Actions

start

Start an instance of a browser.

```
start firefox
start chrome
start ie
start safari
start aurora
start edgeBETA
start chromium
```

!^ (aliased by url)

Go to a url.

```
!^ "http://www.google.com"
url "http://www.google.com"
```

quit

Quit the current browser or the specified browser.

```
//quit current
quit ()
//quit specific
quit browser1
```

<< (write)

Write text to element.

```
"#firstName" << "Alex"

//if you dont like the << syntax you can alias anyway you like, eg:

let write text selector =

    selector << text
```

read

Read the text (or value or selected option) of an element.

```
let selectedState = read "#states"

let firstName = read "#firstName"

let linkText = read "#someLink"
```

click

Click an element via selector or text, can also click selenium `IWebElements`.

```
click "#login"

click "Login"

click (element "#login")
```

doubleClick

Simulates a double click via JavaScript.

```
doubleClick "#login"
```

ctrlClick

Click an element via selector or text while holding down the control key, can also click selenium `IWebElements`.

```
ctrlClick "#list > option"

ctrlClick "Oklahoma"

ctrlClick (element "#list > option")
```

shiftClick

Click an element via selector or text while holding down the shift key, can also click selenium `IWebElements`.

```
shiftClick "#list > option"

shiftClick "Oklahoma"

shiftClick (element "#list > option")
```

rightClick

Right click an element.

```
rightClick "#settings"
```

check

Checks a checkbox if it is not already checked.

```
check "#yes"

//below code will not click the checkbox again, which would uncheck it

check "#yes"
```

uncheck

Unchecks a checkbox if it is not already unchecked.

```
uncheck "#yes"

//below code will not click the checkbox again, which would check it

uncheck "#yes"
```

--> (drag is an alias)

Drag on item to another.

```
".todo" --> ".inprogress"

drag ".todo" ".inprogress"
```

hover

Hover over an element.

```
url "http://lefthandedgoat.github.io/canopy/testpages/"

"#hover" == "not hovered"

hover "Milk"

"#hover" == "hovered"
```

element

Get an element (Selenium `IWebElement`) with given css selector or text (built in waits, automatically searches through `iFrames`). Most useful if you need to write some custom helpers to provide functionality that canopy does not currently have.

```
let logoutHref = (element "#logout").GetAttribute("href")

describe ("logout href is: " + logoutHref)

//continue with your test
```

unreliableElement

Try to get an element without the built in reliability. Throws exception if element not found.

```
let logout = unreliableElement "#logout"
```

elementWithText

Unreliably get the first element with specific text for a selector.

```
let firstBob = elementWithText ".name" "Bob"
```

elementWithin

Get an element by searching within another element (nested).

```
let name = element "#header" |> elementWithin ".name"
```

someElement

Like `element` function except it runs a `Some(IWebElement)` or `None`. Read more about `Option` types here.

```
//create your own exists helper function
let exists selector =
    let someEle = someElement selector
    match someEle with
    | Some(_) -> true
    | None -> false
```

someElementWithin

Like `elementWithin` function except it runs a `Some(IWebElement)` or `None`. Read more about Option types here.

```
//create your own exists helper function
let someName = element "#header" |> someElementWithin ".name"
```

parent

Get the parent element of provided element.

```
element "#firstName" |> parent
```

someParent

Get the `Some`/`None` parent element of provided element.

```
element "#firstName" |> someParent
```

elements

The same as element except you get all elements that match the css selector or text.

```
let clickAll selector =
  elements selector
  |> List.iter (fun element -> click element)

clickAll ".button"
```

unreliableElements

The same as elements except there is no reliability. You get an empty list if there no elements on the first try.

```
let names = unreliableElements ".name"
```

unreliableElementsWithin

Try without reliability to get elements within an existing element.

```
//note that the bellow can be done (better) with selector '#people tr:first'
//the space is 'within' in css selectors
let people = element "#people"
let firstPerson = unreliableElementsWithin "tr:first" people
```

elementsWithText

Unreliably gets elements with specific text for a selector.

```
let daves = elementsWithText ".name" "Dave"
```

elementsWithin

Get elements by searching within another element (nested).

```
let names = element "#header" |> elementsWithin ".name"
```

nth

Get the nth element.

```
click (nth 4 ".button")
```

first

Get the first element.

```
click (first ".button")
```

last

Get the last element.

```
click (last ".button")
```

fastTextFromCSS

Effeciently get the text values for all elements matching a css selector.

```
let names = fastTextFromCSS ".name"
```

