Technical white paper

Cross-Browser Functional Testing Best Practices

HP Unified Functional Testing Best Practices Series

Table of contents

Welcome to this Document

Welcome to *Cross-Browser Functional Testing Best Practices.* This document provides concepts and guidelines for functional testing of applications with different browsers, describes the different types of testing that the term 'cross-browser' encompasses, and presents best practices for different scenarios involving multiple browsers.

This document is intended for automation engineers who are developing functional testing scenarios for web-based applications that run in a browser.

Introduction to Cross-Browser Functional Testing

The term *cross-browser testing* usually means some variation of testing of a web-based application on different browsers. We might have a requirement that our application must run on Internet Explorer, Firefox, Chrome and Safari. Conversely, we might have to support all versions of Internet Explorer from version 8 and later. And what do we mean by 'must run'? Is it enough that the functionality is identical on different browsers? What about the way the page looks – is it acceptable if two buttons appear on the same line in one browser, but appear one above the other in a different browser?

This document will examine some common types of cross-browser functional testing and will present some best practices that you can adopt in order to achieve your cross-browser testing goals.

Types of Cross-Browser Functional Testing

Standard Multi-Browser Testing

Standard multi-browser testing is a basic check that the application under test (AUT) supports more than one browser, for example, Internet Explorer, Firefox, Chrome, and Safari. This type of test can run on a single computer, first on one browser and then on another. Alternatively, it can run on different computers. The same test script is typically run on the different browsers without modification. It usually assumes that only one user is active during the testing.

Multi-Version Testing

This is a test that your AUT works on more than one version of a browser, such as Internet Explorer 9 and 10, or all versions of Firefox from 10 onwards. Typically, only one user is active during the testing. The same test script is used for the different browsers.

In many cases, the tests must be run on different computers, since some browsers do not allow more than one version to be installed on the same computer at one time.

Concurrent Testing

Concurrent testing checks that your AUT works with two or more browsers at the same time. The same user might be logged into the different browsers, or different users could be logged in, depending on the aim of the test and the requirements of the AUT.

Aims of the test include:

- Ensuring that there is no unexpected interaction between the browsers
- Checking that the AUT allows a user to be logged in more than once. If a user can be logged in more than once, check that any updates made by one user are reflected in the other browser(s).

- Checking that the AUT prevents a user from logging in more than once, and that the browser behaves correctly when a user attempts to log in twice. Note that some applications will not allow the second login, while other applications will allow the second login and log the first session out.

Concurrent testing[1] is a generic term that covers the following test types[2]:

- Single Browser Concurrent Testing - The same browser is used on the same computer at the same time. Most browsers support tabbed browsing, which allows multiple web pages to be opened within a single browser window. Typically, each tab is implemented as a separate process, so single browser concurrent testing can be performed by opening up two tabs within the same browser.
- Single Browser Distributed Concurrent Testing – The same browser is used on different computers at the same time
- Multi-Browser Concurrent Testing – Different browsers are used on the same computer at the same time
- Multi-Browser Distributed Testing – Different browsers are used on different computers at the same time

It is also possible to test that the AUT works with two or more different versions of the same browser on the same computer at the same time. This is called *Multi-Version Concurrent Testing*, and will not be addressed further in this document.

Application (or Browser) Compatibility Testing

This is a test that the AUT looks and behaves the same regardless of the browser or browser version used to access it.

Different browsers have different ways of rendering ('displaying') the same HTML, so the same page might look different in two different browsers. This test checks that the page looks the same, or at the very least that there are no glaring discrepancies between browsers. This test is often carried out manually.

Typically, the tester will choose one browser to be the 'baseline' browser, will ensure that all functional tests work correctly, and will check that each page in the baseline browser is rendered correctly. Once the browser has been established as the baseline, the tester will open up another browser and go through each page, comparing the look of the page to the baseline browser.

