



QEngine Technical Paper

Automating AJAX based Web Application 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>4</u>
<u>Using AJAX Element Handling option in Settings</u>	<u>5</u>
<u>Using Built-In Functions</u>	<u>6</u>
<u>Conclusion</u>	<u>10</u>

Abstract :

Web Sites and Applications are becoming crucial to the success of businesses. With the increasing popularity of Web 2.0, a large number of web applications have become more complex using advanced techniques such as Asynchronous JavaScript and XML (AJAX) which gives tremendous usability benefits. AJAX makes asynchronous calls to the web server to create more responsive web applications.

In traditional web applications, the HTML elements are already loaded in the web pages which is fairly easy to test in an automated fashion. Unfortunately, AJAX-based web applications are not as easy or consistent to test as the traditional web applications. The problem with AJAX testing is that the web page under test will be modified asynchronously such as an element will be dynamically added based on a check box click or an element will be removed based on a list element selection, etc.

In this paper, we present an overview of the capabilities QEngine for successful automation of AJAX based application.

Introduction :

The purpose of this paper is to outline the provisions of QEngine product that allows automating AJAX based Web Application.

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 it back as recorded.

How does QEngine identify elements during playback ?

Before recording any actions in the web page, QEngine will record “**setWindow**” action, which will enable QEngine to identify and use the proper window document to search for elements during playback. Each time the page refreshes the “**setWindow**” will be recorded. The window document set using “**setWindow**” function call will be used for the further element identification and for performing actions over the element. If the page is loaded with some delay then QEngine will wait in function call “**setWindow**”, for the page to completely load until maximum timeout period is reached (default: 60 secs) and then proceed to search for the elements in the web page.

Problem Definition:

In case of, AJAX web applications the element can be added to the web page without refreshing the content of the page. In such a scenario, “**setWindow**” will not be recorded in the script. Hence, if the element is loaded into the page with some delay, QEngine might fail to perform the intended / recorded action over the particular element which will result in failure of playback.

What provision does QEngine have to work with AJAX application ?

QEngine has two ways to successfully deal with such a scenario :

1. [Using “AJAX element handling” option in Settings.](#)
2. [Using Built-in functions.](#)

Using “AJAX element handling” option in Settings :

Consider the Following Scenario :

```
launchApplication("http://demo.qengine.com:7001/examples/payrollsystem/dynamic/employee.html")
setWindow( "Employee Details Entry Form",5)
clickButton("emp",1) # Here an AJAX request will be sent to show the employee details form
setText("username","John",13) # The employee form is displayed after 13 secs as recorded here
setText("age","20/07/1978",8)
```

In the above script, the Employee details form is shown only after 13 secs. But if it took more than 13 secs during playback(due to some network delay), then playback will fail to find the other elements in the form or play may behave erratically. This may cause further script playback failure, as some intermittent steps might not have happened.

To overcome this issue, enable the “**Enable AJAX element handling**” in settings as said below :

In the “**Settings->Replay Settings**”, switch on “**Enable AJAX element handling**” and configure the “**Maximum time to wait (in secs)**”.

By enabling this option, QEngine will search for a certain element to perform the recorded action during playback. If QEngine is able to find the needed element in the window document, it will perform the needed action and proceed to the next script activity.

If QEngine is unable to find the needed element, then it will repeatedly search for the same element until it is found. If the maximum time limit is reached before finding the element, the particular element will be reported with “**ELEMENT_NOT_FOUND**” status and proceed with further script activity.

For Example :

Configure the “**Maximum time to wait**” as 25 secs in the Replay Settings. Now start the playback for the above-listed script. In this case, QEngine will wait for the element to be loaded in the document for 25 secs and not 13 secs. Hence even though the element is loaded with minor delay QEngine can identify the element and perform the needed actions. You can also configure “Maximum time to wait” as high as required.

The “**AJAX element handling**” can be enabled / disabled during runtime using the below function :

```
enableWaitForElement(“true / false”, “<wait_time_in_secs>”)
```

Example :

```
enableWaitForElement(“true”, “10”)
```

Also the “**timeout / wait time**” for elements alone can be modified during runtime with the help of the below function:

```
setWaitTimeForElement(wait_time_in_secs)
```

Example :

```
setWaitTimeForElement(20)
```

You can insert the above two functions in the script using “Built-in Function” link available above the script editor.

Using “Built-in Functions” :

Using “**Built-in**” functions, we can wait for the particular element to load in the document where ever it is required. i.e, Instead of setting the “AJAX element handling” for the whole suite, you can use it at a place where it is actually required.