switchTo

Switch to an existing instance of a browser.

```
start firefox
let mainBrowser = browser
start chrome
let secondBrowser = browser
//switch back to mainBrowser after opening secondBrowser
switchTo mainBrowser
```

switchToTab

Switch browser focus between tabs.

```
1: switchToTab 2
```

closeTab

Close a specific tab.

```
1: closeTab 2
```

resize

Resize the browser to a specific size.

```
1: resize (1920, 1080)
```

rotate

Rotate the browser by switching the Height and Width.

```
1: rotate()
```

js

Run JavaScript in the current browser.

```
1: //give the title a border

2: js "document.querySelector('#title').style.border = 'thick solid #FFF467';"
```

screenshot

Take a screenshot and save it to the specified path with specified filename. Returns image as byte array.

```
1
: let path = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + @"\c
  anopy\"
2
: let filename = DateTime.Now.ToString("MMM-d_HH-mm-ss-fff")

3 screenshot path filename
:
```

sleep

Sleep for `x` seconds.

```
1: //sleep for 1 second

2: sleep ()

3: //sleep for 1 second

4: sleep 1

5: //sleep for 3 seconds

6: sleep 3
```

highlight

Place a border around an element to help you identify it visually, used in wip test mode.

```
1: highlight ".btn"
```

describe (aliased as puts)

Describe something in your test, currently writes description to console.

```
1: describe "on main page, testing logout"
```

waitFor

Wait until custom function is true (better alternative to sleeping `x` seconds).

```
1: let fiveNumbersShown () =
```

```
2:     (elements ".number").Length = 5
3:
4: url "http://somepage.com/countdown"
5: waitFor fiveNumbersShown
6: //continue with your test
```

waitFor2

Wait (with message) until custom function is true (better alternative to sleeping `x` seconds).

```
1: let fiveNumbersShown () =
2:     (elements ".number").Length = 5
3:
4: url "http://somepage.com/countdown"
5: waitFor2 "waiting for five numbers to be shown" fiveNumbersShown
6: //continue with your test
```

waitForElement

Wait until an element with a given CSS selector appears in the DOM. This is useful when you need to wait for data being loaded and displayed.

```
1: url "http://somepage.com/countdown"
2: waitForElement ".number"
3: //continue with your test
```

clear

Clear the text of an element.

```
1: clear "#firstName"
```

press

Simulate a key press. Other keys can be sent by first importing OpenQA.Selenium and using the keys defined there.

```
1: press tab

2: press enter

3: press down

4: press up

5: press left

6: press right

7: press esc

8:

9: open OpenQA.Selenium

10: press Keys.Backspace
```

alert

Gets the current alert.

```
1: alert() == "Welcome to Test Page!"
```

acceptAlert

Accepts the current alert.

```
1: acceptAlert()
```

dismissAlert

Dismiss the current alert.

```
1: dismissAlert()
```

pin

Pin a browser to the left, right, or fullscreen (any browser you start is pinned right automatically).

```
pin Left
pin Right
pin FullScreen
```

tile

Tile listed browsers equally across your screen. 4 open browsers would each take 25% of the screen.

```
start chrome
let browser1 = browser
start chrome
let browser2 = browser
start chrome
let browser3 = browser

tile [browser1; browser2; browser3]
```

positionBrowser

Position current browser on the screen - position is in percentages: positionBrowser left top width height

```
positionBrowser 66 0 33 50
```

currentUrl

Gets the current url.

```
let u = currentUrl ()
```

title

Gets the title of the current page.

```
let theTitle = title ()
```

reload

Reload the current page.

```
reload ()
```

navigate

Navigate forward or back.

```
navigate back

navigate forward
```

addFinder

Add a finder to the list of current finders to make your selectors cleaner.

```
let findByHref href f =
    try
        let cssSelector = sprintf "a[href*='%s']" href
        f(By.CssSelector(cssSelector)) |> List.ofSeq
    with | ex -> []

addFinder findByHref
```

Fast selectors

Skip looking through the list of finders for a specific selector, use a specific function.

```
css ".name"
xpath "//div/span"
jquery ".name:first"
```

```
4: label "First Name"

5: text "Last Name"

6: value "Submit"
```

failsWith

Expect a failure with a specific message and pass test if it occurs

```
1: failsWith "An expected error message"
```

Assertions

== (equals)

Assert that the element on the left is equal to the value on the right.

```
1: "#firstName" == "Alex"
```

