DollarX Documentation

Release 1.2.1

Danny Loya

Feb 13, 2022

Contents

`DollarX` is a library dedicated for testing web applications, meant to simplify interactions with the browser and assertions, making it significantly more maintainable, while optimizing performance and minimizing race-conditions. It relies on Selenium WebDriver.

CHAPTER 1

Features

- Thoughtful, expressive API to define W3C elements and interact with the browser
- Eliminates race conditions and optimizes performance
- Easy to use and troubleshoot
- Works on top of Selenium and integrates easily with its API
- Easily extensible using utility functions
- Includes a collection of custom Hamcrest matchers, that are optimized and provide useful error messages
- Support for visual testing
- Two flavours: Standard API supports multiple browser instances. Simplified API supports a single instance of browser.
- Includes Java and Scala implementation (DollarX - Scala, JDollarX - Java)

CHAPTER 2

Example

Basic example:

```
//Boilerplate
InBrowserSinglton.driver = DriverSetup.createStandardChromeDriver(); // assuming we␣
↪have such utility function
InBrowserSinglton.driver.get("http://xxx.yyy.zzz");

// definitions
Path carouselItem = div.withClasses("carousel-cell");
Path viewableItem = carouselItem.withClass("is-selected");

// action
dragAndDrop(firstOccurrenceOf(viewableItem)).to(lastOccurrenceOf(viewableItem));

// assertion
assertThat(viewableItem, isPresent(4).timesOrMore());
```

CHAPTER 3

Introduction

DollarX has 3 components:

- Definitions of `Paths` that defines DOM element
- Actions in the browser
- Assertions

As well as several minor components:

- Visual capture/testing
- AgGrid interactions and assertions
- High-level API - for common interactions with various input elements

CHAPTER 4

DOM Path builder based on XPath

- Flexible API that allows to declare complex xpath easily, and creates immutable objects
- Grammar is intuitive and similar to English
- Easy to troubleshoot, since toString() of an element is basically English text
- Supports almost any relevant xpath(1.0) expression
- Expandable easily using utility functions
- “Collaborates” with standard Selenium WebElements
- Can be used independently from the other DollarX components

CHAPTER 5

Interactions with the browser

- Relies on the Path Builder
- Two flavors:
  1. The standard, with multi-browser instances support.
  2. A simplified API package, for a single browser instance

CHAPTER 6

Custom Matchers:

- Relies on the Path Builder, Browser interactions.
- Extends Hamcrest and ScalaTest matchers
- Optimized for performance and atomicity (minimize race condition issues)
- Provides useful error message for failures
- Two flavors, similarly to the interaction with the browser
- The general purpose, standard version, supports assertions in both the browser and a given W3C document

CHAPTER 7

JavaDoc

See standard JavaDoc here

CHAPTER 8

Contents:

8.1 Background

Testing web applications is a challenge in multiple dimensions, even when ignoring the testing framework itself:

1. Writing correct code (learning curve, a lot of pitfalls)
2. Readability and maintainability
3. Race conditions and other gotchas
4. Performance - especially as the number of scenarios increases
5. Assertions - typically not expressive. Failure create useless errors.
6. Logging/Troubleshooting

8.1.1 Test cases

First Scenario

Let's say we that in our web application we have a scenario in which have a list of names, and we want to assert that "John" is in the list. Naive implementation:

```
List<WebElement> els = driver.findElements(By.cssSelector("li.name"));
List<WebElement> filtered = els.stream().filter(el ->
                              el.getText().equals("John")).
                              collect(toList());
assertThat(filtered, not(empty()));
```

This code looks innocent and reasonable, but has serious problems:

1. If there are 100 elements in the list, it will access the browser 101 times - extremely inefficient.
2. It retrieves potentially many elements that are not needed. Again, inefficient.
3. Race condition can lead to false negative - If the list of names is not empty, but the entry with the name "john" appears after a short delay, the first line will return immediately and we will miss "john", although it is there
4. What if "john" is actually inside an element wrapped by the "li"? We will miss it, since we only examine the "li".
5. If the list updates during execution, some of the elements we have a reference to, might disappear, resulting in a "StaleElementException" being thrown
6. The error message of the assertion, in case of a failure, is: "expected not empty, but was empty". This is not useful.

Besides these, it is also quite brittle to use a string for the selector of the elements.

To illustrate how insidious seemingly innocent code can be, let's examine 2 examples.

Second Scenario

Let's say that in our application there is a large table with a "clear" button, and we want to assert it worked properly. Naive implementation:

```
List<WebElement> all = driver.findElements(By.xpath...);
assertThat(all, is(empty());
```

Again, this code seems reasonable, but has serious issues:

1. Consider the failure (rare) scenario: we retrieve potentially many elements, and not do anything with them - clearly inefficient, but at least does not block.
2. Consider the success (common) scenario: the first line will block for several seconds until reaching the timeout and giving up. Again - very inefficient.
3. What if it takes a short time for the elements to clear? findElements() will return all the elements, and we will get a false failure.
4. Assertion error message is almost useless, without context

Finding elements

Selenium offers several way to find elements. The most commonly used are CSS selector, and Xpath. Xpath is significantly more expressive, thus generally a better solution. The problem is that it has a tendency to be complicated and brittle. For example, an xpath for a DIV element with class "foo", is:

```
"//div[contains(conat(' ', normalize-space(@class), ' '), ' foo ')]"
```

Using a CSS selector instead is much simpler, but CSS is more limited. Besides, even using CSS can be non-trivial. Ideally, we want an API that combines the expressiveness of xpath, but be intuitive and simple (div.withClass("foo")).

8.1.2 DollarX

The goal is to minimize the challenges described above, and abstract the complexity. Let's reimplement the examples above with DollarX and analyze it.

First Scenario

```
assertThat(
     listItem.withClass("name").and(hasText("John")),
     isPresentIn(browser));
```

Let's re-examine the concerns in the previous implementation:

1. Even there are 100 elements in the list, it will access the browser only once, eliminating the previous implementation inefficiency
2. It finds at most a single element from the browser, eliminating the previous implementation inefficiency
3. If it takes the element with text "John" a short time to appear, it will wait until it appears, avoid the race condition issue in the previous implementation
4. What if "john" is actually inside an element wrapped by the "li"? we could use "hasAggregatedText" instead of "hasText".
5. Since the interaction with the browser is atomic, there is no chance of encountering "StateElementException".
6. In case of assertion error, the output is:

   "list item, that has class "name" and has the text "John" is expected to be present, but is absent"

   This is much more useful.

Second Scenario

```
Path row = listItem.withClass("table-row").describedBy("row");
assertThat( row, isAbsentFrom(browser));
```

Let's re-examine the previous implementation issues:

1. Consider the failure (rare) scenario: it will block until it can't find a DOM without this element.
2. Consider the success (common) scenario: It will look for a DOM without this element and returns immediately once it is true.
3. What if it takes a short time for the elements to clear? Again, since isAbsent looks for a page without this element, it will behave correctly and will not be sensitive to race conditions
4. Assertion error message, in case of failure is: "row is expected to be absent, but is present". This message is useful.

8.1.3 Summary

The following anti-patterns are common when writing assertions in the browser:

1. Find all elements, then iterate over them looking for something, or take the nth element
2. Find an element, then look for an element under it, or with another relation to it.
3. Variation: The assertion involves several elements, so look for each of them separately
4. Use various "Sleep" statement to mitigate race conditions
5. Find all elements, and verify size is 0 (or: n, >n, <n)
6. "Enhance" the DOM to make it easier to test, thus changing behavior. This should be done judiciously.

The general approach to deal with it can be to write complicated XPath to find exactly what we expect atomically. The problem with this approach is that xpath is very brittle and complicated. This is where Dollarx comes into the picture. It allows to create an arbitrarily complex XPath that is much easier to build, understand and maintain. Thus it uses the power of XPath but abstract away its challenges.

8.2 Paths

8.2.1 Introduction

`Path` is how an DOM element/elements is defined in DollarX. It allows to build an arbitrarily complex path to an element, using xpath internally.

Why xpath?

Xpath is significantly more expressive than CSS. CSS is especially limited when trying to describe relationships between elements. Even something as simple a div that contains a span can't be expressed in CSS.

Why not use raw xpath?

Raw xpath is difficult to create, understand, troubleshoot and maintain. Although it is expressive, it is quite difficult to work with.

8.2.2 Building Blocks

Note that a path instance is immutable. Any function on it creates and returns a new instance.

The standard implementation is `BasicPath`. It includes predefined elements and allows to create new ones.

Suppose you have a `Path` el. You can declare another `Path` based on it by applying the following:

- Add a property, using a `Path.that`or `Path.and` clause. Simple examples:

```
el.that(hasClass("abc"));
el.that(hasClass("abc"), hasText("John"));
el.that(hasClass("abc").and(hasText("John");
el.that(hasClass("abc").or(hasText("John"));
el.that(not(hasText("John"));
```

  Some common properties can be used directly, without a `Path.that` clause. For example:

```
el.withClass("abc").withText("joe");
```

- Describe which occurrence of that element it is, or its index among its siblings. For example:

```
firstOccurrenceOf(el.withClass("abc").withText("joe"));
lastOccurrenceOf(el.withClass("abc").withText("joe"));

occurrenceNumber(4).of(el.withText("joe"));

childNumber(4).ofType(div.withClass("foo"));
```

- Describe logical operation with another Path element. For example

```
el1.or(el2);  //  matches both

PathOperators.not(el); // mathes any element except for el
```

- Add a relation to another path. Some examples:

```
el.inside(div);
div.contains(el);

el2.after(el1);

el2.beforeSibling(el1);
el2.immediatelyBeforeSibling(el1);
```

An `ElementProperty` used in a `Path.that` clause can be elaborate. It can be: #. a simple property, for example:

```
el.that(hasClass("abc"));

el.that(hasTextContaining("joe"));

el.that(hasId("123")).and(hasText("foo"));
```

  A `Path.that` clause supports multiple properties, as well as logical operations. For example:

```
el.that(hasClass("abc), hasId("123"));

el.that(hasClass("abc).and(hasId("123"));

el.that(hasClass("abc").or(hasClass("123"));
```

  The list of supported properties is long. Take a look at `ElementProperties` for details.

- A relation to another element. Examples:

```
el.that(isInside(otherEl));
```

The list of supported properties of Paths is long. Please refer to the javadoc of `ElementProperties` and `BasicPath`.

There are also easy ways to extend DollarX to support new properties. See the *recipes* section for detail.

8.2.3 Single Browser Instance Paths

Besides `BasicPath`, there is another implementation of Path, specifically for the case there is a single instance of browser we connect to. It add some actions in the browsers that can be used in an OOP way, such as el.click().

This class is `SingleBrowserPath`.

8.2.4 String representation of Paths

One of the useful features in DollarX is the representation of `BasicPath`as string. It is clear, and in many cases English-like representation. This makes troubleshooting/debugging easier. For examples, look at the *recipes* .

The describedBy function

When creating a path the relies on the definitions of other path, the description as strings can be complicated.

`Path.describedBy` allows to provide an alias description, which can be useful to simplify it.

For example:

```
Path thePasswordInput = input.inside(div.afterSibling(label.withText("password")).
↪describedBy("the password input");
 System.out.println(thePasswordInput);
// "the password input"

Path contactsTable = div.withClasses("ag-table", "contacts");
Path row = div.withClass("ag-row");

Path contact = row.inside(table).describedBy("contact");

System.out.println(contact.that(hasAggregatedTextContaining("john smith")));
// output: contact, with aggregated text containing "john smith"
```

This is useful when an exception is thrown or when you have assertions failures.

8.2.5 Predefined elements

Under `BasicPath`, there are many element types that are defined and can be statically imported. See the JavaDoc of BasicPath. If you need to create a new type of element, look at the *recipes*.

8.2.6 Relations to other elements

The following is a list of supported Path element properties that related to other elements. In `ElementProperties`(see JavaDoc for details):

- hasChildren - has at least one child

- hasNoChildren
- isOnlyChild
- hasChild, isParentOf - the element is the direct parent of another element(s). The methods are equivalent.
- isChildOf, hasParent - the opposite of hasChild, isParentOf
- **contains, hasDescendant - contain one or more elements. Two flavors:**
  - Accept one or more elements
  - With a limit on the count of the elements. Such as: contains(exactly(3).occurrencesOf(myElement))
- hasAncestor, isContainedIn - is contained within another element
- **isAfter. is after another elements in the DOM. 2 flavors:**
  - Accept one or more elements
  - With a limit on the count of the elements. Such as: isAfter(exactly(n).occurrencesOf(div)) . The limit can be: exactly, atMost, atLeast .
- isBefore - the opposite of isAfter
- **isSiblingOf. 2 flavors:**
  - Accept one or more elements
  - With a limit on the count of the elements. Such as: isAfterSibling(atLeast(2).occurrencesOf(div))
- isAfterSibling - 2 versions, as in isSiblingOf
- isBeforeSibling - 2 versions, as in isSiblingOf
- isWithIndex, isNthSibling - states the index of the element among its siblings. 0 is first.
- withIndexInRange - similar to isWithIndex, but allows to provide a range
- isLastSibling
- isNthFromLastSibling - states the place of the element from its last sibling. 0 is last.

In addition, the following relation properties are in In `BasicPath`:

- `BasicPath.childOf` - similar to In `ElementProperties.isChildOf`
- `BasicPath.parentOf` - the opposite of `BasicPath.childOf`
- `BasicPath.contains`, `BasicPath.ancestorOf`
- `BasicPath.inside`, `BasicPath.descendantOf` - the opposite of `BasicPath.contains`
- `BasicPath.childNumber` - similar to `ElementProperties.isNthSibling`, but first is 1. For example: childNumber(4).ofType(div.withClass("foo"))
- `BasicPath.occurrenceNumber` - the global occurrence of a given path in the DOM, starting with 1 for example: occurrenceNumber(3).of(listItem)
- `BasicPath.withGlobalIndex` -similar to `BasicPath.occurrenceNumber`, but a different syntax, and first is 0. el.withGlobalIndex(n) is an alias for occurrenceNumber(n + 1).of(el)
- `BasicPath.firstOccurrenceOf` - first occurrence of this element in the DOM
- `BasicPath.lastOccurrenceOf` - last occurrence of this element in the DOM
- `BasicPath.beforeSibling` - is a sibling of the given path parameters and appears before it
- `BasicPath.immediatelyBeforeSibling` - is before sibling and adjacent (right before)

- `BasicPath.afterSibling` - is a sibling of the given path parameters and appears after it
- `BasicPath.immediatelyAfterSibling` - is after sibling and adjacent (right after)

8.2.7 Common Properties

See the *recipes* section for more detail.

Properties related to CSS classes under `ElementProperties`:

- hasClass, hasClasses
- hasClassContaining
- hasNonOfTheClasses
- hasAnyOfClasses

Properties related to text under `ElementProperties`:

- hasSomeText
- hasText
- hasTextContaining
- hasTextEndingWith
- hasTextStartingWith
- hasAggregatedTextEqualTo
- hasAggregatedTextContaining
- hasAggregatedTextStartingWith
- hasAggregatedTextEndingWith

Logical operations on properties:

- `ElementProperty.and`
- `ElementProperty.andNot`
- `ElementProperty.or`
- `ElementProperties.not`

Attributes:

- isEnabled/isDisabled
- isSelected
- isChecked

8.3 Actions

There are 2 versions of the API for browser operations: The standard one, that supports multiple connected browser instance, under `InBrowser`, and the singleton browser version, under `InBrowserSinglton`.

The difference is that the single-browser-instance API is simplified. For example:

```
// standard interface:
// Assume we have: InBrowser browser = .... ;
browser.clickAt(myElement);
browser.dragAndDrop(myElement).to(otherElement);

// Single browser version:
// assume we have static imports from InBrowserSingleton
clickAt(myElement);
dragAndDrop(myElement).to(otherElement);
```

8.3.1 Supported Operations

Supported operations include:

- `InBrowser.clickOn`, `InBrowser.clickAt` - click an element. The difference between the two is that the former expects the element to be clickable.
- `InBrowser.doubleClickOn`
- `InBrowser.dragAndDrop`- to another element or to a location by coordinates
- `InBrowser.scroll`- scroll up,down,left or right.
- `InBrowser.scrollTo`- scroll to the location of an element
- `InBrowser.scrollElement`- scroll within an element. Useful for grid, especially when they are virtualized
- `InBrowser.pressKeyDown`, `InBrowser.releaseKey`- on an element or in the browser in general
- `InBrowser.sendKeys`- send text to element or to browser in general
- `InBrowser.hoverOver` - hover over an element

8.3.2 Select Menu

Since the Select API in Selenium is good for most use cases, dollarx did not re-implement it. Instead, to use it, you would do something like:

```
//For clarity, I avoided any static imports. In real life, I encourage using them.

// my dropdown menu is the first "select" element in the form.
Path selectElement = BasicPath.select.inside(BasicPath.form);

Select myDropdownMenu = InBrowserSingleton.getSelect(selectElement);

// use the Selenium Select API ...
myDropdownMenu.selectByVisibleText("apple");
```

8.4 Assertions

DollarX includes custom Hamcrest Matchers to allow to create assertions. Their benefits are:

- Expressiveness and readability. For example:

```
assertThat(myElement, isPresent(1000).timesOrMore());
```

- Useful error messages. For example, suppose we have the following assertion:

```
assertThat(div.withText("flex-item"), isPresent(1000).timesOrMore());
```

  and there are only 4 elements with "John". The assertion error will be:

```
Expected: browser page contains (div, that has class flex-item) at least␣
↪1000 times
   but: (div, that has class flex-item) appears 4 times
```

- Optimized for atomicity and speed. For example, the previous assertion will construct a single XPath that find a DOM with at least 1000 elements that we look for. This improves correctness and performance.

  Another example: isAbsent() matcher, constructs an xpath that finds a DOM without the element that is expected to be absent. This means that in case of success (the common case), it returns immediately, while the standard way of calling WebDriver.find will block for several seconds, until timeout is reached and then it will throw an exception that will need to be caught by in the code of the test.

All the matchers are in `CustomMatchers` and `singlebrowser.custommatchers.CustomMatchers`. There are 2 flavors - one that supports multiple instances of browsers, and a simplified one that supports a single instance of browser. See the JavaDoc for details.

8.5 Visual Testing

8.5.1 Resizing the browser or an element

In some cases there may be a need to make sure that the browser always has predefined dimensions. This is useful, for example, when capturing/asserting images. There are two classes that allow to do it:

- `WindowResizer` - resize for the browser
- `ElementResizer` - resize an element

Example use:

```
try (WindowResizer windowResizer = new WindowResizer(1000, 768)) {
    try (ElementResizer elementResizer = new ElementResizer(div.that(hasClass("widget-
↪pane")), 600, 400)) {
        System.out.println(String.format("element total dimensions: %d, %d",␣
↪elementResizer.getTotalWidth(), elementResizer.getTotalHeight()));
        System.out.println(String.format("element visible dimensions: %d, %d",␣
↪elementResizer.getVisibleWidth(), elementResizer.getVisibleHeight()));

        // do something with the image inside the resized element
    }
}
// at this point everything is reverted to the original state
```

8.5.2 Capturing and validating images

Dollarx offers the ability to capture an image(ie. screenshoot) of a Path element, and asserting it looks as expected. All the functionality is under `SingltonBrowserImage` and `Images` .

It has separate hanling for HTML 5 canvas elements, which allows to download just the image data for that element, thus is more optimized. It also supports capturing an the source of and HTML img element from its URI.

Displaying an image of an element

It is possible to capture and display the image for a given element in a separate window. Note that this does not work well as an evaluation within the debugger. The classes that deal with images are:

- `SingltonBrowserImage` - supports a single browser instance
- `Images` - supports multiple browser instances

Example:

```
Path map = div.withClass("gm-style");
SingltonBrowserImage mapImage = new SingltonBrowserImage(map);
mapImage.show();

SingltonBrowserImage mapImage = new SingltonBrowserImage(canvas);
// note that show() would also work, but showCanvas should be significantly faster␣
↪for small elements
mapImage.showCanvas();
```

Capturing a reference image to a PNG file

Typically, we want to capture a reference image, and then, in our test, assert that our image is similar to the one we previously captured. So, to capture an image:

```
File outFile = new File("reference.png");
SingltonBrowserImage mapImage = new SingltonBrowserImage( div.withClass("gm-style"));
mapImage.captureToFile(outFile);
```

In case we want to capture an image from the source of an HTML 'img' element, the code would look like:

```
import static com.github.loyada.jdollarx.BasicPath.image;
import static com.github.loyada.jdollarx.BasicPath.occurrenceNumber;

File outFile = new File("reference.png");
SingltonBrowserImage mapImage = new SingltonBrowserImage( occurrenceNumber(10).
↪of(image));
mapImage.captureImgSourceToFile(outFile);
```

Validating an image against a reference image

This assertion comes in two flavors:

- accurate, pixel-perfect comparison - see *SingltonBrowserImage.assertImageIsEqualToExpected*.
- fuzzy comparison - the images are "similar" - see *SingltonBrowserImage.assertImageIsSimilarToExpected*.
- accurate comparison, but allowing crop/shift - see *SingltonBrowserImage.assertImageIsEqualToExpectedWithShiftAndCrop*.
- create an image that highlights the errors when comparing the captured image to an expected image. Useful for troubleshooting - see *SingltonBrowserImage.getErrorImage*.

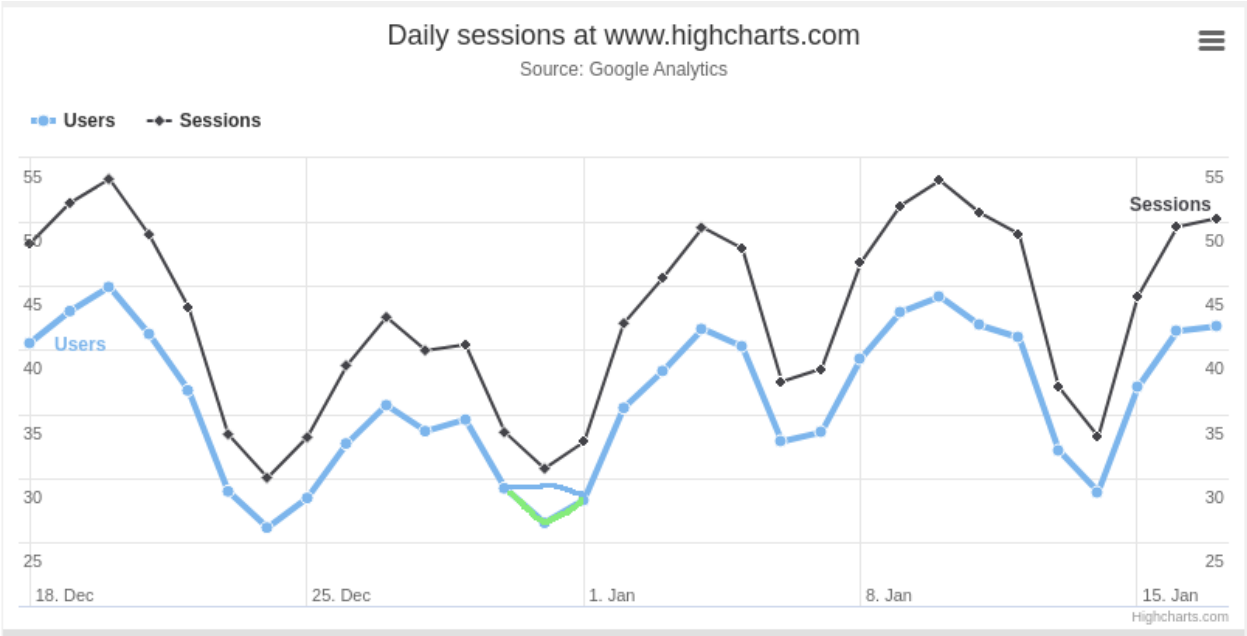
The fuzzy comparison currently uses a simplistic algorithm (transform color space, check weighted difference and normalize it).

