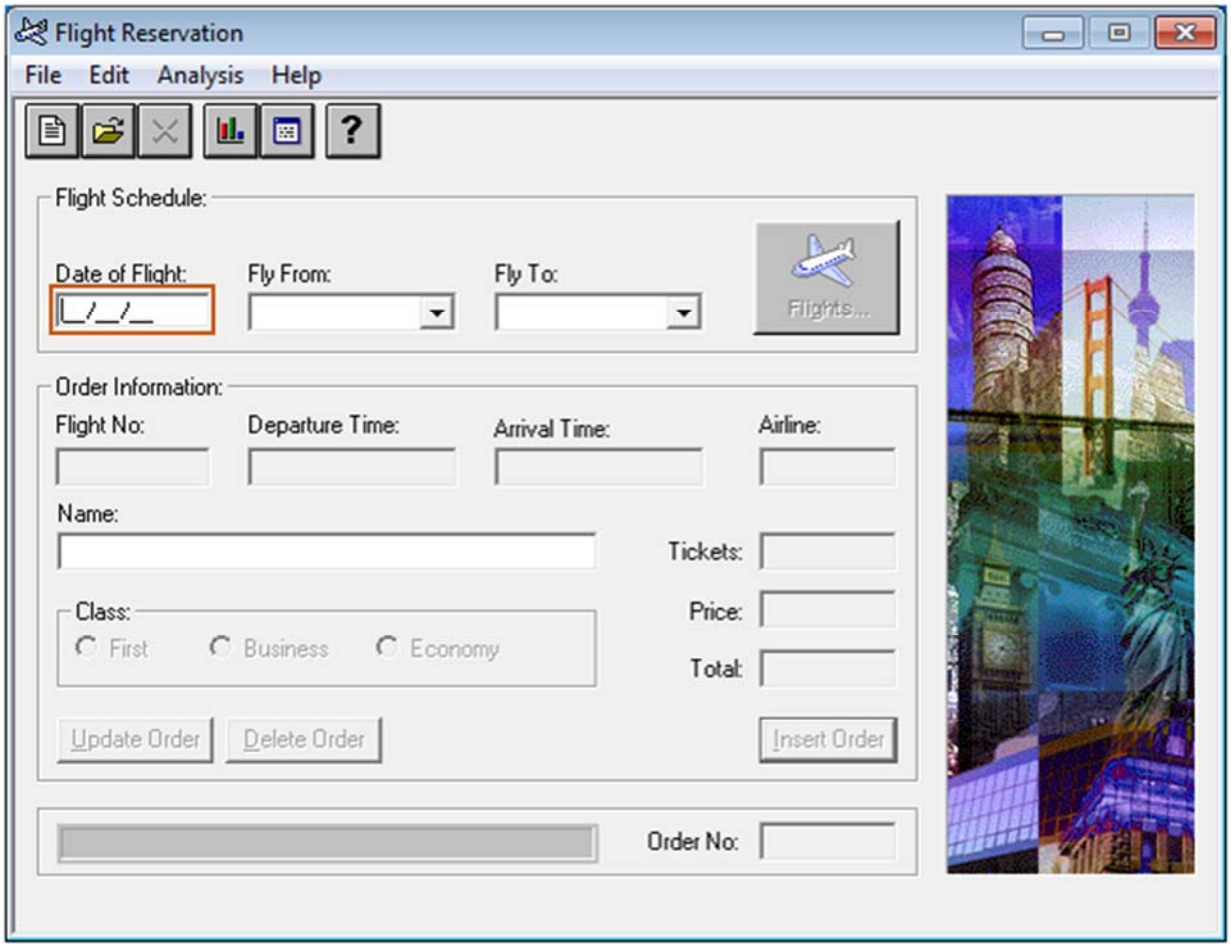
Recording Modes - Low-Level Recording



When the exact co-ordinates of the object are essential then **Low-Level Recording** will be used.

Consider the below image from flight reservation application. Here we need to book an order. We will record booking a ticket in Low Level and normal recording mode to observe how they differ.

- For booking the order, date has to be entered in the “Date of Flight” object and origin in "Fly From" object. We will record this step using Low Level recording mode.
- Then we will record rest of the activities in "Normal" mode.

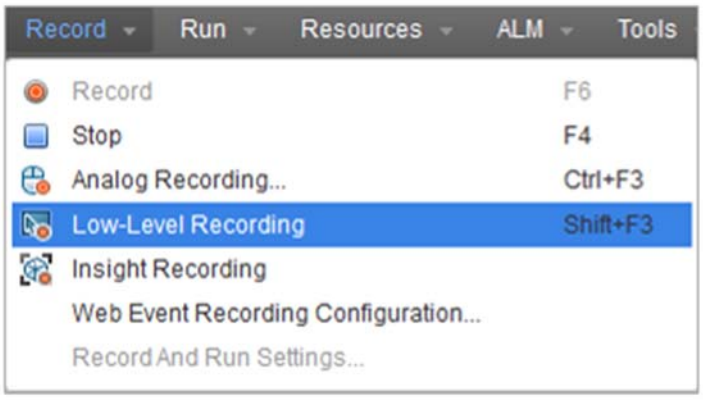


Steps to perform Low-Level Recording

Step 1: Create a new test ->File->New->Test

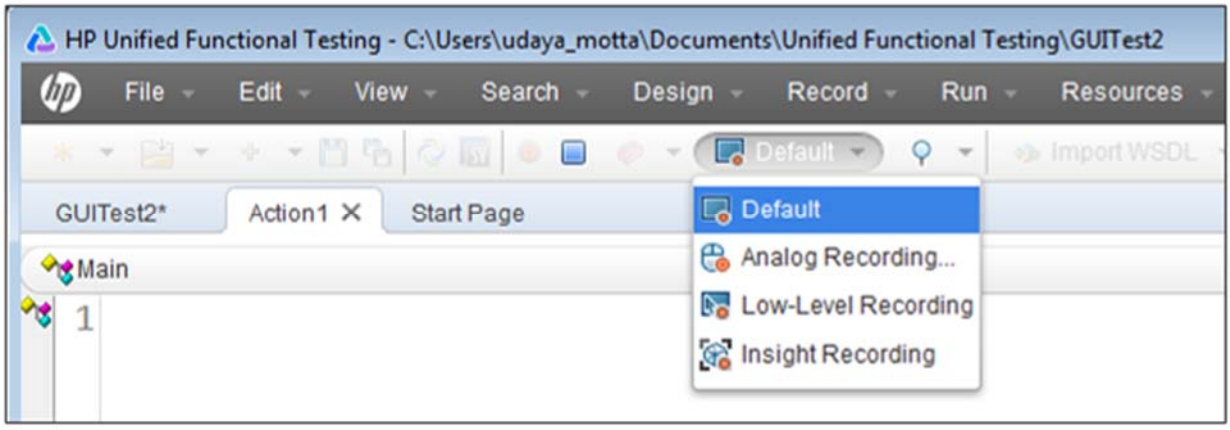
Step 2: Click Record menu item.

Step 3: Click on the Low-Level Recording, from the “Record” menu. This will start the recording in Low-level recording mode.



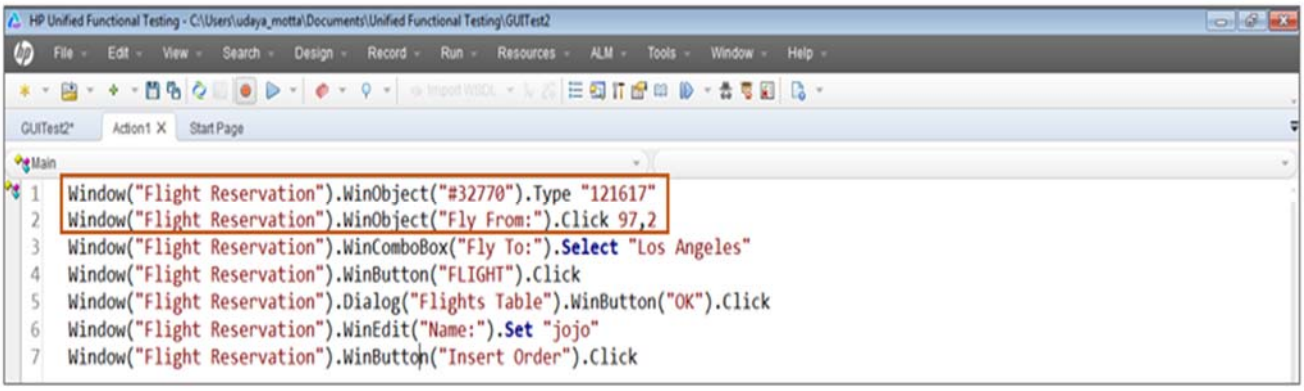
Step 4: Enter date in the 'Date of Flight' object and origin in "Fly From" object

Step 5: Switch back to default recording mode as shown below in UFT



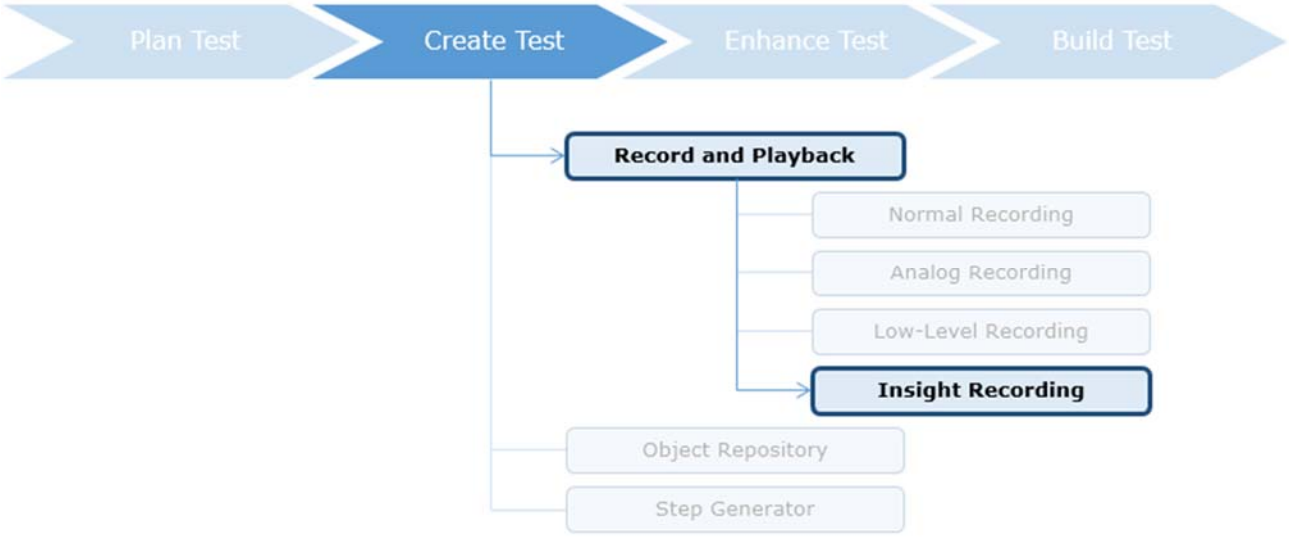
Step 6: Continue the booking process by giving the remaining data and click the insert order button.

Script generated:



We can check the difference between the script present in the highlighted area and rest of the script. Script in the highlighted area has been recorded using ‘Low-Level Recording’ mode and rest using ‘Normal/Default Recording’ mode. Child objects present in the window are captured as ‘WinObject’ in the low-level recording mode and x and y co-ordinates are using for the click method

Recording Modes - Insight Recording



Insight Recording

Insight recording mode can be useful to test objects from an environment that UFT does not support or even from a remote computer

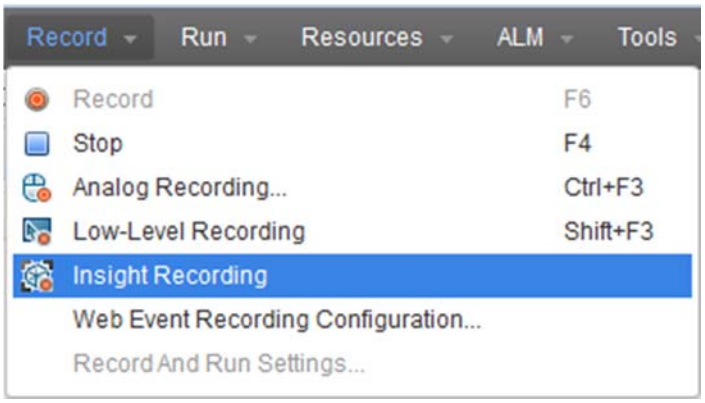
running a non-Windows operating system. Insight mode can be used on unsupported technologies such as Flash.

In Insight recording mode UFT recognizes objects based on their appearance and not on their native properties. When UFT runs the test it recognizes the objects in the application by matching them to the images saved with each of the Insight test objects.

Steps to perform Insight recording

Step 1: Open a new test and click the record button

Step 2: Click on the Insight Recording menu after starting the recording

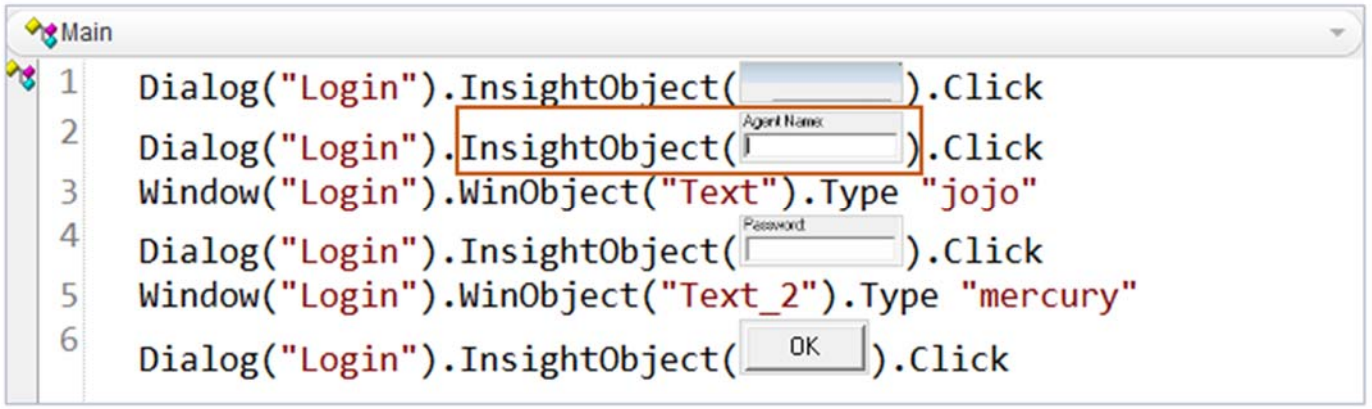


Step 3: Perform the log in operation in Flight Reservation application

Step 4: Stop the recording

Script generated:

Below is the script that gets generated after doing insight recording on the login page of flight reservation.



In the highlighted image, you can see that during insight recording objects are captured as **'InsightObject'**. They are basically the image forms of the objects captured during recording

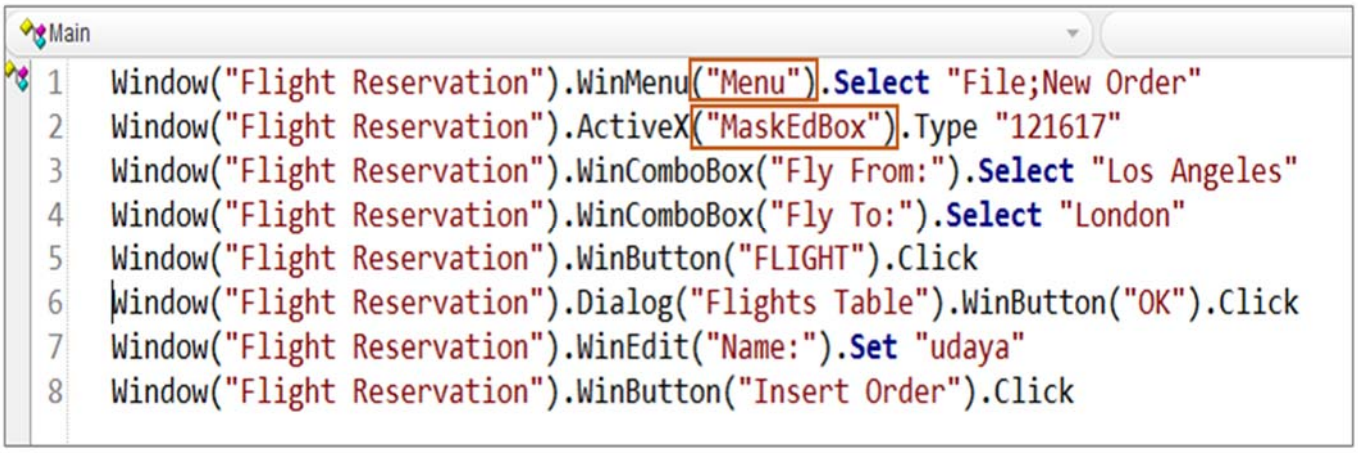
Analyzing Record and Playback - Test Object Model

When we click record, UFT generates the statements (line of a code) for every standard operation performed on application under test (AUT) which we have observed in the earlier recording examples.

Test object

A **test object** is an object that UFT creates in the Object repository i.e. it is a placeholder to store test objects, to represent the actual object in your application.

UFT use these test objects while executing the scripts. Some of them have been marked in the image below.



UFT Test Object Model(TOM)

A collection of object types or classes which represents the different objects in the application. For Example a button object or edit box object in an application are the objects of AUT and represent *Button* class or *Edit box* class. These test object class has a several properties to uniquely identify the objects of the particular class and methods to perform specific actions.

There are two types of objects in **TOM**,

- Test object

- Run-time object.

Ideally for every **"Run-Time object"** type, there should be corresponding UFT Test object or else – we cannot automate the respective **"Run-Time object"**.

GUI Object (WindowApplication)	UFT Test Object
DialogBox	Dialog
TextBox	WinEdit
Button	WinButton
Window	Window
Menu Bar	Menu
Checkbox	WinCheckBox
Dropdown List	WinComboBox
Image	Static
Radio Button	WinRadioButton

GUI Object (Web App)	UFT Test Object
Browser	Browser
WebPage	Page
TextBox	WebEdit
Button	WebButton
Link	Link
Browser	Browser
WebPage	Page

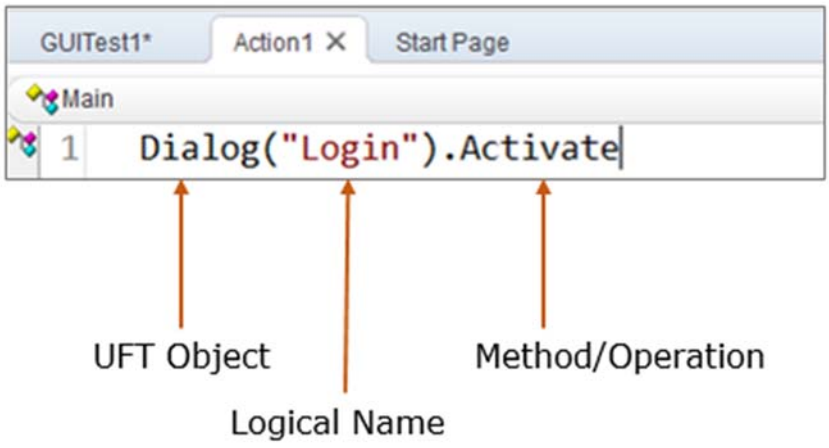
UFT views all the objects present on AUT as ‘Run-Time Object’. They are the objects with their own properties and methods defined by the technology in which they have been designed:

- **Properties** – What are the object’s characteristics? Example: Class of an object, Text present in an object
- **Methods** – What are the operations that can be performed on/by each object

While capturing the objects from the AUT, UFT captures the following information about the objects.

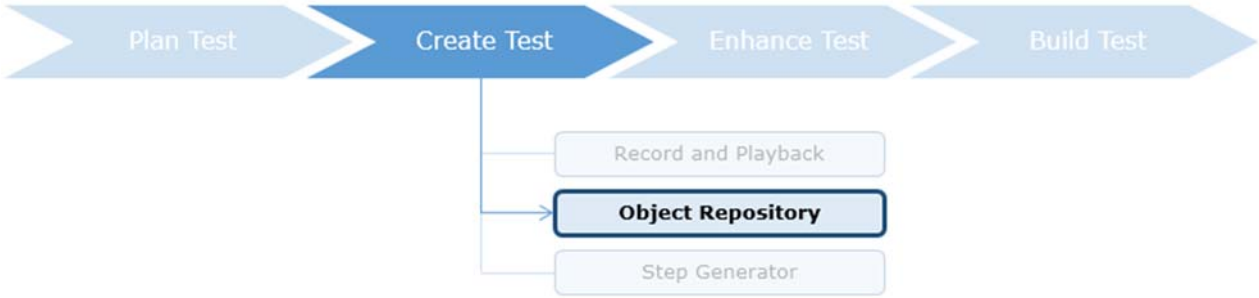
- Class of the object
- Hierarchy (belongs to which parent object in the AUT)
- Description Properties (which properties can be used to identify the object uniquely)

UFT calls these objects as **‘Test object’** and assigns a logical name to it for easy identification.



These captured objects are stored at a place called **object repository**. In the subsequent section we will see what object repository is and how it is created.

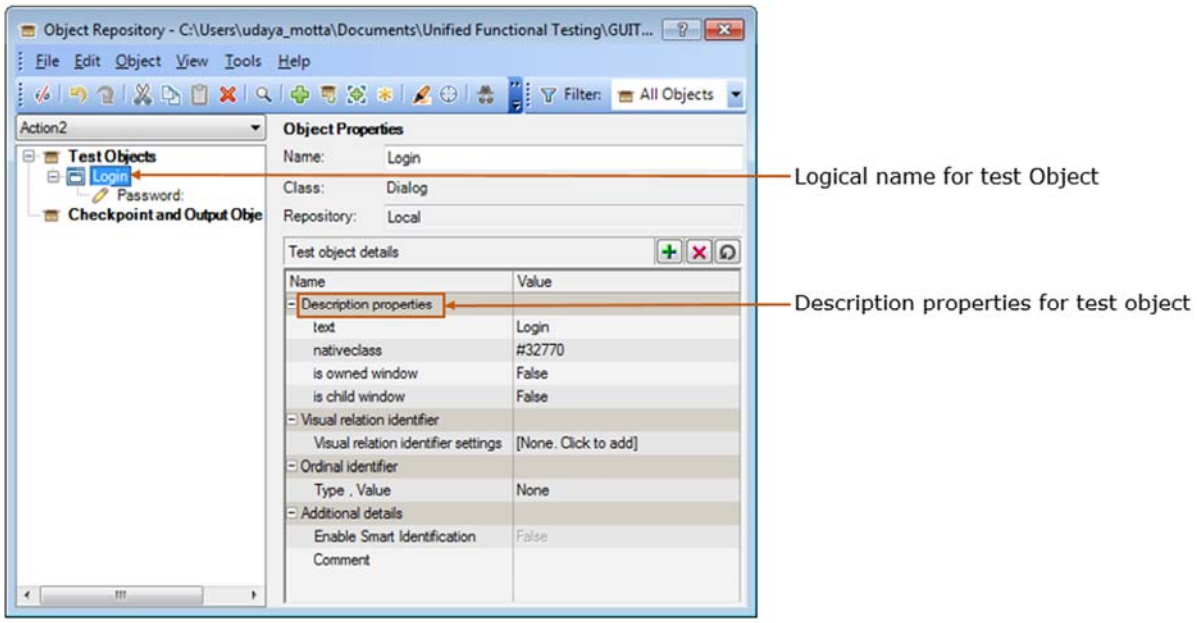
Object Repository



Whenever UFT captures test objects, it needs a place to keep those test objects so that they can be used later for preparing and executing test scripts.

Object repository is a placeholder/file which stores test object's information i.e. the name of the object, the class to which it belongs, and the properties of the UI object to identify it uniquely.

It acts as an interface between the test script and AUT and helps script properly identify the objects on AUT during test execution. It can be accessed from **"Resources->Object Repository"**



In the object repository:

- Description properties of the object are stored under respective logical names. These are the property values used by UFT to recognize the object.

  Example: Class value of an object, Window Id etc

- User can add, rename, delete the objects

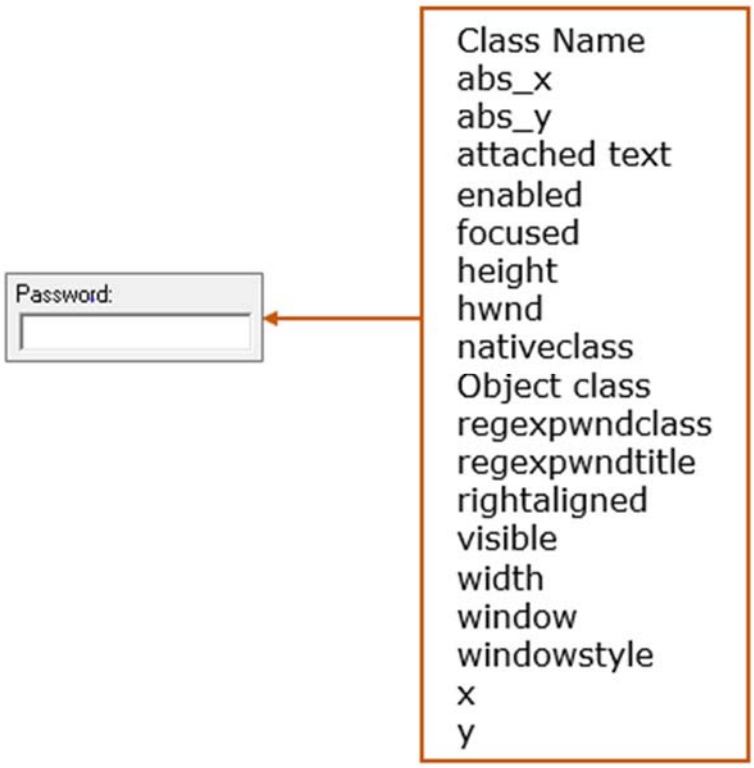
Now that we have come to know about what is test object and where it is stored let's see how description properties are configured for test object in object repository.

Object Identification

The next question that should come to our mind is how does UFT identifies the objects present in AUT and captures them. For making this happen UFT uses something called as as Description propertied which are configured with the help of **'Object Identification'**. In the following section we will be looking at this part of UFT.

An object could have larger number of properties associated with it.

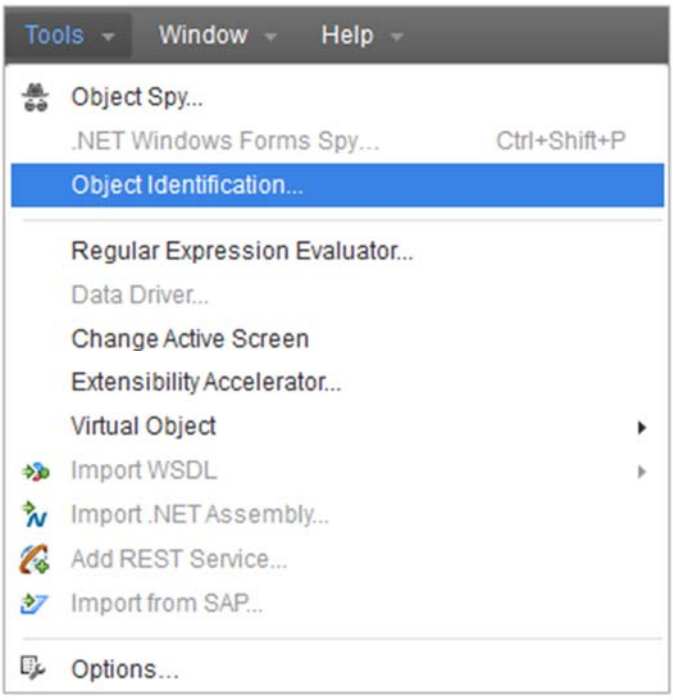
Example: Password edit box can have the following properties associated with it. These properties are provided to UFT by the Add-In available in UFT, which is for the respective technology in which that object has been created in the AUT.



Storing all these properties in Object Repository will not only consume space, but also reduce the performance of the UFT .To solve this problem UFT stores the minimum set of property values based on **object identification** settings available in 'Tools' menu.

Below image shows how to access 'Object Identification'.