!= (does not equal)

Assert that the element on the left is not equal to the value on the right.

```
1: "#firstName" != "Tom"
```

=== (aliased as is)

Assert that the value on the left is equal to the value on right. * Note: does not use retry-ability. Equivalent to Assert.Equals.

```
1: "Not a selector" === "Not a selector"
```

*= (one of many equals)

Assert that at least one element in a list equals a value.

```
1: ".todoItem" *= "Buy milk"
```

*!= (none equals)

Assert that none of the items in a list equals a value.

```
1: ".todoItem" *!= "Sell everything"
```

contains

Assert that one string contains another.

```
1: contains "Log" (read "#logout")
```

containsInsensitive

Assert that one string contains (case insensitive) another.

```
1: containsInsensitive "Log" (read "#logout")
```

notContains

Assert that one string does not contains another.

```
1: notContains "Hello Bob!" (read "#name")
```

count

Assert there are `x` items of given css selector.

```
1: count ".todoItem" 5
```

=~ (regex match)

Assert that an element `regex` matches a value.

```
1: "#lastName" << "Gray"
```

```
2: "#lastName" =~ "Gr[ae]y"

3: "#lastName" << "Grey"

4: "#lastName" =~ "Gr[ae]y"
```

!=~ (regex match)

Assert that an element does not `regex` match a value.

```
1: "#lastName" << "Gr0y"

2: "#lastName" !=~ "Gr[ae]y"

3: "#lastName" << "Gr1y"

4: "#lastName" !=~ "Gr[ae]y"
```

*~ (one of many regex match)

Assert that one of many element `regex` matches a value.

```
1: "#colors li" *~ "gr[ea]y"
```

on

Assert that the browser is currently on a url. Falls back to using `String.Contains` after page timeout.

```
1: url "https://duckduckgo.com/?q=canopy+f%23"

2: on "https://duckduckgo.com/?q"
```

onn

Same as `on` but does not fall back to using `String.Contains`.

```
1: url "https://duckduckgo.com/about"

2: onn "https://duckduckgo.com/about"
```

selected

Assert that a radio or checkbox is selected.

```
1: selected "#yes"
```

deselected

Assert that a radio or checkbox is not selected.

```
1: deselected "#yes"
```

displayed

Assert that an element is displayed on the screen. (Note: Will not walk up the dom. If a parent container is hidden this may give the wrong results, try adding :visible to selector)

```
1: displayed "#modal"
```

notDisplayed

Assert that an element is not displayed on the screen. (Note: Will not walk up the dom. If a parent container is hidden this may give the wrong results, try adding :visible to selector)

```
1: notDisplayed "#modal"
```

enabled

Assert that an element is enabled.

```
1: enabled "#button"
```

disabled

Assert that an element is not enabled.

```
1: disabled "#button"
```

fadedIn

Returns true/false if element has finished fading in and is shown.

```
1: let isShown = (fadedIn "#message")()

2: waitFor <| fadedIn "#message"
```

Configuration

driverHostName

- .net core has a delay when resolving localhost that makes using canopy very slow
- canopy uses 127.0.0.1 instead but allows you to adjust it based on your ipv/ipv6 needs
- Defaults to "127.0.0.1"

```
1: driverHostName <- "localhost"

2: driverHostName <- "::1"
```

chromeDir

- Directory for the chromedriver
- Defaults executing directory (bin\Debug in many cases)

```
1: chromeDir <- @"C:\your\custom\path"
```

chromiumDir

- Directory for the chromedriver for use with chromium
- Defaults to pre-set OS paths
- nix: /usr/lib/chromium-browser
- Windows: C:\

```
1: chromiumDir <- "C:\\"
```

firefoxDir

- Install path for firefox
- Defaults to pre-set OS paths
- OSX: /Applications/Firefox.app/Contents/MacOS/firefox-bin
- nix: /usr/lib/firefox/firefox
- Windows: C:\Program Files (x86)\Mozilla Firefox\firefox.exe

```
1: firefoxDir <- @"C:\Program Files (x86)\Mozilla Firefox\firefox.exe"
```

firefoxDriverDir

- Directory for the firefox gecko driver
- Defaults executing directory (bin\Debug in many cases)

```
1: firefoxDriverDir <- @"C:\your\custom\path"
```

ieDir

- Directory for Internet Explorer
- Defaults executing directory (bin\Debug in many cases)

```
1: ieDir <- @"C:\your\custom\path"
```