Dollarx also supports providing a "filter image" as a reference to the assertion. This image highlights the areas of interest in the image that we want to focus on in the assertion. That allows to ignore areas that either cannot be asserted, or areas that have no useful information. This improves the assertions significantly, and is highly recommended.

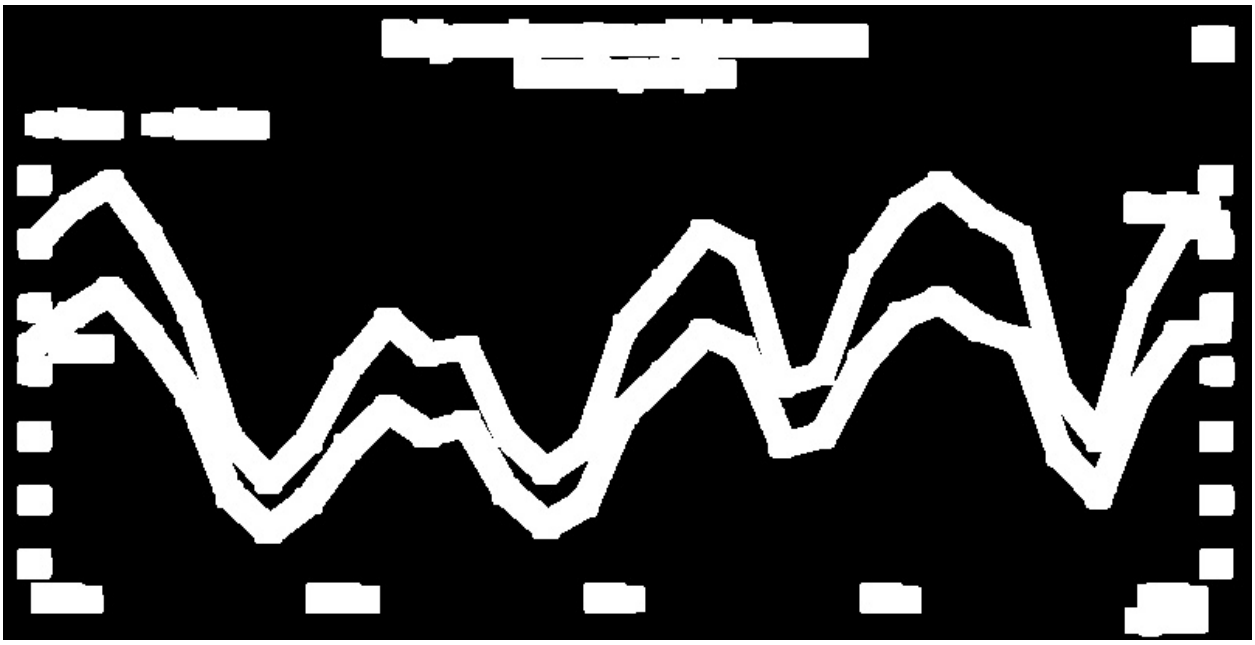
- fuzzy assertion - see `SingltonBrowserImage.assertImageIsSimilarToExpectedWithFilter` and `ImageComparator.verifyImagesAreSimilarFilteringInterestingAreas`
- strict assertion - see `ImageComparator.verifyImagesAreEqualFilteringInterestingAreas`.

As a starting point I recommend creating the "filter image" by processing the reference image with a Canny edge detector, followed by dilation. Here is an example:

Reference image:



Filter image:

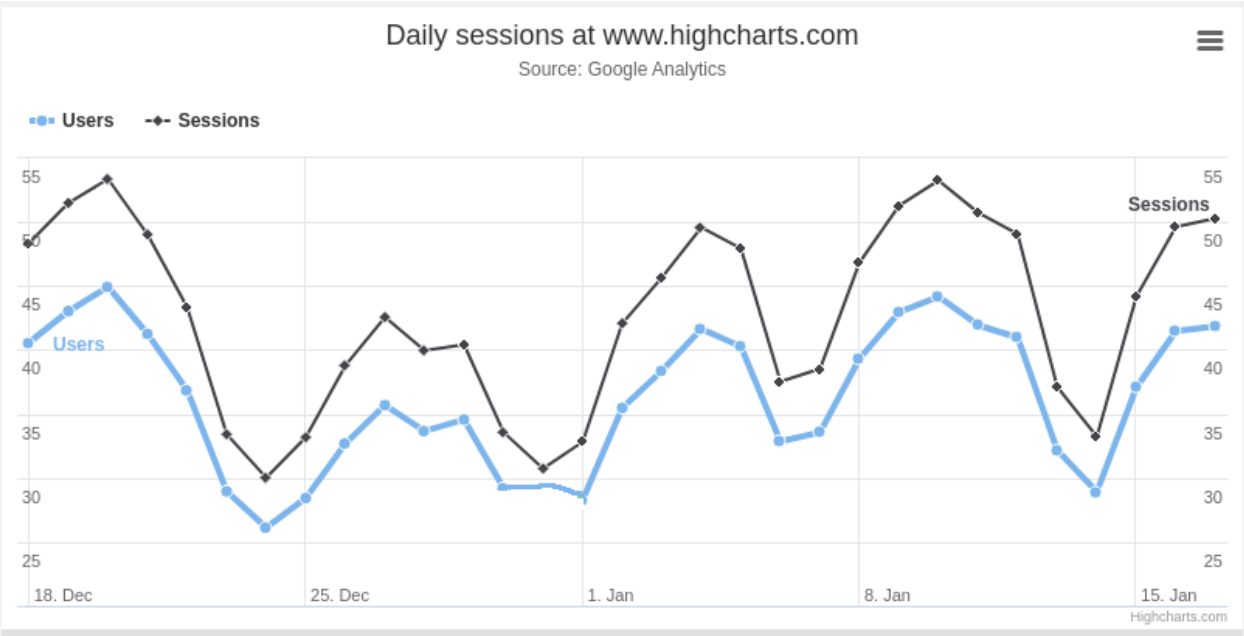


The assertions will ignore any of the locations that are not highlighted in the "filter image", so that there is less "noise" in the comparison.

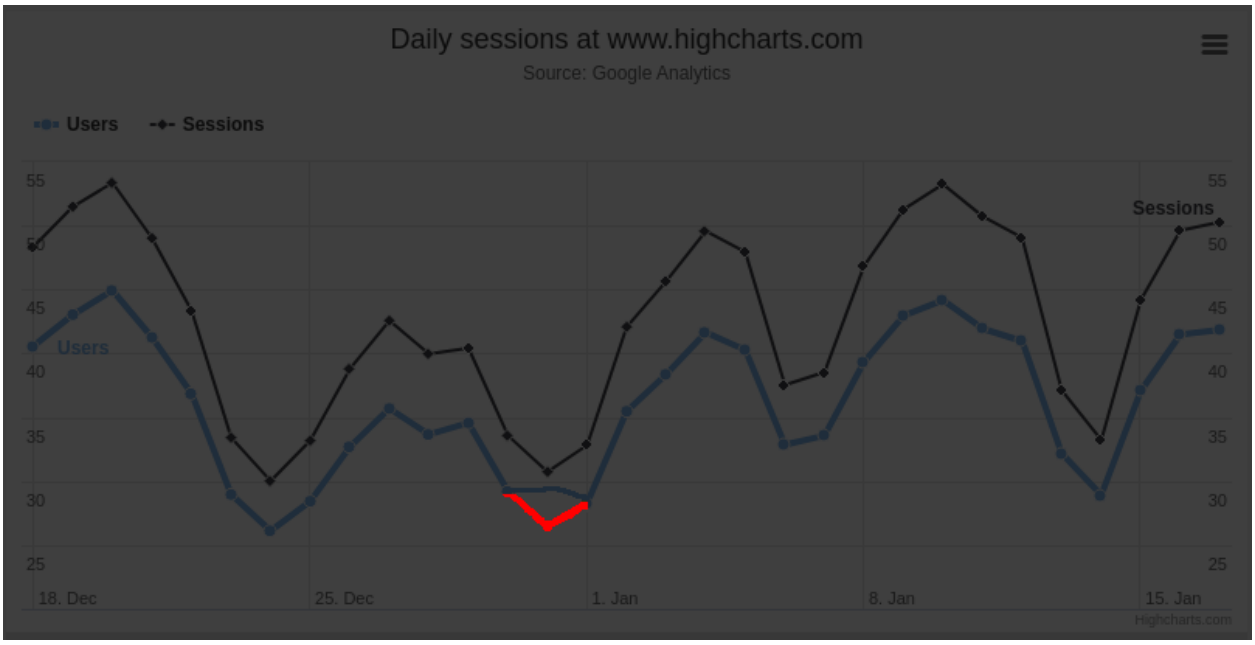
In case the assertion fail you can capture the error image that highlights the locations of errors:

- `ImageComparator.getErrorsImage` -
- `ImageComparator.getErrorImageForSimilarity`

For example, for the example above, let's say the actual image was :



Then the error image is:



Supported image captures

The classes above support 3 types of image captures/validations:

- Standard image capture - captures the entire screen, then crops it to the relevant section.
- Canvas element - more efficient, since it captures only the wanted element.
- The source of an HTML img element - by downloading it from the URI specficied in the "src" attribute. Typically (assuming the static resources are local in your tests) this will be more efficient than capturing the entire browser.

8.5.3 Temporarily make an element invisible

Sometimes we may want to assert an image capture of an element/page against an expected reference, while ignoring certain parts (ie. elements). This could be, for example, because different configurations or browsers render fonts differently, which may invalidate our assertion. To achieve this, use `SingltonBrowserImage.Obscure`. This class temporarily obscure the given elements, allowing you to do what you need, and then revert back to the original style.

For example(taken from ObscureExample.java in the tests) :

```
Path firstJavaSnippet = firstOccurrenceOf(div.withClass("highlight-java"));
assertThat(firstJavaSnippet, isDisplayed());

try (Obscure obscure = new Obscure(firstJavaSnippet)) {
  assertThat(firstJavaSnippet, isNotDisplayed());
  assertThat(obscure.getObscuredElements().size(), is(1));

  // we can now validate that the page ia pixel-perfect
}

// all back to normal
assertThat(firstJavaSnippet, isDisplayed());
```

8.5.4 Recommended recipe for visual assertion in CI

1. **Ensure the entire image you are interested in fits in the browser window, by:**
    1. changing the browser window size in Selenium webdriver, or WindowResizer
    2. scrolling to the appropriate location in the page
2. Use ElementResizer if needed to ensure the image is resized to an accurate size
3. Capture the image to a file - if it is correct and can be your reference, save it under "resources" as a reference. Note that you should do it for each environment you plan to run the tests in (for example, if in Docker, you need to capture the image inside docker using volume mapping).
4. Create a "filter image" as described above. I recommend the opencv library for image manipulation. Save the filter image under "resources".
5. Implement the visual assertion test, either fuzzy or strict, with the API that allows to provid a filter image.
6. Add to the test a hook so that in case the assertion fails, it captures the error image to the test-report/file. This allows to troubleshoot failed tests more easily.

8.6 AgGrid Testing

- *What about accessing a specific cell/row/header in an AgGrid?*
- *Working Examples*

8.6.1 Introduction

Since Ag-Grid is so popular, DollarX provides specific support for it using `AgGrid`.

8.6.2 Defining a grid

The AgGrid class provides a builder that allows to define a grid. Here is an example of a grid with two rows, containing only text:

```
 import com.github.loyada.jdollarx.singlebrowser.AgGrid;

 Map<String, String> row1 = new HashMap<>();
      row1.put("name", "tony smith");
      row1.put("language", "english");
      row1.put("jan","$38,031");
      row1.put("dec","$86,416");
      Map<String, String> row2 = new HashMap<>();
      row2.put("name", "Andrew Connell");
      row2.put("language", "swedish");
      row2.put("jan","$17,697");
      row2.put("dec","$83,386");

// A table with two rows
AgGrid grid2 = AgGrid.getBuilder()
              .withHeaders(Arrays.asList("dec", "jan", "language", "name"))
              .withRowsAsStrings(Arrays.asList(row1, row2))
              .containedIn(div.that(hasId("myGrid")))
              .build();
```

Here is an example of a grid of one row, with more flexible content (e.g. an image):

```
Map<String, ElementProperty> rowWithProperties = new HashMap<>();
rowWithProperties.put("name", hasAggregatedTextEqualTo("tony smith"));
rowWithProperties.put("country", contains(image.that(hasSource("https://flags.fmcdn.
↪net/data/flags/mini/ie.png"))));

// We define a table with a single row, and content that is not just text
AgGrid grid = AgGrid.getBuilder()
             .withHeaders(Arrays.asList("name", "country"))
             .withRowsAsElementProperties(Arrays.asList(rowWithProperties))
             .containedIn(div.that(hasId("myGrid")))
             .build();
```

If you know the order of the column in the table, and you want to potentially speed up the operation in case the order is the way you define it, you can use a declaration that maintains the order, as follows:

```
List<Entry<String, String>> row1 = Arrays.asList(
            new SimpleEntry<> ("name", "tony smith"),
            new SimpleEntry<> ("language", "english"),
            new SimpleEntry<> ("jan","$38,031"),
```

(continues on next page)

(continued from previous page)

```
            new SimpleEntry<> ("dec","$86,416")
    );
List<Entry<String, String>> row2 = Arrays.asList(
          new SimpleEntry<> ("name", "Andrew Connell"),
          new SimpleEntry<> ("language", "swedish"),
          new SimpleEntry<> ("jan","$17,697"),
          new SimpleEntry<> ("dec","$83,386")
  );
// A table with two rows, with known order of columns
AgGrid grid = AgGrid.getBuilder()
          .withHeaders(Arrays.asList("dec", "jan", "language", "name"))
          .withRowsAsStringsInOrder(Arrays.asList(row1, row2))
          .containedIn(div.that(hasId("myGrid")))
          .build();
```

You can also define the grid to be: * non virtualized - if you know your grid is indeed not virtualized, it would make operations much faster * strict - when asserting, would verify that the table contains only the data in your declaration

Note that by default, not all columns or all rows are required to be defined.

Refering To Columns/Headers by ID

If there is a column without a textual header, you may want to still refer to it. This can be achieved by wrapping the ID in curly braces, such as the following example.

```
// match on the column with ID of "country" (as opposed to the text in the header)
Map<String, String> row = Map.of("{country}", "USA");
AgGrid grid = AgGrid.getBuilder()
          .withHeaders(Arrays.asList("{country}"))
          .withRowsAsStrings(Arrays.asList(row))
          .containedIn(div.that(hasId("myGrid")))
          .build();
```

Changing Default Step Size When Scrolling

The Default step size when scrolling through the grid is 60 pixels. You can override that by calling `AgGrid.setScrollStep`. This could increase performance significantly, especially with large grids.

8.6.3 Assertion of the content of an AgGrid

The Hamcrest matcher supporting that, is `AgGrid.isPresent`.

For example, you can define a virtualized grid with a flexible content, such as images:

```
import com.github.loyada.jdollarx.singlebrowser.AgGrid;

Map<String, ElementProperty> rowWithProperties = new HashMap<>();
rowWithProperties.put("name", hasAggregatedTextEqualTo("tony smith"));
rowWithProperties.put("country", contains(image.that(hasSource("https://flags.fmcdn.
↪net/data/flags/mini/ie.png"))));

// We define a table with a single row
AgGrid grid = AgGrid.getBuilder()
```

(continues on next page)

(continued from previous page)

```
            .withHeaders(Arrays.asList("name", "country"))
            .withRowsAsElementProperties(Arrays.asList(rowWithProperties))
            .containedIn(div.that(hasId("myGrid")))
            .build();
assertThat(grid, AgGridMatchers.isPresent());
```

Note that when you run the code above. It scrolls through the grid to find the elements, in order to deal with the virtualization.

If your table just has text, then the definition is a bit simpler:

```
Map<String, String> row1 = new HashMap<>();
      row1.put("name", "tony smith");
      row1.put("language", "english");
      row1.put("jan","$38,031");
      row1.put("dec","$86,416");
      Map<String, String> row2 = new HashMap<>();
      row2.put("name", "Andrew Connell");
      row2.put("language", "swedish");
      row2.put("jan","$17,697");
      row2.put("dec","$83,386");

// A table with two rows
AgGrid grid = AgGrid.getBuilder()
              .withHeaders(Arrays.asList("dec", "jan", "language", "name"))
              .withRowsAsStrings(Arrays.asList(row1, row2))
              .containedIn(div.that(hasId("myGrid")))
              .build();
assertThat(grid, AgGridMatchers.isPresent());
```

The class supports virtualized and non-virtualized tables. If you want your assertion to fail if you have more rows than you define, define the grid as strict.

8.6.4 What about accessing a specific cell/row/header in an AgGrid?

In order to access a specific row in an AgGrid, you provide the index of the row as it appears in the table. The library will scroll until the row is visible (dealing with DOM virtualization), and return a Path element that allows you to access the row. For example, suppose we want to scroll to the 100th row and click it:

```
// override the default timeout to make scrolling through rows faster. This is␣
↪optional.
// When operation is done the timeout will go back to be relatively long.
grid.overrideTimeoutDuringOperation(2);

Path row = grid.ensureVisibilityOfRowWithIndex(100);
clickAt(row);
```

In order to interact with a specific cell, you follow the same pattern - providing the index of the row and the name/title of the column. Suppose I want to click on the checkbox inside the cell of the 100th row, under the "country" column:

```
// scroll until the expected cell, in the 100th row of the grid, under the column
↪"country"
Path cell = grid.ensureVisibilityOfRowWithIndexAndColumn(100, "country");

Path checkbox = span.withClass("ag-icon").that(hasNoneOfTheClasses("ag-hidden"));
clickAt(checkbox.inside(cell));
```

Another example, this time finding a row with a specific content, then interacting with it:

```
 AgGrid grid = AgGrid.getBuilder()
        .withHeaders(Arrays.asList( "language", "name", "country"))
        .withRowsAsElementProperties(new ArrayList<>())
        .containedIn(div.that(hasId("myGrid")))
        .build();

Map<String, ElementProperty> rowWithProperties = new HashMap<>();
rowWithProperties.put("name", hasAggregatedTextEqualTo("Kevin Cole"));
rowWithProperties.put("language", hasAggregatedTextEqualTo("english"));
rowWithProperties.put("country", contains(image.that(hasSource("https://flags.fmcdn.
↪net/data/flags/mini/ie.png"))));

grid.overrideTimeoutDuringOperation(1);

// assert that the row exists in the grid and get its index
int index = grid.findRowIndex(rowWithProperties);

// ensure the row with the given index is visible, and return a path that allows to␣
↪access it.
Path row = grid.ensureVisibilityOfRowWithIndex(index);

clickAt(grid.CELL.inside(row));
```

In order to interact with a column header, we can follow a similar pattern, or use special methods:

```
// "manual" approach - access the header element
Path header = grid.getVisibleHeaderPath("country");
clickAt(header);

// second approach - shortcut methods:
// sort column
grid.clickOnSort("country");

// click on sort until a desired outcome is met
grid.sortBy("language", SortDirection.ascending);

// open header menu
grid.clickMenuOfHeader("country");
```

If you don't know the row number of the content you expect, but you know the column and the content of your cell, you can find it by the column, as follows:

```
// scroll as needed until the expected cell is displayed or throw an exception
Path myCell = grid.ensureVisibilityOfCellInColumn("jan", hasAggregatedTextEqualTo("$4,
↪298"));

// now we can interact with the cell we found, for example:
clickOn(myCell);

// if we want the ROW of that cell:
Path row = grid.ROW.containing(myCell);
```

8.6.5 Working Examples

Look at the various working integration tests pointing to the official AgGrid example, demonstrating real-world scenarions HERE

8.7 High-Level API

The purpose of the high level API package is to provide a simplified API to common interactions with elements of type inputs, radio buttons, checkboxes, dropdowns etc.

The examples below are from the integrations tests of dollarx.

It is less flexible and sometimes suboptimal, but its value is a higher level abstraction.

For example:

```
RadioInput radio = RadioInputs.withLabeledText("Option 123");
radio.select();
assertTrue(radio.isSelected());
```

The code above does not require the developer to know the exact organization of the DOM. It will find it by itself (on a best effort basis), assuming there is a "label" element with the text "Option 123" that corrsponds to the input

If you don't even know if therer is a "label" element for this input, but just know that there is some text the correspond to this field, and you want to let dollarx figure out the right structure, you can do it with the higher level function withTextUnknownDOM():

```
RadioInput myInput = RadioInputs.withTextUnknownDOM("Female", 5, TimeUnit.
↪SECONDS);
myInput.select();
assertTrue(myInput.isSelected());
```

Another example, for checking a checkbox:

```
CheckBox checkbox =  CheckBoxes.checkBoxWithProperties(ElementProperties.
↪isNthSibling(3));
checkbox.check();;
assertTrue(checkbox.isChecked());
assertThat(checkbox.toString(), equalTo("checkbox, that is in place 3 among␣
↪its siblings"));
```

8.7.1 Inputs

- `Inputs.inputForLabel`- input referenced by the label element with the given text
- `Inputs.inputFollowedByUnlabeledText`- it is followed by text, without a label element.
- `Inputs.genericFormInputAfterField`- a generic, reasonable, guess that works for many forms

- `Inputs.genericFormInputBeforeField`- a generic, reasonable, guess that works for many forms
- *`Inputs.clearInput`*- clear the input. If it is unsuccessful, it throws an exception.
- `Inputs.clearInputNonStrict`- clear the input as much as it can
- *`Inputs.changeInputValue`*- replace any existing value in the input with a new one
- *`Inputs.selectInFieldWithLabel`* - select an option in a select element
- *`Inputs.changeInputValueWithEnter`* - update the value and send ENTER

8.7.2 Checkboxes

- *`CheckBoxes.checkBoxWithLabel`*
- *`CheckBoxes.checkBoxWithProperties`*checkbox with a list of properties
- *`CheckBox.isChecked`* - is this checkbox checked?
- *`CheckBox.check`*
- *`CheckBox.uncheck`*

8.7.3 Radio Buttons

- *`RadioInputs.withTextUnknownDOM`*- there is some text next to the input, but the DOM structure is unknown
- *`RadioInputs.withLabeledText`*
- *`RadioInputs.withUnlabeledText`*
- *`RadioInputs.withProperties`*
- *`RadioInput.isSelected`*
- *`RadioInput.select`*

8.8 Recipes

8.8.1 Troubleshooting and Debugging

I created a path instance and I am not sure what it maps to. What can I do?

