Driving Selenium from Dyalog APL

Selenium (http://www.seleniumhq.org) is a widely used open-source tool for automating browsers, with growing support from browser vendors. The GitHub repository Dyalog/Selenium contains code which allows Dyalog applications to drive browsers via Selenium.

The namespace `Selenium` contained in this repository contains quite thin cover-functions for some of the most frequently used features of Selenium, exposed by the C# or Microsoft.NET bindings for Selenium. Two .NET assemblies and depending on the browser used, browser-dependent support executables, need to be installed separately, see README.md in the GitHub repo for installation instructions.

At this time, bindings for other platforms (than Microsoft.NET) are not easy to produce.

Example: Testing TryAPL

A typical example function is provided in file `Samples\TestTryAPL.dyalog` in the GitHub repo:

```
:Namespace TestTryAPL

    ∇ r←Basic;S;result
     ⍝ Verify that TryAPL is working

      S←##.Selenium
      S.InitBrowser''
      S.GoTo'http://tryapl.org'

      'APLedit'S.SendKeys'1 2 3+4 5 6'
      S.('APLedit'SendKeys Keys.Return)
      result←'ClassName'S.Find'result'

      :If '5 7 9'≡result.Text
          r←'TRYAPL is working'
      :Else
          r←'1 2 3+4 5 6 failed'
      :EndIf
    ∇

:EndNamespace
```

The above code assumes that the Selenium namespace has been loaded into the same namespace as the test namespace, so that a reference can be created using `S←##.Selenium`. It proceeds to make the following calls into the Selenium namespace:

Call	Discussion
`S.InitBrowser ''`	Ensure the browser engine is up and running
`S.GoTo 'http://tryapl.org'`	Navigate to a URL
`'APLedit' S.SendKeys '1 2 3+4 5 6'`	Simulate typing into input with id=APLedit
`S.('APLedit' SendKeys Keys.Return)`	Send a special keystroke
`'ClassName' S.Find 'result'`	Get a reference to element with id=result
`'5 7 9'≡result.Text`	Verify that the expected output was produced

Example: MiServer Regression Test Suite

One of the first uses of Selenium for Dyalog has been to create a test framework for MiServer, where each page in the sample web page has (or will have) a corresponding Selenium test script. This regression test suite has already proved to be invaluable in catching issues caused by refactoring of the MiServer core, which is still a very young piece of software. The collection of test functions can be found within the sample MiServer site, in the `MS3/QA` folder.

Option Settings

The Selenium namespace exposes three global variables:

Option and Default	Discussion
`DEFAULTBROWSER←'Chrome'`	The browser to use if `InitBrowser` is called with an empty right argument. The code has been tested with FireFox and Chrome.
`DLLPATH←'C:\Devt\Selenium\'`	Pointer to the folder containing the Microsoft.Net assemblies. You must set this to the correct value before calling InitBrowser.
`RETRYLIMIT←2000`	The number of milliseconds that the `Retry` operator should retry an operation.

Function Reference

Initialisation

`InitBrowser browser`

Attempts to creates an instance of `OpenQA.Selenium.<browser>Driver`, using the current value of `DEFAULTBROWSER` if the right argument is empty. If successful, CURRENTBROWSER is set. If subsequent calls are made to InitBrowser, and the browser would not be changed, InitBrowser attempts to verify whether the current instance is alive and reuse it rather than start a new engine.

If successful, the namespace will contain a ref to the new instance in the variable `BROWSER`.

`GoTo url`

Calls the `Navigate.GoToUrl` method and verifies that the `Url` property subsequently has the desired value. Signals an error if navigation fails.

Finding Elements

A key step in manipulating the browser DOM is extracting references to the objects that you want to operate on.

`ref←{selector} Find id`

Allows searching for DOM elements by Id (the default if no left argument is provided) or one of the other selectors: ClassName, CssSelector, Id, LinkText, Name, PartialLinkText, TagName or XPath. Instances of Selenium Driver classed support a number of methods with names in the form:

```
FindElement[s]By[selector]
```

The left argument is used to define the selector, and if a trailing ‘s’ is provided, as in:

```
'ClassNames' Find 'myclass'
```

… then the corresponding `FindElements*` function is called (in this example FindElementsByClassName), returning a collection of element references. Without a trailing s, the `FindElement*` function is called, returning a single reference. Examples of use:

Call	Discussion
`Find 'mytable'`	Return ref to the element with id mytable
`'ClassName' Find 'result'`	Find all element with class result
`'ClassSelectors' Find '#mytable td'`	All td elements in table with id mytable

`ref←id FindListItems text`

Return refs to all list items found within a list with the id given on the left, which have Text properties which can be found in the vector of text vectors on the right.

Waiting for a Reaction

The Selenium namespace contains a number of functions which manipulate the browser in various ways (see the next section for details). Interaction with a web browser is asynchronous; following any action, an unknown amount of time may pass before the server responds and a response is detectable in the browser. A Retry operator is provided to allow waiting for an expected effect.

`{ok}←(fn Retry) arg`

An operator which invokes fn on arg, and retries until the result is true, or RETRYLIMIT milliseconds have passed. For example:

```
{(Find 'result').Text=⍵} Retry 'You pressed the button!'
```

Retry returns the result of the final function application.

Browser Automation Functions

The following functions manipulate the browser in various ways.

`{selector} Click id`

Clicks on the selected element. Both arguments are passed directly to the `Find` function, and the Click method is invoked on the result. For example:

```
Click 'btn1'
```

`fromid DragNDrop toid`

Drag the element <fromid> and drop it on <toid>. Use by `ListMgrSelect`. For example:

```
('list1' FindListItems 'apples') DragNDrop 'list2'
```

`{open} ejAccordionTab (tabText ctlId)`

Ensure that the Syncfusion ejAccordionTab with the selected tabText has the desired state (open or closed), verifying the state by checking the visibility of the element with Id <ctlId>.

`id ListMgrSelect items`

In a ListManager object with Id <id>, select items with Text properties found in <items>, by dragging them from the list on the left to list on the right.

`fromid DragNDrop toid`

Drag the element <fromid> and drop it on <toid>. Use by `ListMgrSelect`. For example:

```
('list1' FindListItems 'apples') DragNDrop 'list2'
```

Other WebDriver Functionality

Instances of the Selenium WebDriver classes support significant functionality which is not covered by the existing Selenium namespace. The full set of methods and properties exposed by WebDriver.dll and WebDriver.Support.dll, which are documented here on the Selenium web site:

http://www.seleniumhq.org/docs/03_webdriver.jsp

After InitBrowser has been called, the variable BROWSER is a reference to the current instance. It exposes the following methods which are not covered by the Selenium namespace, but can be called from APL after consulting the Selenium documentation:

```
ExecuteAsyncScript  ExecuteScript  GetScreenshot  Manage  Quit  SwitchTo
```

It also exposes a number of properties, some of which look interesting:

```
AcceptUntrustedCertificates  Capabilities  CurrentWindowHandle
FileDetector  Keyboard  Mouse  PageSource  Title  Url  WindowHandles
```

There is scope for future extensions to the tool, contributions and suggestions are welcome!