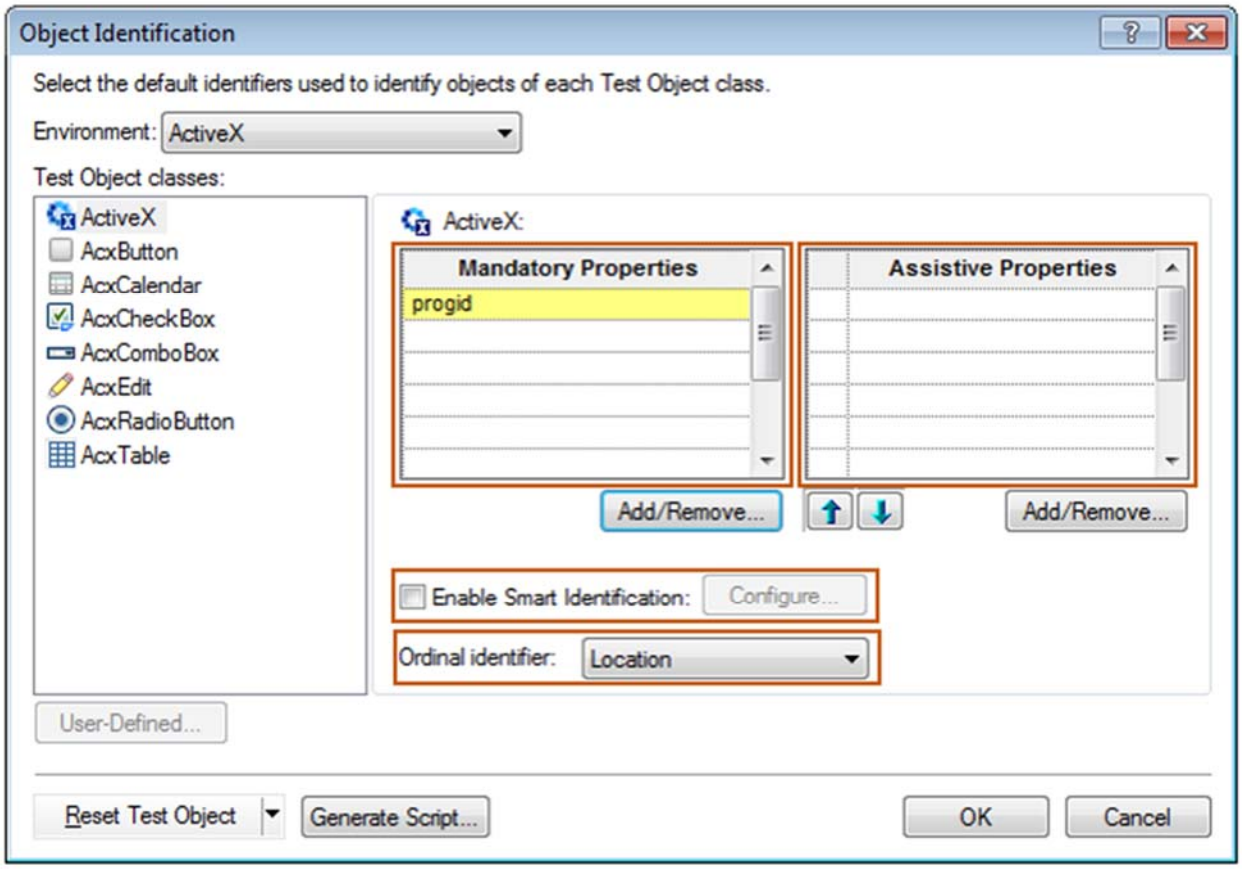
Select tools->Object Identification



There are 4 major identification category which UFT uses for object identification.

- Mandatory Properties
- Assistive Properties
- Ordinal Identifier
- Smart Identification

Below window shows where the mentioned categories can be set.



Below mentioned are the steps which UFT follows to identify the object using these 4 categories. If it locates objects in particular step, it need not go to the next step.

Step 1: During recording UFT will read all the **mandatory property** values and then it checks whether it is sufficient or not, if sufficient it will store it in object repository. If not, it goes to **assistive properties**

Note: Sufficient in the sense – none of the other objects should have same property values in same screen.

Step 2: In assistive properties, UFT will read 1st assistive property value from the application & check whether all the mandatory properties plus 1st assistive property values are sufficient to uniquely identifying the object in the application. If sufficient, it will store them in object repository. If not, it goes to next assistive property.

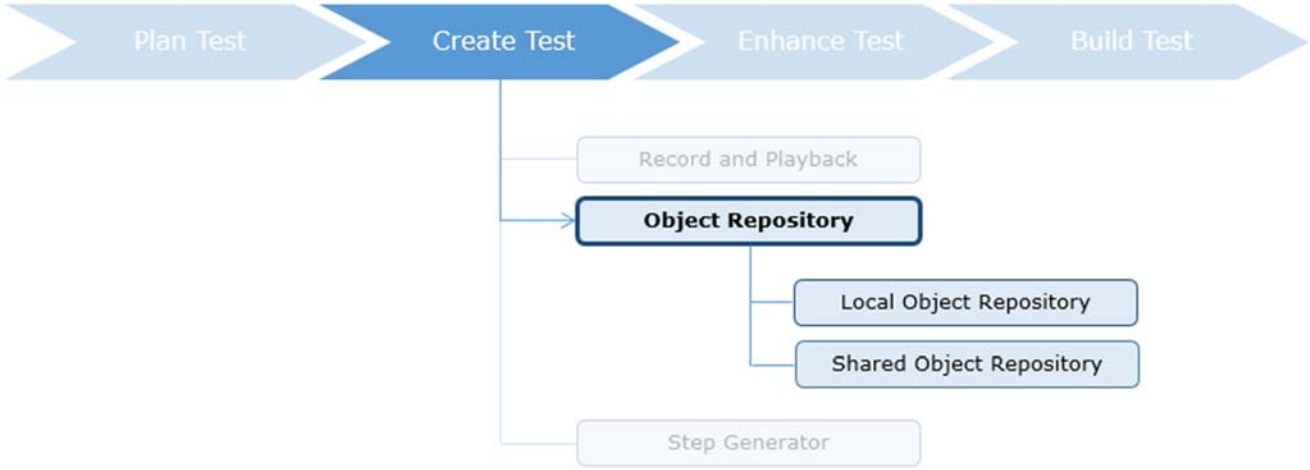
Note: Above step is repeated till the UFT is able to recognize the object uniquely or no more assistive property is in the list. In this case, UFT will go to **ordinal identifier**.

Step 3: Ordinal identifier is the sequential number (0, 1, and 2 etc...) generated by UFT based on order in which the object appears in the application.

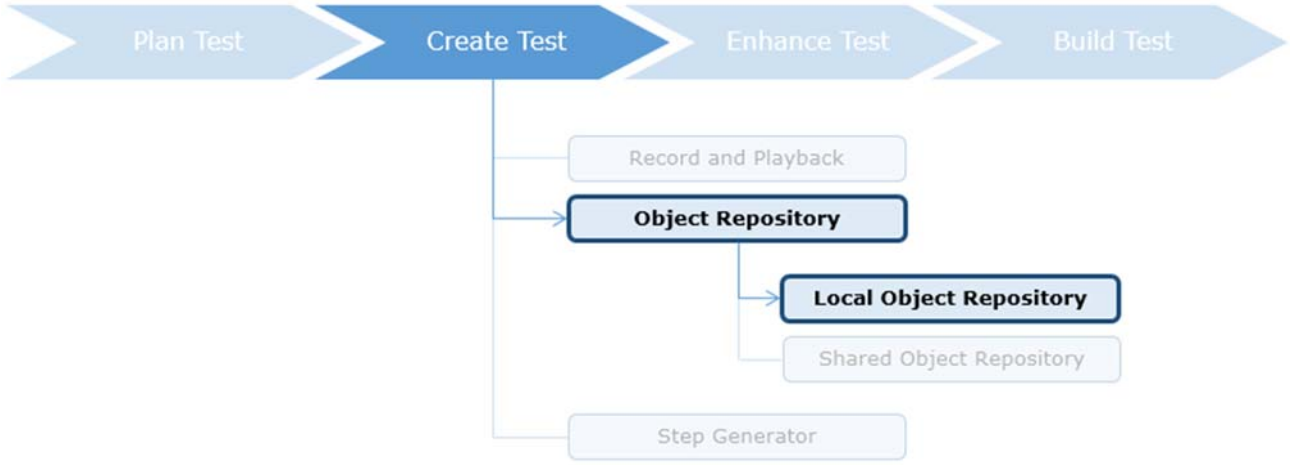
Step 4: If UFT finds more than one object that fits the description properties the UFT ignores the recognized description and uses the **Smart identification** mechanism to recognize the objects uniquely.

Object Repository – Types

Till now we have seen how Description properties are configured for test object. Let's see how test objects are stored in different types of repositories



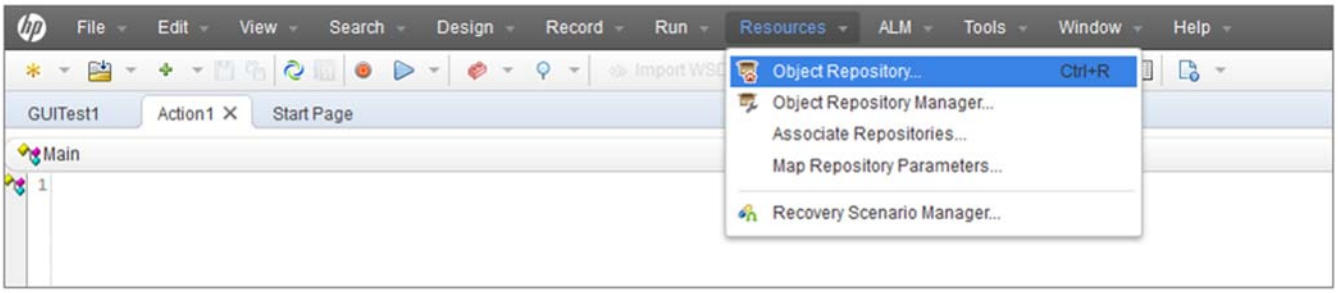
Local Object Repository



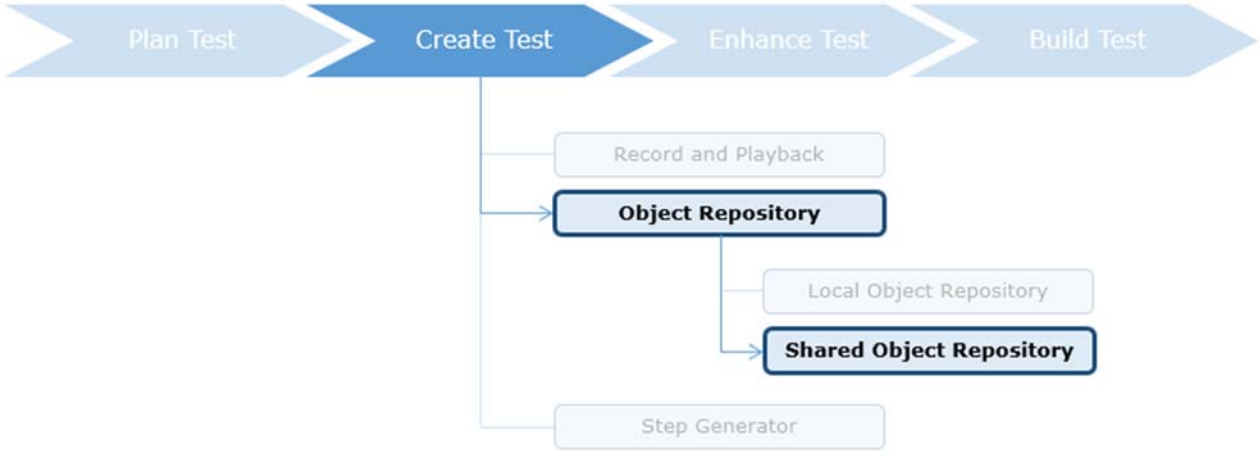
Local object repository is populated by default with test objects whenever recording is performed on the AUT. Whatever objects user has interacted

with during recording they will be captured and stored inside the local object repository. Follow the below step to check out the local object repository after recording a set of events.

Navigate: - Resources -> Object Repository.



Shared Object Repository



Global or Shared Object Repository as the name suggest is the type of object repository which can be shared among various tests.

- Between Shared and Local Object Repository, Shared Object Repository is more commonly used in automation projects
- Shared Object Repository is read-only by default
- It has to be explicitly associated with an action

How to work with Shared Object Repository:

Step 1: Create the Shared Object Repository

There are two different ways of creating the Shared Object Repositories

- Exporting objects from Local Object Repository to Shared Object Repository. This is demonstrated in the following page.

- Create Shared Object Repositories using Object Repository Manager(ORM). We will learn about ORM and how Shared Object Repository is created in ORM in video in further pages.

Step 2: Edit the Shared Object Repository using Object Repository Manager if needed.

Step 3: Associate the Shared Object Repository with your test

Step 4: Create the test scripts using the associated Object Repository

Lets see the above four steps in detail

Demo 2 : Exporting Objects from Local Object Repository to Shared Object Repository

Highlights:

- Create Shared Object Repository from Local Object Repository
- Learn about Shared Object Repository

Demo Steps:

Step 1: Record a test script.

Example: Record the login page of the flight reservation application.

Step 2: Open the object repository created after this recording

Step 3: In the Object repository navigate as below

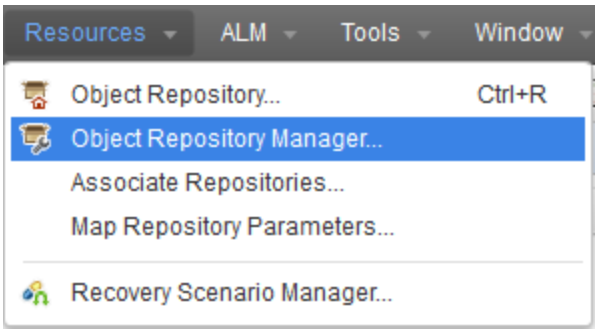
File -> Export and Replace objects ‘or’ "Export local objects".

Step 4: Save the shared object repository created on the file system with a suitable name.

Object Repository Manager

Object repository manager is a utility available in UFT that is used for creating and working with shared object repository.

How to open object repository manager:



Below mentioned are few of the commonly performed operations on the shared object repository using object repository manager:

- **Editing shared object repository**: By default a shared object repository is non-editable. To edit any shared object repository, first open that repository using an object repository manager and then select File -> Enable Editing.
- **Managing objects in shared object repository**: Renaming, deleting etc can only be done in shared object repository using object repository manager.
- **Comparing two shared object repositories**: Let us assume a situation where an automation tester is provided with two or more shared object repositories. Now they want to check whether these repositories are having common objects within them or not. To pursue this, object repository comparison tool is used.

Navigate: Tools -> Object Repository Comparison Tool

This feature enables us to compare two shared object repositories and to view the differences in their objects such as different object name, different object descriptions etc.

- **Merging two shared object repositories**: As the name suggests, this feature is used to merge two shared object repositories to combine them into one.

Navigate: Tools -> Object Repository Merge Tool

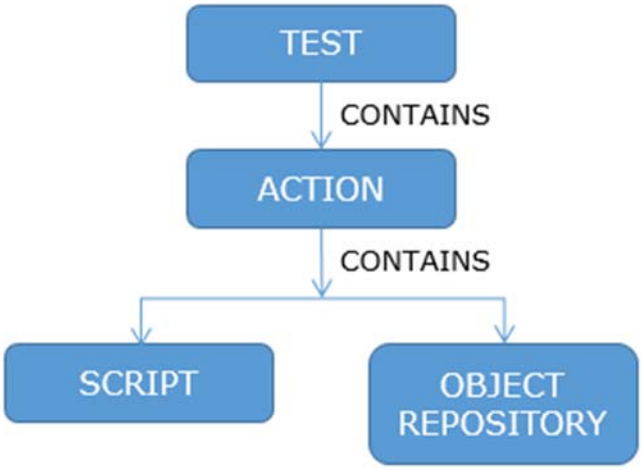
Note: While merging two shared object repositories objects are automatically compared.

Creating script using Shared Object Repository

Till now we understood the types of object repositories which UFT supports. We observed that with local object repository scripts are automatically created by UFT during recording. Now the challenge is with shared object repository, where no recording is done.

Here we will understand how test scripts are created using shared object repository.

Before moving forward let us understand one of the very important relationship in UFT:-



Explanation:- A test in UFT comprises of an object called action i.e. Action is a logical entity inside a test which contains the set of events to be performed on the application in the form of a 'script'. This script requires test objects to identify the objects in the AUT to perform the required operations at run-time. These test objects are present inside object repository.

With this description we can easily infer that an action contains two major entities a script and an object repository. This object repository can be of either types (Local Object Repository or Shared Object Repository). With either of the repository types action will contain the script only. This script can be created either manually or through recording.

Note: Through recording we have already seen how the script will be created.

Let us see how to create it manually using shared object repository.

Steps to follow:

- Create a shared object repository of the relevant objects required for the script.
- Save the shared object repository on the filesystem.
- Create new Test
- Associate shared object repository with the action.
- Drag and drop the objects from shared object repository to the Editor view required for script execution.
- Execute the script.

Local Object Repository vs Shared Object Repository

We have seen the types of repositories in detail and their usage. Let us combine the concepts learnt and look at the basic differences between them.

Local Object Repository	Shared Object Repository
It is the default object repository	Shared/Global object repository is created separately by the user
Local object repository is action-specific and can be used only for that particular action	It can be accessed by multiple tests and actions
Preferred when AUT is not dynamic with respect to time	Preferred when AUT is dynamic with respect to time
It is saved with .bdb extension	Shared object repository is saved with .tsr extension
Local object repository is editable repository	Shared Object repository is read only by default but can be edited by using object repository manager

Object Spy

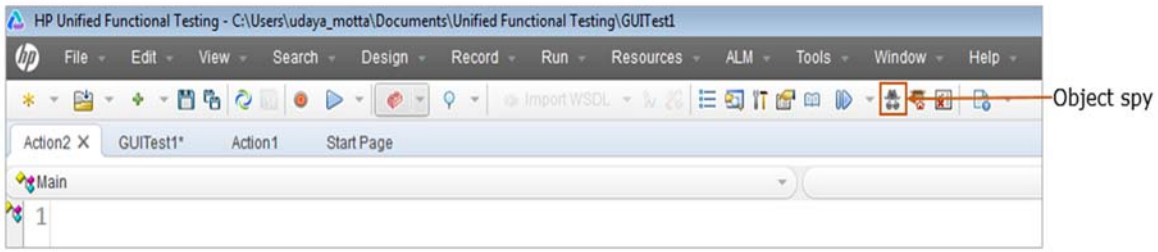
Object repository contains the information about test objects i.e their description properties. Now the point is how to view the properties associated with **run-time** objects?

Object spy is an inbuilt tool in UFT which can be used to get the following information about a **run-time** object from AUT.

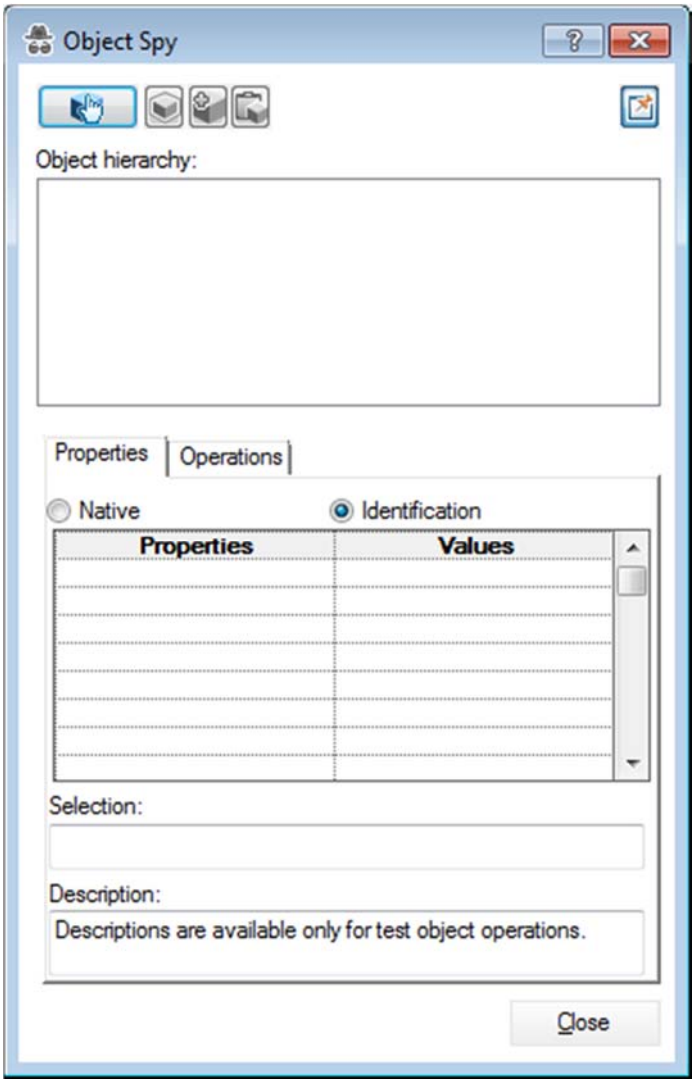
- Native properties and operations of a run-time object
- Identification properties which can be used in test object
- Operations which can be performed using test objects
- Hierarchy structure of the run-time object

Steps to open the Object Spy :

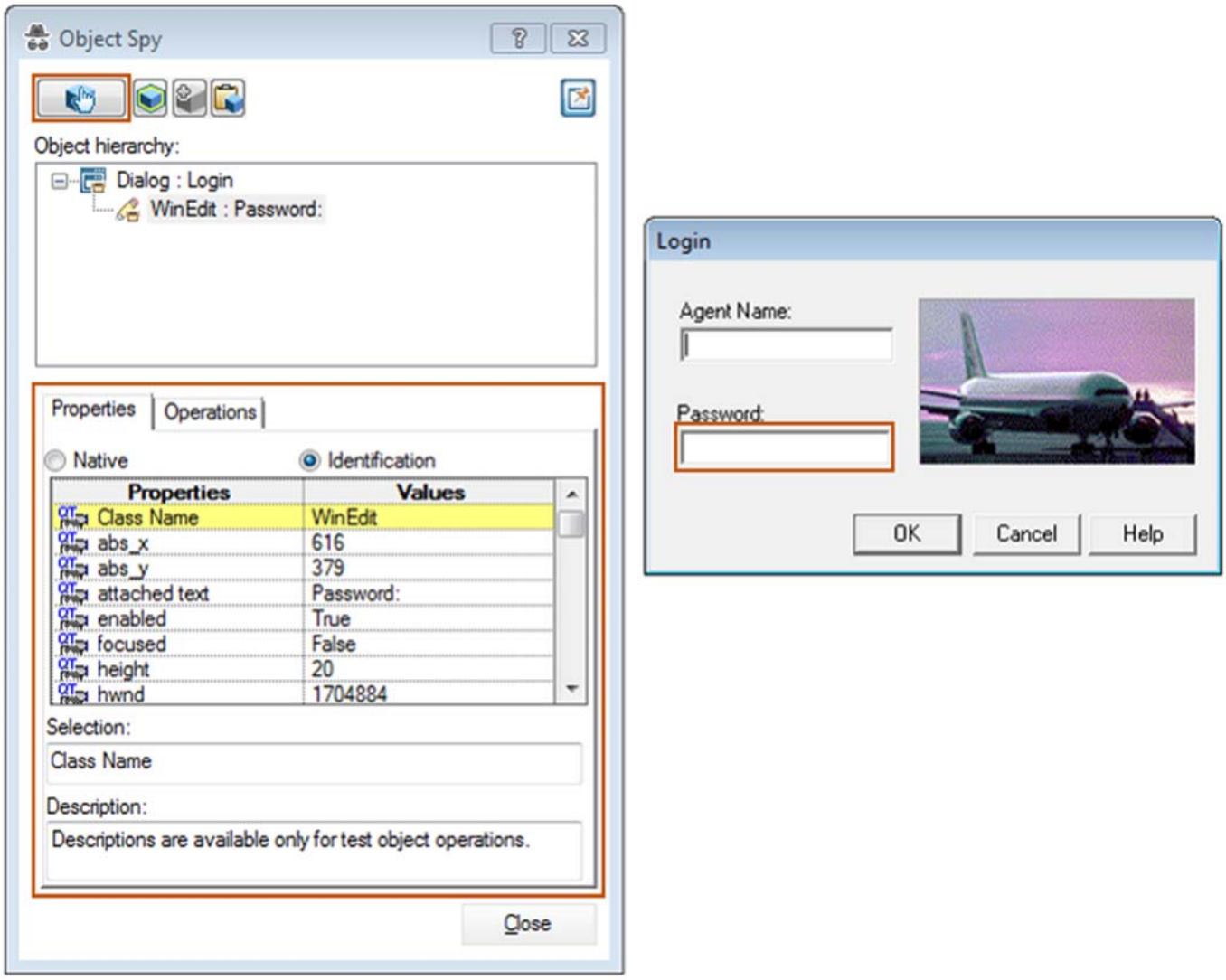
- Create a new test :- File->New->Test
- Click the “Object Spy” button on the toolbar



- "Object Spy” window will appear

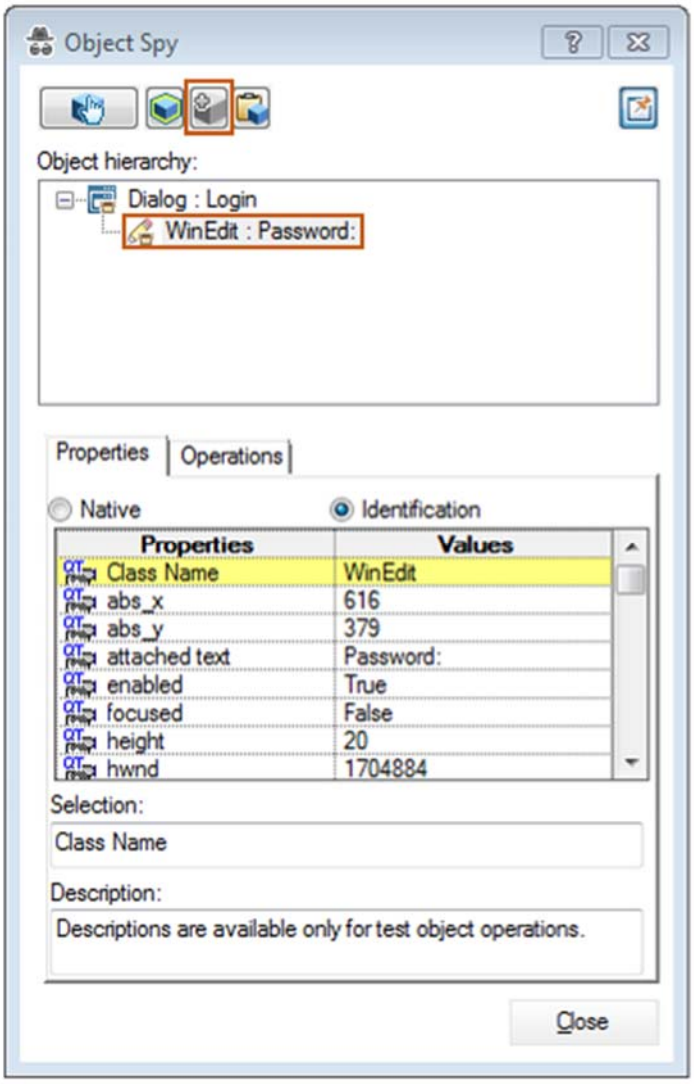


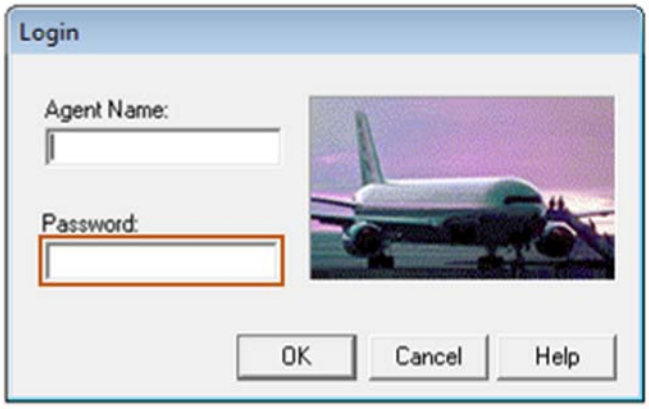
- Click on the Pointing finger button on the window and place the pointing finger on the application. It will display the properties of that object. Check the image below.



