Checking whether page links are broken

Links are the most essential elements on a web page, as they are the connection between different sections on a page, other pages, and external pages. A link must lead to a valid **Uniform Resource Locator** (**URL**). If it leads to a non-existing or otherwise unavailable page, then it will be marked as broken.

A link that is a permanent element of a page is also called a permalink. Such a link is expected to always appear on a web page, and it will always lead to the same URL. Such a link is easy to map, either with OR, or using descriptive programming. However, in many web applications, links lead to dynamically generated pages, such as customer information, search results, and so on. Needless to say, their `href` attribute is also dynamically built, based on data that is known only during runtime. On a search results page, such as those generated by Google and other search engines, even the number of links may vary. This is also true for billing information and call details pages, which the web interface of mobile operators displays to customers.

Testing links is one of the very basic tasks that automation can tackle very efficiently, and hence, you need to free the manual tester to perform other tasks. In this recipe, we will see a very simple method to check that links on a page are not broken.

Getting ready

From the **File** menu on the UFT home page, navigate to **New** | **Function Library**, or use the *Alt* + *Shift* + *N* shortcut. Name the new function library as `Web_Functions.vbs`.

How to do it...

The seemingly obvious approach would be to get the collection of links on the page first, and then retrieve the value of the `href` attribute for each link and click on the `href` value. After the target page loads, check the URL and compare it to the original value taken from `href`. Basically, this is more or less what a manual tester would do. However, this process does not only check if a link is broken, but also checks if it is valid. This process is quite tedious and does not take into account the fact that in many cases, the value of `href` does not predict what would be the target URL. For example, the widespread usage of TinyURL!™ and redirections makes this approach impractical. Another complication is that some links load the target page on the same window and even the same tab, while others do it in a separate tab or window. While using a link is an essential part of the business flow, it is logical to actually open the new page (or navigate to the page on a new tab/window). After the target page loads, the test script can manipulate its elements and hence, continue the test flow as planned.

If, however, we just need to check that the links are not broken, then it is possible to do it using an instance of `MSXML2.XmlHttp`. In the following example, we will declare a global variable for this object and write four functions in `Web_RegisteredFunctions.vbs`:

- `DisposeXMLHttp()`: This function removes the reference to the global `oXMLHttp` variable
- `InitXMLHttp()`: This function creates an instance of `XMLHttp`, and then sets a reference to `oXMLHttp`
- `GetLinks(URL)`: This function retrieves all the links on a web page using a `Description` object

- `CheckLink(strHref)`: This function checks if a given link is broken or not

The code is as follows:

```
Dim oXMLHttp

Function disposeXMLHttp()
    Set oXMLHttp = Nothing
End Function

Function initXMLHttp()
    Set oXMLHttp = CreateObject("MSXML2.XmlHttp")
End Function

function getLinks(oPage)
    Dim oAllLinks, oDesc

    Set oDesc = Description.Create
    oDesc("html tag").value = "a|A"
    oDesc("html tag").regularexpression = true
    Set oAllLinks = oPage.ChildObjects(oDesc)

    set getLinks = oAllLinks
End function

Function checkLink(URL)
    If lcase(typename(oXMLHttp)) <> "xmlhttp" Then
        initXMLHttp()
    End If

    if oXMLHttp.open("GET", URL, false) = 0 then
        On error resume next
        oXMLHttp.send()

        If oXMLHttp.Status<>200 Then
            reporter.ReportEvent micFail, "Check Link", "Link " & URL & '
        Else
            reporter.ReportEvent micPass, "Check Link", "Link " & URL & '
        End If
    End if
End Function
```

We then run `Action1` with the following lines of code:

```
Dim i, j, oPage, oAllLinks, regex, sHref

call initXMLHttp()
'We build a filter to exclude links that are not "real", direct links but
Set regex = new RegExp
regex.pattern = "mailto:|\#|share=(facebook|google\-plus|linkedin|twitter
regex.ignorecase = true
regex.global = true

Set oPage = Browser("name:=.+").Page("title:=.+")

j=0
set oAllLinks = getLinks(oPage)
print "Total number of links: " & oAllLinks.count
For i = 0 to oAllLinks.count-1
    If oAllLinks(i).Exist(0) Then
        On error resume next
        sHref=oAllLinks(i).Object.href

        If not regex.test(sHref) Then
            j=j+1
            print j & ": " & sHref
            call checkLink(sHref)
        else
            reporter.ReportNote i & " - " & sHref & " is a mailto, sectic
        End if
        If err.number <> 0 Then
            reporter.ReportEvent micWarning, "Check Link", "Error: " & er
        End If
        On error goto 0
    else
        reporter.ReportEvent micWarning, "Check Link", oAllLinks(i).GetTC
    End If
Next
print "Total number of processed links: " & j

disposeXMLHttp()
```

How it works...

In the function library, we declared `objXMLHttp` as a variable of global scope. The `InitXMLHttp()` and `DisposeXMLHttp()`functions take care of creating and disposing the instance of the `MSXML2.XmlHttp` class. The `GetLinks(objPage)` function uses a `Description` object to retrieve the collection of all links from a page with a regular expression (HTML `a` or `A` tag). This collection is returned by the `GetLinks(objPage)` function to the calling action, where, for each item (link) in the collection, it retrieves and passes the `href` attribute to the `CheckLink(strHref)` function. The latter method checks the link by opening a connection to the URL given by `strHref` and waiting for a HTTP response to the `send` command. If the target URL is available, then the status of the HTTP response should be `200`. We also check if there is some error during the process, with `On Error Resume Next` as a precaution. (It is important to keep in mind that this may not work together with UFT's out-of-the-box settings for error handling, by navigating to **Test** | **Settings** | **Run**. This will work perfectly; using this setting, proceed to the next step). This is done because sometimes, a link that is retrieved at the start of the process may not be available when we actually wish to execute the checkpoint, as is the case with sliders and galleries with changing content.

There's more...

It is possible to further analyze the returned status with a `Select Case` decision structure to report exactly what the problem is (**404=Page not found**, **500=Internal Server Error**, and so on).

See also

For technical documentation of the `open` method of the `XMLHTTPRequest` object used in this recipe, please refer to http://msdn.microsoft.com/en-us/library/windows/desktop/ms757849(v=vs.85).aspx (http://msdn.microsoft.com/en-us/library/windows/desktop/ms757849(v=vs.85).aspx).

PREV
2. Testing Web Pa…

NEXT
Deleting cookies