There are two options:

- Examine the string representation of it. The toString() method of Path provides a readable description of it, that looks like english.

  For example:

```
Path dialog = div.withClass("ui-dialog");
Path myInput = input.inside(dialog).that(hasAttribute("name", "goo"),␣
↪isAfter(atLeast(2).occurrencesOf(div)));
System.out.println(myInput);
// output: input, inside (div, that has class ui-dialog), that [has name: "goo", is␣
↪after at least 2 occurrences of: div]
```

- Examine the xpath. This is not the recommended approach, since the whole point of Dollarx is to abstract xpath. But if you are unsure, you can always call getXPath(), which returns an Optional of the xpath:

```
System.out.println(myInput.getXPath().get());
// output: "div[contains(concat(' ', normalize-space(@class), ' '), ' ui-dialog ')]//
↪input[@name='goo'][count(preceding::div)>=2]"
```

Note that DollarX will not include the prefix to the xpath by default (e.g. "//") until it is needed to access the element, and then it will be added automatically, so if you copy-pase it to something like Chrome DevTools, remember to add the prefix yourself.

8.8.2 Working With Paths

I am not sure what is the correct way to define the path that I want. It looks like there are many ways that seem similar. What is the right way?

This is by design: one of the key features of DollarX is a flexible grammar that allows you to define paths the way that you find convenient to express, similarly to how we use spoken language. If when you read or output the string representation of two paths, their description seems equivalent in English, then they are equivalent from DollarX's perspective.

In the following example, all the "row" values are equivalent.

```
Path dialog = div.that(hasClass("ui-dialog"));
Path row = element.withClass("condition").inside(dialog);
// or....
row = element.inside(dialog).withClass("condition");
row = element.inside(dialog).that(hasClass("condition"));
row = element.inside(dialog).and(hasClass("condition"));
row = element.that(hasClass("condition"), isInside(dialog));
row = element.that(hasClass("condition").and(isInside(dialog)));
row = element.that(hasClass("condition")).and(isInside(dialog));
row = element.inside(dialog).that(hasClass("condition"));
row = element.withClass("condition").and(isInside(dialog));
row = element.withClass("condition").and(isContainedIn(dialog));
row = element.that(hasAncesctor(dialog)).and(hasClass("condition"));

// if you prefer to break the definition to two steps:
Path condition = element.withClass("condition");
row = condition.inside(dialog);
```

Similarly, let's look at the string representation of equivalent paths...

```
println(element.withClass("condition").inside(dialog));
println(element.that(hasClass("condition")).and(isInside(dialog)));
println(element.that(hasAncesctor(dialog)).and(hasClass("condition")));

output:
any element, that has class condition, inside (div, that has class ui-dialog)
any element, that [has class condition, has ancestor: (div, that has class ui-dialog)]
any element, that [has ancestor: (div, that has class ui-dialog), has class condition]
```

As can be seen, all are logically equivalent, thus the xpath they represent is equivalent.

I created a path and want to easily inspect its matches in the browser page I am connected to using Selenium

Several options:

- Highlight the element in the browser. `DebugUtil.hightlight` , *`DebugUtil.highlightAll`*- highlights the element(s) in the browser for 2 seconds.
- Examine the DOM while troubleshooting. DebugUtil offers several methods to do that. *`DebugUtil.getDOMOfAll`*, *`DebugUtil.getDOM`* - returns the elements that match the given path instance in the currect page, presented in a clear, readable way. Typically, these functions eliminate the need to use Chrome Devtools for debugging.

  The typical steps are:

  1. Add a breakpoint in the place in your code you expect to find the Path element, or elements, in the browser page.
  2. Run in debug mode and stop in the breakpoint
  3. Evaluate getDOM(myPath) or getDOMOfAll(myPath). The result is equivalent or better to what you would in Chrome Devtools.
- Download the page and convert it to a W3C Document, then inspect it looking for the element you are interested in. Luckily DollarX includes functions that do it.
  - *`DebugUtil.getPageAsW3CDoc`* - returns a W3C Document object representation of the page in the browser

- *PathParsers.findAllByPath* - returns a list of nodes of all the matches to the path in the document. BTW - This function is used widely in the unit tests of DollarX to test Paths.

I want to find or interact with an element based on the text content, while ignoring case

The property *ElementProperties.hasText* and the other ones in the examples below are case-insensitive.

```
Path myElement = ....;
Path myElementWithText = myElement.that(hasText("abc"));
// or alternatively
Path myElementWithText = myElement.withText("abc");

// partial match
myElement.that(hasTextContaining("abc"));
myElement.that(hasTextStartingWith("abc"));
myElement.that(hasTextEndingWith("abc"));
```

I want to find or interact with an element based on the text content, but the some of the text might be in nested elements inside my element

Use the property `ElementProperties.hasAggregatedText` - it will aggregate the text under the element, including nested elements, while normalizing any "white space" characters.

Like *ElementProperties.hasText*, it is case-insensitive, and offers partial matching:

ElementProperties.hasAggregatedTextContaining, *ElementProperties.hasAggregatedTextStartingWith*, *ElementProperties.hasAggregatedTextEndingWith*.

I want to assert that a table/list that includes specific rows appears on the page

The guideline is to define a path of the table that contains all the expected rows in their right order, and then assert that this table is present. The reason is that we want to maximize atomicity in order to minimize race conditions and round trips to browser. Let's take a trivial example of a table of names with two rows:

```
Path theTable = div.withClass("the-table");
row = div.withClass("row");
row1 = row.that(hasAggregatedText("john smith"), isWithIndex(0));
row2 = row.that(hasAggregatedText("jane king"), isWithIndex(1));

assert (theTable.that(contains(row1, row2)), isPresent());
```

8.8.3 Working with Grids

I can't find a cell within a grid, but when I scroll I see it exists

This is likely the result of a virtualized grid. This grid updates the DOM to include only cells that may be visible to the user. This kind of optimization can be effective when dealing with large tables.

In such a case, in order to find a cell, you need to scroll until it is present in the DOM.

DollarX allows you to do it with *InBrowser.scrollElement* or *InBrowserSinglton.scrollElement* . The implementation is in *Operations.ScrollElement*.

For example:

```
Path table = div.withClass("ag-body-viewport");

InBrowserSinglton.scrollElement(table).rightUntilElementIsPresent(div.
↪that(hasAggregatedTextContaining("$86,416")));
InBrowserSinglton.scrollElement(table).leftUntilElementIsPresent(div.
↪that(hasAggregatedTextContaining("Tony Smith")));
InBrowserSinglton.scrollElement(table).downUntilElementIsPresent(div.
↪that(hasAggregatedTextContaining("isabella cage")));
```

What about interaction with AgGrid?

Refer to *AgGrid Testing* .

8.8.4 Dealing With Race Conditions

Typically, using DollarX correctly (minimizing interactions with the browser) eliminates most of the issues with race conditions. However, there may be cases in which we may have an intermittent failure. For example - we want to click an element, but it is currently hidden by a modal, and it may take a while for the application to remove that modal.

For these cases, use `Operations.doWithRetries` . For example:

```
Operations.doWithRetries(() -> browser.clickOn(myElement), 5, 10);
```

This code tries to click the myElement.If it fails (for example - the element is not clickable), it will wait 10 milliseconds and then try again. This will continue up to 5 times. Once it reached 5 retries, it will throw the exception thrown by the clickOn().

Another example:

```
doWithRetries(() -> assertThat(myElement, isDisplayed()), 5, 10);
```

This code asserts the myElement is displayed. If the assertion fails, it will wait and retry. After 5 times, it will throw an assertion error.

8.8.5 Extensions and Customization

I found a property that is unsupported by DollarX out-of-the-box. Can I add it?

Definitely!

There are several options to do so:

- If it a new type of element, use `BasicPath.customElement` to define a simple new element. For example:

```
// define a new element type: <label></label>
Path label = customElement("label");
```

- If it's just a matter of <attribute>=<value> on a DOM element, you can use `ElementProperties.hasAttribute`. For example, suppose you want to define your own `ElementProperty`, called "hasRole". it would look like:

```
ElementProperty hasRole(String role) {
   return hasAttribute("role", role);
}

// now you can use it:
Path myInput = input.that(hasRole("password"));
```

- For more flexibility, you can use, `ElementProperties.hasRawXpathProperty` . It allows you to define any constant xpath and string representation. See the JavaDoc for more details.
- For more flexibility, use `CustomElementProperties.createPropertyGenerator`. This method accepts functions that creates the xpath as well as the string representation of the property. Once defined, it can be used as a parameter of `CustomElementProperties.hasProperty`. For example, to define the "hasRole" property, similarly to the previous section, we could do:

```
Function<String, ElementProperty> role = createPropertyGenerator(
           value -> format("@role='%s'", value),
           value -> format("has role %s", value));

 // Now we can use it:
 Path myEl = div.that(hasProperty(role, "foo")).and(hasClass("x"));

 assertThat(myEl.toString(), is(equalTo("div, that has role foo, and has class x
↪")));
```

`CustomElementProperties.createPropertyGenerator` also has a version that accepts BiFunctions, which allows to create properties such as :

```
BiFunction<String, Integer, ElementProperty> dataNum =␣
↪createPropertyGenerator(
          (attr,value) -> format("@data-%s='%d'", attr, value),
          (attr, value) -> format("has data %s of %d", attr, value));

// The following will match a div element with attribute: data-foo="4"␣
↪and the class "x":
Path el = div.that(hasProperty(dataNum, "foo", 4)).and(hasClass("x"));

assertThat(el.toString(), is(equalTo("div, that has data foo of 4, and␣
↪has class x")));
```

I need to use Selenium WebDriver directly. Can I use it in conjunction with DollarX?

Yes!

You can get from `Path` to Selenium WebElement and vice-versa.

- To get a WebElement from a `Path`, call `InBrowser.find` or `InBrowser.findAll`, or the equivalents in `InBrowserSinglton`.
- To get a `Path` from a WebElement, create a `Path` as a wrapper of a WebElement. Example:

```
// InBrowser browser = ......;

WebElement webEl = browser.find(myPath);

Path insideWebEl = BasicPath.Builder().withUnderlying(webEl).build();
```

(continues on next page)

(continued from previous page)

```
// Now I can define a path inside webEl:
Path myFooEl = insideWebEl.withClass("foo");

// and perform an assertion on it:
assertThat(myFooEl, isPresentIn(browser));
```

8.9 Javadoc

8.9.1 com.github.loyada.jdollarx

main package for defining Path DOM elements and browser interaction

BasicPath

public final class **`BasicPath`** implements *Path*
The standard implementation of Path in DollarX

Fields

anchor

public static final *BasicPath* **`anchor`**
An anchor(or "a") element

body

public static final *BasicPath* **`body`**

button

public static final *BasicPath* **`button`**

canvas

public static final *BasicPath* **`canvas`**

div

public static final *BasicPath* **`div`**

element

public static final *BasicPath* **`element`**
Any element

form

public static final *BasicPath* **form**

header

public static final *BasicPath* **header**
Any header element

header1

public static final *BasicPath* **header1**

header2

public static final *BasicPath* **header2**

header3

public static final *BasicPath* **header3**

header4

public static final *BasicPath* **header4**

header5

public static final *BasicPath* **header5**

header6

public static final *BasicPath* **header6**

html

public static final *BasicPath* **html**

iframe

public static final *BasicPath* **iframe**

image

public static final *BasicPath* **image**

input

public static final *BasicPath* **`input`**

label

public static final *BasicPath* **`label`**

listItem

public static final *BasicPath* **`listItem`**
An "li" element

main

public static final *BasicPath* **`main`**

option

public static final *BasicPath* **`option`**

paragraph

public static final *BasicPath* **`paragraph`**

section

public static final *BasicPath* **`section`**

select

public static final *BasicPath* **`select`**

span

public static final *BasicPath* **`span`**

svg

public static final *BasicPath* **`svg`**

table

public static final *BasicPath* **`table`**

td

public static final *BasicPath* **td**

textarea

public static final *BasicPath* **textarea**

th

public static final *BasicPath* **th**

title

public static final *BasicPath* **title**

tr

public static final *BasicPath* **tr**

unorderedList

public static final *BasicPath* **unorderedList**
An "ul" element

Methods

after

public *Path* **after** (*Path path*)
The element appears after the given path

Parameters

- **path** –
  - the element that appear before

Returns a new path with the added constraint

afterSibling

public *Path* **afterSibling** (*Path path*)
The element has a preceding sibling that matches to the given Path parameter

Parameters

- **path** –
  - the sibling element that appears before

Returns a new path with the added constraint

ancestorOf

public *Path* **ancestorOf** (*Path path*)

Parameters

- **path** –
  - the element that is inside our element

Returns a new path with the added constraint

and

public *Path* **and** (*ElementProperty... prop*)

Alias equivalent to that(). Added for readability. Example:

```
div.that(hasClass("a")).and(hasText("foo"));
```

Parameters

- **prop** – a list of element properties (constraints)

Returns a new Path

before

public *Path* **before** (*Path path*)

The element is before the given path parameter

Parameters

- **path** –
  - the element that appear after

Returns a new path with the added constraint

beforeSibling

public *Path* **beforeSibling** (*Path path*)

The element is a sibling of the given path and appears before it

Parameters

- **path** –
  - the sibling element that appears after

Returns a new path with the added constraint

builder

public static *PathBuilder* **builder** ()

childNumber

public static *ChildNumber* **childNumber** (Integer *n*)

the element is the nth child of its parent. Count starts at 1. For example:

```
childNumber(4).ofType(div.withClass("foo"))
```

Parameters

- **n** – the index of the child - starting at 1

Returns a ChildNumber instance, which is used with as in the example.

childOf

public *Path* **childOf** (*Path path*)

Parameters

- **path** –
  - the parent element

Returns a new path with the added constraint

containing

public *Path* **containing** (*Path path*)

Parameters

- **path** –
  - the element that is inside our element

Returns a new path with the added constraint

contains

public *Path* **contains** (*Path path*)

Parameters

- **path** –
  - the element that is inside our element

Returns a new path with the added constraint

customElement

public static *BasicPath* **customElement** (String *el*)

Create a custom element Path using a simple API instead of the builder pattern. Example:

```
Path myDiv = customElement("div");
```

Parameters

- **el** –
  - the element type in W3C. will be used for the toString as well.

Returns a Path representing the element

customNameSpaceElement

public static *BasicPath* **customNameSpaceElement** (String *el*)

descendantOf

public *Path* **descendantOf** (*Path path*)

The element is inside the given path parameter

Parameters

- **path** –
  - the element that is wrapping our element

Returns a new path with the added constraint

describedBy

public *Path* **describedBy** (String *description*)

firstOccurrenceOf

public static *Path* **firstOccurrenceOf** (*Path path*)

First global occurrence of an element in the document.

Parameters

- **path** – the element to find

Returns a new path with the added constraint

getAlternateXPath

public Optional<String> **getAlternateXPath** ()

getDescribedBy

public Optional<String> **getDescribedBy** ()

getElementProperties

public List<*ElementProperty*> **getElementProperties** ()

getUnderlyingSource

public Optional<WebElement> **getUnderlyingSource**()

getXPath

public Optional<String> **getXPath**()

getXpathExplanation

public Optional<String> **getXpathExplanation**()

immediatelyAfterSibling

public *Path* **immediatelyAfterSibling**(*Path path*)

The sibling right before the current element matches to the given Path parameter

Parameters

- **path** –
  - the sibling element that appears right before

Returns a new path with the added constraint

immediatelyBeforeSibling

public *Path* **immediatelyBeforeSibling**(*Path path*)

The sibling right after the element matches the given path parameter

Parameters

- **path** –
  - the sibling element that appears after

Returns a new path with the added constraint

inside

public *Path* **inside**(*Path path*)

Element that is inside another element

Parameters

- **path** –
  - the containing element

Returns a new Path with the added constraint

insideTopLevel

public *Path* **insideTopLevel** ()

Returns an element that is explicitly inside the document. This is usually not needed - it will be added implicitly when needed.

Returns a new Path

lastOccurrenceOf

public static *Path* **lastOccurrenceOf** (*Path path*)

Last global occurrence of an element in the document

Parameters

- **path** – the element to find

Returns a new path with the added constraint

occurrenceNumber

public static *GlobalOccurrenceNumber* **occurrenceNumber** (Integer *n*)

used in the form : occurrenceNumber(4).of(myElement)). Return the nth occurrence of the element in the entire document. Count starts at 1. For example:

```
occurrenceNumber(3).of(listItem)
```

Parameters

- **n** – the number of occurrence

Returns GlobalOccurrenceNumber instance, which is used as in the example.

or

public *Path* **or** (*Path path*)

match more than a single path. Example: div.or(span) - matches both div and span

Parameters

- **path** – the alternative path to match

Returns returns a new path that matches both the original one and the given parameter

parentOf

public *Path* **parentOf** (*Path path*)

Parameters

- **path** –
  - the child element

Returns a new path with the added constraint

textNode

public static *Path* **textNode** (String *text*)

Define a text node in the DOM, with the given text. Typically you don't need to use it, but it is relevant if you have something like:

```
   Male
   Female

input.immediatelyBeforeSibling(textNode("Male"));
```

Parameters

- **text** – the text in the node. Note that it is trimmed, and case insensitive.

Returns a Path of a text node

that

public *Path* **that** (*ElementProperty*... *prop*)

returns a path with the provided properties. For example: div.that(hasText("abc"), hasClass("foo"));

Parameters

- **prop** –
  - one or more properties. See ElementProperties documentation for details

Returns a new path with the added constraints

toString

public String **toString** ()

withClass

public *Path* **withClass** (String *cssClass*)

Equivalent to `this.that(hasClass(cssClass))`

Parameters

- **cssClass** – the class name

Returns a new path with the added constraint

withClasses

public *Path* **withClasses** (String... *cssClasses*)

Equivalent to `this.that(hasClasses(cssClasses))`

Parameters

- **cssClasses** – the class names

Returns a new path with the added constraint

withGlobalIndex

public *Path* **withGlobalIndex** (Integer *n*)

An alias of: `occurrenceNumber(n + 1).of(this)`

Parameters

- **n** –
  - the global occurrence index of the path, starting from 0

Returns a new path with the added constraint

withText

public *Path* **withText** (String *txt*)

Element with text equals (ignoring case) to txt. Equivalent to:

```
path.that(hasText(txt))
```

Parameters

- **txt** –
  - the text to equal to, ignoring case

Returns a new Path with the added constraint

withTextContaining

public *Path* **withTextContaining** (String *txt*)

Equivalent to `this.that(hasTextContaining(txt))`.

Parameters

- **txt** – the text to match to. The match is case insensitive.

Returns a new path with the added constraint

BasicPath.ChildNumber

public static final class **ChildNumber**

Allows to define an element that has a predefined number of similar preceding siblings. Count starts at 1 (same as you would use in English). Should be used through the method *`childNumber(Integer)`* . Example:

```
childNumber(5).ofType(div);
```

Constructors

ChildNumber

public **ChildNumber** (Integer *n*)

Does not return any usable Path by itself. Must be used like: `ChildNumber(5).ofType(div)`

Parameters

- **n** – the number of child. Count starts at 1.

Methods

ofType

public *Path* **ofType** (*Path path*)

an element that has n similar preceding siblings. For example: `ChildNumber(5).ofType(element.withText("john"))` will correspond to the fifth child that has text "john"

Parameters

- **path** – the element to find

Returns a new Path instance

BasicPath.GlobalOccurrenceNumber

public static final class **GlobalOccurrenceNumber**

Not to be used directly, but through the utility functions: *firstOccurrenceOf(Path)*, *lastOccurrenceOf(Path)*, *occurrenceNumber(Integer)*

Constructors

GlobalOccurrenceNumber

GlobalOccurrenceNumber (Integer *n*)

Methods

of

public *Path* **of** (*Path path*)

return the nth global occurrence (in the entire document) of the given path.

Parameters

- **path** – the element to find

Returns a new Path instance, that adds the global occurrence constraint to it

BasicPath.PathBuilder

public static final class **PathBuilder**

A builder for BasicPath. Usually *customElement(String)* is simpler and sufficient.

Constructors

PathBuilder

public **PathBuilder**()

PathBuilder

public **PathBuilder**(Optional<String> *insideXpath*, Optional<String> *xpath*, Optional<String> *xpathExplanation*, Optional<String> *describedBy*, Optional<WebElement> *underlying*, List<*ElementProperty*> *elementProperties*, Optional<String> *alternateXpath*)

Methods

build

public *BasicPath* **build**()

withAlternateXpath

public *PathBuilder* **withAlternateXpath**(String *alternateXpath*)

withAlternateXpathOptional

public *PathBuilder* **withAlternateXpathOptional**(Optional<String> *alternateXpath*)

withDescribedBy

public *PathBuilder* **withDescribedBy**(String *describedBy*)

withDescribedByOptional

public *PathBuilder* **withDescribedByOptional**(Optional<String> *describedBy*)

withElementProperties

public *PathBuilder* **withElementProperties**(List<*ElementProperty*> *elementProperties*)

withInsideXpath

public *PathBuilder* **withInsideXpath**(String *insideXpath*)

withInsideXpathOptional

public *PathBuilder* **withInsideXpathOptional** (Optional<String> *insideXpath*)

withUnderlying

public *PathBuilder* **withUnderlying** (WebElement *underlying*)

withUnderlyingOptional

public *PathBuilder* **withUnderlyingOptional** (Optional<WebElement> *underlying*)

withXpath

public *PathBuilder* **withXpath** (String *xpath*)

withXpathExplanation

public *PathBuilder* **withXpathExplanation** (String *xpathExplanation*)

withXpathExplanationOptional

public *PathBuilder* **withXpathExplanationOptional** (Optional<String> *xpathExplanation*)

withXpathOptional

public *PathBuilder* **withXpathOptional** (Optional<String> *xpath*)

CustomElementProperties

public final class **CustomElementProperties**

Functions to create custom `ElementProperty`, if the property is unsupported out-of-the-box.

