



QEngine Technical Paper

---

## Building Maintainable Test Cases with QEngine

## Table of Contents

|  |           |
|--|-----------|
| <u>Abstract .....</u>  | <u>3</u>  |
| <u>Introduction .....</u>                                      | <u>4</u>  |
| <u>Preface .....</u>   | <u>4</u>  |
| <u>Problem Definition .....</u>                                | <u>5</u>  |
| <u>Reusing Scripts through External Jython Functions .....</u> | <u>6</u>  |
| <u>Reusing Scripts through "callScript" .....</u>              | <u>7</u>  |
| <u>Calling Jython API Functions .....</u>                      | <u>8</u>  |
| <u>Calling Java API inside Test Automation Scripts .....</u>   | <u>9</u>  |
| <u>Conclusion .....</u>  | <u>13</u> |

## **Abstract:**

Test automation involves creating of many test cases and scripts to fulfill the testing requirements. Maintaining those scripts and test cases is very important. Script maintenance involves better planning on categorizing scripts based on usage and modularizing scripts based on application under test.

While automating an application we should also keep in mind that the application might change in future versions. Automation will be successful when it offers more flexibility to future application changes, so that the scripts can be easily modified.

In this paper we will present how the script maintenance can be made simple by creating reusable scripts with QEngine. Also we will detail how power of jython and Java API can be used inside the scripting.

## Introduction:

The purpose of this paper is to explain how to create reusable scripts with QEngine and make script maintainability simple. It will be easy to debug any problem by reusing the scripts.

## Preface:

QEngine is a test automation tool, that can be used to automate web applications by recording the user interaction with the web application and playing back it as recorded.

QEngine script is a collection of step by step actions carried out by the user while interacting with the web application in a script format. Upon recording the user interaction with the web application, QEngine will generate the script in human readable format.

QEngine uses popular scripting language “**Jython**” as the scripting language for QEngine scripts. Jython is the Java Extension of **Python** language. You can utilize the power of jython in the QEngine scripts to make it more powerful.

## **How QEngine Recording works ?**

For Example, If the user searches for “web functional testing” keyword in google, QEngine will record the interaction in the script as below,

```
secs | launchApplication("about:blank") # launches a web browser with the given URL
      | changeURL("http://www.google.com",3) # loads the given URL in the browser with the delay of 3
      | setWindow( "Google",4) # sets the document with the given title for further playback
      | setText("q","web functional testing",4) # sets the given text in to the text field identified by given id
      | clickButton("btnG",2) # click the button in the web page identified by the given id
```

*The numerical value given in each line represents the **delay** before performing the operation.*

The above script is in a better human understandable format. During replay of the script the actions recorded in the script will be performed step by step.

Similarly we can create and record any number of scripts as required. The recorded scripts can be replayed as needed.

## Problem Definition :

Let us assume, the web application under test requires a user login to access any of its resources. Normally you will record the login actions in all the scripts before doing any script specific operation.

The potential problem here is that **Script Maintenance** will become more cumbersome as modifying the script with respect to changes in the application will be difficult.

## **What is Script Maintenance ?**

Script Maintenance is a mechanism of providing simplicity for script storage. It should have flexibility in modifying the scripts for future application changes.

## **What is the problem with Script Maintainability ?**

The main problem if the script is not properly designed is that if there are minor changes to the application UI (login page) in the future, it will become difficult to incorporate the changes in the scripts already created.

### *For example:*

Assume an application requires user login to access its resource, then if the scripts were designed in such a way that login action is present in all the scripts, then any changes to the login UI of the application in the future will make the script modification difficult.

## **How to solve the above problems ?**

The above problems can be resolved by planning script maintenance in such a way that any repetitive tasks can be grouped under a function and can be reused anywhere in the suite.

### *For example:*

In the above scenario, the login actions can be grouped under a single function so that all the scripts can call the login function to perform the login action in the script. This is called reusing of the scripts.

There are two ways to create reusable scripts in QEngine,

1. Define a separate **External Jython Function** and then use it in the QEngine scripts.
2. Create a new separate **QEngine script** to perform login operation and then call it inside the other QEngine scripts using callScript functionality.