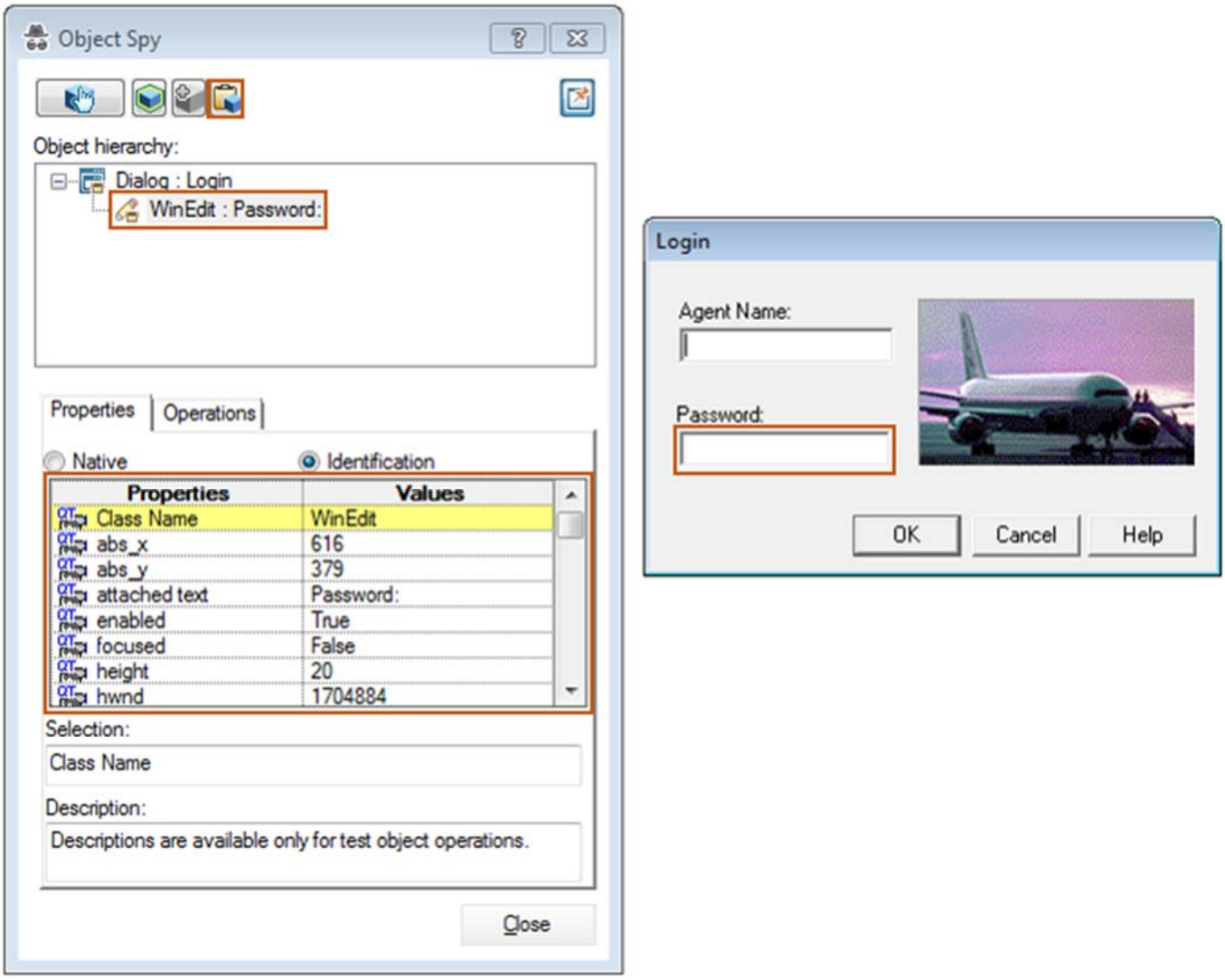
Object Spy can also be used for:

- Adding selected objects to the currently active object repository.





- Copy all identification properties and their values to clipboard. Select the object in the object hierarchy section whose identification properties you want to copy on the clipboard. Then click on the button highlighted on the object spy window to copy all the properties.

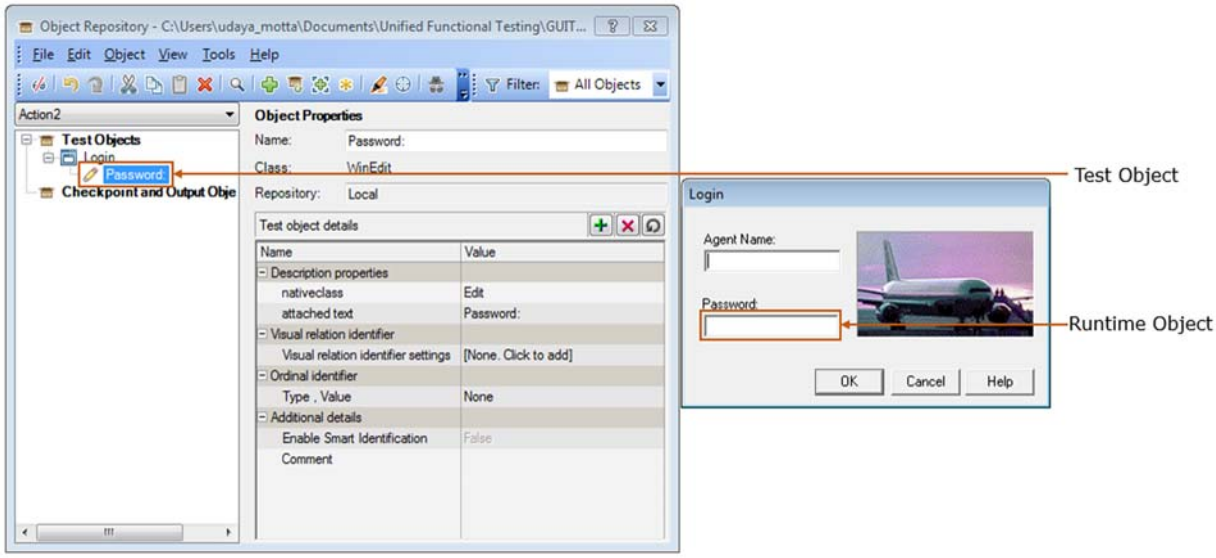


Note: It cannot be used while recording or playing a test.

Test Object vs Run-Time Object

So far we have seen how to work with Test Objects and how they are maintained in object repository. Now let us see the comparison between Test Object and Run-Time Object.

The below image will differentiate between **'Test Object'** and **'Run-time Object'**.



- During recording UFT will learn the properties of the GUI object and stores them in the Object Repository.
- During Runtime UFT will compare the stored object properties with the actual properties of the object available on the AUT. If the perfect match happens then script will perform the required operation on the run-time objects.

Example: During replay when UFT wants to enter password value in the "Password" field, it uses the Test Object properties captured by UFT to identify the run-time object on the application.

Test Objects vs Run-time Objects

Run-time Objects	Test Objects
Reside in the application's Graphical user interface(GUI)/Source code	Reside in the Object repository
Created by application developer	Create by UFT/Automation tester
Used by human users to interact with the application	Used only by UFT test script to identify a runtime object uniquely
Contains all the properties of pertaining to its class	Contains only a subset of properties pertaining to its class, which are sufficient for unique identification
UFT can modify values of only certain properties, that are allowed as per the class definition in the source code using UFT scripts. E.g. Text inside an edit box, on-off status of a radio button ETC.	Properties can be added, removed and any of their values can be modified in UFTs object repository

Now that we know about test objects in UFT and their properties, in the next topic we will see how to identify the properties of run-time objects present on the application.

Object Repository Instances

During execution, the script and the object repository gets loaded in the main memory. Object repository contains test objects with properties and their values. If somehow we have to make any changes in the object's property value we can go and change it inside the object repository. Now what if we have to make any change in the object's property value at the run-time. For this we need to understand about the various instances of object repository.

There are two instance of the Object Repository that comes into action whenever they are used:-

- Design-Time Instance
- Run-Time Instance

Design-Time Instance

Design time instance of the Object repository is the one that gets created whenever a script is recorded/when object repository is created. Below are the Design time instances

- Local Object Repository
- Shared Object Repository

Run-Time Instance

Run-time instance of an Object Repository is the one that gets created in the main memory whenever a script gets executed.

Now to work with the objects present in run-time instance of the object repository and the run-time objects present on the AUT, we would be needing the help of below listed functions:-

1. GetTOProperty – also abbreviated as Get-Test Object-Property

Syntax: GetTOProperty(“propertyname”)

This function is used for fetching a “propertyvalue” of a particular “propertyname” of the test-object in your object repository either local or shared.

2. SetTOProperty – also abbreviated as Set-Test Object-Property

Syntax: SetTOProperty “propertyname”, “propertyvalue”

This function is used for inserting a new “propertyvalue” to a particular “propertyname” of the test-object in your object repository either local or shared.

3. GetROProperty – also abbreviated as Get-RunTime Object-Property

Syntax: GetROProperty(“propertyname”)

This function is used for fetching the property value of a particular “propertyname” from the run-time object present in the AUT.

In the next page let us understand the working of these functions better with the help of a scenario.

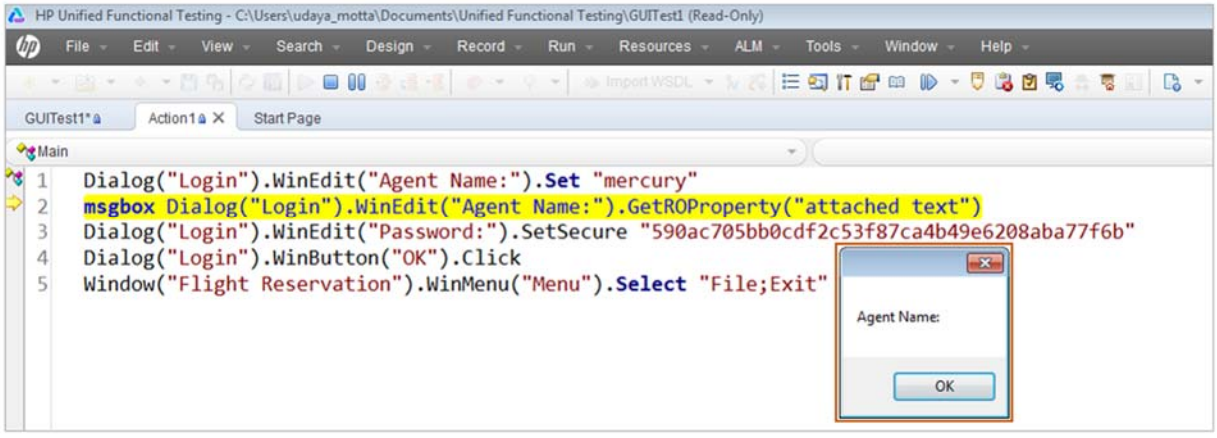
Object Repository Instances – Example

Scenario:

Consider the login page of flight reservation application. User enters the valid credentials in the “Agent Name:” and “Password” field of the login page while recording the login event and after logging in close the flight reservation window. The automation tester wants to fetch the “attached text” property value of “Agent Name” edit box from the object repository and change it to “Infosys” in the object repository at run-time. Also whatever value is present in “Agent Name” object on the application, tester wants to fetch it during the run-time. Follow the below mentioned step to perform the same.

Steps: - Record the login – logout event using normal recording mode, with Agent Name: - “mercury”, Password: - “mercury”.

Script 1: - Script to fetch the “attached text” property value of the Agent Name.

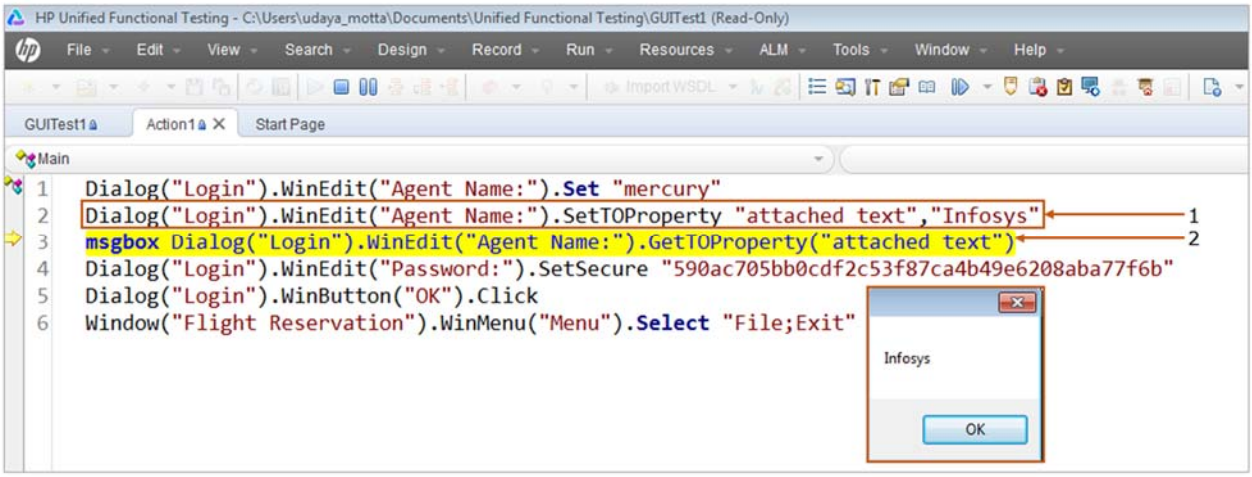


Highlighted statement contains the complete code how to fetch value using gettoproperty().

Note: Proper object hierarchy needs to be maintained before putting gettoproperty function.

Note: msgbox – is an inbuilt function in vbscript used for printing the results on screen**.**

Script 2: - Script to change the “attached text” property value of the Agent Name and check if change occurred or not in the object repository.

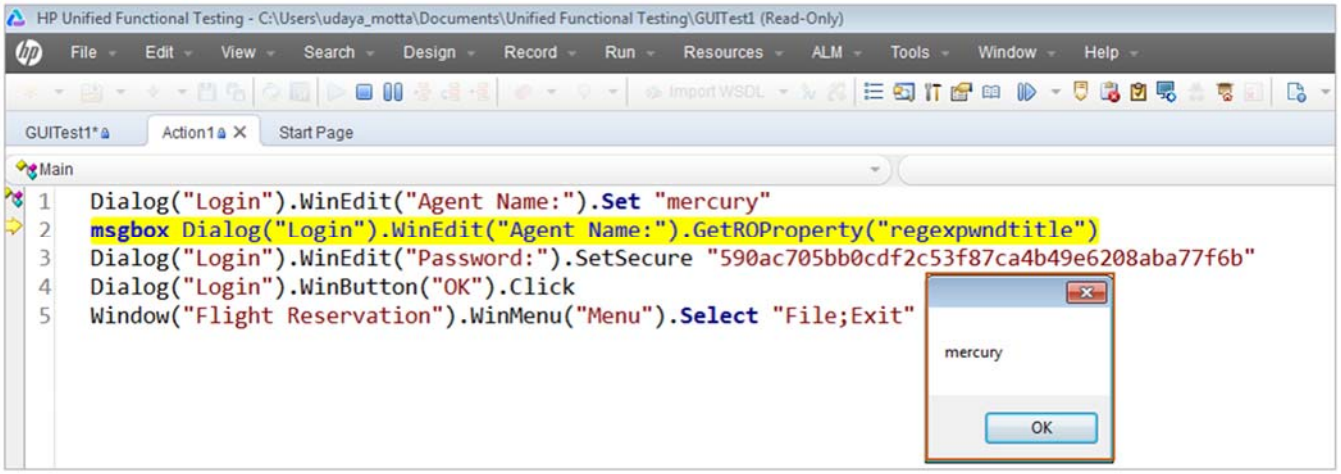


Statement1 – This will change the value in the run-time instance of the object repository.

Statement2 – This will fetch the result from run-time instance of object repository and display on screen.

Note: After executing the script if you open the object repository, changes will not reflect there as settoproperty works on the run-time instance of the object repository.

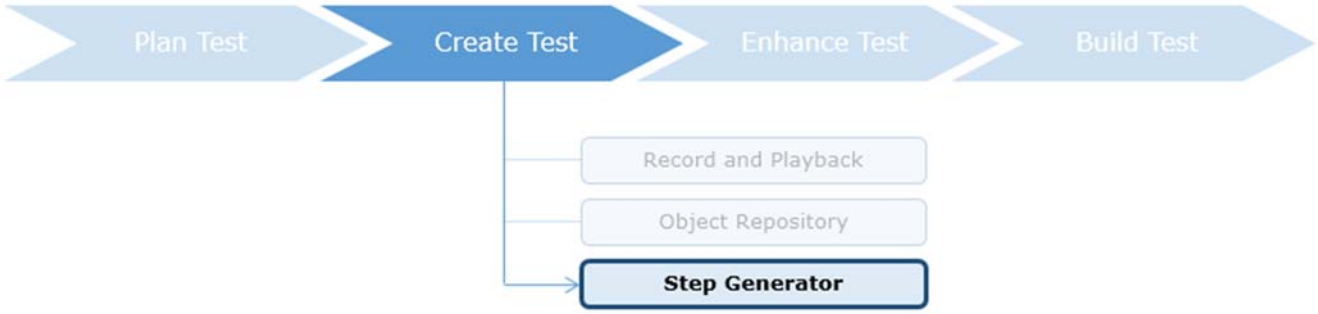
Script 3: - Script to fetch the value present in “Agent Name” object in the AUT.



Highlighted statement contains the complete code on how to fetch value using getroproperty().

Note: - The property name that is kept inside getroproperty() function can be fetched from the object present in the AUT using Object Spy. You can put the relevant propertyname as per the requirement.

Step Generator

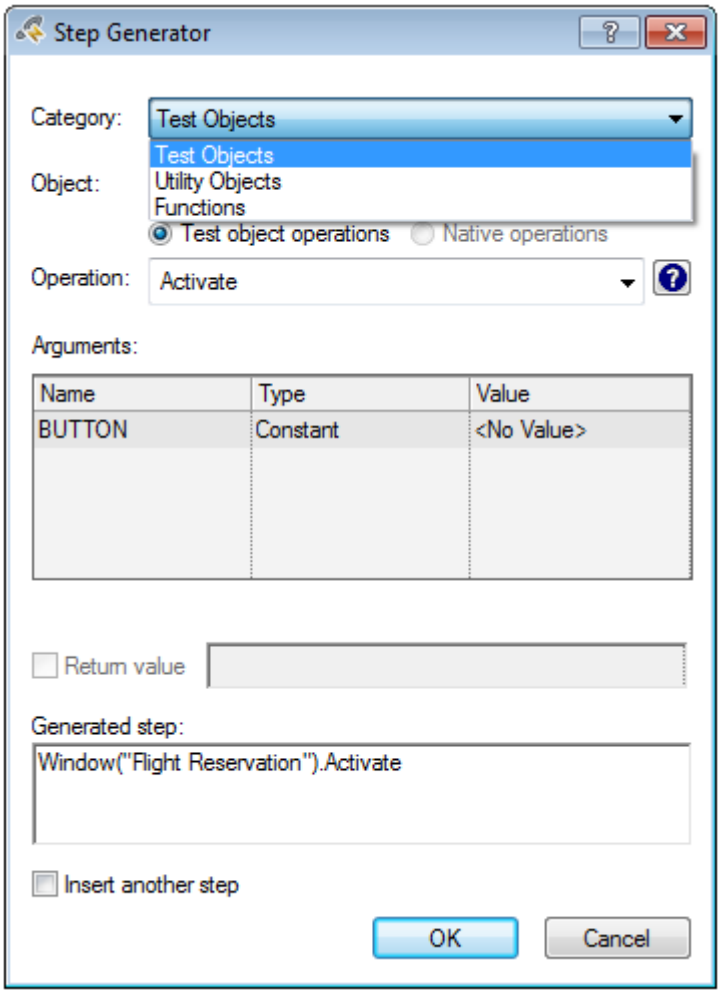


In an already created script, new statements can be added to increase their flexibility. There are various methods to insert logical statements. One of the most commonly used method is Step generator.

Note: - It can also help us in generating the script by adding steps without recording on the AUT. It is very much time consuming and that's why not a recommended option to be used in the real time situation.

Note:- For using step generator, it is mandatory to have test objects in the object repository.

To activate Step Generator **Navigate- : Design->Step Generator**. After selecting "Step Generator" we will get the below window:-

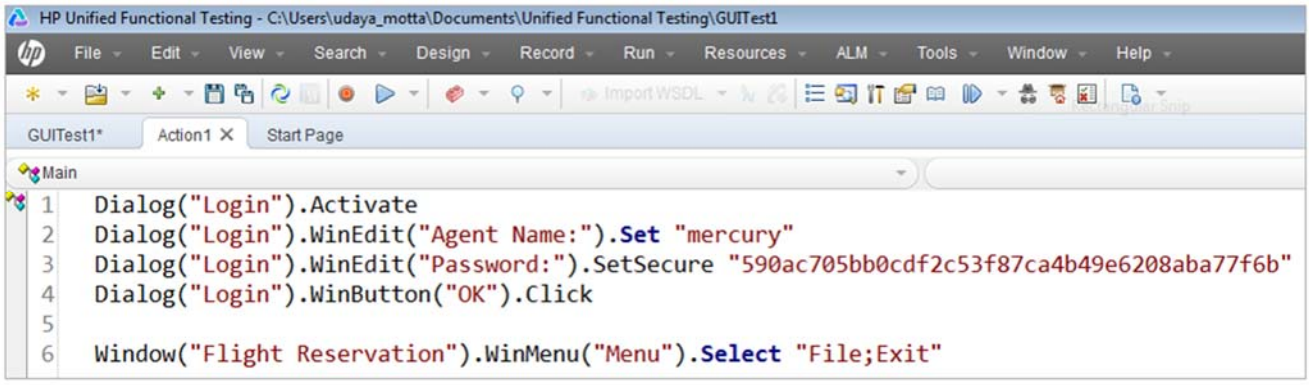


Various categories available in Step generator: -

- **Test Objects** – Used to add test object methods and properties
- **Utility Objects** – Used to add utility object methods and properties
- **Functions** – With this we can provide calls to VBScript functions, internal script functions and library functions

Let us understand it with an example, where we will see how to use “Test Objects” category to add statements in an already created test script.

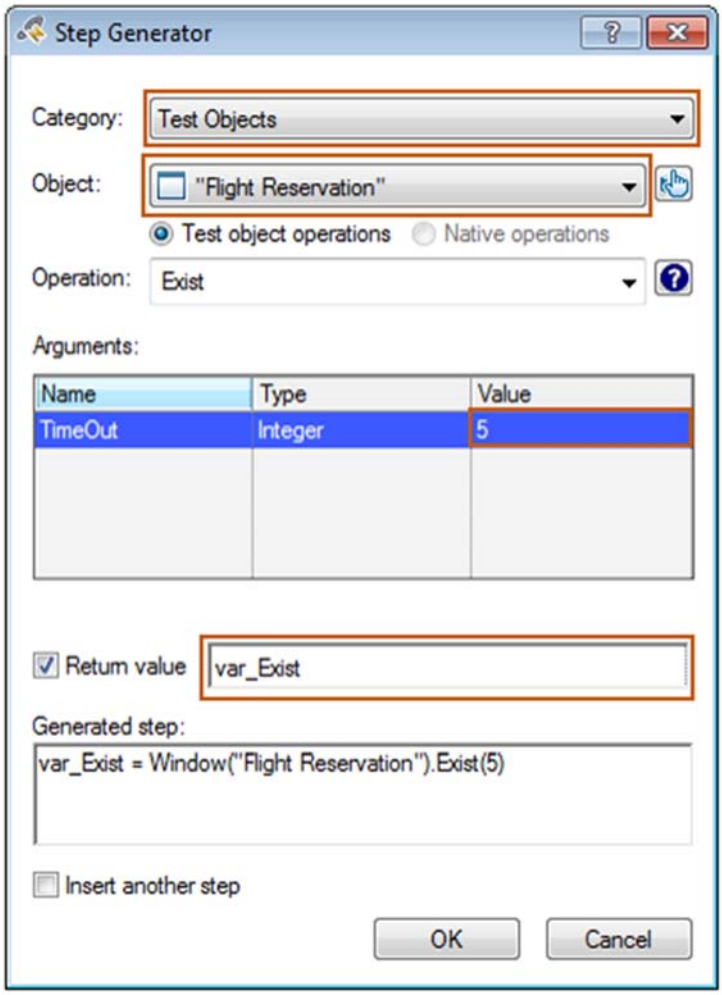
Example: Consider a scenario when an automation tester is working on Flight reservation application. After logging in he wants to place a new order. To achieve this, “New Order” button has to be clicked on the flight reservation window. Below is the script that we get after logging in and clicking on “New Order” button on the flight reservation window:-



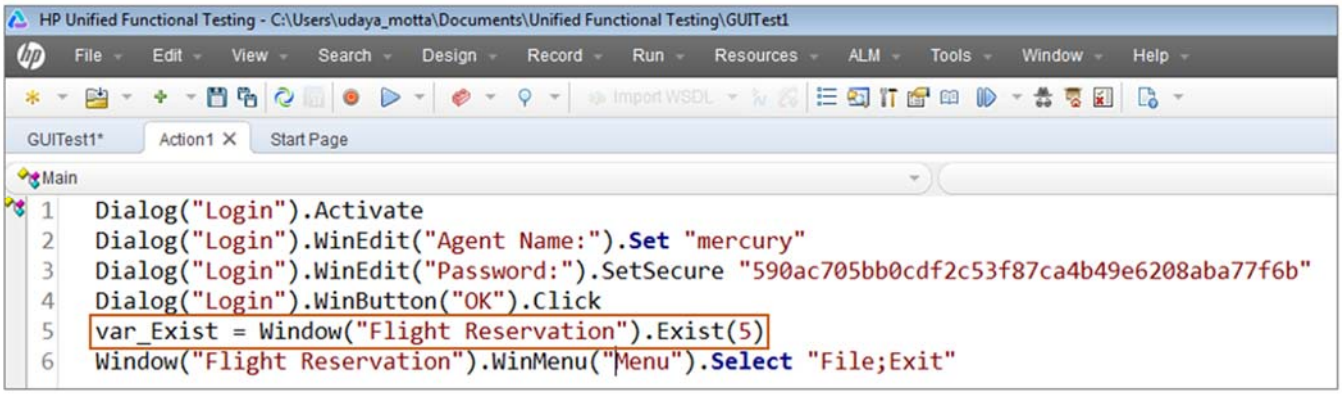
After recording the script, automation tester decides to check if the flight reservation window exists or not.

Note: - For validating if a window exist or not we have a property '.exist', which we are going to use here.

- Place your cursor before the statement where to insert this new step**(Look at the cursor position in the image above.)**
- Go to Design -> Step Generator, the step generator window will get launched.
- Select "Test Objects" category.
- Select the object as "Flight Reservation" as this is the object which we have to check if existing or not.
- Select "Exist" under Operation drop down.
- In the "Arguments" section under "value" column put value=5.



- The return value of the operation be stored in a variable “var_Exist”.
- After clicking on OK, below is the statement that gets added in the script. Result of var_Exist is being displayed using msgbox.

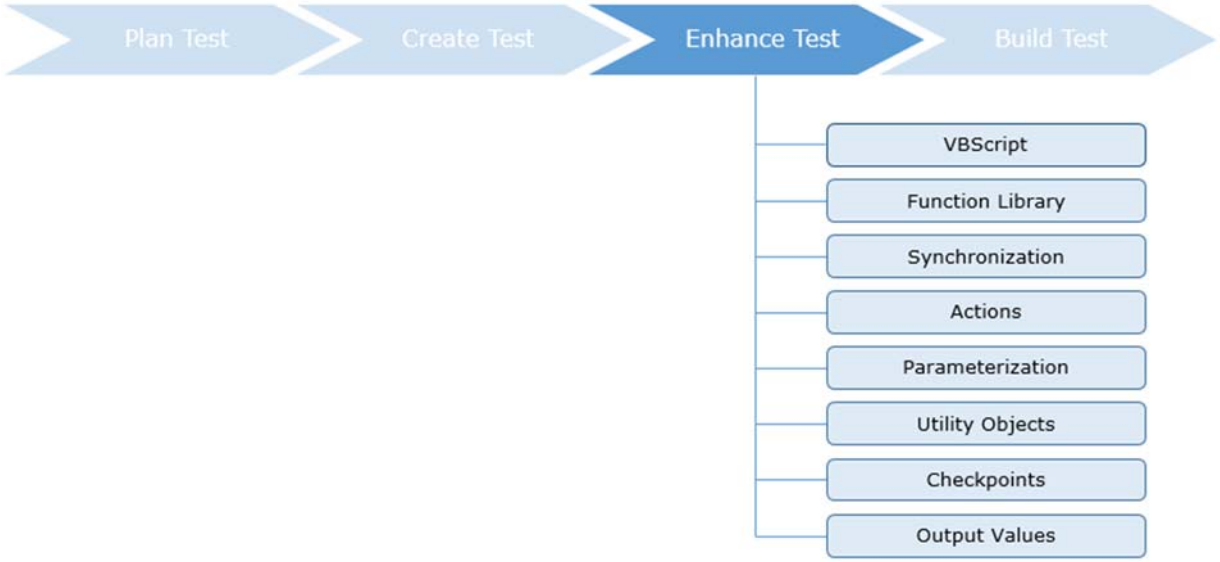


Create Test – Summary

Let us summarize what all we learnt in create test phase:

- What is recording and playback.
- Various modes of recording:
  - Features
  - Implementation
- Importance of Test object model.
- Object repository and it's various types:
  - Local object repository
  - Shared object repository
- Various instances of object repositories.
- Relevance of object spy in identifying the properties of run-time objects.
- How to use step generator to add steps in the script.

