# Driving Selenium from Dyalog APL

Version dated March 2nd, 2020

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](https://github.com/Dyalog/Selenium) contains code which allows Dyalog applications to drive browsers via Selenium.

The namespace Selenium contained in this repository contains cover-functions for some of the most frequently used features of Selenium. The tool uses two Microsoft.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:

∇ 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'

r←result S.WaitFor'5 7 9' '1 2 3+4 5 6 failed'

∇

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

|  |  |
| --- | --- |
| **Expression** | **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' | Reference to element with classname=result |
| result S.WaitFor  '5 7 9' '1 2 3+4 5 6 failed' | Wait until for the expected result. Return the specified error message if this doesn’t happen. |

## Example: MiServer Regression Test Suite

One of the first uses of Selenium for Dyalog has been to create a test framework for MiServer, where most pages in the sample website has 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. The MiServer documentation describes how to set up page tests for a MiSite, and the sample site “MS3” contains a large number of test functions – look for *Testing* in the documentation section of <http://miserver.dyalog.com> for more details.

# Option Settings

The Selenium namespace exposes the following 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 Chrome, Firefox, Edge and HTMLRenderer (which uses CEF, the "Chromium Embedded Framework"). |
| DEFAULTRETRYLIMIT←5 | The number of seconds that the Retry operator and its derivatives should retry an operation. |
| DLLPATH | The path where webdriver.dll, webdriver.support.dll and the .exe for the browser are located. |
| PORT | The address of the port that the browser use to retrieve content |
| [BROWSEROPTIONS] (optional) | A two-element vector which specifies the class of the object that is to be initialized with the options that are given in the JSON-specification in the 2nd element.  Example:  BROWSEROPTIONS←'Chrome.ChromeOptions' '["DebuggerAddress","127.0.0.1:8080"]' |

Underlined options can also be controlled through settings.json.

# settings.json, browser-versions etc.

The repository contains a file settings.json which has default-settings for the most frequently used browsers. Usually each new version of a browser will also require an updated \*driver.exe to control that browser. Since the driver.exe needs to be located in the same folder as the webdriver dlls, that led to a folder-structure which contains 3 files per browser & version.

The default-settings for each browser are stored in settings.json and an be activated using Selenium.ApplySettings'{NameOfConfig}'.

# 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., and also ACTIONS, which is a reference to an instance of Selenium.Interactions.Actions, used to automate mouse movements.

### GoTo url

Calls the Navigate.GoToUrl method and verifies that the Url property subsequently has the desired value. Signals an error if navigation fails. While Navigate.GoToUrl requires a complete URL, you may also specify relative URLs here (ommitting protocol, domain & port).

## 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 |
| 'CssSelectors' Find '#mytable td' | All td elements within tag with id mytable |

Find will retry the search for an element until RETRYLIMIT has elapsed, so if it takes time for a page to render completely, or an element is created by an earlier action, it will wait for a while.

### 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.

## Inspecting the Contents of the Page

The Selenium namespace contains a number of functions which manipulate the browser in various ways (see the next section for details).

### html←PageSource

You can verify whether the page has been correctly loaded by inspecting the PageSource, for example:  
  
∨/'<link href="/Styles/tryapl.css"'⍷Selenium.BROWSER.PageSource

### {ok}←(fn Retry) arg

Following any action, an unknown amount of time may pass before the server responds and a response is detectable in the browser. The Retry operator is provided to allow waiting for an expected effect. The time to wait is controlled by the global variable DEFAULTRETRYLIMIT. Retry invokes fn on arg, and retries until the result is true, or DEFAULTRETRYLIMIT seconds have passed. For example:

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

Retry returns the result of the final application of the function.

### {msg}←element WaitFor text [message]

The left argument can be the id or a reference to an element to be tested. WaitFor will use Retry to wait until the element in question contains the specified text anywhere within its Text property. If the element is of type input, (element.GetAttribute⊂'value') is tested instead.

For example:

'result' WaitFor 'Welcome!'

An optional second character vector can be used in message to replace the default, which is   
Expected output did not appear

## Browser Automation Functions

The following functions manipulate the browser in various ways. Note that, wherever an <id> is used, a reference to an IWebElement (returned by Find) can be used instead.

### {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 DragAndDrop toid

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

('list1' FindListItems 'apples') DragAndDrop '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.

### {action} MoveToElement toid [x y]

Move the mouse to the middle of an element, or optionally to the (x y) coordinates.

If the optional left argument is provided and is a character vector containing one of the strings Click | ClickAndHold | ContextClick | DoubleClick, that mouse action will be performed after the move.

For example:

'DoubleClick' MoveToElement 'btn1' 10 10

### id Select text

Select the item with a given text in a dropdown. For example:

'fruits' Select 'apples'

### id SendKeys text

Send keystrokes to an input control. For example:

'firstname' SendKeys 'Morten'

The right argument can be a special key selected from Selenium.Keys. For a list of special keys, inspect:

Selenium.Keys.⎕nl -2

It should be pointed out that this function can not be used to "send text" to an input, but it rather send keystrokes. This is imporant as soon as you want to send characters that require pressing several keys to input them, as for example APL-Characters or "@" (on a german keyboard).

### id SetInputValue text

This method is an alternative to SendKeys. While SendKeys focusses on sending actual keystrokes, this function provides an easy way to send **text** to an input control. It is recommended when sending "complex" text that may require key-combinations to enter it, for example APL-Keys or special characters like "@".

## 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

The ACTIONS variable is a reference to an instance of an Actions object, which also exposes functions not currently supported by any of the functions in the Selenium namespace:

DragAndDropToOffset KeyDown KeyUp Release

There is scope for future extensions to the tool, suggestions and “pull” requests are most welcome!

## Ending a test

As you have noticed, running a test opens a console-window showing the output of the webdriver that was launched – which will in turn launch the browser. In order to close the windows at the end of a test. it is recommended to call BROWSER.Quit which will close those windows.