Some of the most commonly encountered differences between browsers include:

- Fonts
- Page margins
- Sizes of elements
- Position of elements
- Colors of elements

There are also differences that result from the way that JavaScript is handled by different browsers. These differences can take the form of innocuous differences such as minor layout issues to major differences in behavior. Some examples that we have seen include:

- Events failing to trigger on one of the browsers (such as a drop-down containing States, which is supposed to change according to the selection of a drop-down containing Countries)
- Behavior of drag and drop not working correctly on one of the browsers

Best Practices for Cross-Browser Functional Testing

This section describes some common best practices that can be adopted in order to achieve the different types of cross-browser functional testing described in the previous section. These best practices assume you are using HP Unified Functional Testing (UFT).

[1] Note that in this document, the term *concurrent testing* refers purely to functional testing, and not load or stress testing.
[2] There is an additional type called *Multi-Version Concurrent Testing*, which tests that the AUT works with two or more different versions of the same browser on the same computer at the same time. This is not a common scenario, and will not be addressed further in this document.

Preparing the Browser

There are a number of configurations and settings that should be addressed in order to ensure successful cross-browser functional testing. This section describes some of the most important ones.

Browser Dimensions

In cross-browser functional testing, and particularly in Compatibility Testing, you should take into account differences in the browser dimensions. For example, the browser toolbars can take up real-estate, and this will affect the layout of the browser contents. Similarly, the size of the browser itself can have an effect. It is recommended to execute the tests with the browser in full-screen mode.

Make sure that the font size and page zoom are set to the correct values (usually set to the default) in all browsers.

For Compatibility Testing, it is important to test your browser on the same screen size and resolution as the baseline browser. Keep this in mind if you are testing on a different machine than the one used for the baseline.

Preparing the Browser for Playback

There are some steps that you should consider to ensure that scripts are played back smoothly on the different browsers.

- If you need to play back on Firefox, make sure that the “Unified Functional Testing Extension” is enabled in Firefox. See UFT’s ‘Web Add-in – Quick Reference’ section of the help for details.
- Similarly, to play back on Chrome, make sure that the “Unified Functional Testing Agent” is enabled in Chrome. See UFT’s ‘Considerations – Google Chrome’ section of the help for details.
- If the AUT uses HTTPS (SSL), ensure that any necessary certificates are installed prior to testing.
- In order to ensure consistent testing, you should turn off the browser’s auto-updating feature. Be aware that this means that you may not receive critical security updates. Instead, you should update the browser manually on a regular basis.

To turn off auto-update for the most common browsers, do the following:

- For Internet Explorer: Microsoft provides a toolkit called *Internet Explorer 10 Blocker Toolkit* to disable auto-update of Internet Explorer 10. There are similar toolkits available for versions *9*, *8* and *7*.
- For Firefox: Go to the ‘Update’ tab of Firefox’s ‘Options’ dialog, and choose either ‘Check for updates, but let me choose whether to install them’, or ‘Never check for updates’.
- For Chrome: Disable Google Update. For full details consult *Google’s support site.*

Writing a Portable Test

Most of the cross-browser test types require that a single test script can be run for different browsers without modification. If we need to modify the script to work for a different browser, we end up with multiple copies of the script, each one capable of working for only one browser. This makes maintenance of the test script very difficult, because if the AUT’s functionality changes, each of the test scripts need to be updated separately.

If we plan to execute our tests manually, it is relatively easy to write a script that will run on all browsers. The manual test will contain an instruction to open up a specific browser. From that point on, unless the AUT’s behavior changes depending on the browser, each step will apply to whatever browser is being used, without having to specify any browser-dependent characteristics.

For automated tests, the situation is a bit more complicated. Tools such as UFT maintain a repository of the objects (Object Repository, or OR) that are in each screen of an application, using a combination of identifiers and properties of the object in order to uniquely identify it. When the script is called upon to perform an action on some object, it consults the OR in order to find the required object in the browser’s page.

For many applications, the properties used to identify the objects in one browser are sufficient to identify the equivalent object in a different browser. So the same script and OR can be used on different browsers, with the only difference being the specific browser that is opened at the start of the test (and closed at the end of it). The script and OR can be created by recording a scenario using one browser, and then played back on the same or a different browser.