QEngine has the following “**Built-in Functions**”, that can be inserted into the script using “Built-in Functions” :

1. [waitForElement](#)
2. [waitForElementRemoval](#)
3. [waitForDynamicElement](#)
4. [waitForDynamicElementRemoval](#)
5. [waitForText](#)
6. [waitForTitle](#)

The above functions are present in the “**Function Generator**” under the “**AJAX Functions**” category.

waitForElement :

During playback, QEngine will repeatedly search for the particular HTML element in a web page until it finds the particular element. QEngine will use the mapping data stored in the map file to identify the element in the web page. QEngine will stop searching for the element when any of the following state is reached :

1. QEngine able to find the target element. - Success status will be reported to the reports.
2. Maximum timeout is reached. - ELEMENT_NOT_FOUND status will be reported to the reports.

Syntax:

```
waitForElement("<Element_ID>","<Wait_time_in_secs>")
```

Example :

```
waitForElement("username",25)
```

The above script line will wait for 25 secs for the username text field to be added in the current web page.

waitForElementRemoval :

During playback, QEngine will repeatedly search in the web page for whether the particular HTML element is removed from the web page, until the element is removed from the web page. QEngine will use the mapping data stored in the map file to identify the element in the web page. QEngine will stop searching for the element when any of the following states is reached :

1. QEngine identified the particular element is removed from the web page. - Success status will be reported to the reports.
2. Maximum timeout is reached. - ELEMENT_NOT_REMOVED status will be reported to the reports.

Syntax:

```
waitForElementRemoval("<Element_ID>","<Wait_time_in_secs>")
```

Example :

```
waitForElementRemoval("username",25)
```

The above script line will wait for 25 secs for the username field to be removed from the current web page.

waitForDynamicElement :

The target element will be identified using the tagname and other matching properties given as argument to the function. As described above, QEngine will repeatedly search for the element in the web page until it finds the particular element in the web page. QEngine will stop searching for element if any of the below state is reached :

1. QEngine able to identify the target element. Success status will be reported to the reports.
2. Maximum timeout is reached. - ELEMENT_NOT_FOUND status will be reported to the reports.

Syntax:

```
waitForDynamicElement("<TagName>","<property_name>","<property_value>",index,"<useRegExp>","<Wait_time_in_secs>")
```

Example :

```
waitForElement("input","name","username",1,"false",25)
```

The above script line will wait for 25 secs for the username text field to be added in the current web page.

waitForDynamicElementRemoval :

The target element to be removed will be identified using the tagname and other matching properties given as argument to the function. QEngine will repeatedly check whether the target element is removed from the web page. QEngine will stop searching for the element in any of the below state is reached:

1. QEngine identified the particular element is removed from the web page. - Success status will be reported to the reports.
2. Maximum timeout is reached. - ELEMENT_NOT_REMOVED status will be reported to the reports.

Syntax:

```
waitForDynamicElementRemoval("<TagName>","<property_name>","<property_value>",index,"<useRegExp>","<Wait_time_in_secs>")
```

Example :

```
waitForElementRemoval("input","name","username",1,"false",25)
```

The above script line will wait for 25 secs for the username text field to be removed from the current web page.

waitForText :

During playback, QEngine will repeatedly search for the given text with given prefix, and suffix in the web page. The search text , suffix and prefix text will be passed as arguments to the waitForText function. QEngine will stop searching for the text when any of the following state is reached :

1. QEngine identified the particular search text with give suffix and prefix. - Success status will be reported to the reports.
2. Maximum timeout is reached. - ERROR_TEXT_NOT_FOUND status will be reported in reports.

Syntax:

```
waitForText("<Search_Text>","<Prefix_Text>","<Suffix_Text>",<Wait_time_in_secs>)
```

Example :

```
waitForText("QEngine","AdventNet","WebTest",25)
```

The above script line will wait for 25 secs for the "AdventNet QEngine WebTest" text to be displayed in the current web page.

waitForTitle :

During playback, QEngine will repeatedly search for the window with given window title to be displayed. The title to be searched will be sent as argument to the function. QEngine will search for the window with the given title to be found until any of the following state is reached :

1. QEngine identified the window with the given title. - Success status will be reported in the reports.
2. Maximum timeout is reached. - ERROR_WINDOW_NOT_FOUND status will be reported in reports.

Syntax:

```
waitForTitle("<Window_Title>",<Window_Index>,<Wait_time_in_secs>)
```

Example :

```
waitForTitle("AdventNetPayroll",1,25)
```

The above script line will wait till 25 secs for the window with title "AdventNetPayroll" to be opened.

For more help on the above listed functions refer ajax function reference at

http://www.qengine.com/help/built_in_functions/ajax-functions.html

For help on other functions refer http://www.qengine.com/help/built_in_functions/built_in_functions_intro.html

Conclusion:

This paper discusses automating AJAX based web applications using QEngine . We hope that you now have a better understanding of the automating AJAX based application.

For any clarification on this article, send an email to srinivasar@adventnet.com

Visit our website www.qengine.com for more information.

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

AdventNet offerings include software for IT Operations Management (www.manageengine.com), IT Security Management (www.securecentral.com) and Office Productivity (www.zoho.com) and serve over 20,000 customers worldwide.