Methods

createPropertyGenerator

public static <T> Function<T, *ElementProperty*> **createPropertyGenerator** (Function<T, String> *xpathCreator*, Function<T, String> *toStringCreator*)

Easy way to define a custom property generator that accepts a single parameter. For example:

```
Function<String, ElementProperty> dataFoo = createPropertyFunc(
                      fooVal -> String.format("[@data-foo=%s]", fooVal),
                      fooVal -> "has Foo " + fooVal );
       Path myDataEl = element.that(hasProperty(dataFoo("bar")));
```

Parameters

- **xpathCreator** – function that generates the xpath that represents this attribute
- **toStringCreator** – function that generates the string (text) representation of this attribute
- **<T>** – the type of the parameter the above functions expect

Returns a function that generates an ElementProperty. Use with *hasProperty(Function, Object)*

createPropertyGenerator

public static <T, V> BiFunction<T, V, *ElementProperty*> **createPropertyGenerator**(BiFunction<T, V, String> *xpathCreator*, BiFunction<T, V, String> *toStringCreator*)

Easy way to define a custom property generator that accepts two parameter. For example:

```
Function<String, ElementProperty> dataAttr = createPropertyFunc(
                     (attr, val) -> String.format("[@data-%s=%s]", attr, val),
                     (attr, val) -> String.format("has data-%s of %s", attr, val␣
↪);
      Path myDataEl = element.that(hasProperty(dataAttr("foo", "bar")));
```

Parameters

- **xpathCreator** – function that generates the xpath that represents this attribute
- **toStringCreator** – function that generates the string (text) representation of this attribute
- **<T>** – the type of the first parameter the above functions expect
- **<V>** – the type of the second parameter the above functions expect

Returns a function that generates an ElementProperty. Use with *hasProperty(BiFunction, Object,Object)*

hasProperty

public static <T> *ElementProperty* **hasProperty**(Function<T, *ElementProperty*> *propertyGen*, T *t*)

Syntactic sugar that allows to define properties of the form:

```
 role = createPropertyGenerator(....);
element.that(hasProperty(role, "foo"));
```

Parameters

- **propertyGen** – a function that was generated using createPropertyGenerator()
- **t** –
  - a parameter the property generator expects
- **<T>** – the type of the value to match to

Returns Element property

hasProperty

public static <T, V> *ElementProperty* **hasProperty** (BiFunction<T, V, *ElementProperty*> *propertyGen*, T *t*, V *v*)

Parameters

- **propertyGen** – a function that was generated using createPropertyGenerator()
- **t** –
  - first parameter the property generator expects
- **v** –
  - first parameter the property generator expects
- **<T>** – the type of the first parameter (t)
- **<V>** – the type of the second parameter (v)

Returns Element property

ElementProperties

public final class **ElementProperties**

Various constrains on `Path` instances, that are used with the methods `Path.that` and `Path.and`.

Fields

hasChildren

public static final *ElementProperty* **hasChildren**

The element has 1 or more children (the opposite from hasNoChildren). Examples where it might be useful: an non-empty list, non-empty table, etc.

hasNoChildren

public static final *ElementProperty* **hasNoChildren**

The element has no children. Examples where it might be useful: an empty list, empty table, etc.

hasSomeText

public static *ElementProperty* **hasSomeText**

Element has non-empty text

isChecked

public static final *ElementProperty* **isChecked**
The element is checked

isDisabled

public static final *ElementProperty* **isDisabled**
The element is diabled

isEnabled

public static final *ElementProperty* **isEnabled**
The element is enabled

isHidden

public static *ElementProperty* **isHidden**
Element that is hidden. This is limited to only examine embeded css style, so it not useful in some cases.

isLastSibling

public static final *ElementProperty* **isLastSibling**
The element is the last sibling (ie: last child) of its parent.

isOnlyChild

public static final *ElementProperty* **isOnlyChild**
The element is the only direct child of its parent. It has no siblings. For example: a table with a single row.

isSelected

public static final *ElementProperty* **isSelected**
The element is selected

Methods

contains

public static *ElementProperty* **contains** (*Path... paths*)
The given elements in the parameters list are contained in the current element

Parameters

- **paths** –
  - a list of elements that are descendants of the current element

Returns An element property that can be applied with Path::that

hasAggregatedCaseSensitiveTextContaining

public static *ElementProperty* **hasAggregatedCaseSensitiveTextContaining** (String *txt*)

When aggregating all the text under this element, it contains the given substring. This condition is case sensitive.

Parameters

- **txt** – a substring of the aggregated, case sensitive, text inside the element

Returns An element property that can be applied with Path::that

hasAggregatedCaseSensitiveTextEqualTo

public static *ElementProperty* **hasAggregatedCaseSensitiveTextEqualTo** (String *txt*)

When aggregating all the text under this element, it equals to the given parameter. This condition is case sensitive.

Parameters

- **txt** – the aggregated, case insensitive, text inside the element

Returns An element property that can be applied with Path::that

hasAggregatedTextContaining

public static *ElementProperty* **hasAggregatedTextContaining** (String *txt*)

When aggregating all the text under this element, it contains the given substring (ignoring case)

Parameters

- **txt** – a substring of the aggregated, case insensitive, text inside the element

Returns An element property that can be applied with Path::that

hasAggregatedTextEndingWith

public static *ElementProperty* **hasAggregatedTextEndingWith** (String *txt*)

When aggregating all the text under this element, it ends with the given substring (ignoring case)

Parameters

- **txt** – the suffix of the aggregated, case insensitive, text inside the element

Returns An element property that can be applied with Path::that

hasAggregatedTextEqualTo

public static *ElementProperty* **hasAggregatedTextEqualTo** (String *txt*)

When aggregating all the text under this element, it equals to the given parameter. This condition is not case sensitive.

Parameters

- **txt** – the aggregated, case insensitive, text inside the element

Returns An element property that can be applied with Path::that

hasAggregatedTextStartingWith

public static *ElementProperty* **hasAggregatedTextStartingWith** (String *txt*)

When aggregating all the text under this element, it starts with the given substring (ignoring case)

Parameters

- **txt** – the prefix of the aggregated, case insensitive, text inside the element

Returns An element property that can be applied with Path::that

hasAncesctor

public static *ElementProperty* **hasAncesctor** (*Path path*)

Element is inside the given parameter

Parameters

- **path** – the ancestor of the current element

Returns An element property that can be applied with Path::that

hasAnyOfClasses

public static *ElementProperty* **hasAnyOfClasses** (String... *cssClasses*)

Element that has at least one of the classes given

Parameters

- **cssClasses** –
  - the class names to match to

Returns a element property that can be applied with Path::that

hasAttribute

public static *ElementProperty* **hasAttribute** (String *attribute*, String *value*)

hasCaseSensitiveText

public static *ElementProperty* **hasCaseSensitiveText** (String *txt*)

Element has text equals to the given string parameter. The equality is case-sensitive.

Parameters

- **txt** –
  - the text to match to (case sensitive)

Returns a element property that can be applied with Path::that

hasCaseSensitiveTextContaining

public static *ElementProperty* **hasCaseSensitiveTextContaining** (String *txt*)

The text in the element contains the given parameter. This condition is case=sensitive.

Parameters

- **txt** –
  - the substring to match to (case sensitive)

Returns An element property that can be applied with Path::that

hasChild

public static *ElementProperty* **hasChild** (*Path*... *paths*)

Element is the parent of the given list of elements

Parameters

- **paths** –
  - a list of elements that are children of the current element

Returns An element property that can be applied with Path::that

hasClass

public static *ElementProperty* **hasClass** (String *className*)

Has the class given in the parameter

Parameters

- **className** – the class the element has

Returns An element property that can be applied with Path::that

hasClassContaining

public static *ElementProperty* **hasClassContaining** (String *classSubString*)

Element that has a class with name that contain the given parameter

Parameters

- **classSubString** – a string that should be contained in the class

Returns An element property that can be applied with Path::that

hasClasses

public static *ElementProperty* **hasClasses** (String... *cssClasses*)

Element that has all of the given classes

Parameters

- **cssClasses** –
  - the class names to match to

Returns a element property that can be applied with Path::that

hasDescendant

public static *ElementProperty* **hasDescendant** (*Path... paths*)

The given elements in the parameters list are contained in the current element

Parameters

- **paths** –
  - a list of elements that are descendants of the current element

Returns An element property that can be applied with Path::that

hasId

public static *ElementProperty* **hasId** (String *id*)

Element has ID equals to the given parameter

Parameters

- **id** –
  - the ID to match to

Returns a element property that can be applied with Path::that

hasNChildren

public static *ElementPropertyWithNumericalBoundaries* **hasNChildren** (Integer *n*)

The element has n direct children

Parameters

- **n** – the number of children

Returns a element property that can be applied with Path::that

hasName

public static *ElementProperty* **hasName** (String *name*)

Element with a "name" attribute equal to the given parameter. Useful for input elements.

Parameters

- **name** – the value of the name property

Returns a element property that can be applied with Path::that

hasNonOfTheClasses

public static *ElementProperty* **hasNonOfTheClasses** (String... *cssClasses*)

Element that has none of the given classes

Parameters

- **cssClasses** –
  - a list of class names, none of which is present in element

Returns An element property that can be applied with Path::that

hasParent

public static *ElementProperty* **hasParent** (*Path path*)

Element is direct child of the element matched by the given parameter

Parameters

- **path** –
  - the parent of the current element

Returns An element property that can be applied with Path::that

hasRawXpathProperty

public static *ElementProperty* **hasRawXpathProperty** (String *rawXpathProps*, String *rawExplanation*)

Custom property that allows to state the raw expath of a property, and give a string description of it. Example:

```
Path el = span.that(hasRawXpathProperty("string(.)='x'", "is awesome"),␣
↪isOnlyChild);
 assertThat(el.getXPath().get(), equalTo("span[string(.
↪)='x'][count(preceding-sibling::)=0" +
                                                "and count(following-
↪sibling::)=0]"));
 assertThat(el.toString(), is(equalTo("span, that is awesome, and is only␣
↪child")));
```

Parameters

- **rawXpathProps** –
  - the xpath property condition string. Will be wrapped with [] in the xpath
- **rawExplanation** –
  - a textual readable description of this property

Returns a element property that can be applied with Path::that

hasRef

public static *ElementProperty* **hasRef** (String *ref*)

Element with a "ref" attribute equal to the given role.

Parameters

- **ref** – the value of the role property

Returns a element property that can be applied with Path::that

hasRole

public static *ElementProperty* **hasRole** (String *role*)

Element with a "role" attribute equal to the given role.

Parameters

- **role** – the value of the role property

Returns a element property that can be applied with Path::that

hasSource

public static *ElementProperty* **hasSource** (String *src*)

Element with a "src" attribute equal to the given parameter. Useful for images.

Parameters

- **src** – the URI of the image

Returns a element property that can be applied with Path::that

hasText

public static *ElementProperty* **hasText** (String *txt*)

Element has text equals to the given string parameter, ignoring case.

Parameters

- **txt** –
  - the text to match to

Returns a element property that can be applied with Path::that

hasTextContaining

public static *ElementProperty* **hasTextContaining** (String *txt*)

The text in the element contains the given parameter, ignoring case

Parameters

- **txt** –
  - the substring to match to

Returns An element property that can be applied with Path::that

hasTextEndingWith

public static *ElementProperty* **hasTextEndingWith** (String *txt*)

Element has text that ends with the given parameter

Parameters

- **txt** –
  - the text to match to

Returns a element property that can be applied with Path::that

hasTextStartingWith

public static *ElementProperty* **hasTextStartingWith** (String *txt*)

Element has text that starts with the given parameter

Parameters

- **txt** –
  - the text to match to

Returns a element property that can be applied with Path::that

isAfter

public static *ElementProperty* **isAfter** (*Path*... *paths*)

Element appears after all the given parameters in the document

Parameters

- **paths** –
  - elements that precede the current element

Returns An element property that can be applied with Path::that

isAfter

public static *ElementProperty* **isAfter** (*NPath nPath*)

Element is after at-least/at-most/exactly the given number of the given element. Example use: import static com.github.loyada.jdollarx.atLeast; input.that(isAfter(atLeast(2).occurrencesOf(div)));

Parameters

- **nPath** –
  - at-least/at-most/exactly the given number of the given element

Returns An element property that can be applied with Path::that

isAfterSibling

public static *ElementProperty* **isAfterSibling** (*Path*... *paths*)

Element is a sibling of all the elements defined by the given paths, AND is after all those siblings

Parameters

- **paths** – a list of paths referring to elements

Returns An element property that can be applied with Path::that

isAfterSibling

public static *ElementProperty* **isAfterSibling** (*NPath nPath*)

Element is a sibling of the at-least/at-most/exactly n elements given, and appears after them. Example:

```
Path el = element.that(isAfterSibling(exactly(2).occurrencesOf(div)));
assertThat(el.toString(), is(equalTo("any element, that is after 2 siblings of␣
↪type: div")));
```

Parameters

- **nPath** – a count of elements that are siblings appearing before current elements.

Returns An element property that can be applied with Path::that

isAncestorOf

public static *ElementProperty* **isAncestorOf** (*Path... paths*)

The given elements in the parameters list are contained in the current element

Parameters

- **paths** –
  - a list of elements that are descendants of the current element

Returns An element property that can be applied with Path::that

isBefore

public static *ElementProperty* **isBefore** (*Path... paths*)

Element is before all the elements given in the parameters

Parameters

- **paths** –
  - all the elements that appear after the current element

Returns An element property that can be applied with Path::that

isBefore

public static *ElementProperty* **isBefore** (*NPath nPath*)

Element is before at-least/at-most/exactly the given number of the given element. Example use:

```
import static com.github.loyada.jdollarx.isBefore;
input.that(isBefore(atLeast(2).occurrencesOf(div)));
```

Parameters

- **nPath** –
  - at-least/at-most/exactly the given number of the given element

Returns An element property that can be applied with Path::that

isBeforeSibling

public static *ElementProperty* **isBeforeSibling** (*Path... paths*)

Element is a sibling of all the elements defined by the given paths, AND is before all those siblings

Parameters

- **paths** – a list of paths referring to elements

Returns An element property that can be applied with Path::that

isBeforeSibling

public static *ElementProperty* **isBeforeSibling** (*NPath nPath*)

Element is a sibling of the at-least/at-most/exactly n elements given, and appears before them. Example:

```
Path el = element.that(isBeforeSibling(exactly(2).occurrencesOf(div)));
assertThat(el.toString(), is(equalTo("any element, that is before 2 siblings of␣
↪type: div")));
```

Parameters

- **nPath** – a count of elements that are siblings appearing after current elements.

Returns An element property that can be applied with Path::that

isChildOf

public static *ElementProperty* **isChildOf** (*Path path*)

Element is direct child of the element matched by the given parameter

Parameters

- **path** –
  - the parent of the current element

Returns An element property that can be applied with Path::that

isContainedIn

public static *ElementProperty* **isContainedIn** (*Path path*)

isDescendantOf

public static *ElementProperty* **isDescendantOf** (*Path path*)

Element is inside the given parameter

Parameters

- **path** – the ancestor of the current element

Returns An element property that can be applied with Path::that

isInside

public static *ElementProperty* **isInside** (*Path path*)

Element is inside the given parameter

Parameters

- **path** – the ancestor of the current element

Returns An element property that can be applied with Path::that

isNthFromLastSibling

public static *ElementProperty* **isNthFromLastSibling** (Integer *reverseIndex*)

The element is the nth-from-last sibling. Example usage: find the element before the last one in a list.

Parameters

- **reverseIndex** –
  - the place from last, starting at 0 for the last sibling.

Returns a element property that can be applied with Path::that

isNthSibling

public static *ElementProperty* **isNthSibling** (Integer *index*)

The element is the nth sibling. Example usage: find the 4th element in a list.

Parameters

- **index** –
  - starting at 0 for the first one

Returns a element property that can be applied with Path::that

isParentOf

public static *ElementProperty* **isParentOf** (*Path... paths*)

Element is the parent of the given list of elements

Parameters

- **paths** –
  - a list of elements that are children of the current element

Returns An element property that can be applied with Path::that

isSiblingOf

public static *ElementProperty* **isSiblingOf** (*Path... paths*)

Element is a sibling of all the elements defined by the given paths

Parameters

- **paths** – a list of paths referring to elements

Returns An element property that can be applied with Path::that

isWithIndex

public static *ElementProperty* **isWithIndex** (Integer *index*)

Element that is the nth sibling of its parent

Parameters

- **index** –
  - the index of the element among its sibling, starting with 0

Returns An element property that can be applied with Path::that

not

public static *ElementProperty* **not** (*ElementProperty* *prop*)

Element does NOT have the given property.

Parameters

- **prop** –
  - the property which the element must not have

Returns An element property that can be applied with Path::that

withIndexInRange

public static *ElementProperty* **withIndexInRange** (int *first*, int *last*)

The index among its siblings is between first and last parameters. For example: taking a row from a table, which we know is between row number 2 and 4.

Parameters

- **first** –
  - lower index (inclusive, starting at 0)
- **last** –
  - upper index (inclusive, starting at 0)

Returns a element property that can be applied with Path::that

ElementProperties.And

static final class **And** implements *ElementProperty*

Constructors

And

public **And** (*ElementProperty* *p1*, *ElementProperty* *p2*)

Methods

toString

public String **toString**()

toXpath

public String **toXpath**()

ElementProperties.Not

static final class **Not** implements *ElementProperty*

Constructors

Not

public **Not**(*ElementProperty p*)

Methods

toString

public String **toString**()

toXpath

public String **toXpath**()

ElementProperties.Or

static final class **Or** implements *ElementProperty*

Constructors

Or

public **Or**(*ElementProperty p1*, *ElementProperty p2*)

Methods

toString

public String **toString**()

toXpath

public String **toXpath**()

ElementProperty

public interface **ElementProperty**

The main interface to add an additional constraint on a `Path`. Used with the method `Path.that`, and `Path.and`.

Methods

and

ElementProperty **and**(*ElementProperty prop*)

returns a new property, that is a combination of the current property AND the given property parameter. Meaning the element is required to have both properties. Obviously, this can be used multiple times: prop1.and(prop2).or(prop3).and(prop4)

Parameters

- **prop** – another property to perform a logical "AND" with

Returns a new property that is equivalent to: (this property AND prop)

andNot

ElementProperty **andNot**(*ElementProperty prop*)

returns a new property, that is equivalent to the current property, BUT NOT the property parameter. Obviously, this can be used multiple times: prop1.andNot(prop2).or(prop3.andNot(prop4))

Parameters

- **prop** – another property to perform a logical "NAND" with

Returns a new property that is equivalent to: (this property and not prop)

or

ElementProperty **or**(*ElementProperty prop*)

returns a new property, that is a combination of the current property OR the given property parameter. Meaning the element is required to have any of the two properties. Obviously, this can be used multiple times: prop1.or(prop2).or(prop3).and(prop4)

Parameters

- **prop** – another property to perform a logical "OR" with

Returns a new property that is equivalent to: (this property OR prop)

toXpath

String **toXpath**()

ElementPropertyWithNumericalBoundaries

public interface **ElementPropertyWithNumericalBoundaries** extends *ElementProperty*

Used to define a constraint on a number of elements

Methods

orLess

ElementProperty **orLess**()

Given the a property and and a count of it, returns a property equivalent to at most the count of that property. This works only with specific properties. For example: div.that(hasNChildren(2).orLess())

Returns a property equivalent to at most the count of that property

orMore

ElementProperty **orMore**()

Given the a property and and a count of it, returns a property equivalent to at least the count of that property. This works only with specific properties. For example: div.that(hasNChildren(2).orMore())

Returns a property equivalent to at least the count of that property

HighLevelPaths

public class **HighLevelPaths**

Fields

checkbox

public static *Path* **checkbox**

Methods

hasType

public static *ElementProperty* **hasType**(String *theType*)

Images

public class **Images**

Fields

logger

static Logger **logger**

Methods

assertCanvasImageIsEqualToExpected

public static void **assertCanvasImageIsEqualToExpected**(*InBrowser browser*, *Path el*, InputStream *expectedImageInput*)

Verify that the element's image is pixel-perfect

Parameters

- **browser** –
  - browser
- **el** –
  - canvas to capture and verify
- **expectedImageInput** – reference image file

Throws

- **IOException** –
  - file could not be read

assertHTMLImgSoureIsEqualToExpected

public static void **assertHTMLImgSoureIsEqualToExpected**(*InBrowser browser*, *Path el*, InputStream *expectedImageInput*)

Verify that an image downloaded from an HTML img src attribute, is pixel-perfect

Parameters

- **browser** –
  - browser
- **el** –
  - HTML img element to capture and verify
- **expectedImageInput** – reference image file

Throws

- **IOException** –
  - file could not be read

assertImageIsEqualToExpected

public static void **assertImageIsEqualToExpected**(*InBrowser browser*, *Path el*, InputStream *expectedImageInput*)

Verify that the element's image is pixel-perfect

Parameters

- **browser** –
  - browser
- **el** –
  - element to capture and verify
- **expectedImageInput** – reference image file

Throws

- **IOException** –
  - file could not be read

assertImageIsEqualToExpectedWithShiftAndCrop

public static void **assertImageIsEqualToExpectedWithShiftAndCrop**(*InBrowser browser*, *Path el*, InputStream *expectedImageInput*, int *maxShift*)

Verify that the element's image is pixel-perfect, but allowing some crop/shift

Parameters

- **browser** –
  - browser
- **el** –
  - element to capture and verify
- **expectedImageInput** – reference image file
- **maxShift** – maximum pixels the images are shifted/cropped compared to each other (both on x and y axis)

Throws

- **IOException** –
  - file could not be read

assertImageIsSimilarToExpected

public static void **assertImageIsSimilarToExpected**(*InBrowser browser*, *Path el*, InputStream *expectedImageInput*, int *maxBadPixelsRatio*)

Verify the picture is "similar" to the reference image. Ignores minor differences between the pixels.

Parameters

- **browser** –

  - browser
- **el** –
  - element to capture and validate
- **expectedImageInput** –
  - reference image
- **maxBadPixelsRatio** –
  - a positive number. For example: If it's 100, then 1% of the pixels can have major differences compared to the reference.

Throws

- **IOException** –
  - image file could not be read

captureCanvas

public static BufferedImage **captureCanvas** (*InBrowser browser*, *Path canvas*)

captureCanvasToFile

public static void **captureCanvasToFile** (*InBrowser browser*, *Path el*, File *outputFile*)

Save an HTML5 canvas to file. Optimized for canvas. Will fail if the element is not a canvas.

Parameters

- **browser** –
  - browser
- **el** –
  - Path element to capture
- **outputFile** –
  - output file

captureImgSrcToFile

public static void **captureImgSrcToFile** (*InBrowser browser*, *Path imgEl*, File *outputFile*)

Save the source of an HTML img element to file

Parameters

- **browser** –
  - browser
- **imgEl** –
  - HTML img element to capture
- **outputFile** –
  - output file

captureToFile

public static void **captureToFile** (*InBrowser browser*, *Path el*, File *outputFile*)

Save image to file

Parameters

- **browser** –
  - browser
- **el** –
  - Path element to capture
- **outputFile** –
  - output file

getErrorsImage

public static Optional<BufferedImage> **getErrorsImage** (*InBrowser browser*, *Path el*, InputStream *expectedImageInput*)

create and return an image that highlights the different pixels between the captured image and the reference image

Parameters

- **browser** –
  - browser
- **el** –
  - element to capture and verify
- **expectedImageInput** – reference image file

Throws

- **IOException** –
  - file could not be read
- **AssertionError** –
  - images are not the same size

Returns an image that highlights the different pixels. If the images are equal, returns an empty optional.

show

public static void **show** (*InBrowser browser*, *Path el*)

Display image of an element in a separate window. Does not work as an evaluation within the debugger.

Parameters

- **browser** –
  - browser
- **el** –

– the element to capture and display

showCanvas

public static void **showCanvas** (*InBrowser browser*, *Path el*)

Display image of an HTML5 canvas element in a separate window. Does not work as an evaluation within the debugger.