Unfortunately, life isn’t always this simple. Modern applications that use technologies such as AJAX (used to create asynchronous behavior in the browser) can pose a challenge to testers, for reasons including:

- The state of an application is difficult to determine – clicking the ‘back’ button might revert to the previous state of the AJAX application, or it might return to the full page visited prior to the current page

- Dynamic page updates can interfere with user interaction, and consequently, with test script interaction. For example, the user might initiate an action (such as a search) by pressing a button, and continue working with the page. At some point, that action completes, and a popup is displayed while the user (or the script) is doing something else. This can confuse the test script.
- Objects are created and displayed on the screen dynamically, so these dynamic objects are not always recorded and stored in the OR. Furthermore, they might be displayed differently on different browsers, and consequently need to be located and identified using different criteria depending on the browser.

There are some techniques that can be adopted to overcome these challenges.

- To avoid the problem of determining the state of the application (the example of the 'back' button, above), the test can navigate to a specific URL instead of relying on clicking the 'back' button. If the script initiates an action that will complete 'later', the script can wait until that action completes by polling for the existence of the completion indicator, or by using an appropriate timeout for the Exist property of an element that is expected to appear when the action completes (Note that if you don't provide a timeout, it will use the value in the Test Settings dialog for Object Synchronization Timeout).
- In the case where dynamically created objects are generated, Descriptive programming can be used to overcome the issue. Descriptive programming allows you to specify an object without referring to the OR or the object's name. For example:

  ```
  Browser("Browser").Page("Page").WebEdit("Name:=SearchBox","html tag:=INPUT").set "text to search"
  ```

  This script line will look in the page 'Page' of browser 'Browser' for an edit box with the name of 'SearchBox', with an 'Input' HTML tag, and set its text to 'text to search'. In this example, Page and Browser are in the OR, but the edit box is not.

Avoiding Browser-Specific Behavior in Test Scripts

Whenever possible, you should avoid making your script dependent on a specific browser. Here are some tips that can help.

Property Differences

Link controls are displayed differently on Firefox and Chrome, using different font and color properties. If you design a test or component that runs a checkpoint on a Link object, and the test may run on different browsers, be sure to deselect the *font*, *color*, and *backgroundColor* properties in the checkpoint. Alternatively, you can define regular expressions for these properties, to enable the checkpoint to pass for different values.

Smart Identification

Smart Identification is a feature of UFT that uses heuristics to identify objects that could not otherwise be located. To ensure portable scripts, ensure that Smart Identification in UFT is enabled (it is enabled by default). Open the *File > Settings…* menu in UFT, and click the *Run* node in the tree. Ensure that the *Disable Smart Identification during the run session* checkbox is cleared.

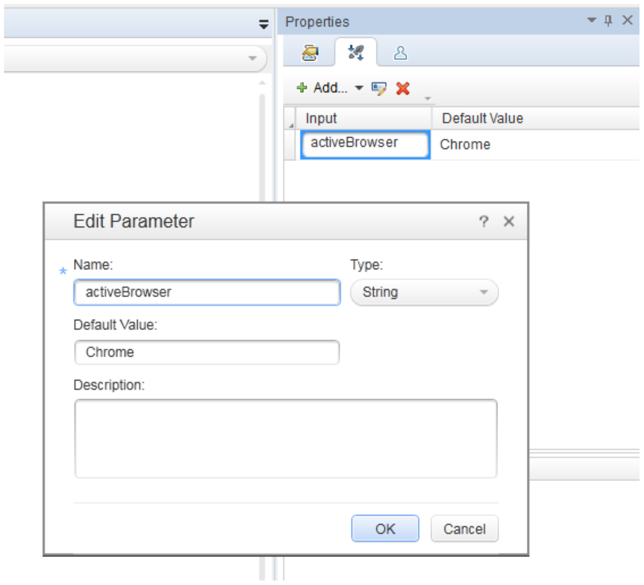
For example, the *type* attribute for a button object is *button* in Internet Explorer, but *submit* in Firefox and Chrome. So if you learn a button object on Internet Explorer, and the type attribute is added to the object's description, UFT has to use smart identification to identify the button object if you test a Web page opened with Firefox or Chrome.