safariDir

- Directory for Safari
- Defaults executing directory (bin\Debug in many cases)

```
1: safariDir <- @"C:\your\custom\path"
```

edgeDir

- Directory for edge driver
- Defaults to C:\Program Files (x86)\Microsoft Web Driver\

```
1: edgeDir <- @"C:\your\custom\path"
```

hideCommandPromptWindow

- Hide drivers command prompt window
- Defaults to false

```
1: hideCommandPromptWindow <- true
```

elementTimeout

- Amount of time for the test runner to search for an element.
- Default is 10.0 seconds

```
1: elementTimeout <- 10.0
```

compareTimeout

- Amount of time for the test runner to spend comparing elements.
- Default is 10.0 seconds

```
1: compareTimeout <- 10.0
```

pageTimeout

- Amount of time for the test runner to wait for the page to load.
- Default is 10.0 seconds

```
1: pageTimeout <- 10.0
```

wipSleep

- Amount of time to spend between WIP tests (Tests marked with &&&&)
- Default is 1.0 seconds

```
1: wipSleep <- 1.0
```

failIfAnyWipTests

- Prevents accidentally allowing wip tests into the build pipeline.
- Set to false locally so tests under development are not affected.
- Set to true in your CI environment to catch wip tests that have been mistakenly commited to trunk/master.
- Default is false

```
1: failIfAnyWipTests <- true
```

runFailedContextsFirst

- Runs failed contexts first if the test suite has already executed.
- Defaults is false

```
1: runFailedContextsFirst <- false
```

failFast

- Stop running test suite after one test fails.
- Defaults is false

```
1: failFast := true
```

failScreenshotPath

- The path to save screenshots that are taken on failure.
- Defaults is AppData\canopy

```
1: failScreenshotPath <- "C:\path\to\screenshot\folder"
```

failScreenshotFileName

- Function that is run to create the filename of screenshot.
- Takes a test and a suite to be optionally used in name creation.
- Defaults is Timestamp with format MMM-d_HH-mm-ss-fff

```
1 failScreenshotFileName <- fun test suite -> System.DateTime.Now.ToString("MMM-d_HH-mm"
: )
```

reporter

- Reporter object that will handle how logs should be stored.
- Must inherit IReporter
- Default is ConsoleReporter

```
1: reporter <- new reporters.ConsoleReporter() :> reporters.IReporter
```

disableSuggestOtherSelectors

- Option that will disable selector suggestion if a selector fails to execute
- Defaults is false

```
1: disableSuggestOtherSelectors <- false
```

autoPinBrowserRightOnLaunch

- Automatically pins the browser to the right of the screen on launch
- Default is true

```
1: autoPinBrowserRightOnLaunch <- true
```

throwIfMoreThanOneElement

- Throws a CanopyMoreThanOneElementFoundException if more than one element is found using a selector
- Default is false

```
1: throwIfMoreThanOneElement <- false
```

configuredFinders

- Defines functions for finding elements based on selectors
- Default is the following sequence
- findByCss
- findByValue
- findByXpath
- findByLabel
- findByText
- findByJQuery

```
1: configuredFinders <- finders.defaultFinders
```

optimizeBySkippingIFrameCheck

- If you need your tests to be faster and don't have any iframes you can turn this to true
- Default is false

```
1: optimizeBySkippingIFrameCheck <- false
```

showInfoDiv

- Allows information to be displayed on the browser when the puts function is called
- Default is true

```
1: showInfoDiv <- true
```

failureScreenshotsEnabled

- Enables/Disables automatic taking of screenshot on failures.
- Default is true

```
1: failureScreenshotsEnabled <- false
```

skipAllTestsOnFailure

- Like fail fast, but instead of not running tests, it skips them.
- Default is false

```
1: skipAllTestsOnFailure <- true
```

skipRemainingTestsInContextOnFailure

- Like skipAllTestsOnFailure, but only skip those in the current context.
- Default is false

```
1: skipRemainingTestsInContextOnFailure <- true
```

skipNextTest

- Skip the next test
- Default is false

```
1: skipNextTest <- true
```

failureMessagesThatShoulBeTreatedAsSkip

- Mark a failed test as skipped if it failed for any of the listed known reasons
- Default is empty list

```
1: failureMessagesThatShoulBeTreatedAsSkip <- ["message 1"; "message 2"]
```

webdriverPort

- Allow specifying a port on which the **WebDriver** instance wil start (instead of a random one)
- Defualt is `None` and it **must** be an `Option` type, i.e. `None` or `Some x`
- Do **NOT** use if running tests in parallel!

```
1: webdriverPort <- Some 4444
```

acceptInsecureSslCerts

- Allow the driver to navigate to sites with self-signed SSL certificates
- Crucial for Chrome Headless testing
- Default is true

```
1: acceptInsecureSslCerts <- true
```