Parameters

- **browser** –
  - browser
- **el** –
  - the element to capture and display

showImage

static void **showImage** (BufferedImage *image*)

Images.ImageComparator

public static final class **ImageComparator**

Internal utility class for images

Methods

getErrorImage

public static Optional<BufferedImage> **getErrorImage** (BufferedImage *img1*, BufferedImage *img2*)

verifyImagesAreEqual

public static void **verifyImagesAreEqual** (BufferedImage *img1*, BufferedImage *img2*)

verifyImagesAreShifted

public static void **verifyImagesAreShifted** (BufferedImage *img1*, BufferedImage *img2*, int *maxShift*)

verifyImagesAreSimilar

public static void **verifyImagesAreSimilar** (BufferedImage *img1*, BufferedImage *img2*, int *maxBadPixelsRatio*)

Images.Obscure

public static class **Obscure** implements AutoCloseable

Fields

js

final JavascriptExecutor **`js`**

obscuredElements

final List<*Path*> **`obscuredElements`**

strict

final boolean **`strict`**

Constructors

Obscure

public **`Obscure`** (*InBrowser browser*, *Path element*)

Obscure

public **`Obscure`** (*InBrowser browser*, List<*Path*> *elements*)

Obscure

public **`Obscure`** (*InBrowser browser*, List<*Path*> *elements*, boolean *strict*)

Methods

close

public void **`close`** ()

getObscuredElements

public List<*Path*> **`getObscuredElements`** ()

InBrowser

public class **`InBrowser`**

A wrapper around Selenium WebDriver, used for interaction with the browser. In case only a single instance of the browser is used, `com.github.loyada.jdollarx.singlebrowser.InBrowserSinglton` offers a simpler API.

Constructors

InBrowser

public **`InBrowser`** (WebDriver *driver*)

Creates a connection to a browser, using the given driver

Parameters

- **`driver`** – a WebDriver instance

Methods

clickAt

public WebElement **`clickAt`** (*Path el*)

Click at the location the first element that fits the given path. Does not require a clickable element.

Parameters

- **`el`** – the element

Returns the clicked on WebElement

clickOn

public WebElement **`clickOn`** (*Path el*)

Click on the first element that fits the given path. Only works for clickable elements. If the element is currently not clickable, will wait up to a second for it to be clickable.

Parameters

- **`el`** – the element

Returns the clicked on WebElement

contextClick

public WebElement **`contextClick`** (*Path el*)

Context-click (right click) at the location the first element that fits the given path. Does not require a clickable element.

Parameters

- **`el`** – the element

Returns the clicked on WebElement

doubleClickOn

public void **`doubleClickOn`** (*Path el*)

Doubleclick the location of the first element that fits the given path.

Parameters

- **`el`** – the element

dragAndDrop

public *Operations.DragAndDrop* **dragAndDrop** (*Path path*)

Drag and drop in the browser. Several flavors of use: browser.dragAndDrop(source).to(target); browser.dragAndDrop(source).to(xCor, yCor);

Parameters

- **path** – the source element

Returns a DragAndDrop instance, that allows to drag and drop to a location or to another DOM element

find

public WebElement **find** (*Path el*)

Finds an element in the browser, based on the xpath representing el. It is similar to WebDriver.findElement(), If el also has a WebElement (ie: getUnderlyingSource() is not empty), then it looks inside that WebElement. This is useful also to integrate with existing WebDriver code.

Parameters

- **el** –
  - the path to find

Returns

- A WebElement instance from selenium, or throws NoSuchElementException exception

findAll

public List<WebElement> **findAll** (*Path el*)

Finds all elements in the browser, based on the xpath representing el. It is similar to WebDriver.findElements(), If el also has a WebElement (ie: getUnderlyingSource() is not empty), then it looks inside that WebElement. This is useful also to integrate with existing WebDriver code.

Parameters

- **el** –
  - the path to find

Returns

- A list of WebElement from selenium, or throws NoSuchElementException exception

findPageWithNumberOfOccurrences

public WebElement **findPageWithNumberOfOccurrences** (*Path el*, int *numberOfOccurrences*, *RelationOperator relationOperator*)

Don't use this directly. There are better ways to do equivalent operation.

Parameters

- **el** – the path to find
- **numberOfOccurrences** – the base number to find

- **relationOperator** – whether we look for exactly the numberOfOccurrences, at least, or at most occurrences

Returns the first WebElement found

findPageWithout

public WebElement **findPageWithout** (*Path el*)

Finds an page in the browser, that does not contain the given path

Parameters

- **el** –
  - the path that must not appear in the page

Returns returns the page element or raises NoSuchElementException

getCssClasses

public List<String> **getCssClasses** (*Path el*)

Get all classes of given Path element.

Parameters

- **el** – the element to look for

Returns a list of classes

getDriver

public WebDriver **getDriver** ()

Returns the underlying WebDriver instance

getSelect

public Select **getSelect** (*Path el*)

Get a Selenium select element, which provides a high level API to interacting with a "select" menu. Since the Selenium API is good enough, there was no need to create a specialized dollarx version.

Parameters

- **el** –
  - must be a "select" path, with "option" elements for the various selectable options.

Returns org.openqa.selenium.support.ui.Select instance

hoverOver

public WebElement **hoverOver** (*Path el*)

Hover over the location of the first element that fits the given path

Parameters

- **el** – the element

Returns the clicked on WebElement

isDisplayed

public boolean **isDisplayed** (*Path el*)

is the element present and displayed? Typically you should not use this method directly. Instead, use CustomMatchers. Also, this is limited to checking the inlined css style, so it is quite limited.

Parameters

- **el** – the element

Returns true if it is present and selected

isEnabled

public boolean **isEnabled** (*Path el*)

is the element present and enabled? Typically you should not use this method directly. Instead, use CustomMatchers.

Parameters

- **el** – the element

Returns true if it is present and enabled

isNotPresent

public boolean **isNotPresent** (*Path el*)

is the element present? Typically you should not use this method directly. Instead, use CustomMatchers.

Parameters

- **el** – the path to find

Returns true if it is not present

isPresent

public boolean **isPresent** (*Path el*)

is the element present? Typically you should not use this method directly. Instead, use CustomMatchers.

Parameters

- **el** – the path to find

Returns true if the element is present

isSelected

public boolean **isSelected** (*Path el*)

is the element present and selected? Typically you should not use this method directly. Instead, use CustomMatchers.

Parameters

- **el** – the element

Returns true if it is present and selected

numberOfAppearances

public Integer **numberOfAppearances** (*Path el*)

Returns the number of elements in the browser that match the given path. Typically you should not use this method directly. Instead, use CustomMatchers.

Parameters

- **el** – the element to find

Returns the number of elements in the browser that match the given path

pressKeyDown

public *Operations.KeysDown* **pressKeyDown** (CharSequence *thekey*)

Press key down in the browser, or on a specific element. Two flavors of use: browser.pressKeyDown(Keys.TAB).inBrowser(); browser.pressKeyDown(Keys.TAB).on(path);

Parameters

- **thekey** – a key to press

Returns returns a KeysDown instance that allows to press a key on the browser in general or on a specific DOM element

releaseKey

public *Operations.ReleaseKey* **releaseKey** (CharSequence *thekey*)

Release key down in the browser, or on a specific element. Two flavors of use:

```
browser.releaseKey(Keys.TAB).inBrowser();
   browser.releaseKey(Keys.TAB).on(path);
```

Parameters

- **thekey** – a key to release

Returns returns a ReleaseKey instance that allows to release on the browser in general or on a specific DOM element

rightClick

public WebElement **rightClick** (*Path el*)

Context-click (right click) at the location the first element that fits the given path. Does not require a clickable element.

Parameters

- **el** – the element

Returns the clicked on WebElement

scroll

public *Operations*.*Scroll* **scroll**()

scroll the browser. Several flavors of use:

```
browser.scroll().to(path);
   browser.scroll().left(50);
   browser.scroll().right(50);
   browser.scroll().up(50);
   browser.scroll().down(50);
```

Returns a Scroll instance that allows to scroll by offset or to a location of a DOM element

scrollElement

public *Operations*.*ScrollElement* **scrollElement**(*Path wrapper*)

scrollElementWithStepOverride

public *Operations*.*ScrollElement* **scrollElementWithStepOverride**(*Path wrapper*, int *step*)

scrollTo

public WebElement **scrollTo**(*Path el*)

Scroll to the location of the first element that fits the given path

Parameters

- **el** – the element

Returns the clicked on WebElement

sendKeys

public *Operations*.*KeysSender* **sendKeys**(CharSequence... *charsToSend*)

send keys to the browser, or to a specific element. Two flavors of use: browser.sendKeys("abc").toBrowser(); browser.sendKeys("abc").to(path);

Parameters

- **charsToSend** – The characters to send. Can be "abc" or "a", "b", "c"

Returns a KeySender instance that allows to send keys to the browser in general, or to a specific DOM element

InBrowserFinder

public class **InBrowserFinder**

Internal implementation.

Methods

find

static WebElement **find** (WebDriver *driver*, *Path el*)

findAll

public static List<WebElement> **findAll** (WebDriver *driver*, *Path el*)

findPageWithNumberOfOccurrences

public static WebElement **findPageWithNumberOfOccurrences** (WebDriver *driver*, *Path el*, int *numberOfOccurrences*)

findPageWithNumberOfOccurrences

public static WebElement **findPageWithNumberOfOccurrences** (WebDriver *driver*, *Path el*, int *numberOfOccurrences*, *RelationOperator relationOperator*)

findPageWithout

static WebElement **findPageWithout** (WebDriver *driver*, *Path el*)

NPath

public final class **NPath**

Internal implementation - not to be instantiated directly

Fields

n

public final int **n**

path

public final *Path* **path**

qualifier

public final *RelationOperator* **qualifier**

Constructors

NPath

public **NPath** (*Path path*, int *n*, *RelationOperator qualifier*)

Methods

atLeast

public static *NPathBuilder* **atLeast** (int *n*)

atMost

public static *NPathBuilder* **atMost** (int *n*)

exactly

public static *NPathBuilder* **exactly** (int *n*)

NPath.NPathBuilder

public static class **NPathBuilder**
Internal implementation - not to be instantiated directly

Fields

n

final int **n**

qualifier

final *RelationOperator* **qualifier**

Constructors

NPathBuilder

NPathBuilder (int *n*, *RelationOperator qualifier*)

Methods

occurrencesOf

public *NPath* **occurrencesOf** (*Path path*)

Operations

public class **Operations**

Internal implementation of various browser operations

Methods

doWithRetries

public static void **doWithRetries** (Runnable *action*, int *numberOfRetries*, int *sleepInMillisec*)

Retry an action/assertion up to a number of times, with delay after each time. For example:

```
doWithRetries(() -> assertThat(div.withClass("foo"), isDisplayedIn(browser)), 5,
↪10);
```

Parameters

- **action** – the action to try. It's a runnable - no input parapeters and does not return anything.
- **numberOfRetries** –
  - maximum number of retries
- **sleepInMillisec** –
  - delay between consecutive retries

doWithRetries

public static <T> T **doWithRetries** (Callable<T> *action*, int *numberOfRetries*, int *sleepInMillisec*)

Retry an action up to a number of times, with delay after each time. For example:

```
WebElement el = doWithRetries(() -> browser.find(div.withClass("foo"), 5, 10);
```

Parameters

- **action** – the action to try. It has no input parameters, but returns a value
- **numberOfRetries** –
  - maximum number of retries
- **sleepInMillisec** –
  - delay between consecutive retries
- **<T>** – any type that the function returns

Throws

- **Exception** – the exception thrown by the last try in case it exceeded the number of retries.

Returns returns the result of the callable

Operations.DragAndDrop

public static class **DragAndDrop**
internal implementation not be instantiated directly - Action of drag-and-drop

Constructors

DragAndDrop

public **DragAndDrop** (WebDriver *driver*, *Path path*)

Methods

to

public void **to** (*Path target*)
drag and drop to the given element's location

Parameters

- **target** –
  - the target(drop) element

Throws

- ***OperationFailedException*** – operation failed. Typically includes the reason.

to

public void **to** (Integer *x*, Integer *y*)
drag and drop to the given coordinates

Parameters

- **x** – coordinates
- **y** – coordinates

Throws

- ***OperationFailedException*** – operation failed. Typically includes the reason.

Operations.KeysDown

public static class **KeysDown**
internal implementation not be instantiated directly - Action of key-down

Constructors

KeysDown

public **KeysDown** (WebDriver *driver*, CharSequence *keysToSend*)

Methods

inBrowser

public void **inBrowser** ()
Send key-down to the browser in general

on

public void **on** (*Path path*)
Send key-down to an element in the browser

Parameters

- **path** – the element to press a key down on

Throws

- ***OperationFailedException*** – operation failed. Typically includes the reason.

Operations.KeysSender

public static class **KeysSender**
internal implementation not be instantiated directly - Action of sending keys to browser

Constructors

KeysSender

public **KeysSender** (WebDriver *driver*, CharSequence... *charsToSend*)

Methods

to

public void **to** (*Path path*)
Send keys to a specific element in the browser

Parameters

- **path** – the element to send the keys to

Throws

- ***OperationFailedException*** – operation failed. Typically includes the reason.

toBrowser

public void **toBrowser** ( )
Send characters tp the browser in general

Operations.OperationFailedException

public static class **OperationFailedException** extends IOException

Constructors

OperationFailedException

public **OperationFailedException** (String *message*, Throwable *cause*)

OperationFailedException

public **OperationFailedException** (String *message*)

Operations.ReleaseKey

public static class **ReleaseKey**
internal implementation not be instantiated directly - Action of releasing a key (key up)

Constructors

ReleaseKey

public **ReleaseKey** (WebDriver *driver*, CharSequence *keysToSend*)

Methods

inBrowser

public void **inBrowser** ( )
releasing a key in the browser in general

on

public void **on** (*BasicPath path*)
release a key on a specific element in the browser

Parameters

- **path** – the element to release the key on

Throws

- *OperationFailedException* – operation failed. Typically includes the reason.

Operations.Scroll

public static class **Scroll**
internal implementation not be instantiated directly - Action of scroll

Constructors

Scroll

public **Scroll** (WebDriver *driver*)

Methods

down

public void **down** (Integer *n*)
scroll down number of pixels

Parameters

- **n** – pixels

left

public void **left** (Integer *n*)
scroll left number of pixels

Parameters

- **n** – pixels

right

public void **right** (Integer *n*)
scroll right number of pixels

Parameters

- **n** – pixels

to

public void **to** (*Path path*)
Scroll until the location of an element

Parameters

- **path** – the element to scroll to

up

public void **up** (Integer *n*)
scroll up number of pixels

Parameters

- **n** – pixels

Operations.ScrollElement

public static class **ScrollElement**
internal implementation not be instantiated directly - Action of scroll within an element

Constructors

ScrollElement

public **ScrollElement** (WebDriver *driver*, *Path wrapper*)

ScrollElement

public **ScrollElement** (WebDriver *driver*, *Path wrapper*, int *stepSizeOverride*)

Methods

down

public void **down** (Integer *n*)
scroll down number of pixels

Parameters

- **n** – pixels

downUntilElementIsPresent

public WebElement **downUntilElementIsPresent** (*Path expectedElement*)
Scroll down until the DOM contains the expected element. Using 40 pixels steps, until the end of the table

Parameters

- **expectedElement** –
  - the element we are looking for

Returns the WebElement or throws an exception of not found

downUntilElementIsPresent

public WebElement **downUntilElementIsPresent**(*Path expectedElement*, int *scrollStep*, int *maxNumberOfScrolls*)

Scroll down until the DOM contains the expected element.

Parameters

- **expectedElement** –
  - the element we are looking for
- **scrollStep** –
  - scroll step in pixels
- **maxNumberOfScrolls** – maximum number of scroll operations

Returns the WebElement or throws an exception of not found

downUntilPredicate

public WebElement **downUntilPredicate**(*Path expectedElement*, Predicate<WebElement> *predicate*)

Scroll down until the DOM contains the expected element, and the given condition for that element is met. Using 40 pixels steps, until the end of the table

Parameters

- **expectedElement** –
  - the element we are looking for
- **predicate** –
  - a condition regarding the expected element that is required to be met

Returns the WebElement or throws an exception of not found

downUntilPredicate

public WebElement **downUntilPredicate**(*Path expectedElement*, int *scrollStep*, int *maxNumberOfScrolls*, Predicate<WebElement> *predicate*)

Scroll down until the DOM contains the expected element, and the supplied condition for that element is met.

Parameters

- **expectedElement** –
  - the element we are looking for
- **scrollStep** –
  - scroll step in pixels
- **maxNumberOfScrolls** – maximum number of scroll operations
- **predicate** –
  - a condition regarding the expected element that is required to be met

Returns the WebElement or throws an exception of not found

left

public void **left** (Integer *n*)
scroll left number of pixels

Parameters

- **n** – pixels

leftUntilElementIsDisplayed

public WebElement **leftUntilElementIsDisplayed** (*Path expectedElement*)
Scroll left until the DOM contains the expected element, and it's displayed. Using 40 pixels steps, until the end of the table

Parameters

- **expectedElement** –
  - the element we are looking for

Returns the WebElement or throws an exception if not found

leftUntilElementIsPresent

public WebElement **leftUntilElementIsPresent** (*Path expectedElement*)
Scroll left until the DOM contains the expected element. Using 40 pixels steps, until the end of the table

Parameters

- **expectedElement** –
  - the element we are looking for

Returns the WebElement or throws an exception of not found

leftUntilElementIsPresent

public WebElement **leftUntilElementIsPresent** (*Path expectedElement*, int *scrollStep*, int *maxNumberOfScrolls*)
Scroll left until the DOM contains the expected element.

Parameters

- **expectedElement** –
  - the element we are looking for
- **scrollStep** –
  - scroll step in pixels
- **maxNumberOfScrolls** – maximum number of scroll operations

Returns the WebElement or throws an exception of not found

leftUntilPredicate

public WebElement **leftUntilPredicate** (*Path expectedElement*, Predicate<WebElement> *predicate*)

Scroll left until the DOM contains the expected element, and the given predicate regarding that element is met. Using 40 pixels steps, until the end of the table

Parameters

- **expectedElement** –
  - the element we are looking for
- **predicate** –
  - a condition regarding the expected element that is required to be met

Returns the WebElement or throws an exception of not found

leftUntilPredicate

public WebElement **leftUntilPredicate** (*Path expectedElement*, int *scrollStep*, int *maxNumberOfScrolls*, Predicate<WebElement> *predicate*)

Scroll left until the DOM contains the expected element and the supplied predicate for the element is met.

Parameters

- **expectedElement** –
  - the element we are looking for
- **scrollStep** –
  - scroll step in pixels
- **maxNumberOfScrolls** – maximum number of scroll operations
- **predicate** –
  - a condition regarding the expected element that is required to be met

Returns the WebElement or throws an exception of not found

right

public void **right** (Integer *n*)

scroll right number of pixels

Parameters

- **n** – pixels

rightUntilElementIsPresent

public WebElement **rightUntilElementIsPresent** (*Path expectedElement*)

Scroll right until the virtualized DOM contains the expect element. Using 40 pixels steps, until the end of the table

Parameters

- **expectedElement** –

– the element we are looking for

Returns the WebElement or throws an exception of not found

rightUntilElementIsPresent

public WebElement **`rightUntilElementIsPresent`** (*Path expectedElement*, int *scrollStep*, int *maxNumberOfScrolls*)

Scroll right until the virtualized DOM contains the expect element.

Parameters

- **`expectedElement`** –
  - the element we are looking for
- **`scrollStep`** –
  - scroll step in pixels
- **`maxNumberOfScrolls`** – maximum number of scroll operations

Returns the WebElement or throws an exception of not found

rightUntilElementIsVisible

public WebElement **`rightUntilElementIsVisible`** (*Path expectedElement*)

Scroll right until the virtualized DOM contains the expect element, and it is visible Using 40 pixels steps, until the end of the table

Parameters

- **`expectedElement`** –
  - the element we are looking for

Returns the WebElement or throws an exception of not found

rightUntilPredicate

public WebElement **`rightUntilPredicate`** (*Path expectedElement*, Predicate<WebElement> *predicate*)

Scroll right until the DOM contains the expected element, and the given predicate regarding that element is met. Using 40 pixels steps, until the end of the table

Parameters

- **`expectedElement`** –
  - the element we are looking for
- **`predicate`** –
  - a condition regarding the expected element that is required to be met

Returns the WebElement or throws an exception of not found

rightUntilPredicate

public WebElement **rightUntilPredicate** (*Path expectedElement*, int *scrollStep*, int *maxNumberOfScrolls*, Predicate<WebElement> *predicate*)

Scroll right until the DOM contains the expected element and the supplied predicate for the element is met.

Parameters

- **expectedElement** –
  - the element we are looking for
- **scrollStep** –
  - scroll step in pixels
- **maxNumberOfScrolls** – maximum number of scroll operations
- **predicate** –
  - a condition regarding the expected element that is required to be met

Returns the WebElement or throws an exception of not found

toLeftCorner

public void **toLeftCorner** ()

Scroll to left-most point

toTopCorner

public void **toTopCorner** ()

Scroll to top-most point

toTopLeftCorner

public WebElement **toTopLeftCorner** (*Path expectedElement*)

Scroll down until the DOM contains the expected element. Using 40 pixels steps, until the end of the table

Parameters

- **expectedElement** –
  - the element we are looking for

Returns the WebElement or throws an exception of not found

toTopLeftCorner

public void **toTopLeftCorner** ()

Scroll to top-left corner

up

public void **up** (Integer *n*)
scroll up number of pixels

Parameters

- **n** – pixels

upUntilElementIsPresent

public WebElement **upUntilElementIsPresent** (*Path expectedElement*)
Scroll up until the virtualized DOM contains the expect element. Using 40 pixels steps, until the end of the table

Parameters

- **expectedElement** –
  - the element we are looking for

Returns the WebElement or throws an exception of not found

upUntilElementIsPresent

public WebElement **upUntilElementIsPresent** (*Path expectedElement*, int *scrollStep*, int *maxNumberOfScrolls*)
Scroll up until the virtualized DOM contains the expect element.

Parameters

- **expectedElement** –
  - the element we are looking for
- **scrollStep** –
  - scroll step in pixels
- **maxNumberOfScrolls** – maximum number of scroll operations

Returns the WebElement or throws an exception of not found

upUntilPredicate

public WebElement **upUntilPredicate** (*Path expectedElement*, Predicate<WebElement> *predicate*)
Scroll up until the DOM contains the expected element, and the given condition for that element is met. Using 40 pixels steps, until the end of the table

Parameters

- **expectedElement** –
  - the element we are looking for
- **predicate** –
  - a condition regarding the expected element that is required to be met

Returns the WebElement or throws an exception of not found

upUntilPredicate

public WebElement **upUntilPredicate**(*Path expectedElement*, int *scrollStep*, int *maxNumberOfScrolls*, Predicate<WebElement> *predicate*)

Scroll up until the DOM contains the expected element, and the supplied condition for that element is met.