Accounting for Browser-Specific Behavior

If your test requires behavior that depends on the browser currently being used, and you can't find a generic way to write the script, you can use a parameter to track which browser is currently being used. The script's code can interrogate that parameter's value to determine how to behave. The value of the parameter will typically be assigned when the test starts. It isn't important what the actual value is as long as it's unique, although it's usually best to give it a name which reflects the browser. So you can define a string called *activeBrowser*, and set it to one of:

- "Firefox"
- "Chrome"
- "IE"
- "Safari"

If the browser is specified as part of the test, the string can be declared and assigned within the test. If it is passed in from outside, the most maintainable way to implement this is to define the parameter as an Action parameter, as follows:



The value of the parameter can be queried in the script by using Parameter("activeBrowser") . Therefore, if you want to implement different behavior for different browsers, you can use the following code:

```
Select Case Parameter("activeBrowser")
   Case "Chrome"
      ' Do something specific for Chrome
   Case "IE"
      ' Do something specific for IE
   Case "Firefox"
      ' Do something specific for Firefox
   Case "Safari
      ' Do something specific for Safari
   Case Else
      ' Browser not supported
End Select
```

However, it is strongly recommended to avoid the need for this kind of code, as it makes maintenance of the test script much more difficult.

Dynamically Replacing the Object Repository

Some web applications employ a custom implementation of a page's content depending on the browser used to access the site. Although the page itself might look similar when rendered in the different browsers, the underlying implementation could be significantly different. This makes it virtually impossible to write a truly portable script. A way around this is to use the Object Repository Manager to create a separate OR for each browser. The script will then select the appropriate OR depending on the browser being used when the test runs. Once the OR has been selected, the rest of the script can be portable, and any browser-specific or implementation-dependent behavior is taken care of by the OR.

In the following example, it is assumed that there is a column in the data table for the browser, and that there is an appropriately named OR for each browser available:

```
If DataTable("browser") = "Chrome" Then
   RepositoriesCollection.Add "C:\Chrome.tsr"
ElseIf DataTable("browser") = "IE" Then
   RepositoriesCollection.Add "C:\IE.tsr"
Else
   RepositoriesCollection.Add "C:\OtherBrowser.tsr"
End If
```

Once this piece of code has been executed in the test, the rest of the script can be browser-independent, for example:

```
Browser("Search").Page("Search").WebEdit("q").Set "some string"
```

Although this enables the script itself to be portable, the disadvantage of this approach is that changes in the application will require each of the ORs to be modified, which complicates the maintenance of the test.

Executing Tests Sequentially on Multiple Browsers on the Same Machine

Situations like Standard Multi-Browser Testing typically require that the test be run once for each browser, as follows:

For each browser:

- Open the browser
- Run the test scenario to completion
- Close the browser

There are a number of different ways this can be achieved. This section describes the most common methods.

Each Script Opens and Closes its Own Browsers

In this scenario, each test script starts the same way: One or more reusable actions are called, which perform the setup for the test, and then the main test script is run. The test script is data driven via a standard global table, and each iteration runs on a different browser.

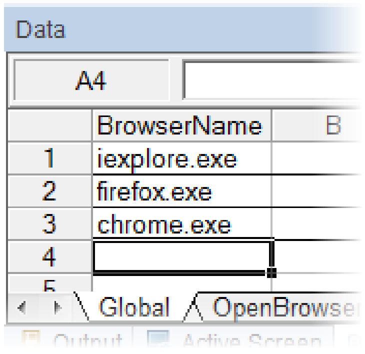
You should create a test which will be used to maintain some general reusable actions that will be used by the scripts. These scripts might include:

- Open Browser
- Login
- Close Browser

The Open Browser action will contain a line which looks something like this:

```
SystemUtil.Run DataTable("BrowserName", dtGlobalSheet), http://www.hp.com
```

This line depends on a column in the global data table called "BrowserName", which should look something like the following:



	BrowserName	B
1	iexplore.exe	
2	firefox.exe	
3	chrome.exe	
4		
5		

You will need to make sure that each test that uses these reusable actions has a column in the global data table called "BrowserName", and that this data table has the relevant values. This will need to be done manually, and if you add a new browser, you will need to add the new browser as a new row to the global table of each test.