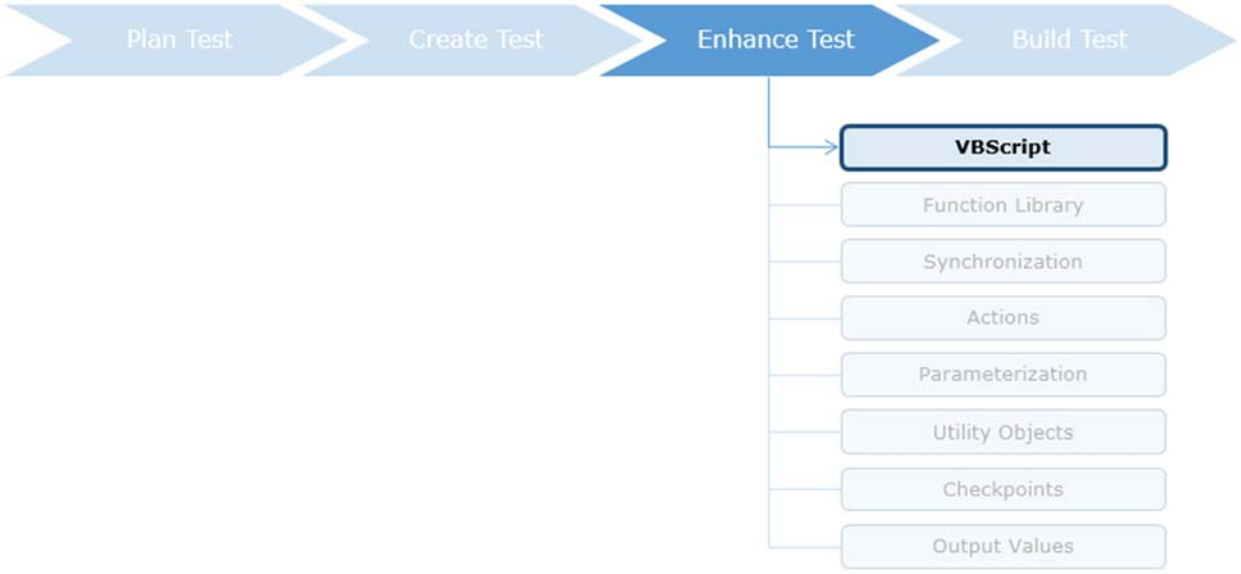
Enhance Test



Most of the times after creating a script an automation tester may need to enhance a test script so that it can be made little more versatile in its operations. Below are the various ways of enhancing a test script:

- VBScript
- Function Library
- Synchronization
- Actions
- Parameterization
- Utility Objects
- Checkpoints
- Output Values

Enhance Test - Introduction to VBScript



While making enhancements in the test scripts, vbscript plays a very crucial role as it is the language which UFT understands.

So here we will understand the basic concepts of vbscript which will help us in enhancing our test scripts.

VBScript

- VBScript stands for Visual Basic Script, syntax of which is very close to Visual Basic
- VB scripting is a client side scripting language
- VBScript is developed by Microsoft

- VBScript can run only on Windows machines and Internet Explorer Browser
- VBScript is case insensitive

VBScript - Variables and Constants

Variables

A variable is a placeholder that refers to a computer memory location where a value or expression can be stored. The value can change during the execution of the script.

Variable names should follow the naming conventions

- Must begin with an alphabetic character
- Cannot contain a period (.)
- Cannot exceed 255 characters

In VBScript, all variables are of type **“variant”** that can store different types of data.

Declare variables

Variables can be declared using the **Dim** statement, the **Public** statement or the **Private** statement.

Syntax:

```
Dim <<varname>>
```

Assigning values to variables

Variables cannot be declared and initialized at the same time in VB scripting. The declaration and initialization should be done in separate statements. This is known as **explicit declaration**.

Syntax:

```
Dim <<varname>>
<<varname>> = <<value>>
```

The variable can also be declared by using its name directly. This is known as **implicit declaration**.

Syntax:

```
<<varname>> = <<value>>
```

Note:- However, this is not a good practice as the script may give unexpected results if the user misspells the variable name by mistake. Vbscript will treat the misspelled variable as a new variable.

Option Explicit

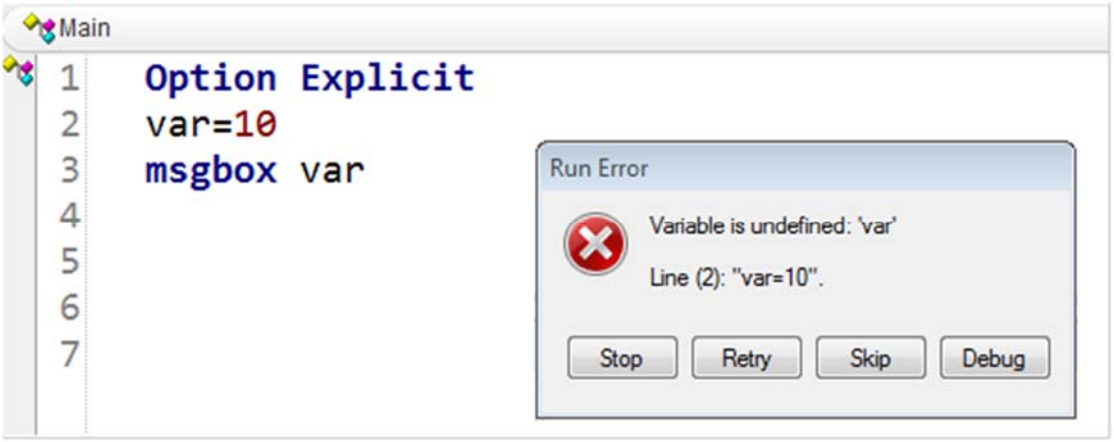
To make the declaration of variables mandatory 'Option Explicit' can be used. This ensures that the variables should be declared using dim, private or public statement before using them. Also, the scope of the variable is clearly defined.

Syntax:

```
Option Explicit
Dim <<varname>>
```

Note:- Option explicit has to be the first statement in the script.

Example:



This example demonstrates the error message that appears when variable is not declared despite writing "Option Explicit".

Constants

A constant is a variable within a program whose value never changes.

To differentiate between variables and constants the user can follow the below naming conventions:

- Using all uppercase characters for constants
- Alternative way is prefixing the name of the constant with “con”

Constant declaration:

Constants can be declared using the keyword "**const".**

Example:

```
const VOTING_AGE = 18
```

Note :- Constants are declared and initialized in the same statement.

In this case a constant named “VOTING_AGE” is declared and the value is initialized to 18.

VBScript - Conditional statements

Controlling the flow of a script is very important from any scripting languages perspective. There are two formats to attain that.

- Conditional statements
- Looping statements

Here first we will see how conditional statements are written in vbscript. Conditional statements help us in performing the different actions depending upon the conditions given.

- Conditional statements will be used in order to evaluate the logic or operation
- It returns a true/false value
- There are three different types of conditional statements, they are:
    - If-Else
    - Else-If ladder
    - Select - Case

If – Else

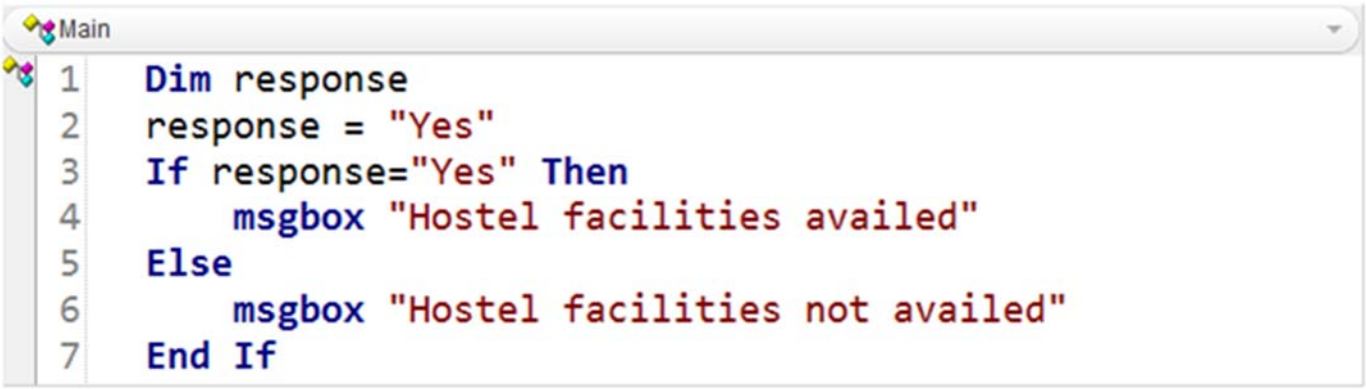
Syntax:

```
If (condition) Then
Statement or a block of statement – set 1
Else
Statement or a block of statement -  set 2
End if
```

If the condition defined is satisfied or ‘true’ then the statement or block of statements – set 1 is executed. Otherwise, the control goes to the Else part and then the statement or block of statements – set 2 will be executed.

Example:

- A student can choose to avail the college accommodation facility. If his choice is ‘yes’ then display “Hostel facilities availed” else display “Hostel facilities not availed”.
- Consider a variable ‘response’ containing the student’s response (Yes or No).



Output:

“Hostel facilities availed” gets printed in a pop up window.

Else- If ladder

Syntax:

```
If (condition) Then
Statement or a block of statement – set 1
Elseif (condition) Then
Statement or a block of statement – set 2
Elseif (condition) Then
Statement or a block of statement - set 3
Else
Statement or a block of statement - set 4
End if
```

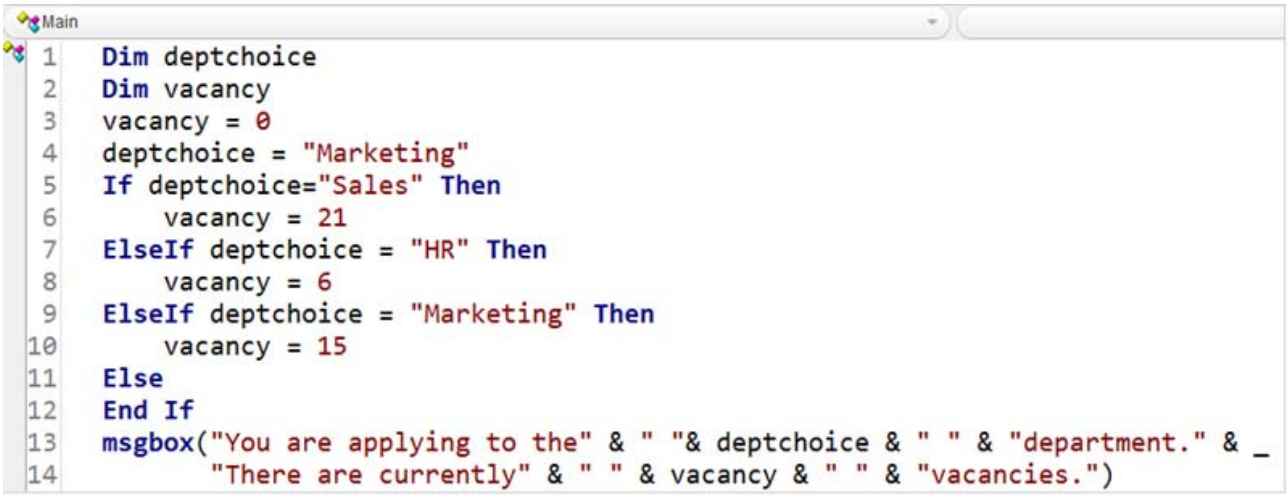
If the condition defined is satisfied or **'True'** then the statement or block of statements – set 1 is executed otherwise, the control goes to the next **'Else**

If' part to check if the condition defined within that is satisfied. If yes, then the statement or block of statements – set 2 gets executed.

If the condition is not met, the control goes to the next **'Else If'** part the statement or block of statements – set 3 gets executed. If none of the conditions are met, then the control goes to the **'Else'** part and executes the statement or block of statements – set 4.

Example:

- An applicant can choose the department he/ she wants to apply to for a job.
- For departments 'Sales', 'HR', 'Marketing', the message "You are applying to the <<applicant's choice of department>> department.
- There are currently <<vacancy>> vacancies." should get displayed in a pop up window. If applicant's choice is any other department, then the message "Currently no openings" should get displayed in a pop up window.
- Consider a variable 'deptchoice' containing the applicant's choice. The code looks like as below:-
-



Output:

"You are applying to the Marketing department. There are currently 15 vacancies." gets displayed in a pop up window.

Select Statement

This is used to pick one out of the many options depending on the condition that comes satisfied. The condition is evaluated once and based on the value it attains one of the following blocks of code which gets executed.

Syntax:

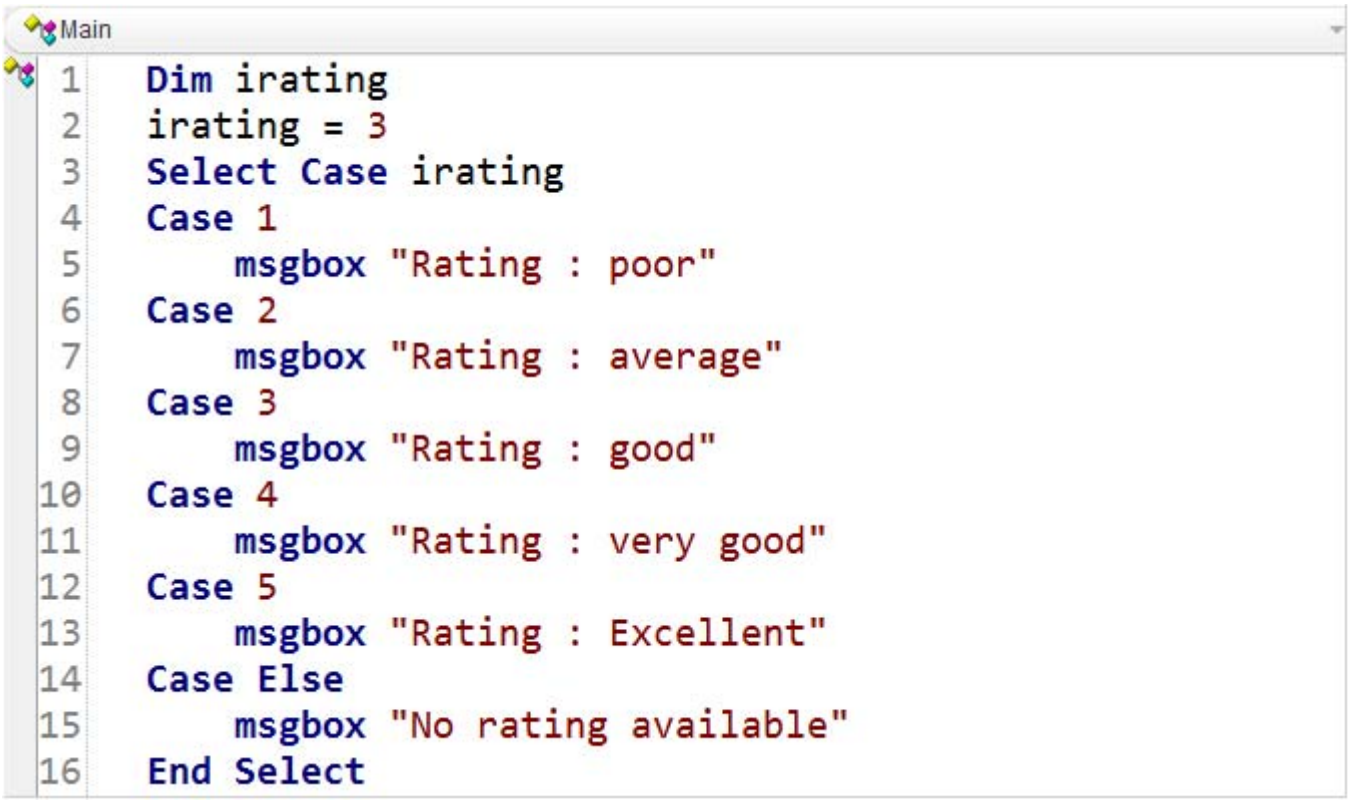
```
Select Case (expression)
Case “case1”
'Block 1 - set of statements
Case “case 2”
'Block 2 - set of statements
…..
Case Else
'Else block
End Select
```

Example:

Consider a scenario to get the user input for the rating of a course. The rating can be between the value ranges 1-5. If any other value is given by the user, the message “no rating available” should get displayed in a pop up window. The rating table is given below.

Rating	Message
1	Poor
2	Average
3	Good
4	Very good
5	Excellent

The code using the select case conditional statement looks as in:



Output:

“Rating : good” gets displayed in a pop up window.

VBScript - Looping statements

Looping statements are the statements which are used to execute one or more statement repeatedly several number of times

Below are the various types of looping related statements: -

- While…..Wend
- Do…..Loop
- Do…..While…..Loop
- Do…..Until…..Loop
- For…..Next
- Exit Do

While…Wend Loop

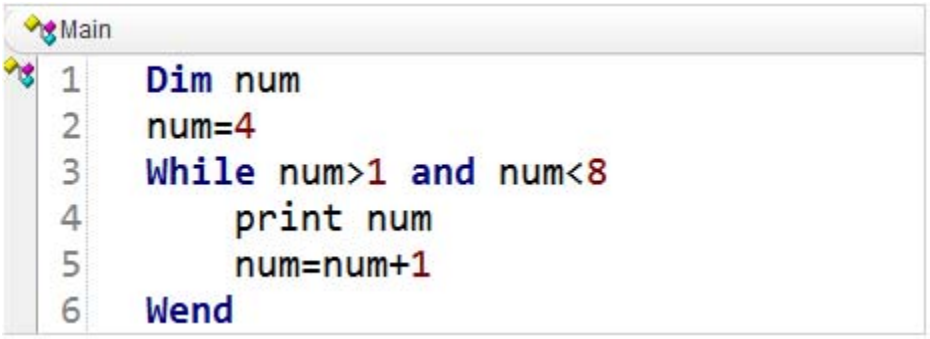
- This is used when a statement or a block of statements are executed as long as condition is true.
- When the condition becomes false, the loop is exited and the control moves to the next statement after Wend keyword.

Syntax:

```
While condition
Statement 1
.
.
.
.
Statement n
Wend
```

Example:

Consider the below image -



Output:

4

5

6

7

While...Wend is an older looping structure. The drawback of this loop is that there is no way to stop the execution in between using statement like Exit Do.

Note: - 'print' is another keyword used for displaying the results in UFT.

Do…Loop

This is used when a statement or a block of statements need to be executed while or until a said condition is true. This type of loop can be implemented using :-

- Do….While…..Loop
- Do…..Until……Loop

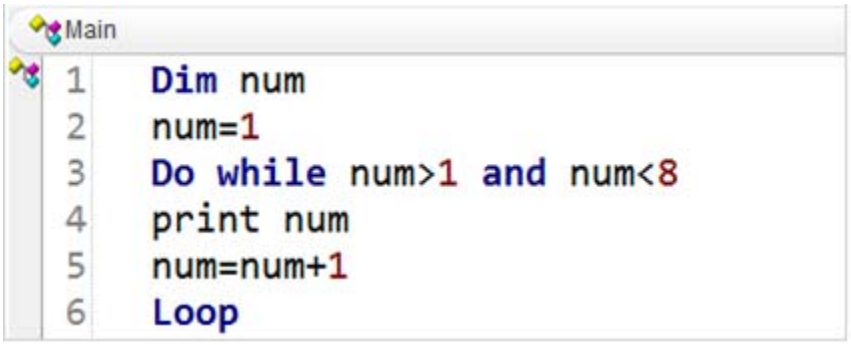
Do..While..Loop

Syntax:

```
Do (While) condition
Statement 1
.
.
.
.
Statement n
Loop
```

The statements within the do loop get executed if the condition holds true.

Example:



Output:

Nothing gets printed since the condition in the do..While.. Loop is not satisfied.

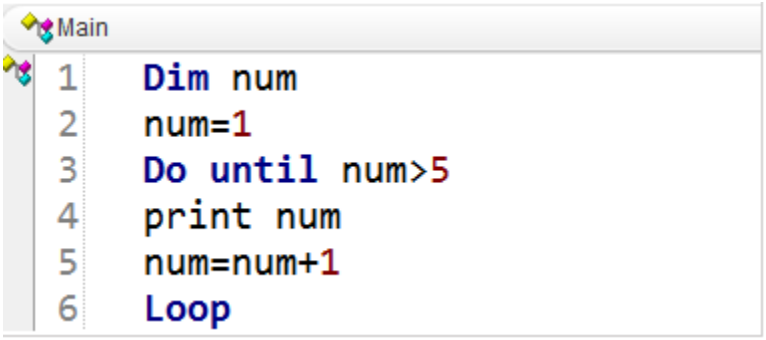
Do...Until...Loop

Syntax:

```
Do (Until) condition
Statement 1
.
.
.
Statement n
Loop
```

The statements within the do loop do get executed till the condition becomes true.

Example:



Output:

1

2

3

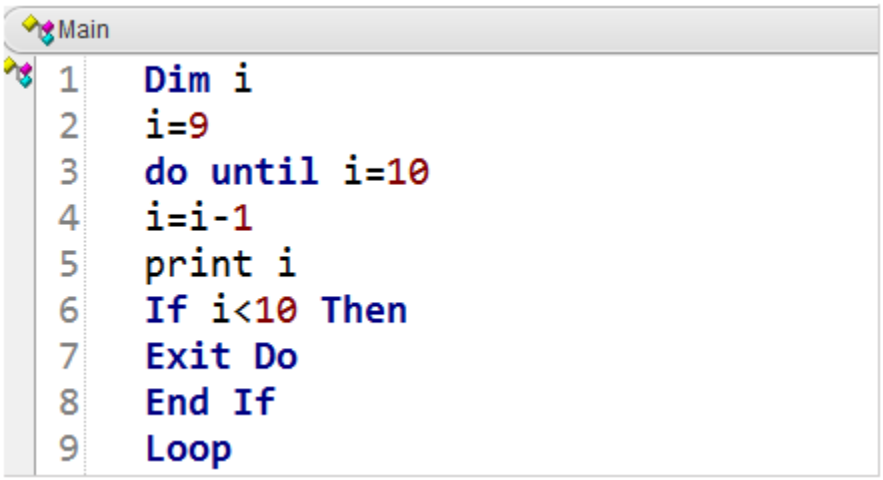
4

5

Exit Do

To avoid infinite loops, we make use of EXIT DO statement to force the loop to exit.

Example:



For...Next

For...Next statement is used when we want a statement or a block of statements to run a certain number of times with a counter that gets incremented and decremented.

Syntax:

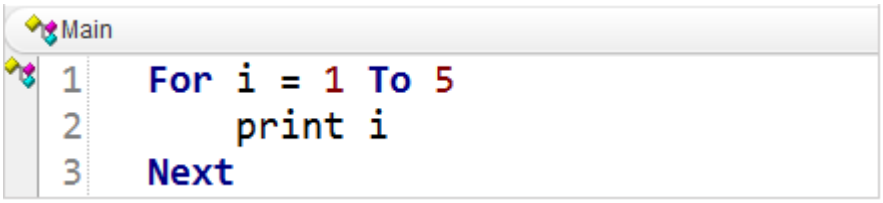
```
For counter = start To end [Step n]
statement 1
.
.
.
.
statement n
Next
```

Next keyword increments the counter variable's value by 1.

Step keyword increments the counter variable's value by any value that the user specifies.

Example 1:

Consider below figure. This example implements For…Next statement without the STEP keyword.



Output:

1

2

3

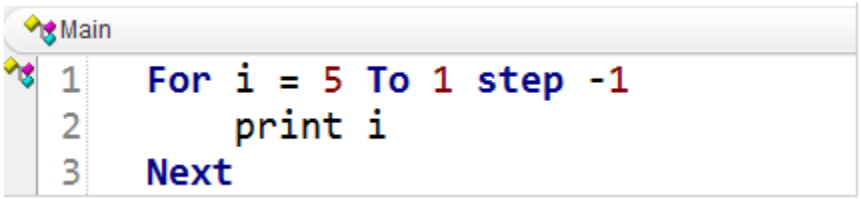
4

5

Note: if you want to exit the 'For' loop, you can use the EXIT FOR statement.

Example 2:

Consider below figure. This example implements For…Next statement with the STEP keyword.



Output:

5

4

3

2

VBScript – SubProgramming

Sub-programming plays a vital role in the programming because it will help the programmer to create reusable codes.

In Vbscript, there are two ways to do sub-programming:

- Function
- Sub procedure

Function

- A function is a series of VBScript statements enclosed by the 'Function' and 'End Function' statements. A function can also return a value.
- A function can take arguments (constants, variables, or expressions that are passed to it by a calling procedure).
- If a function has no arguments, its 'Function' statement must include an empty set of parentheses. A function returns a value by assigning a value to a variable having same name as the function. The return type of a Function is always a Variant.
- [] square brackets shows the content is not mandatory.

Syntax:

```
[Public [Default] | Private] Function function-name [(arglist)]
   [Statements]
   [Name = expression]
   [Exit Function]
   [Statements]
   [Name = expression]
End Function
```

Example:

```
Function Celsius ( fDegrees)
   Celsius = (fDegrees - 32) * 5 / 9
End Function
```

Note: - Here Celsius variable will return the result produced by the expression on right hand side.

- If not explicitly specified using either **Public** or **Private** keywords, function are public by default, that is, they are visible to all other sub-programs in your script. The value of local variables in a function is not preserved between calls to the procedure.
- You cannot define a function inside any other sub-program
- The **'Exit Function'** statement causes an immediate exit from a function. Program execution continues with the statement that follows the statement that called the function. Any number of **'Exit Function'** statements can appear anywhere in a function.

Sub-Procedure

- A Sub procedure is a series of Vbscript statements (enclosed by Sub and End Sub statements) that perform actions but don't return a value.
- A Sub procedure can take arguments (constants, variables, or expressions that are passed by a calling procedure). If a **Sub procedure** has no arguments, its **Sub** statement must include an empty set of parentheses ().

Note: - Major difference between function and sub-procedure is, a function can return value whereas sub-procedure doesn't return a value. If a function doesn't return a value it will act like a sub-procedure only.

Syntax:

```
[Public [Default] | Private] Sub name [(arglist)]
   [statements]
   [Exit Sub]
   [statements]
End Sub
```

Example:

```
Sub ConvertTemp()
   temp = InputBox("Please enter the temperature in degrees F.", 1)
   MsgBox "The temperature is: " & Celsius(temp) & " degrees C."
End Sub
```

Implementation:

Let us assume the value of temp was entered as 64

Output:

“The temperature is: 17.77 degrees C.”

Note: Function “Celsius” defined earlier is called in the Sub Procedure.

- If not explicitly specified using either Public or Private keyword, Sub procedures are public by default, that is, they are visible to all other sub-programs in your script. The value of local variables in a Sub procedure is not preserved between calls to the procedure.
- You can't define a Sub procedure inside any other sub program type.
- The Exit Sub statement causes an immediate exit from a Sub procedure. Program execution continues with the statement that follows the statement that called the Sub procedure. Any number of Exit Sub statements can appear anywhere in a Sub procedure.

Call statement

Transfers control to a Sub-procedure or Function.

Syntax:

[Call] name [argument-list]

You are not required to use the **Call** keyword when calling a **sub-procedure**. However, if you use the **Call** keyword to call a procedure that requires arguments, argument-list must be enclosed in parentheses.

If you omit the **Call** keyword, you also must omit the parentheses around argument-list. If you use either **Call** syntax to call any intrinsic or user-defined function, the function's return value is discarded.