Parameters

- **expectedElement** –
  - the element we are looking for
- **scrollStep** –
  - scroll step in pixels
- **maxNumberOfScrolls** – maximum number of scroll operations
- **predicate** –
  - a condition regarding the expected element that is required to be met

Returns the WebElement or throws an exception of not found

Path

public interface **Path**

The heart of DollarX is the definition of Paths that represent W3C elements, whether in the browser or in a document. Path describes an element, or elements, using API that emulates regular description in English, and can represent almost any element that can be expressed as the XPATH. Path is immutable - any additional constraint creates a new Path instance and does not change the original one. It is important to remember that defining new Paths is very cheap, since it does not interact with the browser. For example, in selenium, if we want to define a WebElement inside another WebElement, it must involve interaction with the browser (which involves potential race conditions):

```
WebElement wrapper = driver.findElement(By.cssSelector("div.foo"));
  WebElement field = wrapper.findElement(By.cssSelector(".bar"));
```

In DollarX it will look like:

```
final Path wrapper = element.withClass("foo");
  final Path field = element.withClass("bad").inside(wrapper);
```

Several points to note:

1. Once defined, Path values are final and can be reused without cost, as opposed to functions.
2. Creating arbitrarily complex Path is easy this way. It is far more maintainable than using explicit xpath.
3. Creating Paths has nothing to do with the browser.
4. The API offers many alternative ways to define equivalent Paths. The guideline is: If it means the same in English, it is mapped to an equivalent Path. For example, the following are equivalent:

```
element.that(hasClass("condition").and(isInside(dialog)));
    element.that(hasClass("condition")).and(isInside(dialog));
    element.inside(dialog).that(hasClass("condition"));
    element.withClass("condition").and(isInside(dialog));
    element.withClass("condition").and(isContainedIn(dialog));
    element.that(hasAncesctor(dialog)).and(hasClass("condition"));
```

(continues on next page)

(continued from previous page)

```
// if you prefer to break the definition to two steps:
Path condition = element.withClass("condition");
condition.inside(dialog);
```

5. Path::toString returns a string that reads like english and expresses exactly the definition of the path. This is very useful when troubleshooting.
6. To check what is the xpath a Path is mapped to, call `path.getXpath().get()`
7. Since it can be used as a wrapper of Selenium WebElement, you are not tied to using only DollarX - but can use it interchangeably with straight Selenium WebDriver.

The pattern in DollarX is to define exactly what you want to interact with or assert, as a Path, and then interact with the browser. This maximizes atomicity and performance, and avoid many of the pitfalls involved with interactions with a dynamic SPA. The standard implementation for Path is `BasicPath`.

Methods

after

Path **after** (*Path path*)

The element appears after the given path

Parameters

- **path** –
  - the element that appear before

Returns a new Path with the added constraint

afterSibling

Path **afterSibling** (*Path path*)

The element is a sibling of the given path, and appears after it

Parameters

- **path** –
  - the sibling element that appears before

Returns a new Path with the added constraint

ancestorOf

Path **ancestorOf** (*Path path*)

The element contains the given path, i.e. the given path parameter is inside the element

Parameters

- **path** –
  - the element that is inside our element

Returns a new Path with the added constraint

and

Path **and** (*ElementProperty*... *prop*)

Alias equivalent to `that`. Added for readability. Example:

```
div.that(hasClass("a")).and(hasText("foo"));
```

Parameters

- **prop** – a list of element properties (constraints)

Returns a new Path

before

Path **before** (*Path path*)

The element appears before the given path

Parameters

- **path** –
  - the element that appear after

Returns a new Path with the added constraint

beforeSibling

Path **beforeSibling** (*Path path*)

The element is a sibling of the given path, and appears before it

Parameters

- **path** –
  - the sibling element that appears after

Returns a new Path with the added constraint

childOf

Path **childOf** (*Path path*)

The element is a direct child of the given path

Parameters

- **path** –
  - the parent element

Returns a new Path with the added constraint

containing

Path **containing** (*Path path*)

The element contains the given path, i.e. the given path parameter is inside the element

Parameters

- **path** –
  - the element that is inside our element

Returns a new Path with the added constraint

contains

Path **contains** (*Path path*)

The element contains the given path, i.e. the given path parameter is inside the element

Parameters

- **path** –
  - the element that is inside our element

Returns a new Path with the added constraint

descendantOf

Path **descendantOf** (*Path path*)

The element is contained in the given path element, i.e. the given path parameter is wrapping it

Parameters

- **path** –
  - the element that is wrapping our element

Returns a new Path with the added constraint

describedBy

Path **describedBy** (String *description*)

A useful method to give a readable description to the path, for example: Suppose that instead of describing it's DOM positions and attributes, you prefer to describe it as "search result". Then you'd call: searchResult = myElement.describedBy("search result"); Now, calling System.out.println(firstOccurrenceOf(searchResult)), will print: "first occurrence of search result" This will replace its toString() result.

Parameters

- **description** – a readable description to that expresses the functionality of the path

Returns a new Path similar to the old one but that is described by the given description

getAlternateXPath

Optional<String> **getAlternateXPath** ()

Returns Should not be used unless you are developing for DollarX.

getDescribedBy

Optional<String> **getDescribedBy**()

Returns optional readable functional description of the Path

getElementProperties

List<*ElementProperty*> **getElementProperties**()

Returns Should not be typically used, unless you are developing for DollarX

getUnderlyingSource

Optional<WebElement> **getUnderlyingSource**()

Returns The WebElement that is used as the underlying reference for the Path. In most cases, this Optional is empty.

getXPath

Optional<String> **getXPath**()

The Optional xpath is maps to. Note that the prefix that marks it is inside the document (for example; "//" as the prefix of the xpath) can be omitted. This is not a concern - it will be added automatically by DollarX when interacting with the browser.

Returns an optional containing the xpath this Path is mapped to. The optional is empty only in case it is used as a wrapper of a WebElement (not the typical case).

getXpathExplanation

Optional<String> **getXpathExplanation**()

immediatelyAfterSibling

Path **immediatelyAfterSibling**(*Path path*)

The sibling right before the element matches the given path parameter

Parameters

- **path** –
  - the sibling element that appears after

Returns a new path with the added constraint

immediatelyBeforeSibling

Path **immediatelyBeforeSibling**(*Path path*)

The sibling right after the element matches the given path parameter

Parameters

- **path** –
  - the sibling element that appears after

Returns a new path with the added constraint

inside

Path **inside** (*Path path*)

Element that is inside another element

Parameters

- **path** –
  - the containing element

Returns a new Path with the added constraint

insideTopLevel

Path **insideTopLevel** ()

Returns an element that is explicitly inside the document. This is usually not needed - it will be added implicitly when needed.

Returns a new Path

or

Path **or** (*Path path*)

match more than a single path. Example: div.or(span) - matches both div and span

Parameters

- **path** – the alternative path to match

Returns returns a new path that matches both the original one and the given parameter

parentOf

Path **parentOf** (*Path path*)

The element is a parent of the given path

Parameters

- **path** –
  - the child element

Returns a new Path with the added constraint

that

Path **that** (*ElementProperty... prop*)

returns a path with the provided properties. For example: div.that(hasText("abc"), hasClass("foo"));

Parameters

- **prop** –
  - one or more properties. See ElementProperties documentation for details

Returns a new path with the added constraints

withClass

Path **withClass** (String *cssClass*)

The element has the given class name

Parameters

- **cssClass** – the class name

Returns a new Path with the added constraint

withClasses

Path **withClasses** (String... *cssClasses*)

The element has the given class names

Parameters

- **cssClasses** – the class names

Returns a new Path with the added constraint

withGlobalIndex

Path **withGlobalIndex** (Integer *index*)

Return the nth occurrence of the element in the entire document. Count starts at 1. The following expressions are equivalent: occurrenceNumber(4).of(myElement)); myElement.withGlobalIndex(4); Return the nth occurrence of the element in the entire document. Count starts at 1. For example: occurrenceNumber(3).of(listItem)

Parameters

- **index** – the number of occurrence

Returns a new Path with the added constraint

withText

Path **withText** (String *txt*)

Element with text equals (ignoring case) to txt. Equivalent to `path.that(hasText(txt))`

Parameters

- **txt** –
  - the text to equal to, ignoring case

Returns a new Path with the added constraint

withTextContaining

Path **withTextContaining** (String *txt*)

The element has text, containing the given txt parameter. The match is case insensitive.

Parameters

- **txt** – the text to match to

Returns a new Path with the added constraint

PathOperators

public final class **PathOperators**

Utilities to manipulate a `Path`

Methods

not

public static *Path* **not** (*Path path*)

Any element that does NOT conform to the definition of the given path parameters

Parameters

- **path** –
  - the path that represent what the element does NOT match

Returns a new path that represents the negation of the given parameter

PathParsers

public final class **PathParsers**

functions to find DOM elements in a W3C document. These functions are also useful to experiment and test with how Paths can be used to extract elements (they are used in many of the unit tests of DollarX).

```
Example use:
Path el = div.before(span);
    String xpath = el.getXPath().get();
    NodeList nodes = findAllByXpath("<div>foo</div><div>boo</div><span></span>",␣
↪el);
    assertThat(nodes.getLength(), is(2));
    assertThat(nodes.item(0).getTextContent(), equalTo("foo"));
```

Methods

findAllByPath

public static NodeList **findAllByPath** (String *docString*, *Path path*)

find all the nodes that match a path in a W3C document

Parameters

- **docString** – a W3C document
- **path** – the path to find.

Returns a node list with the details of all the elements that match the given path

findAllByPath

public static NodeList **findAllByPath** (Document *doc*, *Path path*)
find all the nodes that match a path in a W3C document

Parameters

- **doc** – a W3C document
- **path** – the path to find

Returns a node list with the details of all the elements that match the given path

findAllByXPath

public static NodeList **findAllByXPath** (Document *doc*, String *extractedXpath*)
internal implementation

Parameters

- **doc** – a W3C document
- **extractedXpath** – an xpath

Returns a node list with the details of all the elements that match the given xpath

getDocumentFromString

public static Document **getDocumentFromString** (String *document*)
Convert a string to a `Document`, Assuming utf-8 encoding.

Parameters

- **document** – the document as a string

Returns the document as a @link Document}

PathUtils

public final class **PathUtils**
Internal implementation.

Methods

hasHeirarchy

static boolean **hasHeirarchy** (String *xpath*)

oppositeRelation

static String **oppositeRelation** (String *relation*)

transformXpathToCorrectAxis

static Optional<String> **transformXpathToCorrectAxis** (*Path sourcePath*)

RelationOperator

public enum **RelationOperator**
Internal implementation.

Enum Constants

exactly

public static final *RelationOperator* **exactly**

orLess

public static final *RelationOperator* **orLess**

orMore

public static final *RelationOperator* **orMore**

Methods

opAsEnglish

public static String **opAsEnglish** (*RelationOperator op*)

opAsXpathString

public static String **opAsXpathString** (*RelationOperator op*)

XpathUtils

public final class **XpathUtils**
Internal implementation.

Fields

hasSomeText

public static final String **hasSomeText**

isHidden

public static final String **isHidden**

Methods

aggregatedCaseSensitiveTextEquals

public static String **aggregatedCaseSensitiveTextEquals** (String *text*)

aggregatedTextContains

public static String **aggregatedTextContains** (String *text*)

aggregatedTextEndsWith

public static String **aggregatedTextEndsWith** (String *text*)

aggregatedTextEquals

public static String **aggregatedTextEquals** (String *text*)

aggregatedTextStartsWith

public static String **aggregatedTextStartsWith** (String *text*)

aggregatedcaseSensitiveTextContains

public static String **aggregatedcaseSensitiveTextContains** (String *text*)

caseSensitiveTextContains

public static String **caseSensitiveTextContains** (String *text*)

caseSensitiveTextEquals

public static String **caseSensitiveTextEquals** (String *text*)

doesNotExist

public static String **doesNotExist** (String *path*)

doesNotExistInEntirePage

public static String **doesNotExistInEntirePage** (String *path*)

hasAnyOfClasses

public static String **hasAnyOfClasses** (String... *classNames*)

hasAttribute

public static String **hasAttribute** (String *attribute*, String *value*)

hasClass

public static String **hasClass** (String *className*)

hasClassContaining

public static String **hasClassContaining** (String *className*)

hasClasses

public static String **hasClasses** (String... *classNames*)

hasId

public static String **hasId** (String *id*)

insideTopLevel

public static String **insideTopLevel** (String *xpath*)

nOccurances

public static String **nOccurances** (String *xpath*, int *numberOfOccurrences*, *RelationOperator relationOperator*)

processTextForXpath

public static String **processTextForXpath** (String *txt*)

textContains

public static String **textContains** (String *text*)

textEndsWith

public static String **textEndsWith** (String *text*)

textEquals

public static String **textEquals** (String *text*)

textStartsWith

public static String **textStartsWith** (String *text*)

translateTextForPath

public static String **translateTextForPath** (String *txt*)

8.9.2 com.github.loyada.jdollarx.custommatchers

Custom Hamcrest matchers for assertions in tests - supports multiple instances of browsers, as well as assertions on a W3C Document

CustomMatchers

public class **CustomMatchers**

A collection of Hamcrest custom matchers, that are optimized to be as atomic as possible when interacting with the browser or a W3C document, and return useful error messages in case of a failure.

Methods

hasElement

public static Matcher<*InBrowser*> **hasElement** (*Path el*)

Successful if the browser has an element that corresponds to the given path. Example use:

```
assertThat( browser, hasElement(el));
```

Parameters

- **el** – the path to find

Returns a matcher for a browser that contains the element

hasElements

public static *HasElements* **hasElements** (*Path path*)

Successful if element is present in the browser or a W3C document. Useful especially when you have a reference count. This matcher is optimized. For example:

```
assertThat(browser, hasElements(path).present(5).times());
   assertThat(browser, hasElements(path).present(5).timesOrMore());
   assertThat(document, hasElements(path).present(5).timesOrLess());
```

Parameters

- **path** – The path of the elements to find

Returns a matcher for the number of times an element is present.

hasNoElement

public static Matcher<*InBrowser*> **hasNoElement** (*Path el*)

Successful if given browser has no elements that correspond to the given path. The implementation of this is optimized. For example:

```
assertThat( browser, hasNoElement(path));
```

Parameters

- **el** –
  - the path that is expected not to exist in the browser

Returns a matcher that is successful if the element does not appear in the browser

hasText

public static *HasText* **hasText** (String *text*)

Successful if element has the text equal to the given parameter in the browser/document. Example use:

```
assertThat( path, hasText().in(browser));
```

Parameters

- **text** – the text to equal to (case insensitive)

Returns a custom Hamcrest matcher

isAbsentFrom

public static Matcher<*Path*> **isAbsentFrom** (*InBrowser browser*)

Successful if given browser has no elements that correspond to the given path. Equivalent to hasNoElement() matcher. This is much better than doing not(isPresent()), because in case of success (i.e. the element is not there), it will return immidiately, while the isPresent() will block until timeout is reached. For example:

```
assertThat( path, isAbsentFrom(browser));
```

Parameters

- **browser** – the browser instance to look in

Returns a matcher that is successful if the element does not appear in the browser

isAbsentFrom

public static Matcher<*Path*> **isAbsentFrom** (Document *document*)

Successful if given document has no elements that correspond to the given path. For example:

```
assertThat( path, isAbsentFrom(doc));
```

Parameters

- **document** –
  - a W3C document

Returns a matcher that is successful if the element does not appear in the document

isDisplayedIn

public static Matcher<*Path*> **isDisplayedIn** (*InBrowser browser*)

Successful if given element is present and displayed in the browser. Relies on WebElement.isDisplayed(), thus non-atomic. For example:

```
assertThat( path, isDisplayedIn(browser));
```

Parameters

- **browser** – the browser instance to look in

Returns a matcher that checks if an element is displayed in the browser

isEnabledIn

public static Matcher<*Path*> **isEnabledIn** (*InBrowser browser*)

Successful if given element is present and enabled in the browser. Relies on WebElement.isEnabled(), thus non-atomic. For example:

```
assertThat( path, isEnabledIn(browser));
```

Parameters

- **browser** – the browser instance to look in

Returns a matcher that checks if an element is enabled in the browser

isNotDisplayedIn

public static Matcher<*Path*> **isNotDisplayedIn** (*InBrowser browser*)

Successful if given element is either not present, or present and not displayed in the browser.

Relies on WebElement.isDisplayed(), thus non-atomic. For example: `assertThat( path, isNotDisplayedIn(browser));`

Parameters

- **`browser`** – the browser instance to look in

Returns a matcher that checks if an element is displayed in the browser

isPresent

public static *IsPresentNTimes* **`isPresent`** (int *nTimes*)

Successful if the the element appears the expected number of times in the browser or W3C document. This matcher is optimized. Example use for browser interaction:

```
InBrowser browser = new InBrowser(driver);
  assertThat( myElement, ispresent(5).timesOrMoreIn(browser));
  assertThat( myElement, ispresent(5).timesIn(browser));
  assertThat( myElement, ispresent(5).timesOrLessIn(browser));
```

Same examples apply in case you have a Document (org.w3c.dom.Document).

Parameters

- **`nTimes`** –
  - the reference number of times to be matched against. See examples.

Returns a matcher that matches the number of times an element is present. See examples in the description.

isPresent

public static *IsPresent* **`isPresent`** ()

Successful if element is present in the browser/document. Example use:

```
assertThat( path, isPresent().in(browser));
```

Returns a custom Hamcrest matcher

isPresentIn

public static Matcher<*Path*> **`isPresentIn`** (*InBrowser browser*)

Successful if given element is present in the browser. For example:

```
assertThat( path, isPresentIn(browser));
```

Parameters

- **`browser`** – the browser instance to look in

Returns a matcher that checks if an element is present in a browser

isPresentIn

public static Matcher<*Path*> **isPresentIn** (Document *document*)

Successful if given element is present in the document. For example:

```
assertThat( path, isPresentIn(document));
```

Parameters

- **document** –
  - a W#C document

Returns a matcher that checks if an element is present in a document

isSelectedIn

public static Matcher<*Path*> **isSelectedIn** (*InBrowser browser*)

Successful if given element is present and selected in the browser. Relies on WebElement.isSelected(), thus non-atomic. For example:

```
assertThat( path, isSelectedIn(browser));
```

Parameters

- **browser** – the browser instance to look in

Returns a matcher that checks if an element is selected in the browser

CustomMatchersUtil

public final class **CustomMatchersUtil**

Internal implementation.

Methods

wrap

public static String **wrap** (*Path el*)

CustomMatchersUtil.ISPresentNTimesMatcherForDocument

public static class **ISPresentNTimesMatcherForDocument** extends TypeSafeMatcher<*Path*>

Internal implementation

Fields

foundNTimes

int **foundNTimes**

Constructors

ISPresentNTimesMatcherForDocument

public **ISPresentNTimesMatcherForDocument** (int *nTimes*, *RelationOperator relationOperator*, Document *doc*)

Methods

describeMismatchSafely

protected void **describeMismatchSafely** (*Path el*, Description *mismatchDescription*)

describeTo

public void **describeTo** (Description *description*)

matchesSafely

protected boolean **matchesSafely** (*Path el*)

toString

public String **toString** ()

CustomMatchersUtil.NTimesMatcher

public static class **NTimesMatcher** extends TypeSafeMatcher<*Path*>

Internal implementation

Fields

foundNTimes

int **foundNTimes**

Constructors

NTimesMatcher

public **NTimesMatcher** (int *nTimes*, *RelationOperator relationOperator*, *InBrowser browser*)

Methods

describeMismatchSafely

protected void **`describeMismatchSafely`** (*Path el*, Description *mismatchDescription*)

describeTo

public void **`describeTo`** (Description *description*)

matchesSafely

protected boolean **`matchesSafely`** (*Path el*)

toString

public String **`toString`** ()

HasElementNTimes

public class **`HasElementNTimes`**

Internal implementation - not to be instantiated directly. This matcher is optimized for the success use-case. In that case it match for a single element with exact number of elements wanted. In case of failure, it will make another call to get the actual number of elements on the page, in order to provide a detailed error message. So the trade off is: In case of success it's faster, In case of failure it's slower. It makes sense since most of the time we expect success.

Constructors

HasElementNTimes

public **`HasElementNTimes`** (*Path path*, int *nTimes*)

Methods

times

public Matcher<*InBrowser*> **`times`** ()

matches the exact number given

Returns a matcher

timesOrLess

public Matcher<*InBrowser*> **timesOrLess** ( )
matches the number given, or less

Returns a matcher

timesOrMore

public Matcher<*InBrowser*> **timesOrMore** ( )
matches the number given, or more

Returns a matcher

HasElementNTimes.NTimesMatcher

public static class **NTimesMatcher** extends TypeSafeMatcher<*InBrowser*>
Internal implementation - not to be instantiated directly.

Fields

foundNTimes

int **foundNTimes**

Constructors

NTimesMatcher

NTimesMatcher (*Path path*, int *nTimes*, *RelationOperator relationOperator*)

Methods

describeMismatchSafely

protected void **describeMismatchSafely** (*InBrowser browser*, Description *mismatchDescription*)

describeTo

public void **describeTo** (Description *description*)

matchesSafely

protected boolean **matchesSafely** (*InBrowser browser*)

toString

public String **toString**()

HasElements

public class **HasElements**
Internal implementation.

Constructors

HasElements

public **HasElements**(*Path path*)

Methods

present

public *HasElementNTimes* **present**(int *nTimes*)

HasText

public class **HasText**
Internal implementation.

Constructors

HasText

public **HasText**(String *text*)

Methods

in

public Matcher<*Path*> **in**(*InBrowser browser*)

in

public Matcher<*Path*> **in**(Document *document*)

IsPresent

public class **IsPresent**
Internal implementation.

Constructors

IsPresent

public **IsPresent** ()

Methods

in

public Matcher<*Path*> **in** (*InBrowser browser*)

in

public Matcher<*Path*> **in** (Document *document*)

IsPresentNTimes

public class **IsPresentNTimes**
Internal implementation - not to be instantiated directly. This matcher is optimized for the success use-case. In that case it match for a single element with exact number of elements wanted. In case of failure, it will make another call to get the actual number of elements on the page, in order to provide a detailed error message. So the trade off is: In case of success it's faster, In case of failure it's slower. It makes sense since most of the time we expect success.

Constructors

IsPresentNTimes

public **IsPresentNTimes** (int *nTimes*)

Methods

timesIn

public Matcher<*Path*> **timesIn** (*InBrowser browser*)

timesIn

public Matcher<*Path*> **timesIn** (Document *doc*)

timesOrLessIn

public Matcher<*Path*> **timesOrLessIn** (*InBrowser browser*)

timesOrLessIn

public Matcher<*Path*> **timesOrLessIn** (Document *doc*)

timesOrMoreIn

public Matcher<*Path*> **timesOrMoreIn** (*InBrowser browser*)

timesOrMoreIn

public Matcher<*Path*> **timesOrMoreIn** (Document *doc*)

8.9.3 com.github.loyada.jdollarx.highlevelapi

Package for high level interactions. The API is less flexible than the standard API and sometimes not optimal, but it is simplified.