## Reusing Scripts through External Jython Function:

For example:

Logging into the web application is a common thing that needs to be done before doing any operation in a web application. If the login action can be recorded in a separate script and reused in all other scripts, it will be easy to maintain / debug the scripts.

Following are the script lines that do login action in the web application,

```
launchApplication("<Application URL>") # launches a web browser with the given URL  
  
setWindow("Login",2) # set the document for playback  
setText("username","qengine",2) # set username in to the username textfield as "qengine"  
setText("password",<encrypted password>,2) # set password in the text field  
clickButton("login",2) # click the button to login
```

As per the first method, the above script lines can be grouped under the an External Jython function named "login" in the custom.py file so that it can be called from any QEngine scripts.

In the custom.py file define a function named login and define the actions recorded,

The custom.py file will look like below,

```
# Script actions recorded belong to Framework.py and needs to be imported here,  
from Framework import *  
  
def login(username,password):  
    launchApplication("<Application URL>")  
    setWindow("Login",2)  
    setText("username","qengine",2)  
    pwd = getEncryptedValue(password)  
    setText("password",pwd,2)  
    clickButton("login",2)
```

The above function will take user name and password and login to the application. In the QEngine script where it requires to login to the application, simply call the above function as shown below,

```
from custom import *  
login("qengine","qengine")
```

This will perform the login operation in the web application.

### Where should the custom.py file be kept under QEngine?

The custom.py file can be placed under QEngineWebTest\jars folder. If the file is kept under QEngineWebTest\jars folder, it can be accessed directly inside QEngine scripts.

Incase, if you wish to place the custom.py file under a different folder than the above, the path needs to be defined in the PY\_PATH variable in the setCommonEnv.bat file. To define the PY\_PATH variable, follow the instructions given below:

- a) Open the setCommonEnv.bat file present under QEngineWebTest/bin folder.
- b) Set value to the line set PY\_PATH=<path to the folder where the custom files are kept>
- c) Now save the file and restart the QEngine.

This will properly load the required custom files from the specified path while running the QEngine scripts. To add multiple location in the PY\_PATH, add multiple paths with ;(semicolon) as separator between paths.

### Reusing Script through "callScript":

1. Create a new script named "login" in a suite in the Web Client.
2. Record the login actions alone in the script,

```
useLocalMapFile()  
  
launchApplication("<Application URL>")  
setWindow("Login",2)  
setText("username","qengine",2)  
pwd = getEncryptedValue(password)  
setText("password",pwd,2)  
clickButton("login",2)
```

Now the login can be called in other QEngine scripts as below,

```
callScript("login","login")
```

The above script line will invoke the login script.

The login actions will be stored in a single place, So it will be easy to make any changes to login script alone without affecting the scripts requires login operation.

### Calling Jython API Functions :

Jython / Python has more inbuilt capabilities which can be used inside QEngine scripts for efficient usage to meet the automation requirements. We shall discuss on how to use those capabilities inside QEngine Scripts.

Here we shall discuss on generating random numbers using Jython API,

Let us define a sample function which will generate random numbers,

*# File custom.py (Below are the contents of custom.py file)*

```
import random # This will import the random module present with Jython in to the custom.py
```

```
# The below function can be used to get Random values in the given range a,b
```

```
def getRandomInt(a,b):
```

```
    val = random.randint(a,b) # here we are invoking the API function to generate random number
```

```
    return val; # the randomly generated value is returned to the calling script.
```

The above function `getRandomInt` will receive two arguments and generate a random int with in the given range of a,b.

### How to access the above defined function inside the QEngine script ?

The above defined function can be accessed in QEngine Script after importing the functions in to QEngine Script. You can access the the above-defined function as shown below:

```
# The below line imports the functions of custom.py
```

```
from custom import *
```

```
val = getRandomInt(10,20)
```

```
displayMessage(str(val))
```

The above code will get the random int value within the range of 10 and 20.

Similarly, we can use the various Jython API as per our needs and use the power of Jython in QEngine scripts.