Example:

```
Function callDemo(text)

   MsgBox text

End Function
```

```
Call callDemo("Welcome to Infosys")
```

Output:

Welcome to Infosys

VBScript - String processing

So far we have seen how to work with variables and constants, a variable might even hold a string value in it. Now let us learn what are the different string processing functions that are available in Vbscript.

Here we will be looking at string processing functions of following types:-

- Trimming
- String chopping
- Length
- String comparison

String processing - Trimming functions

String trimming functions:

- At times, there comes a need to handle a string having spaces on either or both ends like “ Infosys “.
- It is required to trim or remove excess spaces from the string in order to process it further for string processing.
- VBScript has 3 functions for trimming -
  - Trim
  - Ltrim
  - Rtrim

Trim

Concept	The `Trim` function removes both the leading and trailing blank spaces of the given string.
Syntax	`Trim(string)`

Implementation:

```
Main
1  Dim str1
2  str1 = "  Infosys  "
3  msgbox "After Trim:"&Trim(str1)&"-"
```

Output:

After Trim:Infosys-

Ltrim

Concept	The `LTrim` function removes leading blank spaces of the given string.
Syntax	`LTrim(string)`

Implementation:

```
Main
1  Dim str1
2  str1 = "  Infosys  "
3  msgbox "After LTrim:"&LTrim(str1)&"-"
```

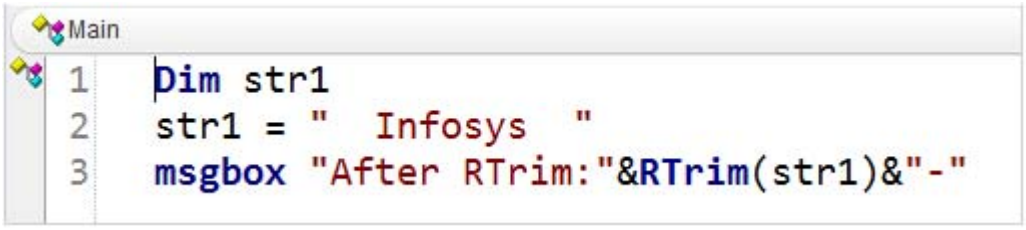
Output:

After Trim:Infosys -

Rtrim

Concept	The `RTrim` function removes trailing blank spaces of the given input string.
Syntax	`RTrim(string)`

Implementation:



Output:

After Trim: Infosys-

String processing - Chopping functions

At times, there occurs requirement to extract first few characters or last few characters or mid few characters out of a string. String chopping is used during those times.

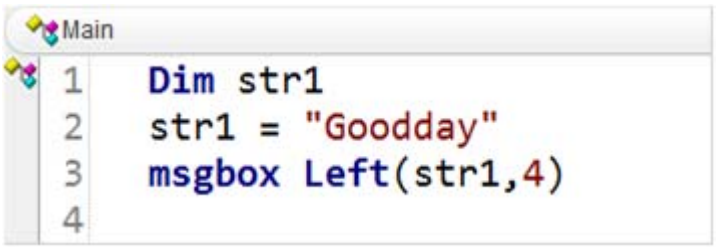
Various chopping functions are:

- Left
- Right
- Mid

Left

Concept	The `Left` function returns a specified number of characters from the left side of a string.
Syntax	`Left(string, length)` `string`: String expression from which the leftmost characters are to be fetched. `length`: Number indicating how many characters to return. If 0, s zero-length string("") is returned.

Implementation:



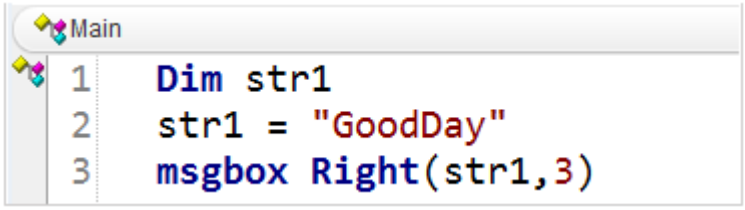
Output:

Good

Right

Concept	The `Right` function returns a specified number of characters from the right side of a string.
Syntax	`Right(string, length)` `string`: String expression from which the rightmost characters are fetched. `length`: Number indicating how many characters to return. If 0, a zero-length string("") is returned.

Implementation:



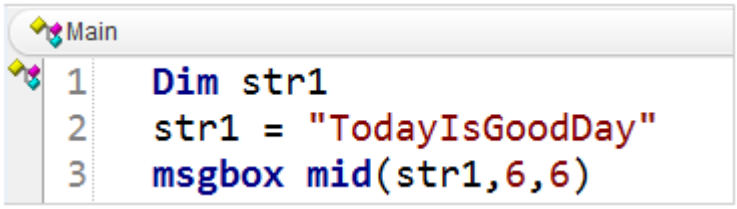
Output:

Day

Mid

Concept	The `Mid` function returns a specified number of characters from a string.
Syntax	`Mid(string, start, length)` `string`: String expression from which the characters are to be fetched. `start`: Character position in the string from which the part be taken begins. `Length`: Number indicating how many characters to return. This parameter is optional, if omitted, all characters from the start position to the end of string are returned.

Implementation:



Output:

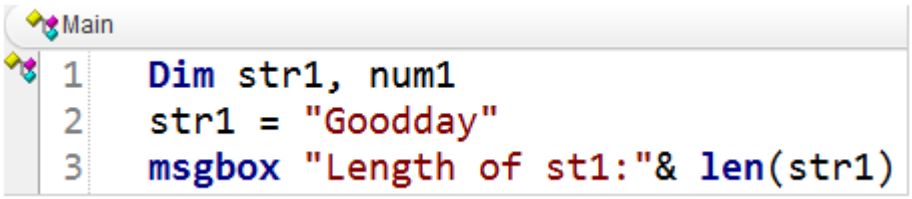
IsGood

String processing - Length and String Comparison

Len

Concept	The Len function returns the number of characters in a string.
Syntax	Len(string)

Implementation:



Output:

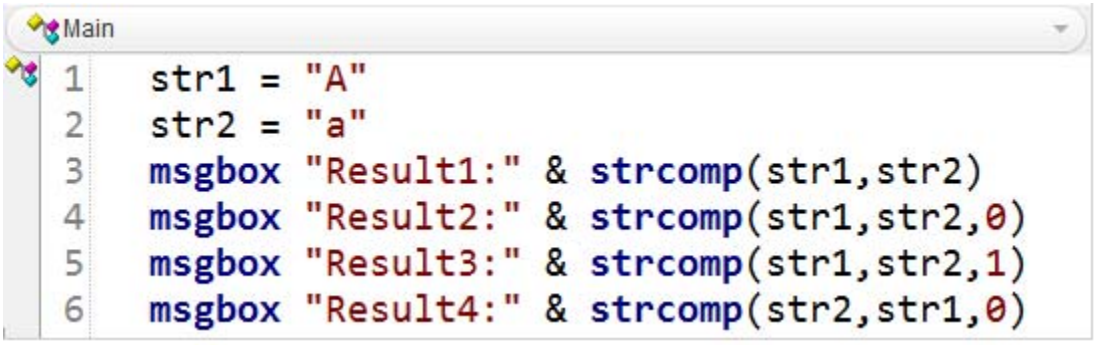
Length of str1:7

Strcomp

String Comparison function as the name suggest is used to compare two string values.

Concept	The StrComp function returns a value indicating the result of a string comparison.
Syntax	StrComp(string1, string2, compare) string1: Any valid string expression string2: Any valid string expression Compare: Numeric value indicating the kind of comparison to use when evaluating strings. It is **optional**. **0**: perform a binary comparison (**default**) **1**: perform a textual comparison

Implementation:



Output:

Result1:-1

Result2:-1

Result3: 0

Result4: 1

Explanation of the output:

- Result1 and Result2 is -1 because 3rd argument compare is default 0 in both the cases which is the binary mode and that compares str1 and str2 in terms of ASCII values. Since ASCII for “A” which is 65 will be less than ASCII for “a” which is 97, it will return -1.
- Result3 is 0 because for textual comparison “a”=”A”.
- Result4 is 1 because for binary comparison, ASCII for “a” is greater than ASCII for “A”.

VBScript - Conversion functions

Below mentioned are few of the conversion functions provided by vbscript, for converting data from one sub-type to another.

Various Conversion Functions:-

CStr - Convert to string

Concept	The `CStr` function returns an expression that has been converted to a string.
Syntax	`CStr(expression)` `expression`: Usually expression is a number to be converted to string.

Implementation:

```
Main
1   num1 = 56.79
2   str1 = "56.79"
3   if(num1) = str1 then
4     msgbox "true"
5     else
6     msgbox "false"
7   End if
8   If cstr(num1) = str1 Then
9       msgbox "true"
10      else
11      msgbox "false"
12  End If
```

Output:

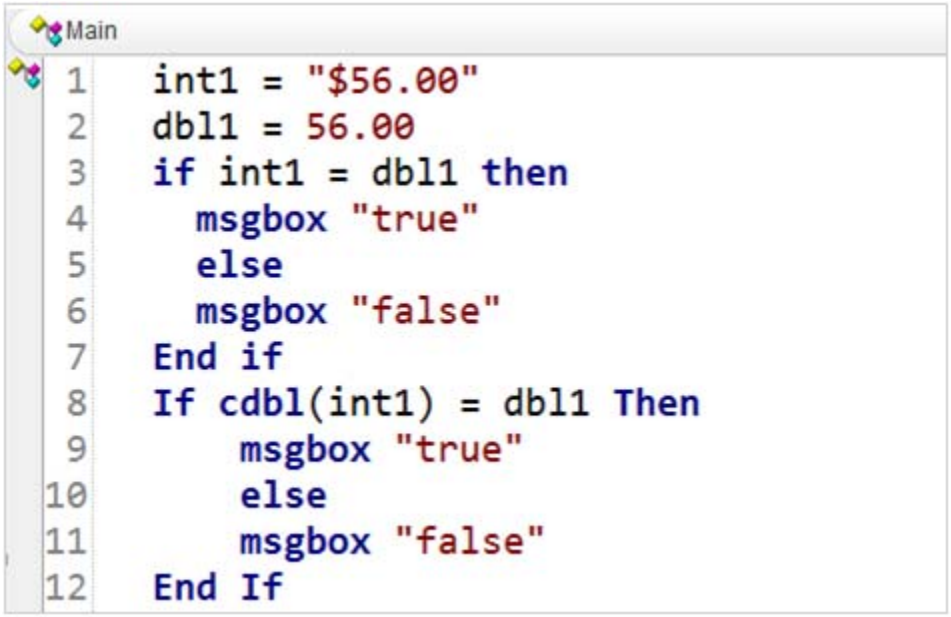
false

true

CDbl - Convert to Double

Concept	The CDbl function returns an expression that has been converted to a double.
Syntax	CDbl(expression) expression: Usually expression is a string/currency to be converted to double.

Implementation:



Output:

false

true

Cint - Convert to Integer

Concept	The CInt function returns an expression that has been converted to an integer.
Syntax	CInt(expression) expression: Usually expression is a double to be converted to an integer.

Implementation:

Main

```
1   int1 = 56
2   dbl1 = 56.44
3   if int1 = dbl1 then
4     msgbox "true"
5     else
6     msgbox "false"
7   End if
8   If int1 = cint(dbl1) Then
9       msgbox "true"
10      else
11      msgbox "false"
12  End If
```

Output:

false

true

CLng - Convert to Long

Concept	The CLng function returns an expression that has been converted to a long.
Syntax	CLng(expression) expression: Usually expression is a double to be converted to long.

Implementation:

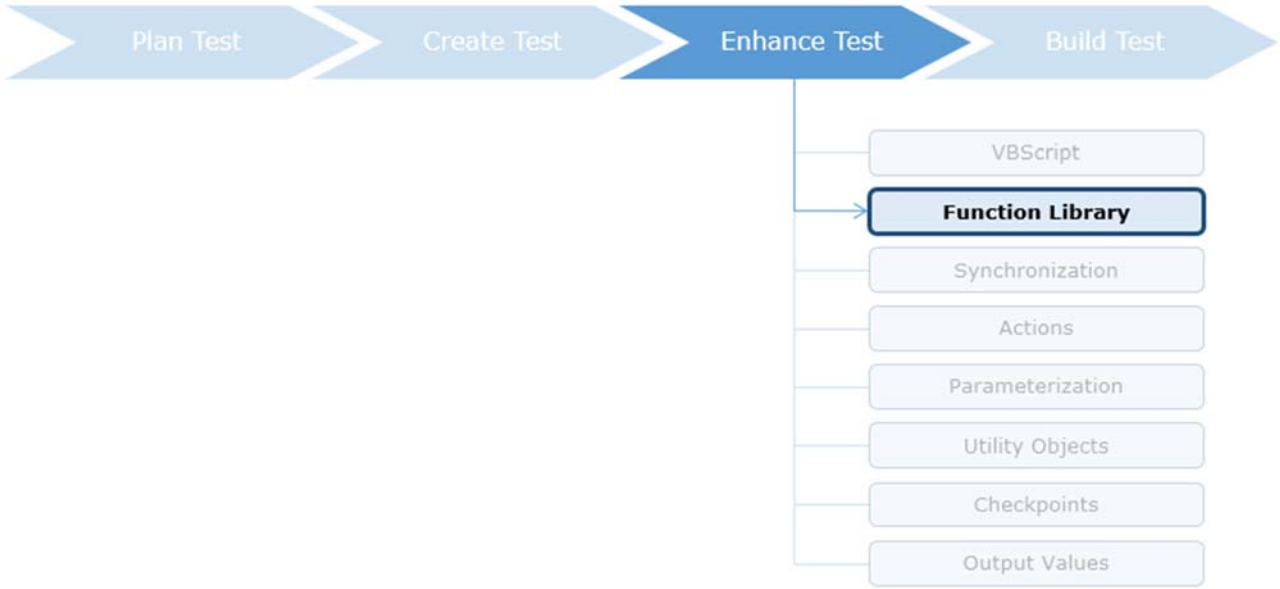
```
Main
1   lng1 = 24556
2   dbl1 = 24556.4436
3   if lng1 = dbl1 then
4     msgbox "true"
5     else
6     msgbox "false"
7   End if
8   If lng1 = clng(dbl1) Then
9       msgbox "true"
10      else
11      msgbox "false"
12  End If
```

Output:

false

true

Enhance Test - Function Library



Introduction

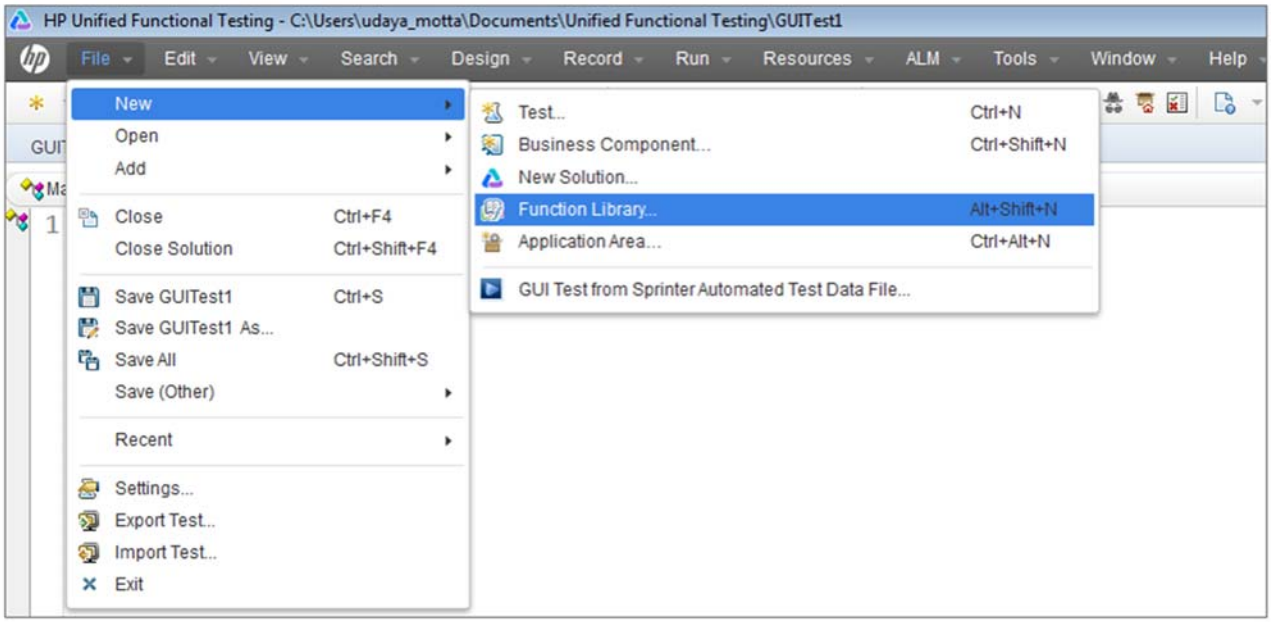
Through function libraries we can define our own vbscript functions, subroutines, statement and so on, and then call those functions through our test. A function library is a separate document that contains Visual Basic Script. Any text file written in standard VBS syntax can be used as a function library. There are two ways of associating/loading function library in a test:-

- Statically
- Dynamically

Statically

This is one of the way of associating a function library present on the filesystem. Below mentioned are the steps followed to associate it:-

- Create new function library.
    - **Navigate:-** File -> New -> Function Library



- Write the piece of code using vbscript in the function library directly or in a notepad. Save the file on the filesystem **(.vbs extension for Notepad & .qfl for function library file)**.

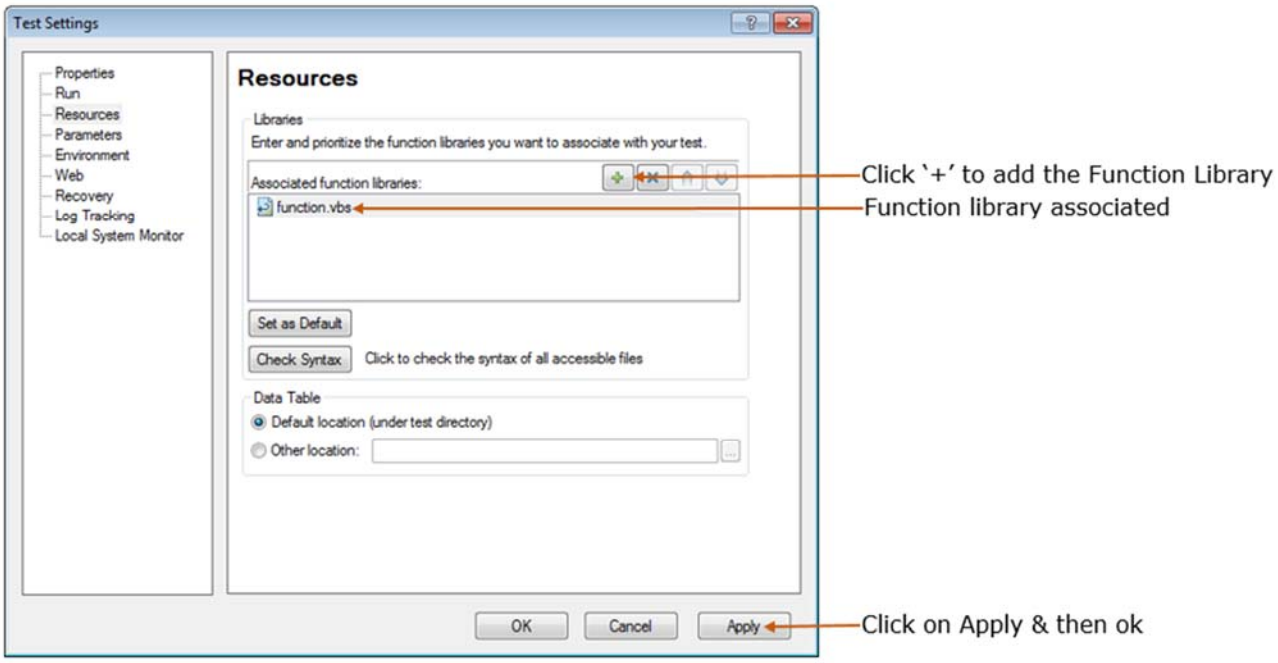
Example:-

```
Function add()
Dim a,b,c
a=20
b=10
c=cint(a) + cint(b)
```

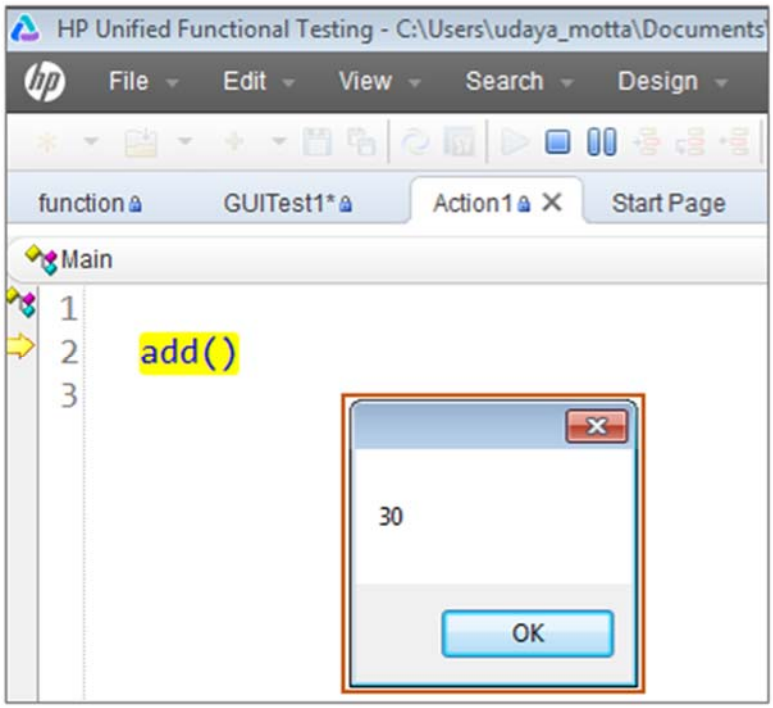
```
msgbox c
End Function

Function subt()
Dim a,b,c
a=20
b=10
c=cint(a) - cint(b)
msgbox c
End Function
```

- Associate the function library with the test.
    - **Navigate:-** File -> Settings -> Resources -> Click  -> Browse and select the function library file to be associated.



Output:-

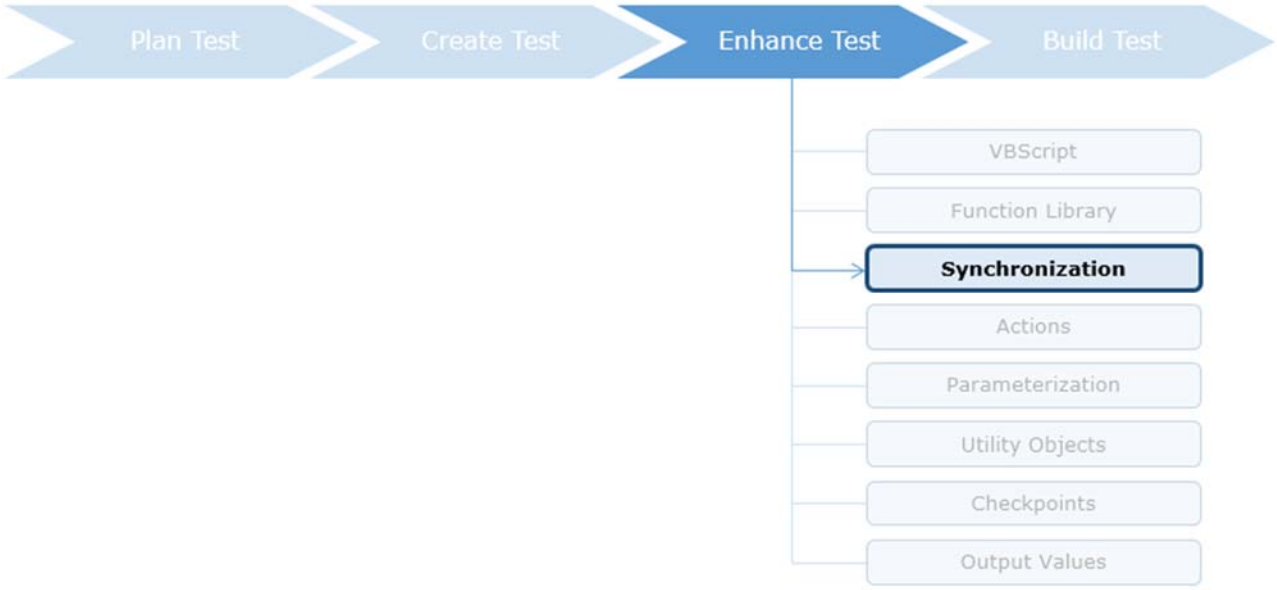


Dynamically

By loading the function library using **"loadfunctionlibrary"** statement. Below mentioned are the steps to be followed

- Create new function library:- **File -> New -> Function Library**.
- Write the piece of code using vbscript in function library.
- Save the function library on the filesystem.
- Open the Action in your test. Write the statement
  - **loadfunctionlibrary <<<Function library file with complete Path >>>**

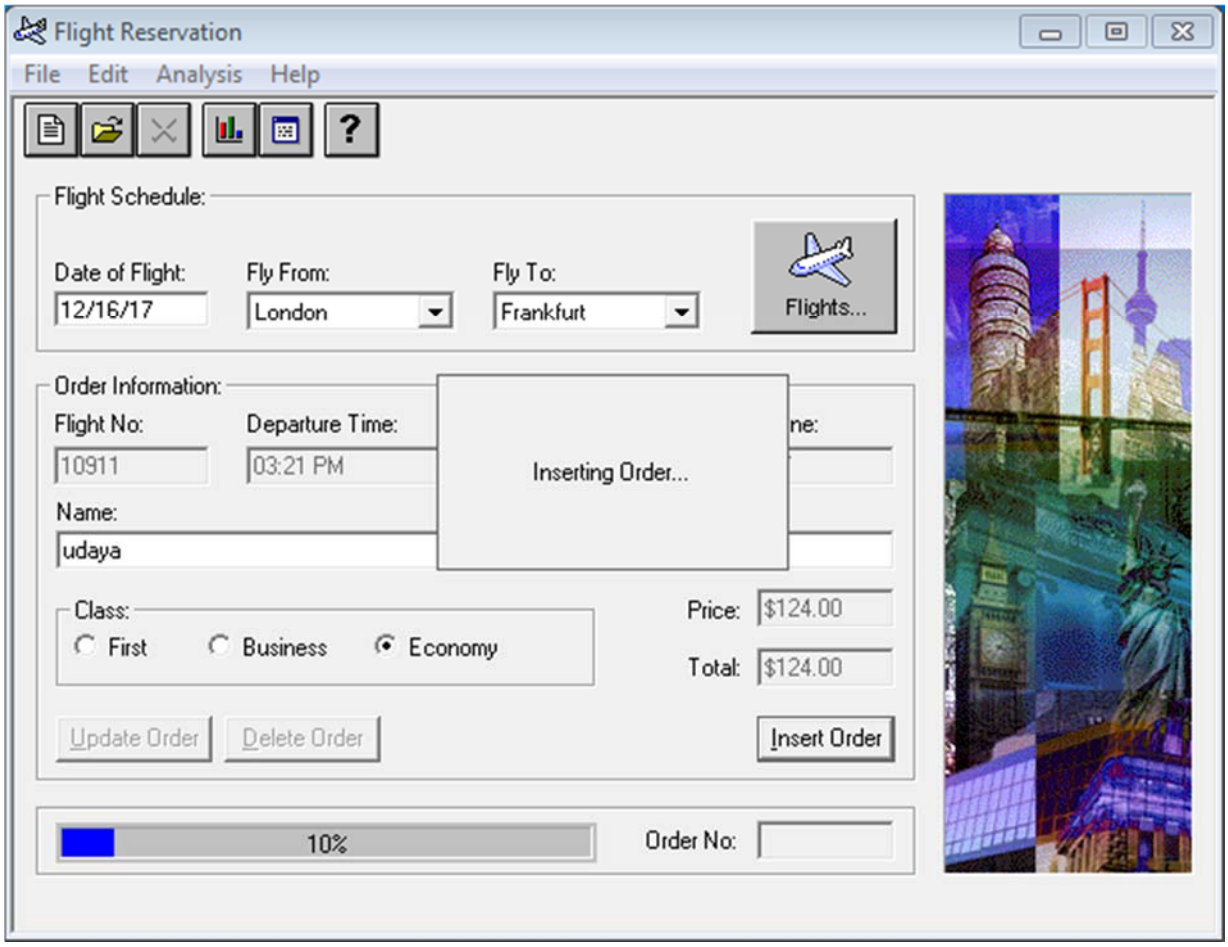
Enhance Test – Synchronization



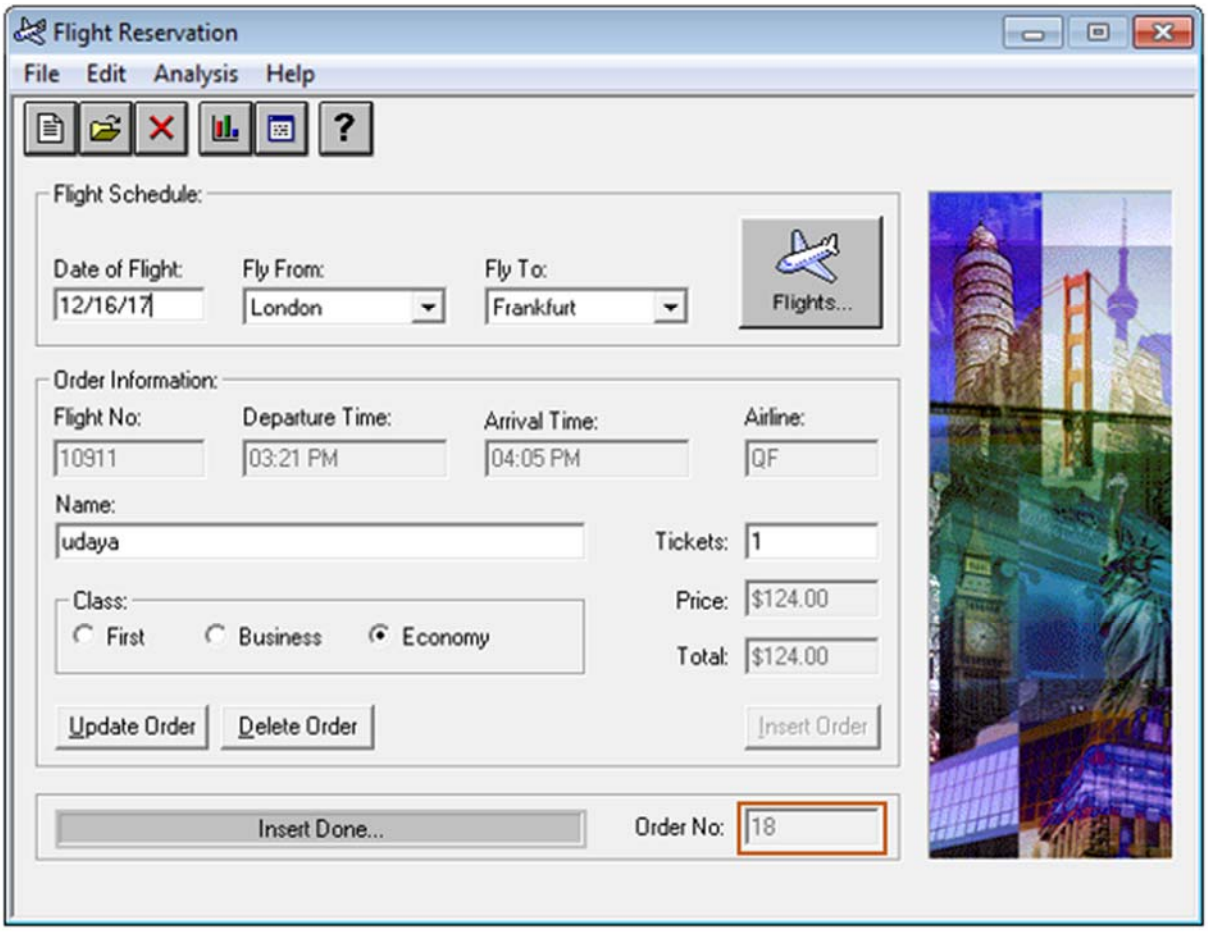
In real time situation whenever a script executes there is a potential failure that arises because of the mismatch in the execution speed between the script and AUT. This issue is solved using the concept of synchronization.