CheckBox

public class **CheckBox**

High-level wrapper to define and interact with a checkbox input. High level API's are not optimized. A definition of an element may interact with the browser to understand the structure of the DOM.

Constructors

CheckBox

public **CheckBox** (*InBrowser browser*, String *labelText*)

input of type "checkbox" with a label

Parameters

- **labelText** –
  - the text in the label

CheckBox

public **CheckBox** (*InBrowser browser*, *ElementProperty*... *props*)

input of type "checkbox" with the given properties

Parameters

- **props** –
  - properties of the checkbox

Methods

check

public void **check** ( )
Check it

isChecked

public boolean **isChecked** ( )
Is it checked?

Returns whether it is checked

toString

public String **toString** ( )

uncheck

public void **uncheck** ( )
Unchecked it

Inputs

public final class **Inputs**
High-level API to define and interact with various input elements. High level API's are not optimized. A definition of an element may interact with the browser to understand the structure of the DOM.

Methods

changeInputValue

public static void **changeInputValue** (*InBrowser browser*, *Path field*, String *text*)
Change input value: clear it and then enter another text in it

Parameters

- **browser** – the browser
- **field** – Path to the input field
- **text** – the text to enter in the input field

Throws

- ***OperationFailedException*** – failed to perform the operation

changeInputValueWithEnter

public static void **changeInputValueWithEnter** (*InBrowser browser*, *Path field*, String *text*)

Similar to changeInputValue, but adds an ENTER after setting the value of the input

Parameters

- **browser** – the browser
- **field** – Path to the input field
- **text** – the text to enter in the input field

Throws

- ***OperationFailedException*** – failed to perform the operation

checkboxType

public static *Path* **checkboxType** (*Path inp*)

Takes an input element and returns such an input of type checkbox.

Parameters

- **inp** – the input element

Returns a Path to the input

clearInput

public static void **clearInput** (*InBrowser browser*, *Path field*)

Clear operation on an input element

Parameters

- **browser** – the browser
- **field** – the input element

inputFollowedByUnlabeledText

public static *Path* **inputFollowedByUnlabeledText** (String *text*)

Input followed by text that does not have its on label element.

Parameters

- **text** – the text following the input

Returns a Path to the input element

inputForLabel

public static *Path* **inputForLabel** (*InBrowser browser*, String *labelText*)

A lazy way to find an input based on the label. Mote that unlike It looks for a label element that has an ID. If it finds one, it returns a Path to an input with that ID. Otherwise it returns a Path to an input inside the label element.

Parameters

- **browser** – the browser
- **labelText** – the label to look for

Returns a Path to the input, on a best effort basis

radioType

public static *Path* **radioType** (*Path inp*)

Takes an input element and returns such an input of type radio.

Parameters

- **inp** – the input element

Returns a Path to the input

selectInFieldWithLabel

public static *Path* **selectInFieldWithLabel** (*InBrowser browser*, String *labelText*, String *option*)

Perform a selection of an option in a select element. It expects to find the label element with the given text before the select element

Parameters

- **browser** – the browser
- **labelText** – The text of the select label
- **option** – The option text

Returns the Path of the select element

RadioInput

public class **RadioInput**

High-level API to define and interact with. High level API's are not optimized. A definition of an element may interact with the browser to understand the structure of the DOM.

Constructors

RadioInput

public **RadioInput** (*InBrowser browser*, *Path thePath*)

a radio button input with the given path. The given path is not validated.

Parameters

- **thePath** – the Path of the radio button

RadioInput

public **RadioInput** (*InBrowser browser*, *ElementProperty... props*)

a radio input with some properties

Parameters

- **`props`** – the properties of the radio button

Methods

isSelected

public boolean **`isSelected`**()
is it currently selected?

Returns whether this radio button is selected

select

public void **`select`**()
Ensure it is selected

toString

public String **`toString`**()

withLabeledText

public static *RadioInput* **`withLabeledText`**(*InBrowser browser*, String *labelText*)
create and return a RadioInput, that has a "label" element with the given text. Note that this is not a pure declaration and it looks for the label in the browser.

Parameters

- **`labelText`** –
  - the text in the label

Returns

- a RadioInput instance

withTextUnknownDOM

public static *RadioInput* **`withTextUnknownDOM`**(*InBrowser browser*, String *text*, int *originalImplicitWait*, TimeUnit *timeUnit*)
In case the organization of the DOM is unclear, it will try both labeled input and unlabeled input. When doing so, it will change the implicit wait temporarily to a small value, and then revert the implicit timeout to the values provided. Use this only if you are not sure about the structure of the DOM.

Parameters

- **`text`** –
  - the text following the radio button
- **`originalImplicitWait`** –
  - the current implicit wait

- **timeUnit** –
  - the current time unit of the implicit wait

Returns a RadioInput instance

withUnlabeledText

public static *RadioInput* **withUnlabeledText** (*InBrowser browser*, String *text*)

create and return a RadioInput, that has straight text after it (not in a "label" element). i.e.:

```
Male
Female
```

Parameters

- **text** –
  - the text following the radio button

Returns

- a RadioInput instance

8.9.4 com.github.loyada.jdollarx.singlebrowser

Simplified API for interaction with a single instance of browser

AgGrid

public class **AgGrid**

Custom class to validate the presence of an AgGrid, and interact with it, since it can be tricky. It supports virtualized and non-virtualized tables. It should be used like other custom matchers in the package.

Fields

AgBody

public static final *Path* **AgBody**

AgGridRoot

public static final *Path* **AgGridRoot**

AgList

public static final *Path* **AgList**

AgListOption

public static final *Path* **AgListOption**

CELL

public static final *Path* **CELL**

CHECKBOX

public static final *Path* **CHECKBOX**

COL_ID

public static final String **COL_ID**

HEADER_CELL

public static final *Path* **HEADER_CELL**

HEADER_MENU

public static final *Path* **HEADER_MENU**

HEADER_TXT

public static final *Path* **HEADER_TXT**

ROW

public static final *Path* **ROW**

Methods

clickMenuOfHeader

public void **clickMenuOfHeader** (String *headerText*)

Click on the menu of a the column with the given header

Parameters

- **headerText** –
  - the header text, or the column ID. A string wrapped with curly braces is interpreted as the column ID.

clickOnSort

public void **clickOnSort** (String *headerText*)

Click on the 'sort' column with the given header

Parameters

- **headerText** –
  - the header text, or the column ID. A string wrapped with curly braces is interpreted as the column ID.

ensureVisibilityOfCellInColumn

public *Path* **ensureVisibilityOfCellInColumn** (String *columnTitle*, *ElementProperty* *cellContent*)

Find a specific cell under a column, without knowing the row, ensure it is displayed, and return its Path

Parameters

- **columnTitle** – the title of the column to look under
- **cellContent** – a property that describes the content of the expect cell

Returns the Path of the found cell. allows to interact with it

ensureVisibilityOfRowWithIndex

public *Path* **ensureVisibilityOfRowWithIndex** (int *n*)

Scroll until the row with the given index is visible, and return a Path element that matches it. Useful for performing operations or accessing fields in the wanted row.

Parameters

- **n** – the number of row in the table, as visible to the user

Returns a Path element that allows to access the row

ensureVisibilityOfRowWithIndexAndColumn

public *Path* **ensureVisibilityOfRowWithIndexAndColumn** (int *index*, String *columnTitle*)

Scroll until the row with the given index, as well as the given column, is visible. It return a Path element that matches the wanted cell in row. Useful for performing operations or accessing fields in the wanted cell (for example: edit it)

Parameters

- **index** – the number of row in the table, as visible to the user
- **columnTitle** – the header title of the wanted cell in the row

Returns the Path element to access the wanted cell in the wanted row

findRowIndex

public int **findRowIndex** (Map<String, *ElementProperty*> *row*)

Find internal index of row within table. This method typically will make sure the row is also visible if it exists, in case the user needs to interact with it, but in some cases ensureVisiblityOfRow will be required.

Parameters

- **row** –
  - the definition of the row content

Returns the internal index of the row, if it was found

findTableInBrowser

public void **findTableInBrowser**()

getBuilder

public static *AgGridBuilder* **getBuilder**()

getRowIndex

public int **getRowIndex**(*Path row*)

assuming the row is already present in the DOM, get its internal index in the table.

Parameters

- **row** – the row we are interested in. Should be the value returned from findRowInBrowser() or ensureVisibilityOfRowWithIndex()

Returns the internal index of the row in the table

getRowIndexOfCell

public int **getRowIndexOfCell**(*Path cell*)

assuming the row is already present in the DOM, get its internal index in the table.

Parameters

- **cell** –
  - the cell in the row we are interested in. Should be the return value of ensureVisibilityOfRowWithIndexAndColumn()

Returns the internal index of the row in the table

getVisibleHeaderPath

public *Path* **getVisibleHeaderPath**(String *headerText*)

Make sure the given column header is visible, and returns a Path element to access it. This is useful to perform direct operations on that element or access other DOM elements contained in the header.

Parameters

- **headerText** –
  - the header text, or the column ID. A string wrapped with curly braces is interpreted as the column ID.

Returns the Path element to access the column header

hasIndex

static *ElementProperty* **hasIndex** (int *ind*)

isVirtualized

public boolean **isVirtualized** ()

openColumnFilterTabAndGetMenu

public *Path* **openColumnFilterTabAndGetMenu** (String *headerText*)
open the popup filter for the column

Parameters

- **headerText** –
  - the header text, or the column ID, to open the popup menu from. A string wrapped with curly braces is interpreted as the column ID.

Returns the Path to the popup menu

openColumnMenuTabAndGetMenu

public *Path* **openColumnMenuTabAndGetMenu** (String *headerText*)
open the popup menu for the column

Parameters

- **headerText** –
  - the header text, or the column ID, to open the popup menu from. A string wrapped with curly braces is interpreted as the column ID.

Returns the Path to the popup menu

openColumnsSelectionMenuAndGetMenu

public *Path* **openColumnsSelectionMenuAndGetMenu** (String *headerText*)
open the popup columns show/hide selection by using a popup of the given column

Parameters

- **headerText** –
  - the header text, or the column ID, to open the popup menu from. A string wrapped with curly braces is interpreted as the column ID.

Returns the Path to the popup menu

openColumnsSelectionMenuAndGetMenu

public *Path* **openColumnsSelectionMenuAndGetMenu** ()
open the popup columns show/hide selection by using a popup of the first column (assumes it is active)

Returns the Path to the popup menu

overrideTimeoutDuringOperation

public void **overrideTimeoutDuringOperation** (int *millisecs*)

Override the default timeout threshold for finding elements while scrolling the table. The default is 5 milliseconds

Parameters

- **millisecs** –
  - the timeout in milliseconds

overrideTimeoutWhenDone

public void **overrideTimeoutWhenDone** (int *millisecs*)

Override the default timeout threshold it reverts to when table operations are done. The default is 5000 milliseconds

Parameters

- **millisecs** –
  - the timeout in milliseconds

rowOfGrid

public static *Path* **rowOfGrid** (*Path gridContainer*)

setFinalTimeout

public void **setFinalTimeout** ()

setScrollStep

public void **setScrollStep** (int *size*)

Override the default step size of scrolling when moving through a grid

Parameters

- **size** – step size in pixels

showAllColumnsUsingFirstColumn

public void **showAllColumnsUsingFirstColumn** ()

Show all columns, by opening the popup menu of the first column. Assumes that the first column has the popup menu.

showAllColumnsUsingMenuOfColumn

public void **showAllColumnsUsingMenuOfColumn** (String *headerText*)

- Show all columns by using the popup menu of the given header.

Parameters

- **headerText** –
  - the header text, or the column ID, to open the popup menu from. A string wrapped with curly braces is interpreted as the column ID.

showSpecificColumnsUsingMenuOfColumn

public void **showSpecificColumnsUsingMenuOfColumn** (String *headerText*, List<String> *columns*)

Show only specific columns, by opening the popup menu of the given column

Parameters

- **headerText** –
  - the header text, or the column ID, to open the popup menu from. A string wrapped with curly braces is interpreted as the column ID.
- **columns** –
  - the columns to show

showSpecificColumnsUsingMenuOfColumn

public void **showSpecificColumnsUsingMenuOfColumn** (List<String> *columns*)

Show only specific columns, by opening the popup menu of the first column. Assumes that the first column has the popup menu.

Parameters

- **columns** –
  - the columns to show

sortBy

public void **sortBy** (String *headerText*, *SortDirection direction*)

Click on 'sort' so that the given column is sorted in the direction provided.

Parameters

- **headerText** –
  - the header text, or the column ID. A string wrapped with curly braces is interpreted as the column ID.
- **direction** –
  - wanted direction. either descending or ascending.

Throws

- *OperationFailedException* – operation failed - typically the configuration of the grid does not allow to sort as wanted.

toString

public String **toString**()

AgGrid.AgGridBuilder

public static class **AgGridBuilder**

Fields

isVirtualized

boolean **isVirtualized**

Methods

build

public *AgGrid* **build**()

Create an AgGrid definition

Returns AgGrid instance

containedIn

public *AgGridBuilder* **containedIn**(*Path container*)

optional - define the container of the grid

Parameters

- **container** – the Path of the container of the grid

Returns AgGridBuilder

isStrict

public *AgGridBuilder* **isStrict**()

The assertions will be strict - if there are extra rows, it will fail.

Returns AgGridBuilder

withHeaders

public *AgGridBuilder* **withHeaders**(List<String> *headers*)

The headers of the columns

Parameters

- **headers** –
  - the headers of the columns. In case you prefer to use a column ID, wrap it with {}. For example, "{the-id}" will refer to a header with a column ID of "the-id". This is useful when a column has no textual header.

Returns AgGridBuilder

withRowsAsElementProperties

public *AgGridBuilder* **withRowsAsElementProperties** (List<Map<String, *ElementProperty*>> *rows*)

Define the rows in the table, in order.

Parameters

- **rows** –
  - A list of rows, where each row is a map of the column name(or column ID) to the property that describes the expected content. To use a column Id as a key, wrap it with curly braces.

Returns AgGridBuilder

withRowsAsElementPropertiesInOrder

public *AgGridBuilder* **withRowsAsElementPropertiesInOrder** (List<List<Map.Entry<String, *ElementProperty*>>> *rows*)

Define the rows in the table, in order. This version can be faster, in case the columns are ordered as they appear in the table, and the table is virtualized

Parameters

- **rows** –
  - A list of rows, where each row is a map of the column name to the property that describes the expected content

Returns AgGridBuilder

withRowsAsStrings

public *AgGridBuilder* **withRowsAsStrings** (List<Map<String, String>> *rows*)

Define the rows in the table, in order.

Parameters

- **rows** –
  - A list of rows, where each row is a map of the column name(or column ID) to the text. To use a column Id as a key, wrap it with curly braces.

Returns AgGridBuilder

withRowsAsStringsInOrder

public *AgGridBuilder* **withRowsAsStringsInOrder** (List<List<Map.Entry<String, String>>> *rows*)

Define the rows in the table, in order. This version can be faster, in case the columns

Parameters

- **rows** –
  - A list of rows, where each row is a map of the column name(or column ID) to the text. To use a column Id as a key, wrap it with curly braces.

Returns AgGridBuilder

withoutVirtualization

public *AgGridBuilder* **withoutVirtualization**()
without virtualization. The default is with virtualization.

Returns AgGridBuilder

AgGrid.SortDirection

public enum **SortDirection**

Enum Constants

ascending

public static final *AgGrid.SortDirection* **ascending**

descending

public static final *AgGrid.SortDirection* **descending**

none

public static final *AgGrid.SortDirection* **none**

Methods

byCssClass

static *SortDirection* **byCssClass**(String *cssClassName*)

getAllClasses

static String[] **getAllClasses**()

getCssClassName

String **getCssClassName**()

AgGridHighLevelOperations

public final class **AgGridHighLevelOperations**
High level utilities for definitions of simplified grids and operations

Constructors

AgGridHighLevelOperations

public **AgGridHighLevelOperations** (*Path gridContainer*)

Methods

buildMinimalGridFromHeader

public *AgGrid* **buildMinimalGridFromHeader** (List<String> *headers*)

cellInGrid

public *Path* **cellInGrid** (int *rowNumber*, String *columnName*)
Ensure a specific cell is visible and return a Path to it

Parameters

- **rowNumber** – row number
- **columnName** – column name

Returns the request cell

changeSimpleInputValueByRowNumber

public void **changeSimpleInputValueByRowNumber** (String *columnName*, int *rowNumber*, String *newValue*)
select an option from a dropdown in a cell

Parameters

- **columnName** – column name
- **rowNumber** – row number
- **newValue** – new Value

changeSimpleInputValueByValue

public void **changeSimpleInputValueByValue** (String *columnName*, String *oldValue*, String *newValue*)
select an option from a dropdown in a cell

Parameters

- **columnName** – column name

- **oldValue** – row number
- **newValue** – new Value

clickOnColumnWithValue

public *Path* **clickOnColumnWithValue** (String *columnName*, String *value*)

Find a the first cell in the given column with the given value, ensure it is visible, and click on it.

Parameters

- **columnName** – the column name
- **value** – the value of the cell we are looking for

Returns the cell element

ensureCellValueIsPresent

public void **ensureCellValueIsPresent** (int *rowNumber*, String *columnTitle*, String *expectedValue*)

Ensure(or assert) that the cell in specific row and column has the expected value

Parameters

- **rowNumber** –
  - number of row of the cell
- **columnTitle** –
  - the column of the cell
- **expectedValue** –
  - the value we assert in that cell

getMinimalGrid

public *AgGrid* **getMinimalGrid** (String *columnName*)

create a minimal grid definition that has the column we are intererested in

Parameters

- **columnName** – the column name

Returns a grid object

goToEditModeInCell

public *Path* **goToEditModeInCell** (String *columnName*, int *rowNumber*)

Find a cell, and doubleclick it

Parameters

- **columnName** – column name
- **rowNumber** – row number

Returns the cell

goToEditModeInCell

public *Path* **goToEditModeInCell** (String *columnName*, String *value*)
Find a cell, and doubleclick it

Parameters

- **columnName** – column name
- **value** – value of cell

Returns the cell

hoverOverCell

public *Path* **hoverOverCell** (int *rowNumber*, String *columnTitle*)
Hover over speicic cell, after ensuring it is visible

Parameters

- **rowNumber** –
  - row number
- **columnTitle** –
  - column

Returns the cell

selectInCell

public void **selectInCell** (String *columnName*, int *rowNumber*, String *option*)
select an option from a dropdown in a cell

Parameters

- **columnName** – column name
- **rowNumber** – row number
- **option** – option to choose

unorderedGrid

public *AgGrid* **unorderedGrid** (List<Map<String, String>> *rows*)
define AgGrid with unordered rows

Parameters

- **rows** – a list of the rows, ignoring the order

Returns an AgGrid object

HighLevelPaths

public class **HighLevelPaths**

Methods

inputFor

public static *Path* **inputFor** (String *labelText*)

InBrowserSinglton

public final class **InBrowserSinglton**

A simplified API built to interact with a single instance of a running browser. See `com.github.loyada.jdollarx.InBrowser` for an API that supports multiple browser instances.

Fields

driver

public static WebDriver **driver**

Methods

clickAt

public static WebElement **clickAt** (*Path el*)

Click on the location of the element that corresponds to the given path.

Parameters

- **el** – a Path instance

Returns the WebElement clicked at

clickOn

public static WebElement **clickOn** (*Path el*)

Click on the element that corresponds to the given path. Requires the element to be clickable.

Parameters

- **el** – a Path instance

Returns the WebElement clicked on

contextClick

public static WebElement **contextClick** (*Path el*)

Context click (right click) on the location of the element that corresponds to the given path.

Parameters

- **el** – a Path instance

Returns the WebElement clicked at

doubleClickOn

public static void **doubleClickOn** (*Path el*)

Doubleclick on the element that corresponds to the given path. Requires the element to be clickable.

Parameters

- **el** – a Path instance

dragAndDrop

public static *Operations.DragAndDrop* **dragAndDrop** (*Path path*)

Drag and drop in the browser. Several flavors of use:

```
dragAndDrop(source).to(target);
dragAndDrop(source).to(xCor, yCor);
```

Parameters

- **path** – the path of the element to drag and drop

Returns a DragAndDrop instance that allows to drag and drop to another element or to another location

find

public static WebElement **find** (*Path el*)

Equivalent to WebDriver.findElement(). If the Path contains a WebElement than it will look for an element inside that WebElement. Otherwise it looks starting at the top level. It also alters the xpath if needed to search from top level correctly.

Parameters

- **el** – a Path instance

Returns returns a WebElement or throws an ElementNotFoundException

findAll

public static List<WebElement> **findAll** (*Path el*)

Equivalent to WebDriver.findElements(). If the Path contains a WebElement than it will look for an element inside that WebElement. Otherwise it looks starting at the top level. It also alters the xpath if needed to search from top level correctly.

Parameters

- **el** – a Path instance

Returns a list of WebElements.

getCssClasses

public static List<String> **getCssClasses** (*Path el*)

Get all classes of given Path element.

Parameters

- **el** – the element to look for

Returns a list of classes

getSelect

public static Select **getSelect** (*Path el*)

Get a Selenium select element, which provides a high level API to interacting with a "select" menu. Since the Selenium API is good enough, there was no need to create a specialized dollarx version.

Parameters

- **el** –
  - must be a "select" path, with "option" elements for the various selectable options.

Returns org.openqa.selenium.support.ui.Select instance

hoverOver

public static WebElement **hoverOver** (*Path el*)

Hover over on the location of the element that corresponds to the given path.

Parameters

- **el** – a Path instance

Returns the WebElement found

isDisplayed

public static boolean **isDisplayed** (*Path el*)

Relies on Selenium WebElement::isDisplayed, thus non-atomic.

Parameters

- **el** – the path of the element to find

Returns true if the element is present and displayed

isEnabled

public static boolean **isEnabled** (*Path el*)

Relies on Selenium WebElement::isEnabled, thus non-atomic.

Parameters

- **el** – the path of the element to find

Returns true if the element is present and enabled

isPresent

public static boolean **isPresent** (*Path el*)

Parameters

- **el** – a Path instance

Returns true if the element is present.

isSelected

public static boolean **isSelected** (*Path el*)

Relies on Selenium WebElement::isSelected, thus non-atomic.

Parameters

- **el** – the path of the element to find

Returns true if the element is present and selected

numberOfAppearances

public static Integer **numberOfAppearances** (*Path el*)

Typically should not be used directly. There are usually better options.

Parameters

- **el** – a Path instance

Returns tbe number of appearances of an element.

pressKeyDown

public static *Operations.KeysDown* **pressKeyDown** (CharSequence *thekey*)

Press key down in the browser, or on a specific element. Two flavors of use:

```
pressKeyDown(Keys.TAB).inBrowser();
pressKeyDown(Keys.TAB).on(path);
```

Parameters

- **thekey** – the key to press

Returns a KeysDown instance that allows to send to the browser in general or to a specific element in the DOM. See example.

releaseKey

public static *Operations.ReleaseKey* **releaseKey** (CharSequence *thekey*)

Release key in the browser, or on a specific element. Two flavors of use:

```
releaseKey(Keys.TAB).inBrowser();
releaseKey(Keys.TAB).on(path);
```

Parameters

- **thekey** – the key to release

Returns a ReleaseKey instance that allows to send to the browser in general or to a specific element in the DOM. See example.

rightClick

public static WebElement **rightClick** (*Path el*)

Context click (right click) on the location of the element that corresponds to the given path.