You can use the following links to get help on Jython / Python,

Python Tutorial - <http://www.python.org/doc/tut/tut.html>

Python Library Reference - <http://docs.python.org/lib/lib.html>

Python Reference Manual - <http://docs.python.org/ref/ref.html>

### Calling Java API inside Test Automation Scripts:

#### How can the Standard Java Classes be invoked inside the QEngine script ?

We can access the standard Java API's inside the jython script to perform customary operations.

For example:

Let us do a sample script that accesses a file and appends some content into it using java.io.File and java.io.FileOutputStream API.

Following are the contents of *custom.py*,

```
# To use File operations and File API of Java we have to import  
# appropriate Java API in the jython script.  
import java.io.File as file  
import java.io.FileOutputStream as outStream  
import java.lang.String as string  
from Framework import *  
  
def writeInFile(filePath,contentToWrite):  
    try:  
        f = file(filePath)  
        exist = f.exists()  
        if(exist==0):  
            f.createNewFile()  
            displayMessage("File created newly")  
        fos = outStream(f, 1)  
        s = string(contentToWrite)  
        fos.write(s.getBytes())  
        fos.close()  
        displayMessage("File written successfully ")  
    except Exception,e:  
        displayMessage(type(e))  
        raise
```

In the above-defined function, it takes two arguments - path of the file and content to write in the file. The above function can be called in the QEngine scripts as below:

```
from custom import *  
writeInFile("C:/custom.txt","Testing file writing in python")
```

The above function call will invoke the writeInFile function and write the given content in the file.

### How to customize the script using custom java class inside the script ?

We can access our own Java API to perform any customary actions inside the QEngine scripts.

For example:

Let us write a Java API which will sort the array of integer values in ascending / descending order and return it back.

Below is the content of Sort.java

```
package com.mytest;  
  
public class Sort  
{  
    public int[] ascend(int[] values)  
    {  
        for(int i=0;i<values.length;i++)  
        {  
            for(int j=i+1;j<values.length;j++)  
            {  
                if(values[i] > values [j])  
                {  
                    int temp = values[i];  
                    values[i] = values[j];  
                    values[j] = temp;  
                }  
            }  
        }  
        return values;  
    }  
}
```

```

public int[] descend(int[] values)
{
    for(int i=0;i<values.length;i++)
    {
        for(int j=i+1;j<values.length;j++)
        {
            if(values[i] < values [j])
            {
                int temp = values[i];
                values[i] = values[j];
                values[j] = temp;
            }
        }
    }
    return values;
}
}

```

Below is the content of custom.py

```

import com.mytest.Sort as sort

def sortAscend(arrVal):
    s = sort()
    val = s.ascend(arrVal)
    return val

def sortDescend(arrVal):
    s = sort()
    val = s.descend(arrVal)
    return val

```

The above functions can be accessed inside QEngine script as below,

```
from custom import *  
  
values= [45, 12, 33, 55, 50]  
sortedVal = sortAscend(values)  
  
for i in range(0, len(sortedVal)):  
    displayMessage(sortedVal[i])
```

The above snippet will get the integer array stored in “values” sorted in ascending order and print the same.

## Conclusion:

This paper concludes on reusing QEngine scripts for better script maintenance and script debugging . We hope that you now have a better understanding of the designing, maintaining and reusing scripts in QEngine and also a better understanding of using Python API and Java API inside QEngine scripts for better efficiency.

For any clarification on this article mail [srinivasar@zohocorp.com](mailto:srinivasar@zohocorp.com)

Visit our website [www.qengine.com](http://www.qengine.com) for more information.

ManageEngine Applications Manager ([www.appmanager.com](http://www.appmanager.com)) provides out of the box performance monitoring for J2EE Application Servers, Databases and Servers.

ManageEngine is the leader in low-cost enterprise IT management software. The ManageEngine suite offers enterprise IT management solutions including Network Management, HelpDesk & ITIL, Bandwidth Monitoring, Application Management, Desktop Management, Security Management, Password Management, Active Directory reporting, and a Managed Services platform. ManageEngine products are easy to install, setup and use and offer extensive support, consultation, and training. More than 30,000 organizations from different verticals, industries, and sizes use ManageEngine to take care of their IT management needs cost effectively. ManageEngine is a division of ZOHOO Corporation. For more information, please visit [www.manageengine.com](http://www.manageengine.com).