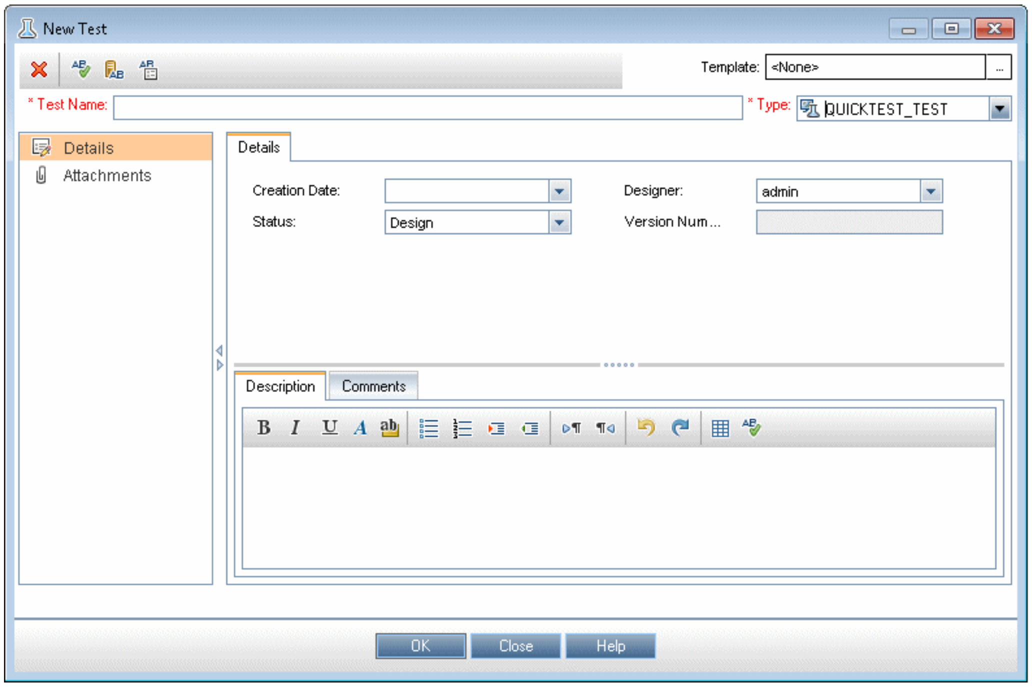
The global table will typically contain additional data that you need to maintain to account for differences between the browsers. This additional data will also need to be included in each test that uses these reusable actions.

The advantage of this method is that all of the setup and teardown is contained within each test's report, making it easier to understand exactly what happened during the test.

There are also a number of disadvantages:

- Each test needs to implement the same data in the global data table, which results in the data being difficult to maintain.
  - A way around this is to use a mechanism for having a single copy of the data, which is then associated with each test prior to it running. You can use HP ALM's Data Management feature for to create a data table in ALM, and associate the table to each test. Although some management is necessary when creating a test set, it allows a single copy of the data to be maintained.

- Each test needs to contain calls to each of the reusable actions in the correct order, in order to open the browser, set it up, and close it.
  - If you use HP ALM, you can create a 'template test' which contains all of the steps (as well as any required add-ins, function libraries and recovery scenarios) required. All new tests created from the template test will automatically be configured according to the template test:



This is the preferred method for creating tests which can run on multiple browsers on the same machine.

Scripts are Driven Externally

This method relies on a number of test scripts being run in sequence. A setup script is run which opens a specified browser. Then the test script which contains the actual test is run. Finally, a script which cleans up is run.

Mechanisms that can be employed to accomplish this include:

- Running the tests from a batch file
- Creating a test set in HP Quality Center or ALM
- Using UFT's Test Batch Runner to create a list of tests and run them in order
- Defining a job in a Continuous Integration system such as Jenkins, which runs the tests in the specified order

This has the advantage that the browser can be opened first, and then multiple test scripts can be run on the same browser.