Parameters

- **el** – a Path instance

Returns the WebElement clicked at

scroll

public static *Operations.Scroll* **scroll** ()

scroll the browser. Several flavors of use:

```
browser.scroll().to(path);
   browser.scroll().left(50);
   browser.scroll().right(50);
   browser.scroll().up(50);
   browser.scroll().down(50);
```

Returns a Scroll instance that allows to scroll by offset or to a location of a DOM element

scrollElement

public static *Operations.ScrollElement* **scrollElement** (*Path el*)

scroll within the given element. Useful especially when working with grids.

Parameters

- **el** – a Path instance

Returns the WebElement found

scrollElementWithStepOverride

public static *Operations.ScrollElement* **scrollElementWithStepOverride** (*Path el*, int *step*)

scroll within the given element. Useful especially when working with grids.

Parameters

- **el** – a Path instance
- **step** – step size override

Returns the WebElement found

scrollTo

public static WebElement **scrollTo** (*Path el*)

scroll to the location of the element that corresponds to the given path.

Parameters

- **el** – a Path instance

Returns the WebElement found

sendKeys

public static *Operations.KeysSender* **sendKeys** (CharSequence... *charsToSend*)

send keys to the browser, or to a specific element. Two flavors of use:

```
sendKeys("abc").toBrowser();
     sendKeys("abc").to(path);
```

Parameters

- **charsToSend** – the keys to send. Can be "abc", or "a", "b", "c"

Returns a KeySender instance that allows to send to the browser in general or to a specific element in the DOM

Obscure

public class **Obscure** extends *Images.Obscure* implements AutoCloseable

class that allows to hide elements temporarily. This is useful when doing visual testing, while ignoring elements that are not interesting for the test. For example - testing a chart while ignoring certain labels. This is an Autocloseable: it reverts to the original state when leaving the try{} block.

Constructors

Obscure

public **Obscure** (*Path element*)

Make the first element matching the given path temporarily hidden. If the element is not found, it ignores it.

Parameters

- **element** – the path of the element to obscure

Obscure

public **Obscure** (List<*Path*> *elements*)

Make the elements matching the given paths temporarily hidden. In case there are multiple matches, it will hide the first one. If the element is not found, it ignores it.

Parameters

- **elements** – the elements to obscure

Obscure

public **Obscure** (List<*Path*> *elements*, boolean *strict*)

Make the elements matching the given paths temporarily hidden. In case there are multiple matches, it will hide the first one.

Parameters

- **elements** – the elements to obscure
- **strict** – in strict mode, if the element is not found, it throws am exception and stops

SingleBrowserPath

public final class **SingleBrowserPath** implements *Path*

An implementation of `Path` that is tailored to a singleton browser, thus allows some additional API's for actions (for those who favor object-oriented API style)

Fields

anchor

public static final *SingleBrowserPath* **anchor**

body

public static final *SingleBrowserPath* **body**

button

public static final *SingleBrowserPath* **button**

div

public static final *SingleBrowserPath* **div**

element

public static final *SingleBrowserPath* **element**

form

public static final *SingleBrowserPath* **form**

header

public static final *SingleBrowserPath* **header**

header1

public static final *SingleBrowserPath* **`header1`**

header2

public static final *SingleBrowserPath* **`header2`**

header3

public static final *SingleBrowserPath* **`header3`**

header4

public static final *SingleBrowserPath* **`header4`**

header5

public static final *SingleBrowserPath* **`header5`**

header6

public static final *SingleBrowserPath* **`header6`**

html

public static final *SingleBrowserPath* **`html`**

input

public static final *SingleBrowserPath* **`input`**

listItem

public static final *SingleBrowserPath* **`listItem`**

section

public static final *SingleBrowserPath* **`section`**

span

public static final *SingleBrowserPath* **`span`**

svg

public static final *SingleBrowserPath* **svg**

unorderedList

public static final *SingleBrowserPath* **unorderedList**

Constructors

SingleBrowserPath

public **SingleBrowserPath** (*BasicPath path*)

Methods

after

public *Path* **after** (*Path another*)

afterSibling

public *Path* **afterSibling** (*Path another*)

ancestorOf

public *Path* **ancestorOf** (*Path another*)

and

public *Path* **and** (*ElementProperty... prop*)

before

public *Path* **before** (*Path another*)

beforeSibling

public *Path* **beforeSibling** (*Path another*)

childOf

public *Path* **childOf** (*Path another*)

click

public void **click**()
click at the location of this element

containing

public *Path* **containing**(*Path another*)

contains

public *Path* **contains**(*Path another*)

descendantOf

public *Path* **descendantOf**(*Path another*)

describedBy

public *Path* **describedBy**(String *description*)

doubleClick

public void **doubleClick**()
doubleclick at the location of this element

dragAndDrop

public *Operations.DragAndDrop* **dragAndDrop**()
drag and drop this element, to another element or another location. Examples:

```
element.dragAndDrop().to(anotherElement);
   element.dragAndDrop().to(50, 50);
```

Returns DragAndDrop instance. See examples for usage.

find

public WebElement **find**()
Find the (first) element in the browser for this path

Returns the WebElement

findAll

public List<WebElement> **findAll**()
Find all elements in the browser with this path

Returns a list of all WebElements with this path

getAlternateXPath

public Optional<String> **getAlternateXPath**()

getDescribedBy

public Optional<String> **getDescribedBy**()

getElementProperties

public List<*ElementProperty*> **getElementProperties**()

getUnderlyingSource

public Optional<WebElement> **getUnderlyingSource**()

getXPath

public Optional<String> **getXPath**()

getXpathExplanation

public Optional<String> **getXpathExplanation**()

hover

public void **hover**()
hover over the element with this path in the browser

immediatelyAfterSibling

public *Path* **immediatelyAfterSibling**(*Path another*)

immediatelyBeforeSibling

public *Path* **immediatelyBeforeSibling**(*Path another*)

inside

public *Path* **inside** (*Path another*)

insideTopLevel

public *Path* **insideTopLevel** ()

or

public *Path* **or** (*Path another*)

parentOf

public *Path* **parentOf** (*Path another*)

rightClick

public void **rightClick** ()
right click at the location of this element

scrollTo

public WebElement **scrollTo** ()
scroll the browser until this element is visible

Returns the WebElement that was scrolled to

sendKeys

public void **sendKeys** (CharSequence... *charsToSend*)
send keys to element

Parameters

- **charsToSend** – the keys to send. Examples:

```
input.sendKeys("abc"); input.sendKeys("a", "bc");
```

Throws

- ***Operations.OperationFailedException*** –
  - operation failed. Includes root cause.

that

public *Path* **that** (*ElementProperty*... *prop*)

toString

public String **toString**()

withClass

public *Path* **withClass**(String *cssClass*)

withClasses

public *Path* **withClasses**(String... *cssClasses*)

withGlobalIndex

public *Path* **withGlobalIndex**(Integer *index*)

withText

public *Path* **withText**(String *txt*)

withTextContaining

public *Path* **withTextContaining**(String *txt*)

SingltonBrowserImage

public class **SingltonBrowserImage**

Capturing and asserting the image (screenshot) of a Path element. It supports both canvas and a standard element image. It supports both accurate and fuzzy assertion. It provides utility functions to display an element in a separate window.

Constructors

SingltonBrowserImage

public **SingltonBrowserImage**(*Path el*)

Parameters

- **el** – The element the represents the image we are interested in

Methods

assertCanvasImageIsEqualToExpected

public void **assertCanvasImageIsEqualToExpected**(InputStream *expectedImageInput*)

Verify that the canvas image is pixel-perfect

Parameters

- **expectedImageInput** –
  - reference png image

Throws

- **IOException** –
  - file could not be read

assertImageIsEqualToExpected

public void **assertImageIsEqualToExpected**(InputStream *expectedImageInput*)

Verify that the element's image is pixel-perfect

Parameters

- **expectedImageInput** –
  - reference png image

Throws

- **IOException** –
  - file could not be read

assertImageIsEqualToExpectedWithShiftAndCrop

public void **assertImageIsEqualToExpectedWithShiftAndCrop**(InputStream *expectedImageInput*, int *maxShift*)

Verify that the element's image is pixel-perfect, but allowing one to be a cropped/shifted version of the other.

Parameters

- **expectedImageInput** –
  - reference png image
- **maxShift** – maximum pixels the images are shifted/cropped compared to each other

Throws

- **IOException** –
  - file could not be read

assertImageIsSimilarToExpected

public void **assertImageIsSimilarToExpected**(InputStream *expectedImageInput*, int *maxBadPixelsRatio*)

Verify the picture is "similar" to the reference image. Ignores minor differences between the pixels. Does not identify offsets and rotation. It uses a VERY simplistic approach (no wavelets or any other transform).

Parameters

- **expectedImageInput** –
  - reference png image

- **maxBadPixelsRatio** –
  - a positive number. For example: If it's 100, then 1% of the pixels can have major differences compared to the reference.

Throws

- **IOException** –
  - file could not be read

assertImgSourceIsEqualToExpected

public void **assertImgSourceIsEqualToExpected** (InputStream *expectedImageInput*)

Verify that the HTML img source is pixel-perfect

Parameters

- **expectedImageInput** –
  - reference png image

Throws

- **IOException** –
  - file could not be read

captureCanvasToFile

public void **captureCanvasToFile** (File *outputFile*)

Capture the image of an HTML5 canvas as a png, and save it to the given file. If the element given is not a canvas, this will fail. Note that it is more optimized - it downloads only the section of the canvas as an image.

Parameters

- **outputFile** –
  - output file

captureImgSourceToFile

public void **captureImgSourceToFile** (File *outputFile*)

Capture the source of an img element as a png, and save it to the given file

Parameters

- **outputFile** –
  - output file

captureToFile

public void **captureToFile** (File *outputFile*)

Capture the image of an element as a png, and save it to the given file

Parameters

- **outputFile** –

- output file

getErrorImage

public Optional<BufferedImage> **getErrorImage** (InputStream *expectedImageInput*)

compare captured image to a reference image and return an image that highlights the differences. Both images are expected to have the same dimensions, otherwise it throws in AssertionError.

Parameters

- **expectedImageInput** –
  - reference png image

Throws

- **IOException** –
  - file could not be read
- **AssertionError** –
  - images are not the same size

Returns an image that highlights the different pixels. If the images are equal, returns an empty optional.

show

public void **show** ()

Display the element image in a separate window. This is useful for troubleshooting/development. Note that this will not work well if you do it inside a debugger evaluation.

showCanvas

public void **showCanvas** ()

Similar to show(), but optimized for an HTML5 canvas element

8.9.5 com.github.loyada.jdollarx.singlebrowser.custommatchers

Custom Hamcrest matchers for assertions in tests - for a singleton instance of browser

AgGridMatchers

public class **AgGridMatchers**

Hamcrest matchers for an AgGrid

Methods

isPresent

public static Matcher<*AgGrid*> **isPresent**()

Verify that the grid, as defined, is present in the browser. In case of an assertion error, gives a useful error message. The assertion can be strict, in which case only the defined rows are expected to exist.

Returns a Hamcrest matcher

CustomMatchers

public final class **CustomMatchers**

A collection of Hamcrest custom matchers, that are optimized to be as atomic as possible when interacting with the browser or a W3C document, and return useful error messages in case of a failure. This is a simplified API, relevant when there is a singleton browser.

Methods

hasText

public static Matcher<*Path*> **hasText**(String *text*)

Successful if element has the text equal to the given parameter in the browser/document. Note that internally it creates a new path that includes the "hasText" constraint, and then searches for it, so it is atomic. Example use:

```
assertThat( path, hasText("John"));
```

Parameters

- **text** – the text to equal to (case insensitive)

Returns a custom Hamcrest matcher

isAbsent

public static Matcher<*Path*> **isAbsent**()

Successful if the browser has no elements that correspond to the given path. The implementation of this is optimized. This is much better than doing not(isPresent()), because in case of success (i.e. the element is not there), it will return immidiately, while the isPresent() will block until timeout is reached. For example: `assertThat( path, isAbsent());`

Returns a matcher that is successful if an element does not appear in the browser.

isDisplayed

public static Matcher<*Path*> **isDisplayed**()

Successful if given element is present and displayed in the browser. Relies on WebElement.isDisplayed(), thus non-atomic. For example: `assertThat( path, isDisplayed());`

Returns a matcher that checks if an element is displayed in the browser

isEnabled

public static Matcher<*Path*> **isEnabled**()

Successful if given element is present and enabled in the browser. Relies on WebElement.isEnabled(), thus non-atomic. For example: `assertThat( path, isEnabled());`

Returns a matcher that checks if an element is enabled in the browser

isNotDisplayed

public static Matcher<*Path*> **isNotDisplayed**()

Successful if given element is either not present, or present and not displayed in the browser. Relies on WebElement.isDisplayed(), thus non-atomic. For example: `assertThat( path, isNotDisplayed());`

Returns a matcher that checks if an element is displayed in the browser

isNotSelected

public static Matcher<*Path*> **isNotSelected**()

Successful if given element is present and is not selected in the browser. Relies on WebElement.isSelected(), thus non-atomic. For example: `assertThat( path, isSelected());`

Returns a matcher that checks if an element is selected in the browser

isPresent

public static *IsPresentNTimes* **isPresent** (int *nTimes*)

Successful if the the element appears the expected number of times in the browser. This matcher is optimized. Example use for browser interaction:

```
assertThat( path, ispresent(5).timesOrMore());
assertThat( path, ispresent(5).times());
assertThat( path, ispresent(5).timesOrLess());
```

Parameters

- **nTimes** –
  - the reference number of times to be matched against. See examples.

Returns a matcher that matches the number of times an element is present. See examples in the description.

isPresent

public static Matcher<*Path*> **isPresent**()

Successful if the the element is present in the browser. Example: `assertThat( path, ispresent());`

Returns a matcher that checks if an element is present in the browser

isSelected

public static Matcher<*Path*> **isSelected**()

Successful if given element is present and selected in the browser. Relies on WebElement.isSelected(), thus non-atomic. For example: `assertThat( path, isSelected());`

Returns a matcher that checks if an element is selected in the browser

IsPresentNTimes

public class **IsPresentNTimes**

Internal implementation - not to be instantiated directly. This matcher is optimized for the success use-case. In that case it match for a single element with exact number of elements wanted. In case of failure, it will make another call to get the actual number of elements on the page, in order to provide a detailed error message. So the trade off is: In case of success it's faster, In case of failure it's slower. It makes sense since most of the time we expect success.

Constructors

IsPresentNTimes

public **IsPresentNTimes**(int *nTimes*)

Methods

times

public Matcher<*Path*> **times**()

timesOrLess

public Matcher<*Path*> **timesOrLess**()

timesOrMore

public Matcher<*Path*> **timesOrMore**()

8.9.6 com.github.loyada.jdollarx.singlebrowser.highlevelapi

Package for high level interactions. The API is less flexible than the standard API and sometimes not optimal, but it is simplified.

CheckBoxes

public final class **CheckBoxes**

High-level wrapper to define a checkbox input. High level API's are not optimized. A definition of an element may interact with the browser to understand the structure of the DOM.

Methods

checkBoxWithLabel

public static *CheckBox* **checkBoxWithLabel** (String *labelText*)
input of type "checkbox" with a label

Parameters

- **labelText** –
  - the text in the label

Returns a high level instance of CheckBox

checkBoxWithProperties

public static *CheckBox* **checkBoxWithProperties** (*ElementProperty*... *props*)
input of type "checkbox" with the given properties

Parameters

- **props** – properties of the checkbox

Returns a high level instance of CheckBox

Inputs

public final class **Inputs**
High-level API to define and interact with various input elements. High level API's are not optimized. A definition of an element may interact with the browser to understand the structure of the DOM.

Methods

changeInputValue

public static void **changeInputValue** (*Path field*, String *text*)
Change input value: clear it and then enter another text in it

Parameters

- **field** – Path to the input field
- **text** – the text to enter in the input field

Throws

- ***Operations.OperationFailedException*** – failed to perform the operation

changeInputValueWithEnter

public static void **changeInputValueWithEnter** (*Path field*, String *text*)
Similar to changeInputValue, but adds an ENTER after setting the value of the input

Parameters

- **field** – Path to the input field
- **text** – the text to enter in the input field

Throws

- *Operations.OperationFailedException* – failed to perform the operation

clearInput

public static void **clearInput** (*Path field*)

Clear operation on an input element

Parameters

- **field** – the input element

inputFollowedByUnlabeledText

public static *Path* **inputFollowedByUnlabeledText** (String *text*)

Input followed by text that does not have its on label element.

Parameters

- **text** – the text following the input

Returns a Path to the input element

inputForLabel

public static *Path* **inputForLabel** (String *labelText*)

A lazy way to find an input based on the label. Mote that unlike It looks for a label element that has an ID. If it finds one, it returns a Path to an input with that ID. Otherwise it returns a Path to an input inside the label element.

Parameters

- **labelText** – the label to look for

Returns a Path to the input, on a best effort basis

selectInFieldWithLabel

public static void **selectInFieldWithLabel** (String *labelText*, String *option*)

Perform a selection of an option in a select element. It expects to find the label element with the given text before the select element

Parameters

- **labelText** – The text of the select label
- **option** – The option text

RadioInputs

public final class **`RadioInputs`**

High-level API to define a with high level instance of radio input High level API's are not optimized. A definition of an element may interact with the browser to understand the structure of the DOM.

Methods

withLabeledText

public static *RadioInput* **`withLabeledText`** (String *labelText*)

create and return a RadioInput, that has a "label" element with the given text. Note that this is not a pure declaration and it looks for the label in the browser.

Parameters

- **`labelText`** –
  - the text in the label

Returns

- a RadioInput instance

withProperties

public static *RadioInput* **`withProperties`** (*ElementProperty*... *props*)

return a radio button with some custom properties

Parameters

- **`props`** –
  - some custom properties for the radio button

Returns a radio input with some custom properties

withTextUnknownDOM

public static *RadioInput* **`withTextUnknownDOM`** (String *text*, int *originalImplicitWait*, TimeUnit *timeUnit*)

In case the organization of the DOM is unclear, it will try both labeled input and unlabeled input. When doing so, it will change the implicit wait temporarily to a small value, and then revert the implicit timeout to the values provided. Use this only if you are not sure about the structure of the DOM.

Parameters

- **`text`** –
  - the text following the radio button
- **`originalImplicitWait`** –
  - the current implicit wait
- **`timeUnit`** –
  - the current time unit of the implicit wait

Returns a RadioInput instance

withUnlabeledText

public static *RadioInput* **withUnlabeledText** (String *text*)

create and return a RadioInput, that has straight text after it (not in a "label" element). i.e.:

```
Male
Female
```

Parameters

- **text** –
  - the text following the radio button

Returns

- a RadioInput instance

8.9.7 com.github.loyada.jdollarx.singlebrowser.sizing

Custom manipulations and evaluations of dimensions of elements or browser window size - for a singleton instance of browser

ElementResizer

public class **ElementResizer** implements AutoCloseable

An AutoCloseable of a resizer for a Path element. When closing, it reverts the the original state

Constructors

ElementResizer

public **ElementResizer** (*Path path*, int *expectedWidth*, int *expectedHeight*)

Resize an element in the browser

Parameters

- **path** – The element to resize
- **expectedWidth** – expected width
- **expectedHeight** – expected height

Methods

close

public void **close** ()

Revert state

getTotalHeight

public Long **getTotalHeight**()
get total scrollable height of the element

Returns height

getTotalWidth

public Long **getTotalWidth**()
get total scrollable width of the element

Returns width

getVisibleHeight

public Long **getVisibleHeight**()
get visible height of the element

Returns height

getVisibleWidth

public Long **getVisibleWidth**()
get visible width of the element

Returns width

getVisibleWidth

public static Long **getVisibleWidth**(*Path el*)
get visible width of the element

Parameters

- **el** –
  - the path to examine

Returns width

SizingUtils

class **SizingUtils**

Fields

HEIGHT

static final String **HEIGHT**

WIDTH

static final String **WIDTH**

Methods

getScrollableDimensions

static Map<String, Long> **getScrollableDimensions** (*Path path*)

getStylingDimensions

static Map<String, String> **getStylingDimensions** (*Path path*)

getVisibleDimensions

static Map<String, Long> **getVisibleDimensions** (*Path path*)

setDimensions

static void **setDimensions** (*Path path*, int *width*, int *height*)

setDimensions

static void **setDimensions** (*Path path*, String *width*, String *height*)

WindowResizer

public class **WindowResizer** implements AutoCloseable

An AutoCloseable resizer for the browser. When closing, it reverts the the original state

Constructors

WindowResizer

public **WindowResizer** (int *expectedWidth*, int *expectedHeight*)

Resize a browser to the requested dimensions. First it changes the window size, and then it updates the size of the html inside it.

Parameters

- **expectedWidth** – expected width
- **expectedHeight** – expected height

Methods

close

public void **close**()
Revert state

getTotalHeight

public Long **getTotalHeight**()
get total scrollable height of the browser

Returns height

getTotalWidth

public Long **getTotalWidth**()
get total scrollable width of the browser

Returns width

getVisibleHeight

public Long **getVisibleHeight**()
get visible height of the browser

Returns height

getVisibleWidth

public Long **getVisibleWidth**()
get visible width of the browser

Returns width

8.9.8 com.github.loyada.jdollarx.utils

Utilities for troubleshooting

DebugUtil

public final class **DebugUtil**
Several utilities that are useful for troubleshooting of existing browser pages. The utilities assume the use of *com.github.loyada.jdollarx.singlebrowser.InBrowserSinglton*.

Methods

getDOM

public static Optional<Element> **getDOM** (*Path el*)

Same as `getDOMOfAll(Path)`, but returns an optional of the first match.

Parameters

- **el** – the path we are looking for

Returns the first Element that matches the path in the current page

getDOMOfAll

public static List<Element> **getDOMOfAll** (*Path el*)

Get all matches of the path as a list of `Element`. JSoup `Element` are a nice, readable way to examine DOM objects. This is useful for troubleshooting. This method relies on `com.github.loyada.jdollarx.singlebrowser.InBrowserSinglton`, and on the library JSoup.

Parameters

- **el** – the path we are looking for

Returns all the elements that match it in the current page

getPageAsW3CDoc

public static org.w3c.dom.Document **getPageAsW3CDoc** ()

Download the current page and convert it to a W3C Document, which can be inspected using the `com.github.loyada.jdollarx.PathParsers` methods

Returns a W3C document

highlight

public static void **highlight** (*Path el*)

Highlight the first element that match the path in the browser, for 2 seconds.

Parameters

- **el** –
  - the definition of the element to highlight

highlightAll

public static void **highlightAll** (*Path el*)

Highlight all the elements that match the path in the browser, for 2 seconds.

Parameters

- **el** –
  - the definition of the elements to highlight

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X