Let us understand this concept with the help of a situation:-

- A user wants to book a ticket on a “Flight Reservation” application. He selects the date, source city,destination city and selects the needed flight.
- He navigates to insert order button and clicks on it, the processing of the order takes few seconds of time before displaying the order number. We need to automate this scenario using UFT.



- The order number that gets displayed need to be captured using GetRoProperty.



- If we record this scenario and play back the script, UFT will capture the order number as soon as the Insert order button is clicked and blank value will be fetched. Reason behind this is beacuse UFT is running ahead of the application in terms of execution speed.
- This time lag can be handled by implementing **synchronization**.

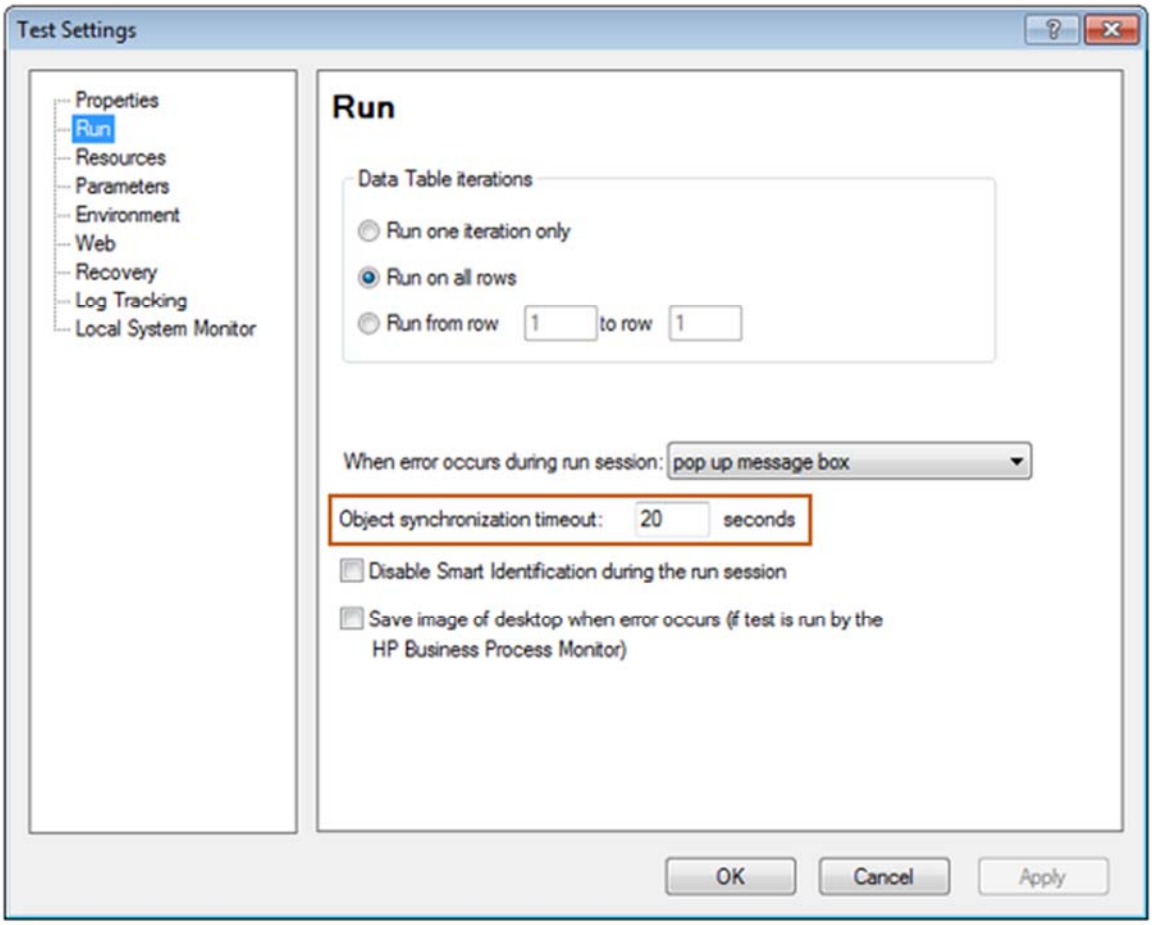
Synchronization can be achieved through:

- Object synchronization timeout
- Wait
- WaitProperty

Object Synchronization Timeout

- By changing the time out value in :- File -> Settings -> Run Tab (Default value is 20secs). **Note:-** UFT will wait for maximum 20Secs for each and every line in the script

- The drawback is it effects all VB-script statements in the script.



Wait

- By using Wait statement.
    - **Syntax:- Wait (x)** x -> time out value in seconds
- The drawback is the script waits the amount of time specified in the wait statement even though the application has completed its task.

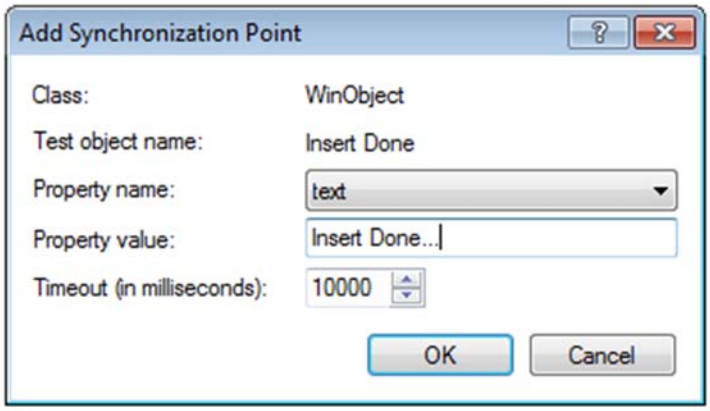
WaitProperty

- “WaitProperty” method is a line in the test script that instructs UFT to wait intelligently for a certain response from the application during playback.
- When a synchronization point is used, a **“WaitProperty”** step is inserted into the script
- Ensures the test runs only when the AUT is ready to continue
- It can synchronize:

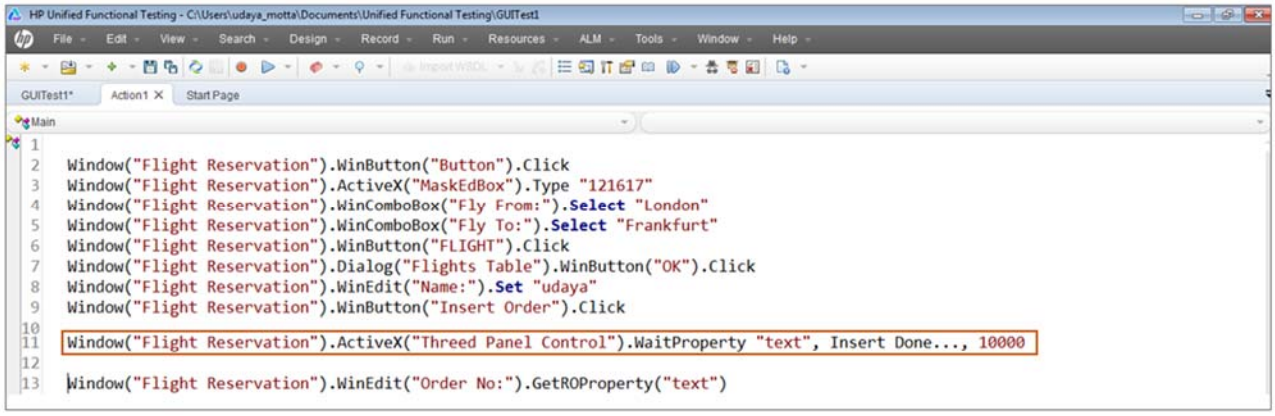
- For a specific amount of time
- Until an object state achieves a certain value
- Response time can be set as a controlling

How to add synchronization point:-

- Navigate to the required step in the script
- In UFT choose Design > Synchronization point during a recording session
- Using the cursor click on the object to synchronize
- Use the Synchronization point creation dialog box

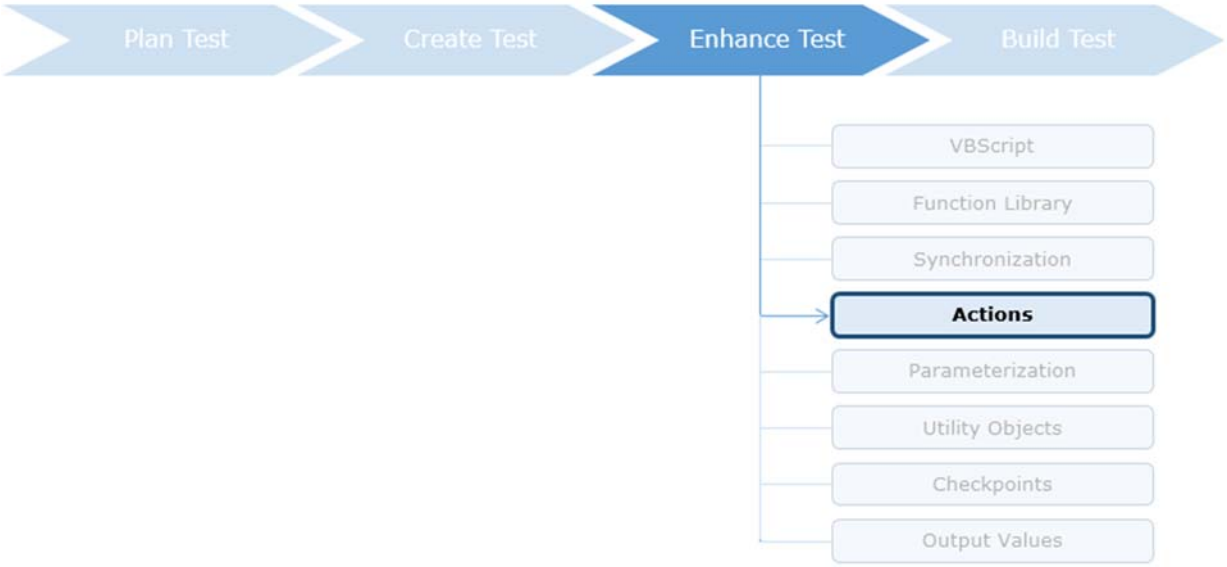


- Select the property name and its corresponding value which you want to use for the synchronization point
- Enter the synchronization point timeout (in milliseconds) and Click OK. A Wait property step is added to your test. **Example**:- For above discussed scenario we are taking **"text"** as the property to achieve the synchronization.



- Total time of script execution :-
    - Time parameter as set in the script + Object synchronization timeout value(from Test Settings)

Enhance Test – Actions



This is one of the important concept in UFT as it will help the user to create the modular test for simplified maintenance.

What are Actions

- Actions help to divide the test into logical sections/units. Breaking up the tests into multiple actions makes the tests modular and efficient.
- An action has its own test script containing all of the steps recorded in that action and all objects in its local object repository.

- When a test is created it includes one action by default. All the steps that are recorded and all the modifications that are made while editing the test are part of that action.

There are 4 types of actions that are available in UFT:-

- **Reusable action**

  An action that can be called multiple times by the test in which it was created (the local test) and by other tests is a reusable action. **By default, new actions are reusable**.

- **Non-reusable action**

  Non-reusable action is an action that can be used only in the test in which it was created.

- **External action**

  A reusable action stored with another test.

- **Nested action**

  An action calling another action.

Calling Action

Let us assume we want to test the important functionalities of a "Flight Reservation “application i.e., booking a Flight, opening the order details. Each time the user has to log into the application and log out to test these functionalities. This scenario has to be automated using UFT.

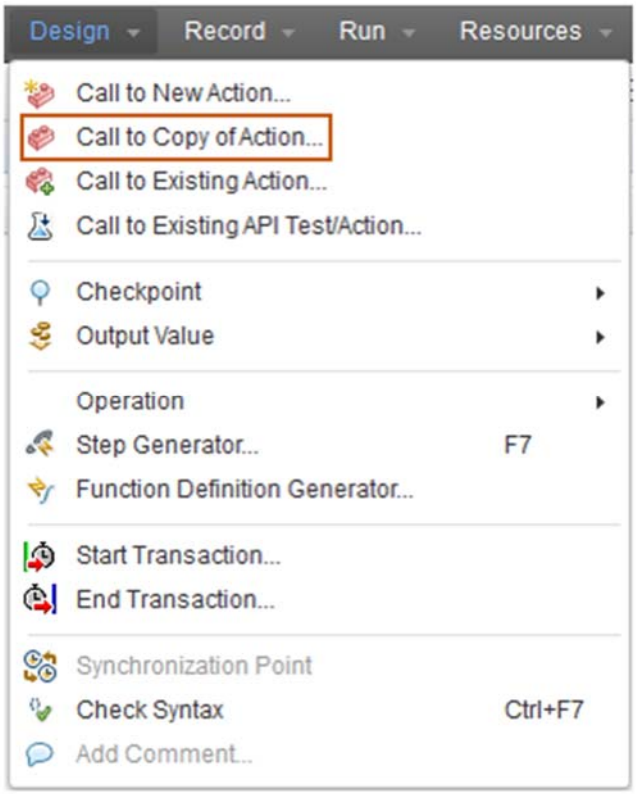
- While planning the tests for business processes, we realize that each test requires the same login and logout steps. In order to increase reusability, we can create one action that contains the steps required for the login process and another for the logout steps.
- These login and logout actions can be called by multiple tests, thus reducing the redundancy in creating the actions.

So as to attain this, we can call actions in two ways: -

- Call to copy of an action
- Call to an existing action

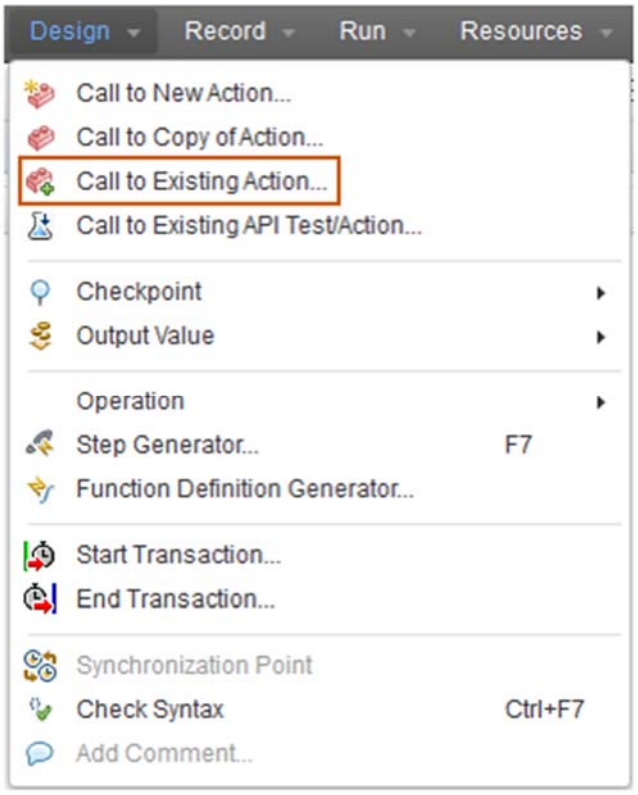
Call to copy of an action

- When copy of an action is called into a test, the action is copied in its entirety, including checkpoints, parameterization, and the corresponding action tab in the Data Table.
- Modifications can be made into the copied action according to the requirements. These changes will not affect nor be affected by other test.

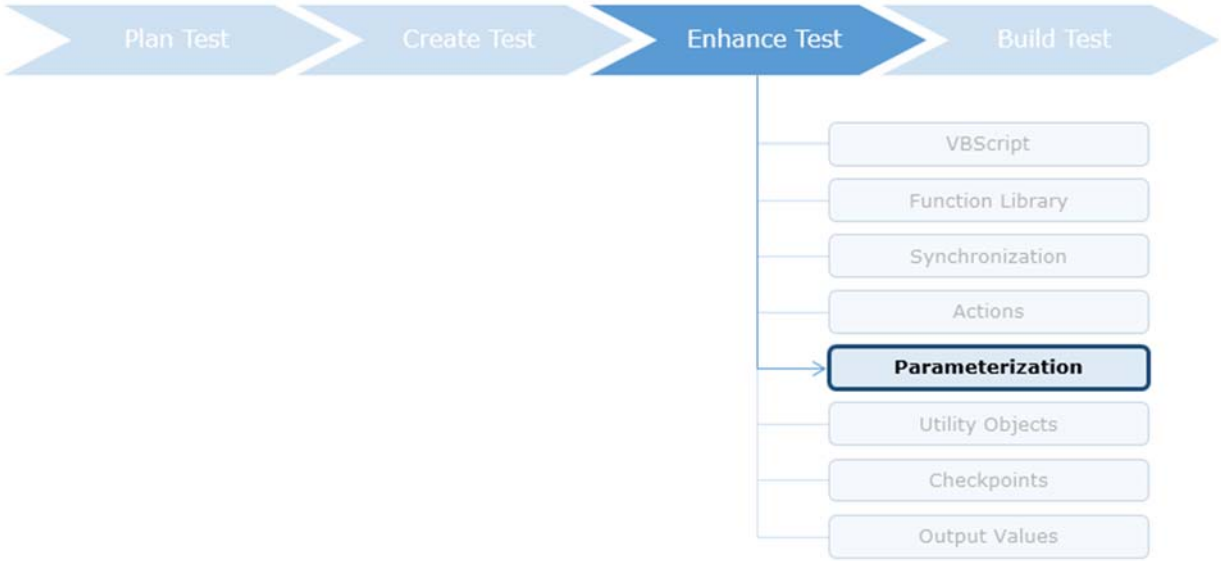


Call to an existing action

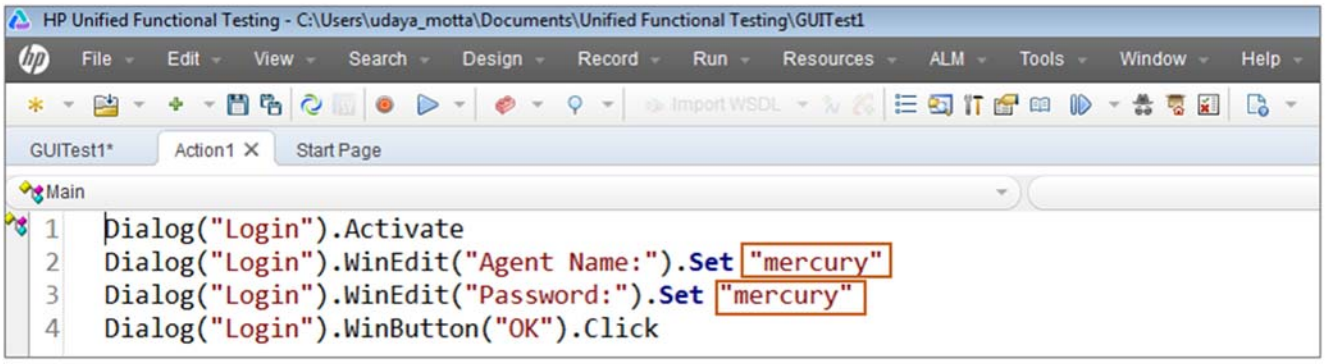
- When an existing action is called from the test, it is in read-only mode. This action can be modified only in the test in which it was created.
- Inserting calls to existing actions helps to use the same action in several tests and makes the maintenance of tests easier.
- Only reusable actions can be called into a test.



Enhance Test – Parameterization



Till now whatever script we have generated has the test data hard coded in it. For example:



Parameterization feature of UFT is used to test a particular operation of the application under test (AUT) with multiple set of values.

Note:- Consider a scenario where the login functionality of "Flight reservation" application has to be tested. The user has to enter the valid username and password values and click on the the login button in order to launch the book of flight page. We can generate a test 'Test01' to check the login functionality. But the script would have the hard coded value of the username and password that were entered during the time of recording the script. We can replace these hard coded values by making the 'Username' and 'Password' fields as **parameters**.

Converting a constant value in the script to a parameter is called **parameterization**.

- Parameterization allows to test the application functionality with different sets of input data. There by it reduces the script maintenance effort.

The different parameter types available in UFT are:

- DataTable
- Environment Variable
- Random Number
- Test / Action parameters

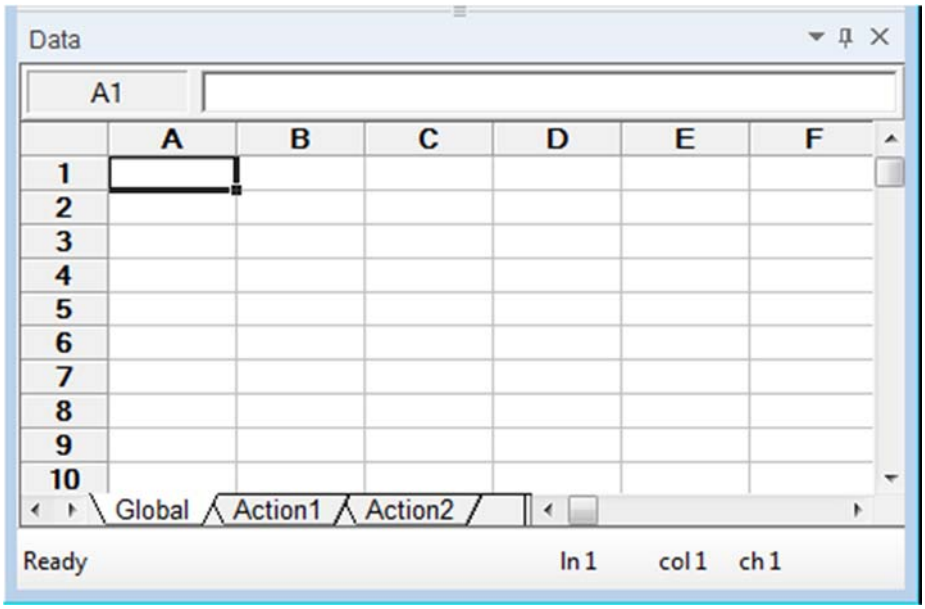
Parameterization – DataTable

What is DataTable?

Data Table is a spreadsheet-like sheet with columns and rows. It contains the data that are applicable to the functional test.

A Date Table contains **two types of data sheets:**

- **Global**
  - Used to store data that is available to all the actions in the test
  - It has the sheet Id as 1
- **Action**
  - Used to store data that is available to only a particular action in the test
  - Action1 has the sheet Id=2 and the other Action sheets that follow would take sheet ids in the sequence.



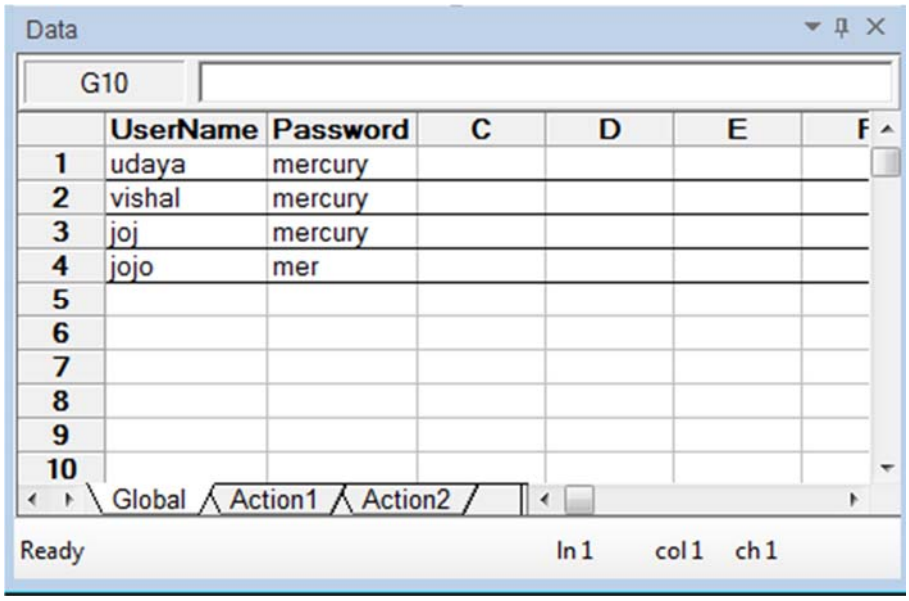
Note: The data table contains one Global tab and additional tab for each action of the test.