There are again a number of disadvantages:

- There is no single report that contains all of the information from all of the test scripts that were run.
- If the tests need to pass information to each other (eg. The name of the browser that was opened), the tester will have to create a mechanism to transfer the data (eg. The test that opens the browser writes it to a file, and the script that contains the actual test reads it from the file). This process is difficult to maintain.
- Dependencies between tests are introduced. A test might rely on an earlier test putting the browser into a specific state, and this violates the principle that tests should be independent. Not only is it difficult to document these dependencies, it is also difficult to ensure that the dependencies are enforced.

This method is not recommended. Use the method described in the previous section ('Each Script Opens and Closes its Own Browsers') instead.

Executing Concurrent Tests on Multiple Browsers on the Same Machine

Some advanced concurrent testing scenarios require the test to access more than one browser on a single machine, and interact with these browsers in turn during the test. For example, a web application might require that if one browser updates data in the application, another browser should reflect the change almost immediately.

To test this, we could theoretically write two separate tests, one for each browser, and add a mechanism to synchronize the tests. But since UFT only allows one test to run at any given time on the same machine, a solution is to write a single test which consists of a series of actions as follows:

1. Action to open up the browsers
2. Multiple portable actions to perform small pieces of functionality in the browser specified by an action parameter that is passed to the action each time it's called. This will allow, for example, a 'login' action to be called twice, once for each browser
3. Action to close the browsers

The test should be repeated so that the purposes of the browsers are swapped. Therefore, for example, if we wanted to test a scenario that if Chrome is open and the user is at screen A, and then Firefox is opened and the user executes a flow that leads to screen B, we should repeat the test with the order swapped, so that Firefox would be opened at screen A, and the flow leading to screen B would be in Chrome. This is usually achieved using multiple iterations of the test, where each iteration opens the browsers in a different order.

It should be noted that scenarios involving executing browsers in parallel are not common for automated testing, and are usually tested only in organizations that have reached the most mature levels of test automation. Most organizations test these scenarios manually.

Executing Concurrent Tests on Multiple Browsers on Different Machines

There are two situations which require running tests on different machines with different browsers. This section explains them both.

Independent Tests

The requirement is to run the tests on the different machines independently, such that the test running on a browser on one machine can run independently of the tests on the other machines. Each test can start and finish in its own time, and doesn't need to wait for or synchronize with any of the tests running on the other machines.

In this scenario, the following methods can be used to execute them:

- Creating a test set in HP Quality Center or ALM, and assigning a different machine for each test in the set. Furthermore, the tests can be scheduled to run in parallel.
- Defining a job in a Continuous Integration system such as Jenkins[3], where each test can be run on a different slave node.

Dependent Tests

It becomes even more complicated if the situation involves a test that drives two browsers on two different machines, with dependencies in the test. An example of this situation might be a web application that requires that if one browser updates data in the application, another browser should reflect the change almost immediately. The implication for testing is that the test on one machine needs to wait for (and get results from) the browser on the other machine before it can continue.

One method that can be used to implement this might be to configure a master test which drives slave tests on the remote machines running the browsers, such that each of the tests on the remote machines will write a notification to some file after each step, and wait for a signal to continue. Typically, the master test will monitor the file, and wait for the remote tests to write to it. Once the remote tests have written to it, the master test provides a signal for the slaves to continue. This can be done by writing to a different file, which the remote tests monitor.

[3] See the HP Application Automation Tools plugin for Jenkins at https://wiki.jenkins-ci.org/display/JENKINS/HP+Application+Automation+Tools for more information

Note that the remote computers must be configured correctly in order to run tests on them. See the section 'How to Run Automation Scripts on a Remote Computer' in UFT's Help for details.

However, as mentioned previously, situations involving executing browsers in parallel are not common for automated testing, and are usually tested only in organizations that have reached the most mature levels of test automation.

Summary

Cross-browser functional testing is a generic term that can mean any one of many different types of test that involves more than one browser. This document has described the most common types of cross-browser tests, and presented a number of strategies and best practices to help you implement cross-browser functional testing in your organization.

Learn more at hp.com/go/uft