Testing

run

Starts test suite after is defined. Usually at the bottom of your Program.fs

```
1: run()
```

runFor

Starts test suite and runs the suite with each of the listed browsers. Usually at the bottom of your Program.fs

```
1: runFor [chrome; firefox; ie]
```

context

Define the context of the tests. A default context is defined and used if one is not provided. You can have as many contexts as you like. Each context gets a new once/before/after/lastly function.

```
1: context "Login page tests"
2: //some tests
3:
4: context "Search page tests"
5: //different tests
6:
7: context "Reset password page tests"
8: //different tests
```

once

Function that is run once time at the beginning of a test suite. (per context)

```
1: once (fun _ ->
```

```
2:     ()//do this one time at the beginning of the most recently defined context
3: )
```

before

Function that is run before each test in a context. (per context)

```
1: before (fun _ ->
2:     ()//do this before every test of the most recently defined context
3: )
```

after

Function that is run after each test in a context. (per context)

```
1: after (fun _ ->
2:     ()//do this after every test of the most recently defined context
3: )
```

lastly

Function that is run once at the end of a context. (per context)

```
1: lastly (fun _ ->
2:     ()//do this after the very last test of the most recently defined context
3: )
```

onPass

Function that is run after a test passes. (per context)

```
1: onPass (fun _ ->
2:     ()//do this after a test passes
```

```
3: )
```

onFail

Function that is run after a test fails. (per context)

```
1: onFail (fun _ ->
2:     ()//do this after a test fails
3: )
```

test

Standard test definition (name defined automatically by the test number eg: Test #1).

```
1: test (fun _ ->
2:     //go somewhere
3:     //interact with page
4:     //assert
5:     ()
6: )
```

&&& (named test, aliased as ntest)

Standard test definition with a name.

```
1: "go somewhere, do some stuff, assert" &&& fun _ ->
2:     //go somewhere
3:     //interact with page
4:     //assert
5:     ()
6: //Or
7: ntest "go somewhere, do some stuff, assert" (fun _ ->
```

```
 8:     //go somewhere
 9:     //interact with page
10:     //assert
11:     ()
12: )
```

&&&& (work in progress, aliased as wip)

Used for debugging. Test runs slower and highlights the elements that it is interacting with to help debug. If one test is marked wip, only wip tests are ran. Other tests are skipped.

```
 1: //this test is run slow and only with other tests marked as wip
 2: "go somewhere, do some stuff, assert" &&&& fun _ ->
 3:     //go somewhere
 4:     //interact with page
 5:     //assert
 6:     ()
 7: //Or
 8: wip (fun _ ->
 9:     //go somewhere
10:     //interact with page
11:     //assert
12:     ()
13: )
```

&&! (skip, aliased as xtest)

Do not run a test.

```
1: //skipped
```

```
 2: "go somewhere, do some stuff, assert" &&! fun _ ->
 3:     //go somewhere
 4:     //interact with page
 5:     //assert
 6:     ()
 7: //Or
 8: xtest (fun _ ->
 9:     //go somewhere
10:     //interact with page
11:     //assert
12:     ()
13: )
```

&&&&& (always run, in both standard and wip modes)

Test will always be run. If some tests are marked work in progress, tests marked as always will also run. The test will run slow with wip tests, but run at normal speed when there are no wip tests.

```
1
:
  //this test is run slow with wip test and regular speed with standard tests, test will
2 always run.
:
  "go somewhere, do some stuff, assert" &&&&& fun _ ->
3
:     //go somewhere
4     //interact with page
:     //assert
5     ()
:
6
:
```

many

Run a single test X times. Helps with troubleshooting tests that sometimes fail.

```
many 20 (fun _ ->
    //go somewhere
    //interact with page
    //assert
    ()
)
```

nmany

Run a single named test X times. Helps with troubleshooting tests that sometimes fail.

```
nmany 20 "description" (fun _ ->
    //go somewhere
    //interact with page
    //assert
    ()
)
```

todo

Mark a test as todo to fill in later. LiveHtmlReporter will mark todo tests in the output.

```
"go somewhere, do some stuff, assert" &&& todo
```