Types of DataTable

Data Table can be categorized as:

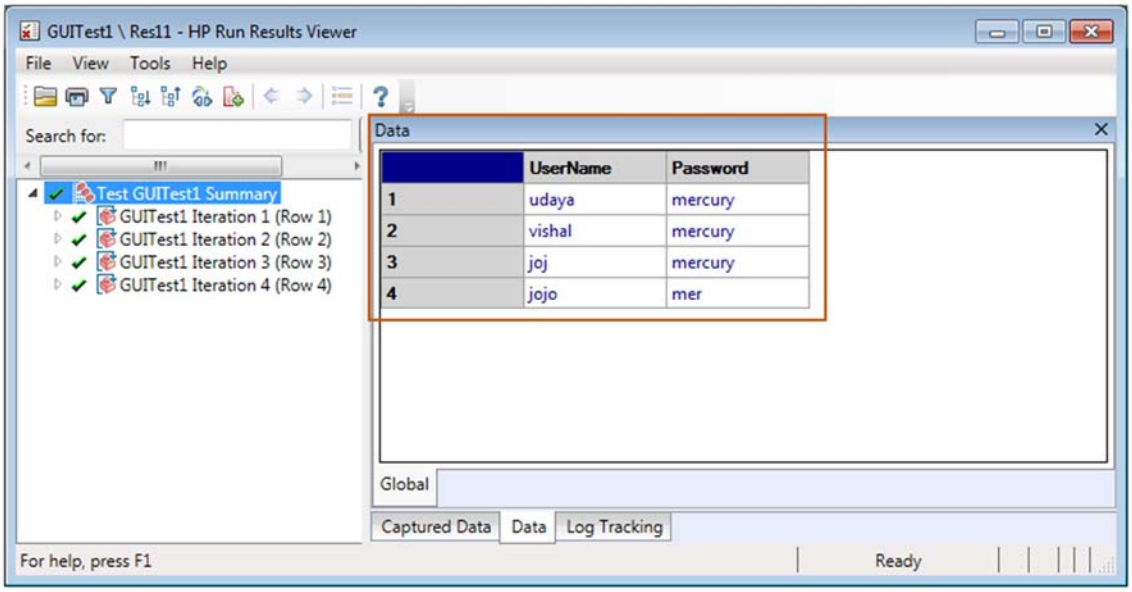
Design-Time Data Table

- The data applicable to the tests are stored here
- This is displayed in the ‘Data’ pane as shown in figure below
- The values in this data table can be modified without executing the script



Run-Time Data Table

- It is a live version of the data table associated with your test, i.e. it is the data table that is used during the run time of your test
- This is displayed in the 'Data' pane in the Run Result Viewer as shown in figure below
- The values in this data table cannot be modified statically but can be modified during run-time only



DataTable Methods

Below are few of the properties/methods that we can use with run-time instance of datatable object:-

Method	Description
AddSheet	This method adds the specified sheet to the run-time data table. **Syntax:** DataTable.AddSheet("Name of Sheet")
Import	This method imports the entire workbook into the run-time data table. **Syntax:** DataTable.Import "Path of the file"
ImportSheet	This method imports a specified sheet of the workbook in the run-time data table. **Syntax:** DataTable.ImportSheet "Path of the file", SourceSheet Id, DestinationSheet Id
GetSheetCount	This method returns the total number of sheets available in run-time data table. **Syntax:** DataTable.GetSheetCount
GetSheet	This method returns the specified sheet from the run-time data table **Syntax:** DataTable.GetSheet(SheetNumber)
GetRowCount	This method returns the total number of rows in specified sheet of the run-time data table **Syntax:** DataTable.GetRowCount
GetCurrentRow	This method returns the active row in the first sheet of the run-time data table i.e. global sheet **Syntax:** DataTable.GetCurrentRow
SetCurrentRow	This method sets the specified row number as the active row in the first sheet of the run-time data table i.e. global sheet **Syntax:** DataTable.SetCurrentRow(NumberOfRow)
SetNextRow	This method sets the row after the current (active) row as the active row in the first sheet of the run-time data table i.e. global sheet **Syntax:** DataTable.SetNextRow
SetPrevRow	This method sets the row before the current (active) row as the active row in the first sheet of the run-time data table i.e. global sheet **Syntax:** DataTable.SetPrevRow
Export	This method exports the run-time data table in the specified file. **Syntax:** DataTable.Export "Path of the file"
ExportSheet	This method exports a specified sheet of the run-time data table to a specified file. **Syntax:** DataTable.ExportSheet "Path of the file",Source SheetNumber
DeleteSheet	This method deletes the specified sheet from the run-time data table. **Syntax:** DataTable.DeleteSheet(SheetNumber)

DataSheet Methods

Below listed are few of the commonly used methods of data sheet:-

Method	Description
AddParameter	This method is used to add a new column to any sheet in a run-time data table. **Syntax:** DataTable.GetSheet(SheetId).AddParameter "NewColumnName","Row1Value" DataTable.AddSheet("Name of Sheet").AddParameter "NewColumnName","Row1Value" **Example:** Script wants to add a new column/parameter named "TrackName" in the global sheet. DataTable.GetSheet(1).AddParameter"TrackName","IVS"
GetParameter	This method is used to retrieve the specified parameter from the run-time data table. In order to get a particular value in the parameter, the property '**value**' can be used. **Syntax:** DataTable.GetSheet(SheetId).GetParameter "ColumnName" DataTable.GetSheet(SheetId).GetParameter ("ColumnName").value **Example:** Script wants to retrieve the value at the first row of "colname" parameter Msgbox DataTable.GetSheet(3).GetParameter("colname").value
GetParameterCount	This method is used to return the total number of parameters in the run-time data table sheet. **Syntax:** DataTable.GetSheet(SheetId).GetParameterCount 'OR' DataTable.GetSheet("SheetName").GetParameterCount **Example:** Script wants to fetch the total columns in a global sheet Msgbox DataTable. GetSheet(1).GetParameterCount
DeleteParameter	This method is used to delete the specified parameter from the sheet in the run-time data table. **Syntax :** DataTable.GetSheet(SheetId).DeleteParameter"Name of the column to delete" **Example:** Script wants to delete a parameter named "TrackStrength" in the global sheet. DataTable.GetSheet(1).DeleteParameter"TrackStrength"

Test and Action Iterations

Iterations also known as repetitions, can be managed for Test and Actions. Iterations for either of these objects are defined on the basis of number of active rows that are available on Global or Local Sheet.

Iteration settings can be set from Test settings or Action call properties for either Test or Action respectively.

The three iteration options are

- Run one iteration only
- Run on all rows
- Run from row __ to row ___

In order to modify Test iterations:-

- Go to **File -> Settings -> Run** tab
- Select the required options present under DataTable iterations settings -> which has got following options:
    - **Run one iteration only** – which executes the scripts by taking the data from the 1st row of the datatable
    - **Run on all rows** – which executes the script by taking the data present in all the rows of the datatable
    - **Run from row – to row** –which executes the script by taking the data present in all the rows of the datatable

Note:- In run from **row .. to row** – starting range should be less than or equal to ending range

In order to modify Action iterations:-

- Go to Canvas view
- Right click the action whose iterations you want to change
- Select "Action Call Properties "
- Select the required option under "Data Table Iterations"

Note:- Active rows in global sheet defines the number of iterations for the test whereas active rows in local sheet defines the number of iterations for the respective action.

Parameterization - Environment Variable

What is Environment Variable?

Environment variables in UFT can be compared with global variables in other programming languages. It can be accessed through any part of the script. The values of these variables remain same unless changed through scripting. It is very useful when we need to use same variable across different reusable actions.

Types of Environment Variables

Environment variables are of two types:

1. Built-In

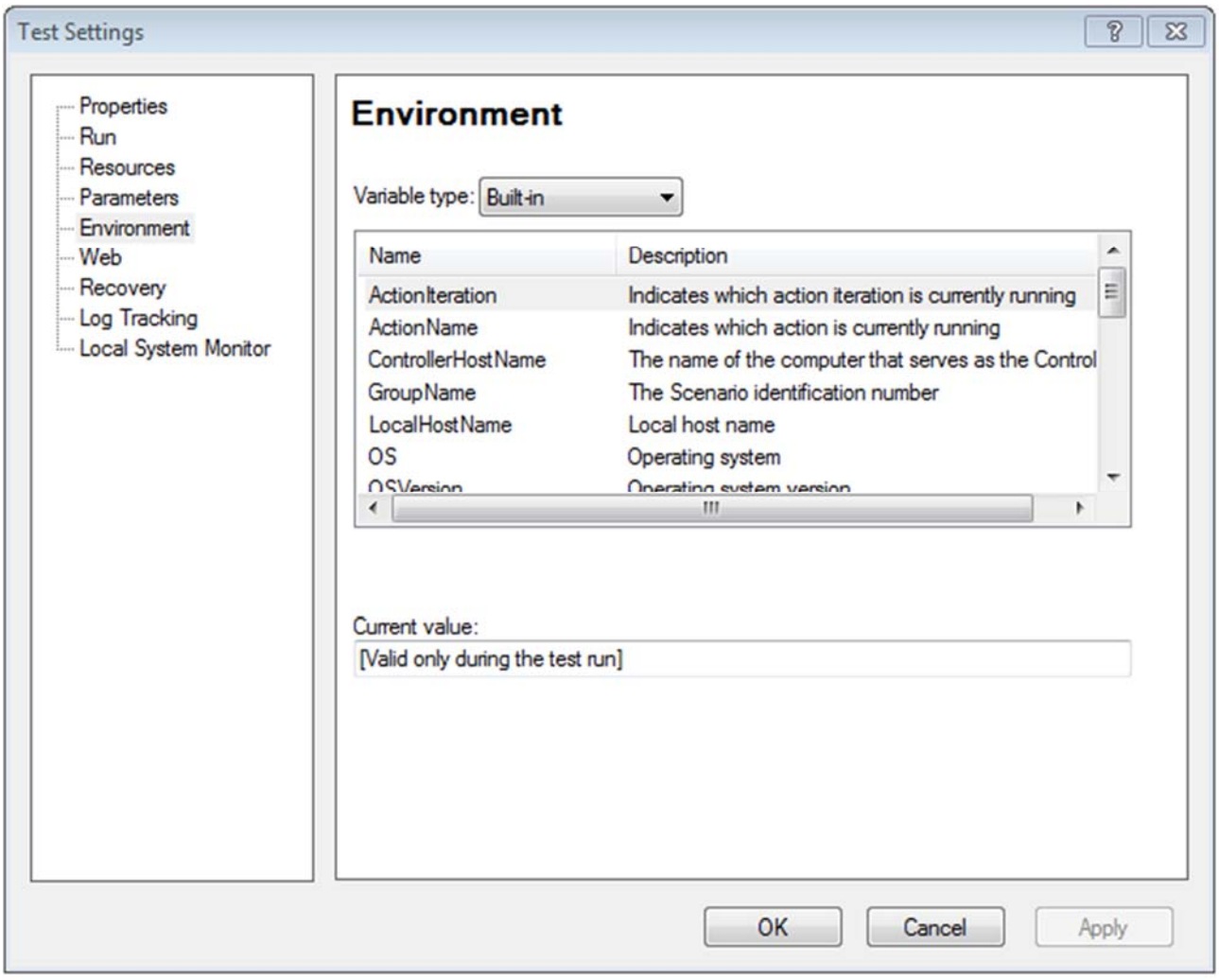
2. User-defined

Built-in

As the name suggests these are the set of predefined variables present in the UFT. These built-in environment variables provide various useful information like System name, OS version, Path of the test etc.

Syntax: strTestName = Environment(“TestName”)

Navigate:- File -> Settings -> Environment



User-defined

Environment variables can also be declared by the users. These variables can be used in the scripting throughout various actions. These variables are local to the test in which it is defined.

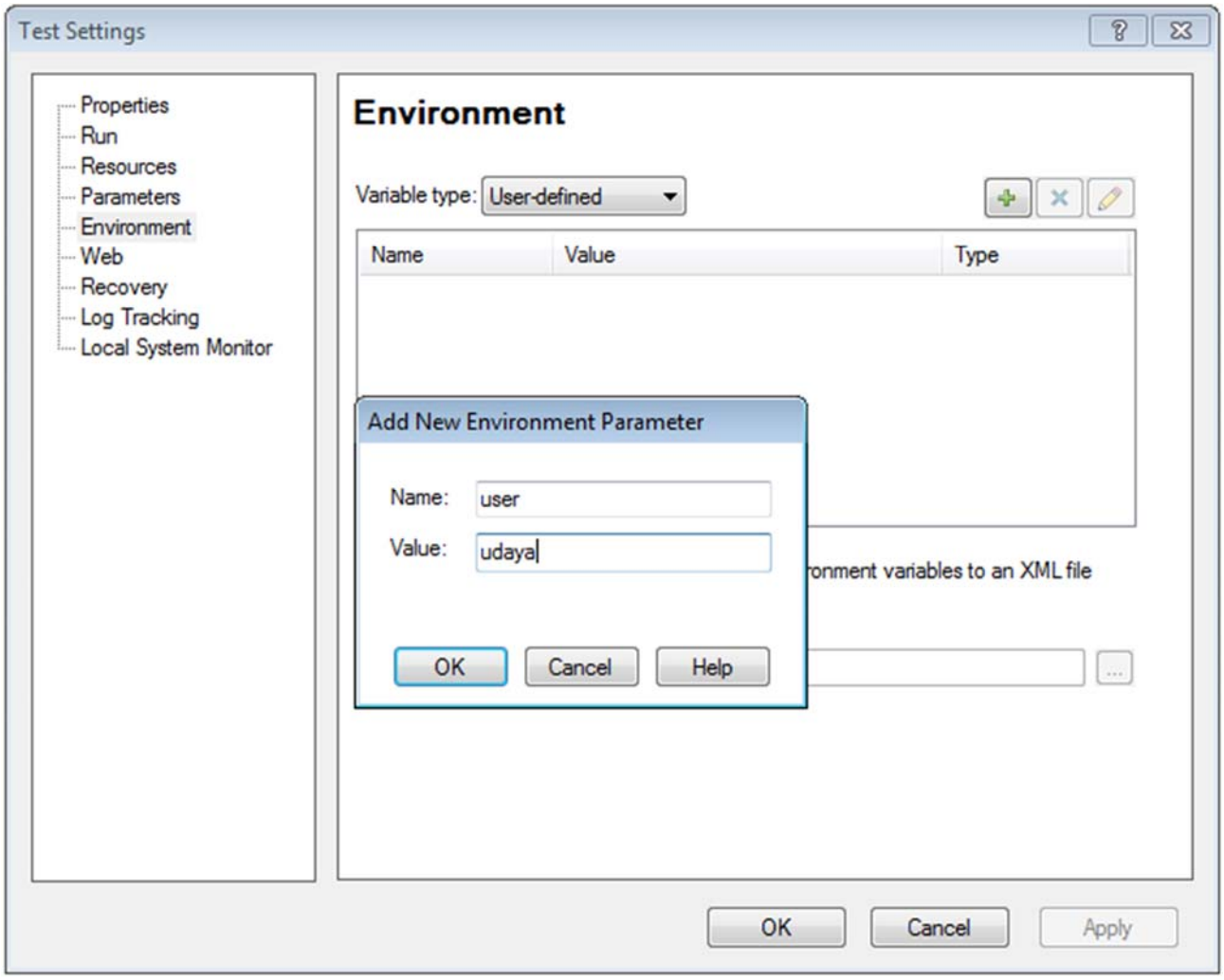
They are of 2 types:-

1. **Internal** : The variables and values are defined by user.
2. **External** : The variable values are fetched from an external xml file.

Steps to create user-defined internal variable -

- **Navigate -** File→ Settings → Environment Tab
- Select User-Defined from the drop down
- Click  button.
- Enter the name and assign a value to the variable
- We access the defined variable anywhere in the script by using the below syntax :-

  Environment.value(“Name_of_the variable”)



Steps to create user-defined external variable -

- Sometimes we need to have the entire list of environment variables during testing, then user has an option of creating them externally and associating it to the test.
- Create an XML file with the list of environment variables in below format

<Environment>

<Variable>

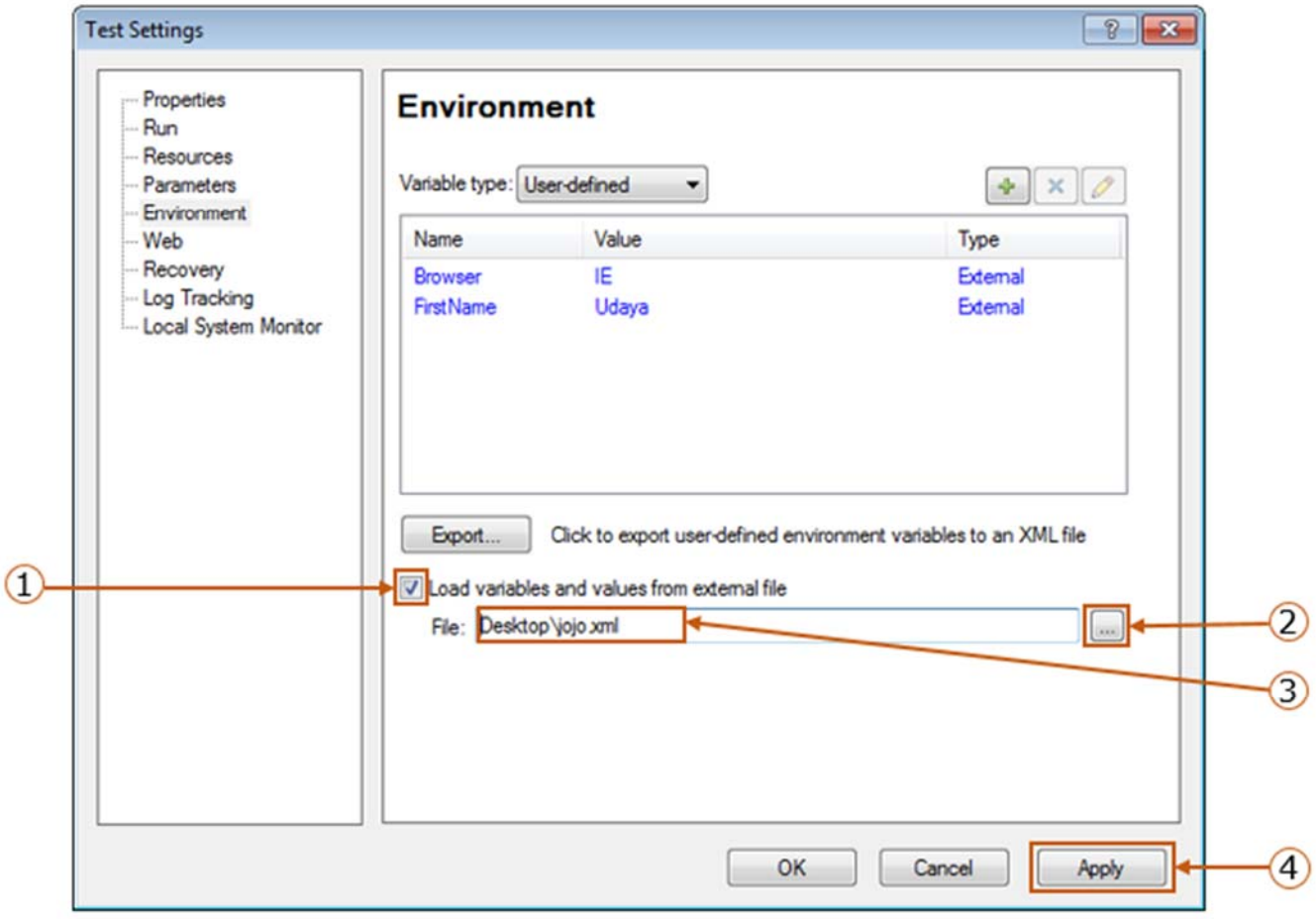
<Name> Browser </Name>

<Value> IE </Value>

<Name> First Name </Name>

<Value > Udaya </Value>

- Follow the remaining steps as shown in the image below:-



Step 1 : Enable the checkbox Load variables and values from external file

Step 2 : Click on browse button and select the target xml file

Step 3 : Selected file will get displayed

Step 4 : Click on Apply and OK

Parameterization - Random Number

So far we have seen two different methods of achieving the parameterization which will help to parameterize different types of data like String values, Numeric values and Alpha-Numeric values. Sometimes we would be only in need of the numeric data to check the application functionalities, let's see how to implement it by using random number.

What is Random Number

Enables you to insert random numbers as values in your test.

For example: To check how your application handles small and large ticket orders, you can instruct UFT to generate a random number and insert it in a number of tickets edit box.

How to create Random number

You can generate a value for the specified random number parameter using the RandomNumber object.

Syntax: RandomNumber (ParameterNameOrStartNumber, [EndNumber])

ParameterName

The name of a random number parameter that has already been defined in the Parameter Options or Value Configuration Options dialog box

StartNumber

The start number for the range within which the random number is generated.

EndNumber

- Number Relevant only when you specify a start number for the first argument.
- The end number for the range within which the random number is generated. Maximum allowable value is 2147483647.

Example

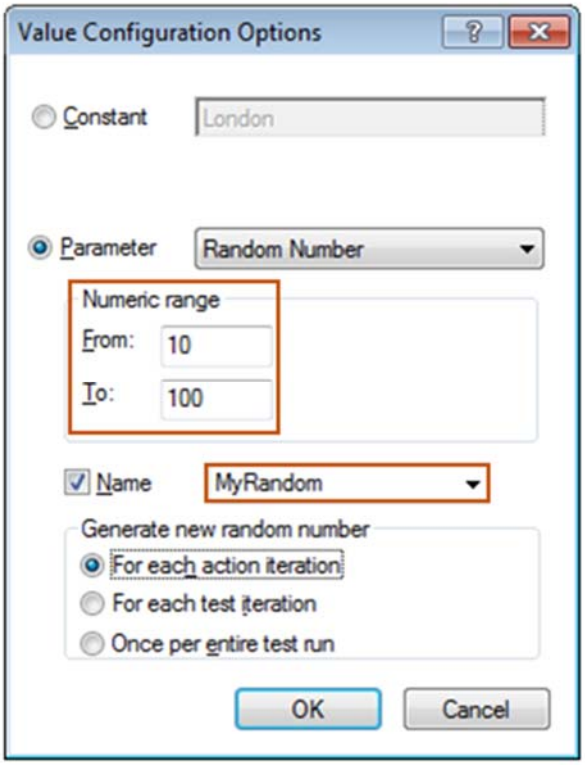
- **Syntax1** – For generating the random numbers between 0 and 100.

  x=RandomNumber (0,100)

- **Syntax2** – For generating the random numbers between 0 and 100 using “MyRandom” parameter name.

  x=RandomNumber ("MyRandom")

- Goto View—>Keyword View
- Click on the value column and select the parameter option as Random number
- Now give the from and to limit and give the name for the random number.



Note: - The above example works only if you have already defined a Random Number parameter called MyRandom in the Parameter Options or Value Configuration Options dialog box.

Parameterization - Input/Output Parameters

What is Input/Output Parameters

Parameter Object i.e. Input/Output parameters are used to parameterize a step in a test script.

They could be:

- Value(s) passed on while calling the action/test (Input parameters)
- Return value from an action/test (Output parameters) that can be used later in the test.

Input/Output Parameters are of two types:-

- Action input/output parameter
- Test input/output parameter

Steps to create Action Input/output Parameters.

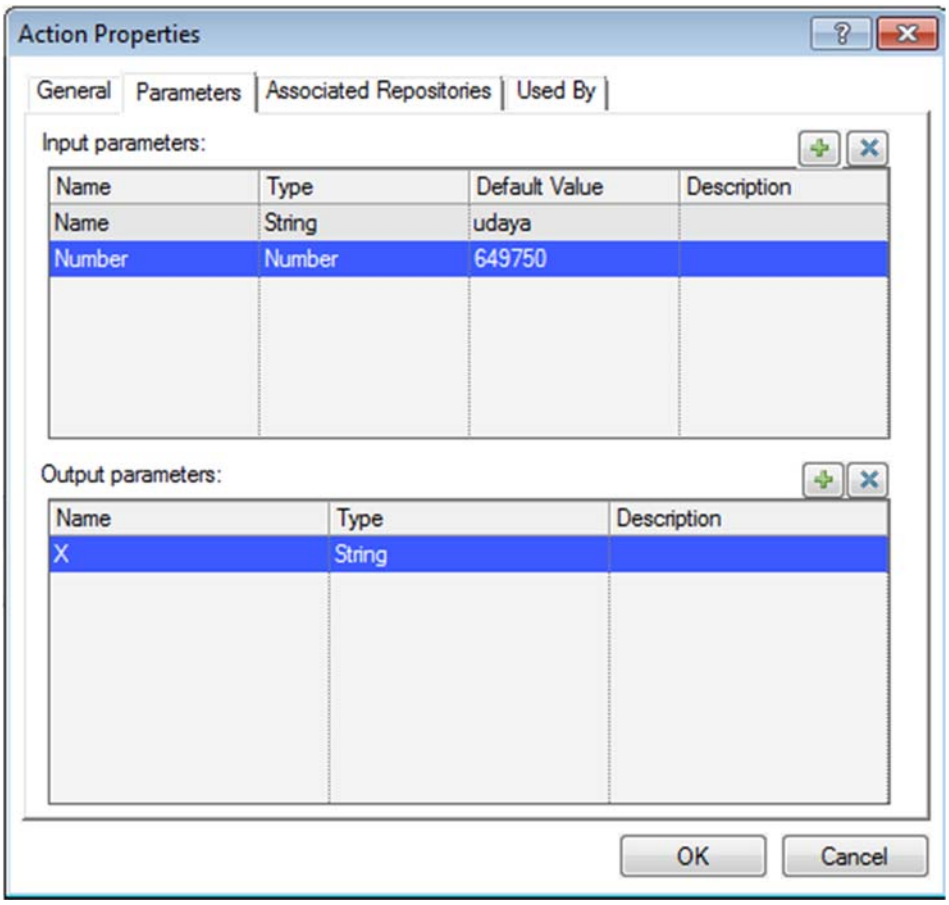
- Open a new Test
- Navigate to the Keyword view -> Right Click on Action1
- Choose Action properties
- Go to parameter tab and create input variable as shown in the fig
- In the Expert View of the Action 1 type:

  s1 = parameter (“Name”)

  s2 = parameter (“Number”)

  Parameter (“X") =s1+s2

  Note:- Here s1, s2 are variables. 'Name', 'Number' are the input parameters and 'X' is the output parameter.

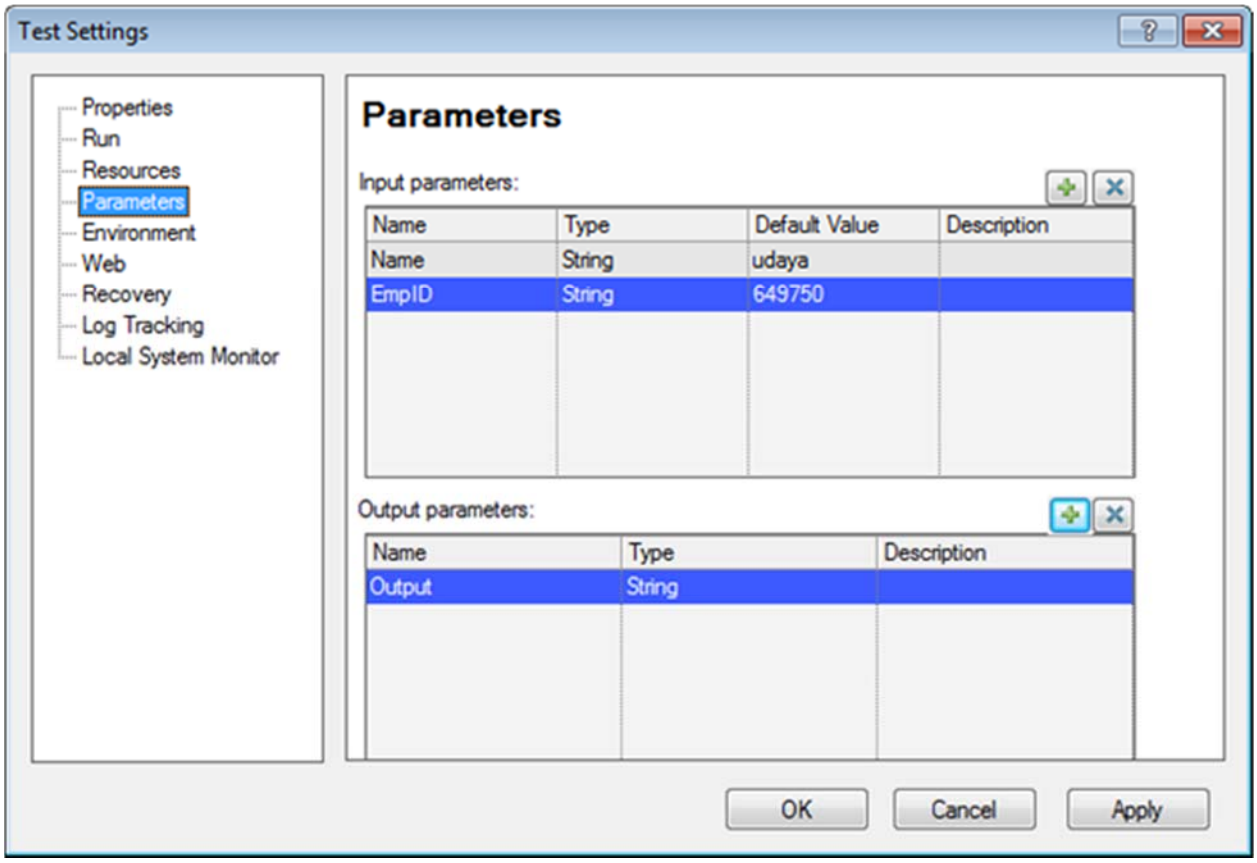


Steps to create Test Input/output Parameters

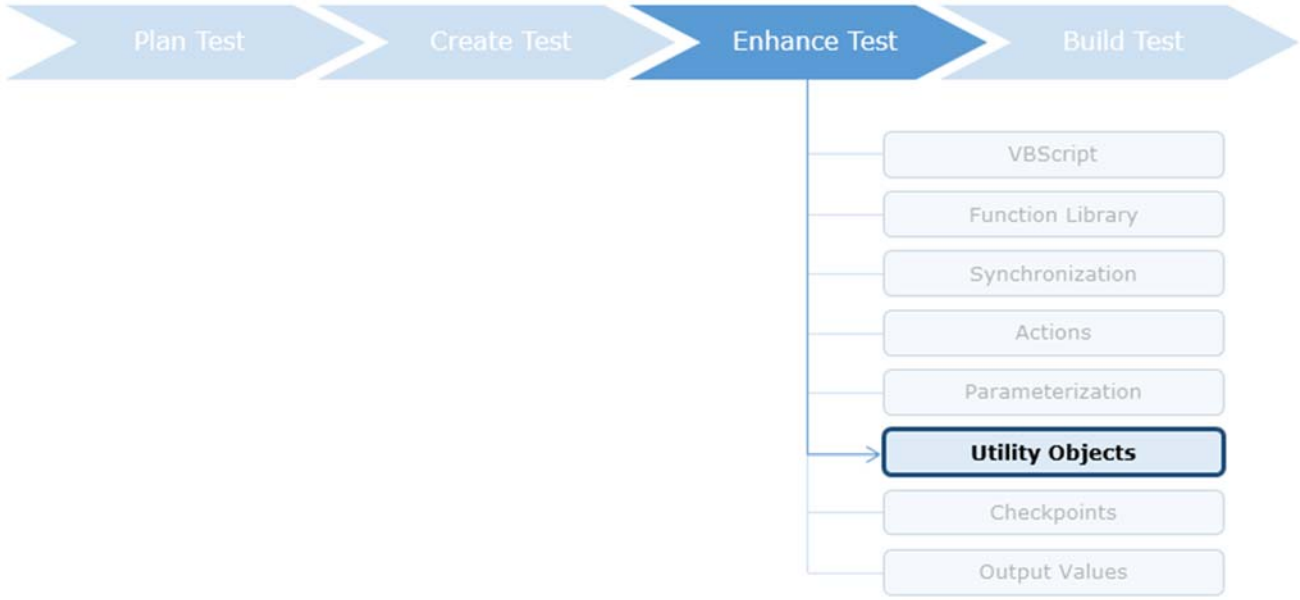
To Access the Test Parameter we use **"TestArgs"** just as we use Parameter to access Action Parameters.

- Open a new Test
- Navigate to File -> Settings -> Parameters
- Create the Input and Output parameter as shown in the image below
- In the Expert View of the Action 1 type:

  TestArgs(“output”) = TestArgs(“Name”)+ TestArgs(“EmpID”)



Enhance Test - Utility Objects



What is Utility Object?

- UFT provides utility Objects to enhance the power of Scripting
- Utility Objects are the reserved objects in UFT .These objects can be used in reporting preferences during run time.

Types of utility Object?

Below are the major type of utility objects used in UFT:-

- Crypt Object
- SystemUtil.Run
- SystemUtil.Close
    - SystemUtil.CloseProcessByName
    - SystemUtil.CloseProcessByWndTitle
    - SystemUtil.CloseProcessByID
    - SystemUtil.CloseProcessByHwnd
    - SystemUtil.CloseDescendentProcess
- Reporter Object

Utility Objects - Crypt Object

- It is used to encrypt a string which needs to be protected.
- It uses the Encrypt method.

Example:

user_name=Crypt.Encrypt(“udaya”)

Print “result is:” & user_name

Output:-

Result is: 58fbd03db7aaf4e50fa6026581594d07

‘It will show the string in an encrypted format’

Note: - The encrypted string produced will be different every time code is run.

Utility Objects - SystemUtil.Run

This utility is used to invoke the application available on the system.

Syntax:

SystemUtil.Run file, [params], [dir], [op], [mode]

Argument	Description
file	**Description:** String value [mandatory]. Name of the file you want to run. **Example:** To open Internet Explorer with the default page `SystemUtil.Run "iexplore.exe"`
params	**Description:** String value [optional]. If the specified file argument is an executable file, use the **'params'** argument to specify any parameters to be passed to the application. **Example:** To open a particular url "www.google.com" in Internet Explorer `SystemUtil.Run "iexplore.exe","http://www.google.com"`
dir	**Description:** String value [optional]. Default directory of the application or file.
op	**Description:** String value [optional]. The action to be performed – open, edit, explore, find & print. If this argument is blank(" "), the open operation is performed. **Example:** To print the value specified in a 'file' `SystemUtil.Run "D:\Sample.txt", "", "", "print"`
mode	**Description:** Integer value [optional]. Specifies how the application is displayed when it opens. Default = 1. **Example:** To activate the Sample.txt file window `SystemUtil.Run "notepad.exe", "D:\Sample.txt", "", "", 1`

Utility Objects - SystemUtil.Close

This method can be used to close an application that is open. Below are the various ways in which this utility can be used:-

Method	Using this method
SystemUtil.CloseProcessByName	The application is closed by identifying its process name. **Syntax:** SystemUtil.CloseProcessByName "ProcessName" **Example:** To close the internet explorer browser window. SystemUtil.CloseProcessByName "iexplore.exe"
SystemUtil.CloseProcessByWndTitle	The application window is closed by identifying the value of it's 'window title' property. **Syntax:** SystemUtil.CloseProcessByWndTitle "NameofTheApplicationWindow" **Example:** To close the Home window of an application. SystemUtil.CloseProcessByWndTitle "Application - Home"
SystemUtil.CloseProcessByID	The Application is closed by identifying it's Process ID (PID). **Syntax:** SystemUtil.CloseProcessByID "Process ID" **Example:** Y=Window ("Notepad").GetROProperty ("Process ID") SystemUtil.CloseProcessByID Y
SystemUtil.CloseProcessByHwnd	The application is closed by identifying the owner of the window with the specified handle (hwnd property value). **Syntax:** SystemUtil.CloseProcessByHwnd "hwnd_value" **Example:** To Retrieve the window handle using hwnd property use the below code snippet. P_wid=Window ("Notepad").GetROProperty ("hwnd") Systemutil.CloseProcessByHwnd (P_wid)
SystemUtil.CloseDescendentProcesses	All the applications opened by UFT will be closed. **Syntax:** SystemUtil.CloseDescendentProcesses **Example:** To Close "notepad.exe" and "calc.exe" opened by the UFT Systemutil.Run "notepad.exe" Systemutil.Run "calc.exe" Systemutil.CloseDescendentProcesses

Utility Objects - Reporter Object

Reporter Object is used for sending information/message to the UFT test results. Below are the methods used for doing the same:-

ReportEvent Method:

Reports an event to the run results.

Syntax:

Reporter.ReportEvent EventStatus, ReportStepName, Details, [ImageFilePath]

- **EventStatus** - This attribute is used for passing on the test status to the result sheet. Below are the test status's available in UFT

    - **0 or micPass** - The test passes and is reported in the test results
    - **1 or micFail** - The test fails and is reported in the test results
    - **2 or micDone** - The message is sent to the test result without affecting the test result. It is used to convey the test has run end to end
    - **3 or micWarning** - The message is sent to the test result without affecting the test result
- **ReportStepName** - Used to name the step displayed in the Run Results window
- **Details** - This contains description of the run results event performed

Filter Method :

Using Filter property, we can instruct UFT about what type of event to display in the result

Syntax:

Reporter. Filter = Mode

The mode can be -

- **0**- Test result contains all reported events. Default mode
- **1**- Test result contains only reported events with warning and fail status
- **2**- Test result contains only reported events with fail status
- **3**- All events are disabled in the test result

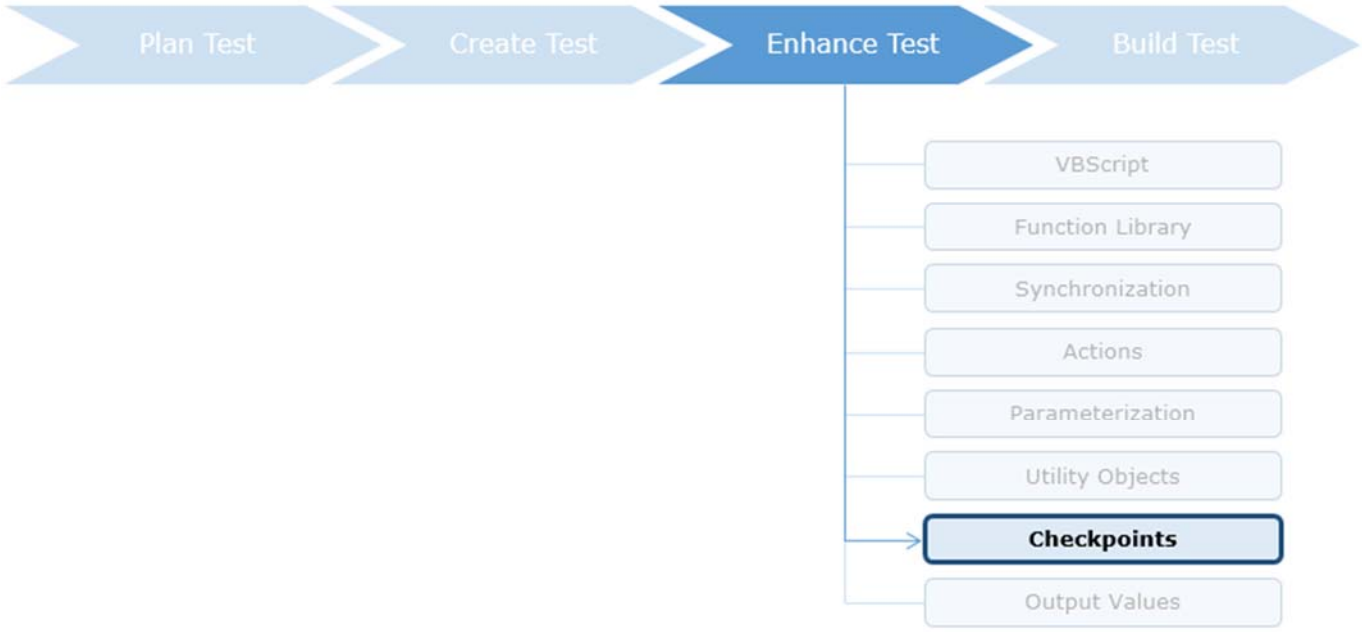
Report Path Property :

This method retrieves the folder path in which the current test's results are stored

Syntax:

```
Path = Reporter.ReportPath    ' Path is a variable to store the result.
```

Enhance Test – Checkpoints



Until now we saw how a test flow is created. However a test case is incomplete without validating the actual behavior of the application against the expected behavior as per the requirements.

What is Checkpoint?

- Checkpoints are specialized class of test objects which can be used to read data/characteristics from runtime objects and validate them against expected values.
- When a checkpoint is added in UFT, it inserts a check checkpoint step in the action script document and a corresponding checkpoint object in the object repository.
- During the test run, UFT compares the expected results of the checkpoint to the current results. If the results do match, the checkpoint passes and if they do not match the checkpoint fails.
- The checkpoint status is reported to the test results automatically.

Note: A failed checkpoint does not stop the execution of the test.

Type of Checkpoints

Subsequent sections contains the brief introduction of the below mentioned checkpoints available in UFT:-

- Standard Checkpoint
- Text Checkpoint

- Text Area Checkpoint
- DataBase Checkpoint
- XML Checkpoint
- Bitmap Checkpoint
- Table Checkpoint
- Accessibility Checkpoint
- File Content Checkpoint
- Page Checkpoint

Standard Checkpoint

Standard checkpoints compare the expected values of object properties to the object's current values during a run session. Standard checkpoints can be created for all supported testing environments (as long as the appropriate add-ins are loaded).

Note:- Standard checkpoint step can be added to the test while recording or editing it.

Example:- Let us assume there is a radio button 

present on the application. We want to check that a radio button is activated after it is selected or not. This can be accomplished by using standard checkpoint.

Standard checkpoint can be added :-

- During recording
- After recording using Active Screen

Text/Text Area Checkpoint

Text Checkpoint

Text checkpoint is used to check that the text is displayed in a screen, window, or Web page, according to specified criteria.

Example 1:

Suppose your application or Web page displays the sentence “Departing from New York to San Francisco”. You can create a text checkpoint that checks that the words “New York” are displayed.

Example 2:

Suppose while checking the text in a page you want to check which text precedes and/or follows it and to which occurrence of the specified text string you are referring.

Text Area Checkpoint

This checkpoint enables you to check that a text string appears within a defined area in a Windows application. When checking text displayed in a Windows-based application, it is often advisable to define a text area larger than the actual text UFT has to check.

Example 1:

Suppose there is an application after logging on which a **"Welcome <<username>>"** message appears on the top right corner of the screen. Now as the text here that we have to check is position specific, then text area checkpoint is the best fit.

DataBase Checkpoint

DataBase Checkpoint

- It checks the contents of a database accessed by your application.
- This database checkpoint can be used to verify the data present in database with the help of DSN (Data Source Name). It stores the expected data and compares this with the data stored in the database.
- When you use a database checkpoint in your script, it connects to the database and sends the query to the database to retrieve the current data from the record set. UFT now compares the current data with the expected data stored in the checkpoint and gives you the result as pass or fail.
- You can create a database checkpoint to confirm that the data being stored in the database does not introduce any error

DSN (Data Source Name) – is a unique name under which connection string is stored i.e. detailed information which is required to connect to database such as database type location, username, password & drives etc.

- We can find the system DSN in the following location:

- Go to Start - > Control Panel - > System and Security
- Click Administrative Tools
- Click Data Sources(ODBC)
- Click System DSN Tab (In this tab you will find all the system DSN's present in the system)

Example:- Suppose in the flight reservation application, user insert's an order. After placing the order a new order number gets generated. Now it has to be validated if the order number generated by the application is entering the right table in the right database. For validating this particular situation database checkpoint is the best fit.

XML Checkpoint

XML Checkpoint

- It checks the data content of XML documents in XML files or XML documents in Web pages and frames
- It can be used to verify data changes across builds
- It can be used to verify whether the structure of the XML file matches with the schema

There are two ways to apply xml checkpoint:-

- XML checkpoint from application
- XML checkpoint from resource

XML Checkpoint from application

- This checkpoint can be used to verify the content of embedded xml (xml present inside the webpage)
- By adding XML checkpoints to your test, you can check the contents of individual XML data files or documents that are part of your web application.
- An XML checkpoint is a verification point that compares a current value for a specified XML element, attribute and/or value with its expected value.
- When you insert a checkpoint, UFT adds a checkpoint step in the Keyword View and adds a Check Checkpoint statement in the Expert View.

- When you run the test, UFT compares the expected results of the checkpoint to the current results. If the results do not match, the checkpoint fails.

XML checkpoint from resource

- This checkpoint can be used to verify the XML document present in specified location.

Example:- Let us consider a situation where an application under test exports data on the web i.e. Say a list of customers who has purchased a particular item from the online store, into an XML file and we have to verify that the application generates the xml file correctly.

Bitmap Checkpoint

Bitmap Checkpoint

- This checkpoint can be used to verify the snapshot of any object. Bitmap checkpoint can be used to compare an area of an application or page, an object or any part of an object. On setting this checkpoint it captures the chosen portion of the screen as a bitmap and compares it with the result at run time.
- Therefore, Bitmap checkpoint captures the visible parts of your AUT and compares them as bitmaps, pixel by pixel.
- Typically this is used to check maps, logos or any other diagrams in your AUT.

Important points to note while applying bitmap checkpoint:

- Bitmap checkpoints are dependent on factors like screen resolution, Operating systems and RGB settings. So any changes to any of these factors will cause the checkpoint to fail.
- When creating the checkpoint, UFT does not record any part that is scrolled off the screen or hidden by any other object.
- While capturing the bitmap if another app is overlapping your AUT then that part of the screen is also captured.
- It can be added while recording or from the active screen.

Example:

Suppose you have a Web site that can display a map of a city the user specifies. The map has control keys for zooming. You can record the new

map that is displayed after one click on the control key that zooms in the map. Using the bitmap checkpoint, you can check that the map zooms in correctly.

Limitations of Bitmap checkpoint:

- Bitmap checkpoint is pixel dependent i.e., change in number of pixels or change in color of pixel will fail the bitmap checkpoint.
- We cannot change expected bitmap. For this, we have to re – insert the checkpoint.
- Using bitmap checkpoint, we cannot compare 2 image files present on the filesystem.

Table Checkpoint

Table Checkpoint

This checkpoint is used to validate information within a table.

Note: Table checkpoint can implemented using Standard checkpoint on tables

Example:

Suppose your application or Web site contains a table listing all available flights from New York to San Francisco. You can add a table checkpoint to check that the time of the first flight in the table is correct.

New York to San Francisco 4/24/2017

SELECT	FLIGHT	DEPART	STOPS
◉	Blue Skies Airlines 360	5:03	non-stop
	Price: $270 (based on round trip)		
○	Blue Skies Airlines 361	7:10	non-stop
	Price: $271 (based on round trip)		
○	Pangaea Airlines 362	9:17	non-stop
	Price: $274 (based on round trip)		
○	Unified Airlines 363	11:24	non-stop
	Price: $281 (based on round trip)		

Accessibility/File Content/Page Checkpoint

Accessibility Checkpoint

It identifies areas of your Web site that may not conform to the World Wide Web Consortium (W3C) Web Content Accessibility Guidelines.

Example:

Guideline 1.1 of the W3C Web Content Accessibility Guidelines requires you to provide a text equivalent for every non-text element. You can add an Alt property check to check whether objects that require the Alt property under this guideline, do in fact have this tag.

File Content Checkpoint

- File Content Checkpoint can be used to compare the content of a file generated run-time (actual file) with the content of the source (expected file).
- The source document is converted into text format. The content gets displayed on which the checkpoint can be configured using the file content checkpoint properties dialog box. When the file content checkpoint step fails during run, the captured data pane in the Run Result Viewer displays a comparison of the generated document and source document.
- For more information, refer to HP UFT > Help > Product Movies > File Content Checkpoints

Page Checkpoint

This checkpoint is another variant of standard checkpoint which as the name indicates comes up when created on a Web Page. This checkpoint check the links and the sources of the images
on a Web page. Page checkpoint can be used to check for broken links. Page checkpoint can be added in two ways:-

- Manually adding a page checkpoint during recording or after recording i.e. During editing.
- UFT can be instructed to put page checkpoint automatically for every page during recording.

Example:

Let us consider a web based application which consist of huge set of web-links. Now while testing that application we have to check whether all the links are working or is there any link/s not working. Manually testing this will be a time taking process so with page checkpoint we can get it validated quickly.

Custom Checkpoint

Till now we have seen how the validations are done using checkpoint. Whatever checkpoints we have added has been applied using the features provided by UFT.

Custom checkpoint is the process of verifying the information on the AUT by writing the program.

Following are the steps that we follow while creating custom checkpoints:-

- Get the expected value
- Get the actual value
- Compare expected and actual value
- Report the status and display it in the run result viewer window.

GetROProperty:

This method is available under all UFT objects which will return value of specified property name from the object present in the application during run time.

Example:

Let us understand the concept of custom checkpoint with the help of a situation where we have to verify the customer name displayed on the flight reservation window. Below is the code for same:

```
Dim ev,av
ev="Infosys"
av = Window("Flight Reservation").WinEdit("Name").GetROProperty("text")
If ev=av Then
Reporter.ReportEvent micPass,"TC_001","Text Matching"
else
Reporter.ReportEvent micFail,"TC_001","Text Not Matching"
EndIf
```

Exist:

This method is available under all UFT objects which returns True if specified objects exist or else returns False after the Timeout. If the timeout is not specified, it takes object synchronization timeout which is present under File -> Settings -> Run tab

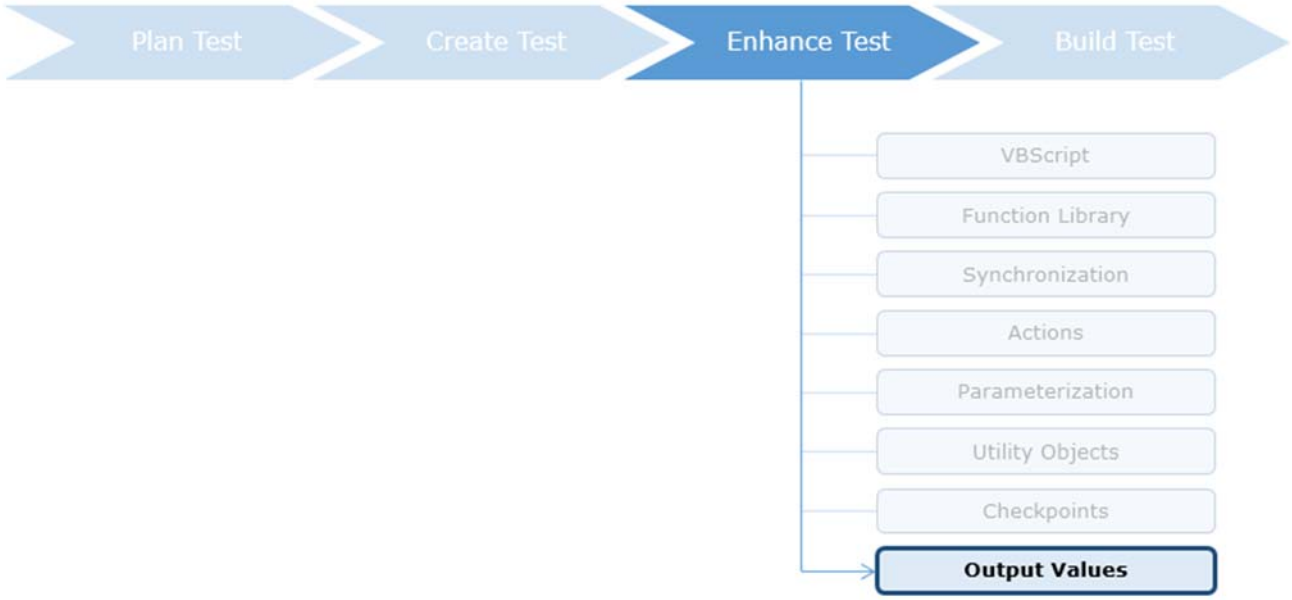
Example:

Write a UFT Script to verify that LOGIN dialog is displayed when we launch Flight Reservation application. Below is the code for same:

```
Dim DialogExist
SystemUtil.Run "C:\Program Files\HP\Unified Functional Testing\samples\flight\app\flight4a.exe"
DialogExist = Dialog("Login").Exist(2)
If DialogExist then
Reporter.ReportEvent micPass,"TC_001","Dialog Box Present"
Else
Reporter.ReportEvent micFail,"TC_001","Dialog Box is not Present"
End IF
```

Note:- In the code Exist(2) – 2 is the timeout in seconds.

Enhance Test - Output Values



Sometimes it is required to capture the data from the application during run time and save them, so that it can be reused wherever required.

Example:

To understand the point let us consider a scenario where there are two flows, from flight reservation application.

First Flow:

- Login Page + Insert Order

Output of this flow is: - Order Number

In First Flow we are doing the Login operation followed by Insert order & the output of the this flow is an order number generated after placing a new order. Now consider the second Flow

Second Flow:

- Open Order - Order Number (Output of first flow)
- Fax Order - Open order(need to fax the order that has been opened)

This scenario needs to be tested for the 50 combination of test data used for inserting a new order and sending a fax for those orders placed and hence we need to use the output values. As the output of first flow will act as an input to the second flow.

What is output value

- Output value is used to output the property value of an UI object to the runtime datatable and pass it as input to another object in the same action or different action.
- The captured value can be stored in Environment Variables as well.

Types of output values -

Table below shows the various types output value options in UFT.

Output value type	Description	Example of use
Standard output value	Outputs the property values of any object	Use standard output values to output text strings by specifying the text property of the object as an output value
Text/Text Area output value	Create a text output value from a text string displayed on a Web page or application. You can define the output value as part of the displayed text, and specify the text before and/or after the output text.	Use Text/Text Area output value to store the error message that appears after a particular step with the known title bar value
Database output value	Create database output values by defining a query to retrieve data from the database and selecting the values to output from the query result set	Use database output values to output the value of the contents of database cells, based on the results of a query defined
XML output value	Create XML output value steps from any XML document contained in an XML Web page or frame, directly from an XML file, or from test objects that support XML.	Use XML output values to output the values of XML elements and attributes in XML documents

Note:- Apart from this there is "File content output value" whose target is to fetch the data from a file.

Enhance Test – Summary

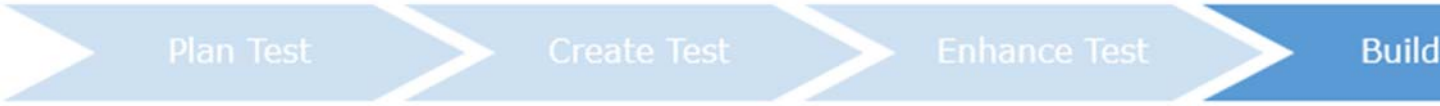
Let us summarize what we have learnt in Enhance Test phase.

In the Enhance test phase we learnt about multiple ways of enhancing a script created in UFT:

- VBScript
  - Introduction to VBScript
  - Condition and Looping statements
  - Sub Programming
  - String Manipulation
  - Conversion functions
- Function Library
  - How to create and associated function library in UFT
- Synchronization
  - Multiple ways to synchronize the script execution speed with the execution speed of UFT
- Actions
  - Introduction to actions
  - Types of actions
  - Various ways of calling actions
- Parameterization
  - What is parameterization
  - Various types of parameterization options available in UFT
- Utility Objects
  - Types of utility objects
- Checkpoints
  - What is a checkpoint
  - Types of checkpoints available in UFT

- Output Values

Build Test



Main objective of this phase is to understand how a complete test is created using the features, that we have learnt in "**Create Test**" and "**Enhance Test**" phases earlier.Let us understand this with the help of a scenario:-

Scenario Description:

Assume there are 3 agents who wants to login to the flight reservation application one by one and after logging in they want to book tickets for 5 different customers. After booking their orders they have to check whether the booking is actually done or not with the help of suitable checkpoint. As an automation tester your job is to automate this scenario using all the suitable components available in UFT.

Agent Details:

Agent Name	Password
Udaya	Mercury
Vishal	Mercury
Sasmita	Mercury

Customer Details:

Customer Name
John
Mary
Donnie
Sam
Smith

Below are the steps that we may consider while creating the script:-

- **Infrastructure** – Tool is provided to the tester with suitable license.
- **Add-In** – After looking at the technical documents, we can find that it is a window based application with some other components, so below mentioned add-ins would be required:
    - Windows Application
    - ActiveX
- Below are the steps to identify while creating and enhancing the script:
    - We need the following actions :- Login, Booking, Logout
        - **Login** – This action contains the script for logging into the application.
        - **Booking** – This action is going to contain script for placing a new order everytime a user wants to.
        - **Logout** – This action will contain the code for logging out from the application.
    - As the scenario requires multiple data points to be used for creating the script. On analyzing the situation we can figure out that login details are kept in Global sheet of the data table and customer details are kept inside the respective local sheet.
    - As we have to validate whether booking happened or not, so as to validate this we are going to use Standard Checkpoint in the script.

- Create the script using record and replay feature with normal recording mode enabled while recording all the events performed on the application.
- Save the Test in solution explorer pane window of UFT with required actions, as **“Scenario1”**.
- Execute the test and view the results in **“Run Result Viewer”** window.

Conclusion:- Create and execute the above mentioned scenario. **"Scenario1"** is the test that UFT has created for the requirements mentioned above.

Course Summary

The course thus provided us an overview of how to work with UFT to automate application under test. The following diagram gives a summary of the same: