TagUI

Release 6.0.0

Ken Soh, Siow Yi Sheng

Mar 10, 2020

CONTENTS:

1 Installation 3

2 Main concepts 5

3 Advanced concepts 13

4 Reference 17

5 Tools 31

6 Frequently Asked Questions 33

An open-source, cross-platform, command-line RPA tool that allows you to automate your desktop web, mouse and keyboard actions easily.

Here's what a simple TagUI flow looks like:

```
https://www.typeform.com

click login
type username as user@gmail.com
type password as 12345678
click btnlogin

download https://admin.typeform.com/xxx to report.csv
```

CHAPTER ONE

INSTALLATION

1.1 Windows

1. Download TagUI v6.0.0 for Windows.
2. Unzip the contents to `C:\`.
3. Install OpenJDK for Windows.
4. Install Chrome.
5. Open *Command Prompt*.
6. Copy, paste and run these commands:

```
setx path "%path%;c:\tagui\src"

tagui c:\tagui\src\samples\1_yahoo.tag
```

You have run your first TagUI flow!

Having problems?

1.2 macOS/Linux

1. Download TagUI v6.0.0 (macOS, Linux).
2. Unzip the contents to your desktop on macOS, or `/home/your_userid` on Linux.
3. Install OpenJDK (macOS, Linux).
4. Install Chrome.
5. Open *Terminal*.
6. Copy, paste and run these commands, replacing `your_tagui_path` as needed:

```
ln -sf /your_tagui_path/tagui/src/tagui /usr/local/bin/tagui

tagui your_tagui_path/tagui/src/samples/1_yahoo.tag
```

You have run your first TagUI flow!

Having problems?

CHAPTER

TWO

MAIN CONCEPTS

2.1 Flows

TagUI automates your actions by running *flows*, which are just text files with the `.tag` file extension.

You can run a flow in the *Command Prompt/Terminal* like this:

```
tagui my_flow.tag
```

TagUI looks for `my_flow.tag` in your current working directory. You can also provide the full path to your flow:

```
tagui c:\tagui\samples\1_yahoo.tag
```

You can also *run flows on a fixed schedule*.

2.1.1 Run by double-click

You can create a shortcut file with:

```
tagui my_flow.tag deploy
```

This creates a shortcut (my_flow.cmd) to run your flow just by double clicking the shortcut. The shortcut is in the same folder as your flow, but you can move it to your desktop or anywhere else for easy access.

If you want to create the shortcut file with options like `headless`, you can just add them in the same line like this:

```
tagui my_flow.tag headless deploy
```

Note: If you move your flow file to another folder, you will need to create a new shortcut file.

2.1.2 Run from a URL

You can also run a flow directly from a URL:

```
tagui https://raw.githubusercontent.com/kelaberetiv/TagUI/master/src/samples/1_yahoo.
↪tag
```

2.1.3 Hide the browser

You can run web flows without showing the web browser by running TagUI with the `headless` option.

```
tagui my_flow.tag headless
```

This allows you to continue using your desktop normally while the flow is running, but it will not work if your flow uses visual automation, because visual automation reads and clicks what is on your screen.

2.2 Steps

Flows are made out of *steps*. Below are some common steps. You can see all the steps in the *step reference*.

2.2.1 click

One of the most common steps is click. You can use it to click on a web element:

```
click Getting started
```

This command tells TagUI to try to click on any element which has "Getting started" as it's "id", "name", "class" or "title" attributes (*How to find an element's attributes*), or as a last resort, has "Getting started" in its text.

This method usually works for targeting what you want, but you can be more explicit by providing an XPath. XPath is a powerful way to choose which web element you want to target. Use it like this:

```
click //a[@class="icon icon-home"]
```

You can also click on a certain point on your screen:

```
click (500,300)
```

Here, 500 and 300 are x-y coordinates. This command clicks on a point which is 500 pixels from the left of your screen and 300 pixels from the top of your screen. A good way to discover which coordinates to input is to use the `mouse_xy()` *helper function* in live mode.

Lastly, you can use visual automation to click where it matches a previously saved image. This command looks for button.png in the same folder as your flow, then looks for a similar image on your screen, and clicks it:

```
click button.png
```

It's often a good idea to keep your flows and images organised. You can create a folder (eg. named images) for your images and use the image like this instead:

```
click image/button.png
```

2.2.2 visit

You can visit a webpage simply by entering the url:

```
https://somewebsite.com
```

2.2.3 type

You can type into web inputs. This command finds the element "some-input" in the same way as for the **click** step and types "some-text" into it:

```
type some-input as some-text
```

You can use [clear] to clear the input and [enter] to hit the Enter key:

```
type some-input as [clear]some-text[enter]
```

You can also use an image as the target, just like with the **click** step:

```
type some-input.png as some-text
```

2.2.4 assign

You can assign values into variables. This makes them easier to reference and work with.

This example uses the `count()` *helper function*, counts the number of elements found with id/name/text with 'row' in them and assigns it to a variable `row_count` for later use:

```
row_count = count('row')
```

2.2.5 read

The **read** step allows you to save text from web elements or from the screen into a variable.

This command finds the element "some-element" and saves its value into a variable called "some-variable":

```
read some-element to some-variable
```

read can also use visual automation and OCR to read text from a region of your screen. The output from this may not be completely accurate as it relies on OCR.

This command reads all the text in the rectangle formed between the points (300,400) and (500,550):

```
read (300,400)-(500,550) to some-variable
```

You can also use XPath to read some attribute values from web elements. This command reads the id attribute from the element:

```
read //some-element/@some-attribute to some-variable
```

2.3 Identifiers

You have probably noticed that different steps have different ways that they target elements, called **identifiers**. Let's look at the different types of identifiers.

Note: The DOM and XPath identifiers only work for Chrome. To automate other browsers, use the Point/Region and Image identifiers.

2.3.1 DOM

```
click Getting started
```

This matches an element in the DOM (Document Object Model) of the web page, matching either the *id, name, class attributes* or the text of the element itself.

2.3.2 XPath

```
click //body/div[1]/nav/div/div[1]/a
```

This matches the *XPath* of an element in the web page. It is a more explicit and powerful way of targeting web elements.

Note: You can use CSS selectors too in place of XPath, but XPath is preferred.

2.3.3 Point

```
click (200,500)
```

This matches the point on the screen 200 pixels from the left of the screen and 500 pixels from the top of the screen. This uses *visual automation*.

2.3.4 Region

```
read (300,400)-(500,550) to some-variable
```

This matches the rectangle formed between the two points (300,400) and (500,550). See *Point*. This uses *visual automation*.

2.3.5 Image

```
click button.png
```

This matches any area on the screen that looks similar to an image file `button.png` (in the folder of the flow). You first need to take a screenshot of `button.png`. This uses *visual automation*.

```
click image/button.png
```

This allows you to look for `button.png` within the `image` folder.

2.4 Live mode

We recommend using live mode when you want to write your own flows or try out some step. In *Command Prompt/Terminal*:

```
tagui live
```

This starts up a live session, where you can run steps one line at a time and immediately see the output.

2.5 If statements

You may want your flow to do something different depending on some factors. You can use an if statement to do this.

For example, if the URL contains the word "success", then we want to click some buttons:

```
if url() contains "success"
{
  click button1.png
  click button2.png
}
```

`url()` is a *helper function* that gets the url of the current webpage. Note the use of `{` and `}`. The steps within these curly braces will only be run if the condition is met, ie. the url contains the word "success".

Another common case is to check if some element exists. Here, we say that "if some-element doesn't appear after waiting for the timeout, then visit this webpage".

```
if !exist('some-element')
{
  https://tagui.readthedocs.io/
}
```

The ! negates the condition and comes from JavaScript, which TagUI code eventually translates to.

In this next example, we check if a variable row_count, which we assigned a value earlier, is equal to 5:

```
if row_count equals 5
{
  some steps
}
```

Here's how we check if it is more than or less than 5:

```
if row_count is more than 5
{
  some steps
}
```

```
if row_count is less than 5
{
  some steps
}
```

2.6 Loops

You can use loops to do the same thing many times within the same flow. In order to run one flow many times with different variables, the standard way is to use *datatables*.

In this example, we repeat the steps within the curly braces `{` and `}` a total of 20 times:

```
for n from 1 to 20
{
  some steps
}
```

2.7 Helper functions

Helper functions are useful JavaScript functions which can get values to use in your steps.

Each helper function is followed by brackets `()`. Some helper functions take inputs within these brackets.

You can see all the helper functions in the *reference*.

2.7.1 csv_row()

Turns some variables into csv text for writing to a csv file. It takes variables as its input, surrounded by square brackets `[]` (which is actually a JavaScript array).

```
read name_element to name
read price_element to price
read details_element to details
write csv_row([name, price, details]) to product_list.csv
```

2.7.2 clipboard()

Gets text from the clipboard:

```
dclick pdf_document.png
wait 3 seconds
keyboard [ctrl]a
keyboard [ctrl]c
text_contents = clipboard()
```

You can also give it an input, which puts the input *onto* the clipboard instead. This can be useful for pasting large amounts of text directly, which is faster than using the **type** step:

```
long_text = "This is a very long text which takes a long time to type"
clipboard(long_text)
click text_input
keyboard [ctrl]v[enter]
```

2.7.3 mouse_x(), mouse_y()

Gets the mouse's x or y coordinates.

This is useful for modifying x or y coordinates with numbers for using in steps like `read` and `click`.

The example below clicks 200 pixels to the right of `element.png`:

```
hover element.png
x = mouse_x() + 200
y = mouse_y()
click (`x`,`y`)
```

2.7.4 mouse_xy()

In live mode, you can use find out the coordinates of your mouse using `echo mouse_xy()` so that you can use the coordinates in your flows:

```
echo mouse_xy()
```

CHAPTER

THREE

ADVANCED CONCEPTS

3.1 Object repositories

Object repositories are optional *csv files* which can store variables for use in flows. They help to separate your flows from your personal data (like login information for web flows), and allow you to share common information between multiple flows for easy updating.

Each flow has a **local object repository** and all flows share the **global object repository**. The local object repository is the `tagui_local.csv` in the same folder as the flow. The global object repository is the `tagui_global.csv` in the `tagui/src/` folder.

An object repository could look like this:

object	definition
email	user-email-textbox
create account	btn btn–green btn-xl signup-btn

Within the flow, TagUI can use the objects `email`, `create account` as variables and they will be replaced directly by the definitions before it is run. Local definitions take precedence over global definitions.

If `user-email-textbox` was the identifier for some web text input, then you could use the following in your flow:

```
type `email` as my_email@email.com
```

3.2 Datatables

Datatables are *csv files* which can be used to run your flows multiple times with different inputs.

A datatable (`trade_data.csv`) could look like this:

#	trade	username	password	pair	size	direction
1	Trade USDSGD	test_account	12345678	USDSGD	10000	BUY
2	Trade USDSGD	test_account	12345678	USDJPY	1000	SELL
3	Trade EURUSD	test_account	12345678	EURUSD	100000	BUY

To use it, you run your flow with `tagui my_flow.tag trade_data.csv`. TagUI will run `my_flow.tag` once for each row in the datatable (except the header). Within the flow, TagUI can use the variables `trade`, `username`, `password`, etc as if they were in the *local object repository* and the values will be from that run's row.

You can run the flow with the `speed` option like this: `tagui my_flow.tag trade_data.csv speed` to remove the delay between runs.

3.3 Running other flows within a flow

A flow can run another flow, like this:

```
tagui login_crm.tag
```

Variables in the parent flow are accessible in the child flow.

3.4 Visual automation tricks

If you make the background of a UI element in a `.png` file 100% transparent using an image editor, TagUI will be able to target the element regardless of its background.

Conversely, you can also remove the foreground content near some anchor element like a frame, to allow you to OCR varying content in the empty area using the **read** step.

3.5 Writing Python within flows

You can write Python code in TagUI flows. Python needs to be installed separately.

The `py` step can be used to run commands in Python (TagUI will call `python` on the command line). You can pass string values back to TagUI with *print()*. The `stdout` will be stored in the `py_result` variable in TagUI.

```
py a=1
py b=2
py c=a+b
py print(c)
echo py_result
```

You can also use `py begin` and `py finish` before and after a Python code block:

```
py begin
a=1
b=2
c=a+b
print(c)
py finish
echo py_result
```

You can pass a variable to Python like this:

```
phone = 1234567
py 'phone = ' + phone
py print(phone)
echo py_result
```

3.6 Saving flow run results

You can save an html log of the run and the flow run results to `tagui/src/tagui_report.csv` with the `report` option.

```
tagui my_flow.tag report
```

The CSV file will show one line for each run, when it started, how long it took to complete, any error message during run, and the link to the log file for that run.

3.7 Create log files for debugging

To do advanced debugging, you can create log files when running flows by creating an empty `tagui_logging` file in `tagui/src/`.

- `my_flow.log` stores output of the execution.
- `my_flow.js` is the generated JavaScript file that was run.
- `my_flow.raw` is the expanded flow after parsing modules.

CHAPTER

FOUR

REFERENCE

Use this section to look up information on steps, helper functions and run options you can use. You can explore using the navigation headers.

4.1 Steps

The steps you can use in TagUI are listed here.

4.1.1 Mouse and Keyboard

click

Left clicks on the identifier.
Can use *DOM*, *XPath*, *Point*, *Image* identifiers.

```
click [DOM/XPath/Point/Image]
```

Examples

```
click Main concepts
click //nav/div/div[2]/ul/li[4]/ul/li[1]/a
click (500,200)
click button.png
```

rclick

Right clicks on the identifier.
Can use *DOM*, *XPath*, *Point*, *Image* identifiers.

```
rclick [DOM/XPath/Point/Image]
```

See *click* for examples.

dclick

Double left clicks on the identifier.
Can use *DOM*, *XPath*, *Point*, *Image* identifiers.

```
dclick [DOM/XPath/Point/Image]
```

See *click* for examples.

hover

Moves mouse cursor to the identifier.
Can use *DOM*, *XPath*, *Point*, *Image* identifiers.

```
hover [DOM/XPath/Point/Image]
```

See *click* for examples.

type

Types into a web input. You can use [clear] to clear the field and [enter] to hit the Enter key.
Can use *DOM*, *XPath*, *Point*, *Image* identifiers.

```
type [DOM/XPath/Point/Image] as [text to type]
```

Examples

```
type search-term as John Wick
type //input[@name="search"] as John Wick
type (500,200) as John Wick
type input_field.png as John Wick

type search-term as [clear]John Wick[enter]
type //input[@name="search"] as [clear]John Wick[enter]
type (500,200) as [clear]John Wick[enter]
type input_field.png as [clear]John Wick[enter]
```

keyboard

Enters keystrokes directly.

```
keyboard [keys]
```

You can use the following special keys:

- [clear]
- [shift] [ctrl] [alt] [cmd] [meta] [enter]
- [win] [space] [tab] [esc] [backspace] [delete]

- [up] [down] [left] [right] [pageup] [pagedown]
- [home] [end] [insert] [f1] .. [f15]
- [printscreen] [scrolllock] [pause] [capslock] [numlock]

Examples

```
keyboard [win]run[enter]
keyboard [printscreen]
keyboard [ctrl]c
keyboard [tab][tab][tab][enter]

keyboard [cmd][space]
keyboard safari[enter]
keyboard [cmd]c
```

mouse

Explicitly sends a mouse event at the current mouse position.
In most cases, you want you use *click* instead.

```
mouse down
mouse up
```

4.1.2 Web

visit

Visits the provided URL.

```
[URL]
```

Examples

```
https://google.com
```

select

Selects a dropdown option in a web input.
Can use *DOM*, *XPath* identifiers.

```
select [DOM/XPath of select input element] as [option value]
```

Examples

```
select variant as blue
```

table

Saves html table data to a csv file.
Uses *XPath* identifier only.

```
table [XPath] to [filename.csv]
```

Examples

```
table //table[1] to exchange-rates.csv
```

popup

Modifies the next steps to be run in a new tab.

```
popup [unique part of new tab's URL]
{
  [steps]
}
```

Examples

```
popup confirm
{
  click Confirm
}
```

frame

Modifies the next steps to use the DOM or XPath in a frame or subframe.

```
frame [frame name]
{
  [steps]
}

frame [frame name] | [subframe name]
{
  [steps]
}
```

Examples

```
frame navigation
{
  click Products
}

frame main | register
{
  click Register
}
```

download

Downloads a file at a URL and saves it.
Saves to the flow's folder by default, but you can also provide a full path to save to.

```
download [file url] to [filename]
```

Examples

```
download https://github.com/kelaberetiv/TagUI/releases/download/v5.11.0/TagUI_Windows.
↪zip to tagui.zip
```

upload

Uploads file to a website.
Can use *DOM*, *XPath* identifiers.

```
upload [DOM/XPath of upload input element] as [filename]
```

Examples

```
upload //input[@name="attach"] as report.csv
```

api

Call a web API and save the response to the variable `api_result`.

```
api https://some-api-url
```

Examples

```
api https://api.github.com/repos/kelaberetiv/TagUI/releases
js obj = JSON.parse(api_result)
js author = obj[0].author.login
```

4.1.3 Using Variables

read

Gets some text or value and stores it in a variable.
Can use *DOM*, *XPath*, *Region*, *Image* identifiers.

```
read [DOM/XPath/Region/Image] to [variable]
```

When you provide a Region or Image identifier, TagUI uses OCR (Optical Character Recognition) to read the characters from the screen.

Examples

```
read //p[@id="address"] to address

read //p[@id="address"]/@class to address-class

read (500,200)-(600,400) to id-number

read frame.png to email
```

assign

Saves text to a variable.

```
[variable] = [value]
```

When using text in the value, surround the text in quotes, like “some text”.
This is actually treated by TagUI as JavaScript, so you can assign numbers to variables or use other JavaScript functions.
The variable name needs to be a single word and cannot start with a number.

Examples

```
count = 5
username = "johncleese"
fullname = firstname + lastname
```

4.1.4 File Saving/Loading

write

Saves a new line of text to an existing file.

```
write [text] to [filename]
write [`variable`] to [filename]
```

Examples

```
write firstname,lastname to names.csv
write `fullreport` to report.txt
```

dump

Saves text to a new file.

```
dump [text] to [filename]
dump [`variable`] to [filename]
```

See *dump* for examples.

load

Loads file content to a variable.

```
load [filename] to [variable]
```

Examples

```
load report.txt to report
```

snap

Saves a screenshot of the whole page, an element or a region.
Can use *DOM*, *XPath*, *Region*, *Image* identifiers.

```
snap [DOM/XPath/Region/Image/page] to [filename]
```

If you use `page` as the identifier, it takes a screenshot of the whole webpage.

Examples

```
snap logo to logo.png

snap page to webpage.png
```

4.1.5 Showing output

echo

Shows some output on the command line.

```
echo [text]
echo [`variable`]
```

Examples

```
echo Flow has started
echo The user is `username`
```

show

Shows element text directly on the command line.
Can use *DOM*, *XPath* identifiers.

```
show [DOM/XPath]
```

Examples

```
show review-text
```

check

Shows some output on the command line based on a *condition*.

```
check [condition] | [text if true] | [text if false]
```

Examples

```
check header_home_text equals to "Home" | "header text is correct" | "header text is
↪wrong"
```

4.1.6 Custom code

js

Runs JavaScript code explicitly. TagUI has direct access to the JavaScript variables.

```
js [JavaScript statement]

js begin
[JavaScript statements]
js end
```

Examples

```
js obj = JSON.parse(api_result)
dump `obj` to result.json

js begin
obj = JSON.parse(api_result)
randomInteger = Math.floor(Math.random() * Math.floor(5)) + 1
js end
dump `obj` to result.json
```

py

Runs Python code and saves the stdout to the variable `py_result` as a string.

```
py [Python statement]

py begin
[Python statements]
py end
```

Examples

```
py result = 2 + 3
py print(result)
echo `py_result`

py begin
import random
random_integer = random.randint(1,6)
print(random_integer)
```

(continues on next page)

(continued from previous page)

```
py end
echo `py_result`
```

run

Runs a command in Command Prompt or Terminal and saves the stdout to the variable `run_result`.

```
run [shell command]
```

Examples

```
run mkdir new_directory
```

vision

Runs Sikuli code.

```
vision [Sikuli statement]

vision begin
[Sikuli statements]
vision end
```

Examples

```
vision click("button1.png")
```

dom

Runs code in the browser dom and saves the stdout to the variable `dom_result`.

```
dom [JavaScript statement to run in the DOM]

dom begin
[JavaScript statements to run in the DOM]
dom end
```

Examples

```
dom intro = document.getElementById("intro")
```

r

Runs R statements and saves the stdout to the variable `r_result`.

```
r [R statement]

r begin
[R statements]
r end
```

4.1.7 Miscellaneous

wait

Explicitly wait for some time.

```
wait [seconds to wait]
wait [seconds to wait] s
wait [seconds to wait] seconds
```

Examples

```
wait 5.5
wait 10 s
wait 20 seconds
```

timeout

Changes the auto-wait timeout when waiting for web elements to appear.

```
timeout [seconds to wait before timeout]
```

Examples

```
timeout 300
```

ask

Prompts user for input and saves the input as the variable `ask_result`.

```
ask [prompt]
```

Examples

```
ask What is the date of the receipt? (in DD-MM-YYYY)
type search as `ask_result`
```

live

Wait for user confirmation before continuing. The user must enter "done" before the flow continues.

```
live
```

tagui

Runs another TagUI flow. Checks the flow's folder.

```
tagui [flow file]
tagui [folder/flow file]
```

Examples

```
tagui update-forex.tag
tagui flows/update-forex.tag
```

comment

Adds a comment.

```
// [comment]
```

Examples

```
// updates the forex rates
```

4.2 Run options

You can use the below options when running `tagui`.

For example, the command below runs `my_flow.tag` without showing the web browser, while storing the flow run result in `tagui_report.csv`.

```
tagui my_flow.tag headless report
```

4.2.1 headless

Runs the flow without a visible browser (does not work for visual automation).

4.2.2 report

Tracks flow run result in `tagui/src/tagui_report.csv` and saves html logs of flows.

4.2.3 my_datatable.csv

Uses the specified csv file as the datatable. See *datatables*.

4.2.4 speed

Runs a datatable flow, skipping the default 3s delay and restarting of Chrome between datatable iterations.

See other deprecated options.

4.3 Helper functions

4.3.1 csv_row()

Formats an array for writing to csv file.

Examples

```
read name_element to name
read price_element to price
read details_element to details
write csv_row([name, price, details]) to product_list.csv
```

4.3.2 count()

Gets the number of elements matching the identifier specified. Note that the identifier needs to be in single quotes `''`.

Examples

```
rows = count('table-rows')
```

4.3.3 clipboard()

Puts text onto the clipboard, or gets the clipboard text (if no input is given).

Examples

```
clipboard('some text')
keyboard [ctrl]v

keyboard [ctrl]c
contents = clipboard()
```

4.3.4 url()

Gets the URL of the current web page.

Examples

```
if url() contains 'success'
{
  click button1
}
```

4.3.5 title()

Gets the title of the current web page.

Examples

```
if title() contains 'Confirmation'
{
  click button1
}
```

4.3.6 text()

Gets all text content of the current web page.

Examples

```
if text() contains 'success'
{
  click button1
}
```

4.3.7 timer()

Gets the time elapsed in seconds in between each running of this function.

Examples

```
timer()
click button1
click button2
click button3
echo timer()
```

4.3.8 exist()

Waits until the timeout for an element to exist and returns a JavaScript `true` or `false` depending on whether it exists or not.
Note that the identifier is surrounded by quotes.
Can use *DOM*, *XPath*, *Image* identifiers.

```
exist('[DOM/XPath/Image]')
```

Examples

```
if exist('//table')
{
  click button1
}
```

4.3.9 present()

Same as *exist()* except that it does not wait until the timeout and immediately returns `true` or `false`.
Note that the identifier is surrounded by quotes.
Can use *DOM*, *XPath*, *Image* identifiers.

Examples

```
read name_element to name
read price_element to price
read details_element to details
write csv_row([name, price, details]) to product_list.csv
```

4.3.10 mouse_xy()

Gets the x, y coordinates of the current mouse position.
Particularly useful in *live mode*.

Examples

```
echo mouse_xy()
```

4.3.11 mouse_x()

Gets the x coordinate of the current mouse position as a number, eg 200.

Examples

```
hover element.png
x = mouse_x() + 200
y = mouse_y()
click (`x`,`y`)
```

4.3.12 mouse_y()

Gets the y coordinate of the current mouse position as a number, eg 200.

Examples

```
hover element.png
x = mouse_x() + 200
y = mouse_y()
click (`x`,`y`)
```

CHAPTER

FIVE

TOOLS

These are separate apps which help you in writing TagUI flows. **TagUI For Python** is a Python API for TagUI.

5.1 TagUI Chrome Extension

The TagUI Chrome extension (Download) helps you write web flows.

It records steps such as page navigation, clicking of web elements and entering information. It then displays the steps for you to paste into your flow.

5.1.1 Usage

1. Go to the website URL you want to start the automation at.
2. Click the TagUI icon, then **Start**.
3. Carry out the steps you want to automate, or right click on elements to record other steps.
4. Click the TagUI icon, then **Stop**.
5. Click **Export** to view the generated TagUI steps.

The recording isn't foolproof (for example, the underlying recording engine cannot capture frames, popup windows or tab key input). It's meant to simplify flow creation with some edits, instead of typing everything manually.

See this video for an example of recording a sequence of steps, editing for adjustments and playing back the automation.

5.2 TagUI Writer, Screenshoter & Editor

TagUI Writer is a Windows app helps write TagUI flows. When pressing Ctrl + Left-click, a popup menu appears with the list of TagUI steps for you to paste into your text editor.

TagUI Screenshoter app helps in capturing screenshots for TagUI visual automation.

TagUI Editor allows you to edit and run TagUI scripts via AutoHotKey.

Download these here.. These third-party tools are created by Arnaud Degardin @adegard.

5.3 RPA For Python

RPA for Python is a Python package (`pip install rpa` to install) which allows you to use TagUI through a Python API. Check out the documentation. It was created and is maintained by TagUI's creator Ken Soh @kensoh.

CHAPTER

SIX

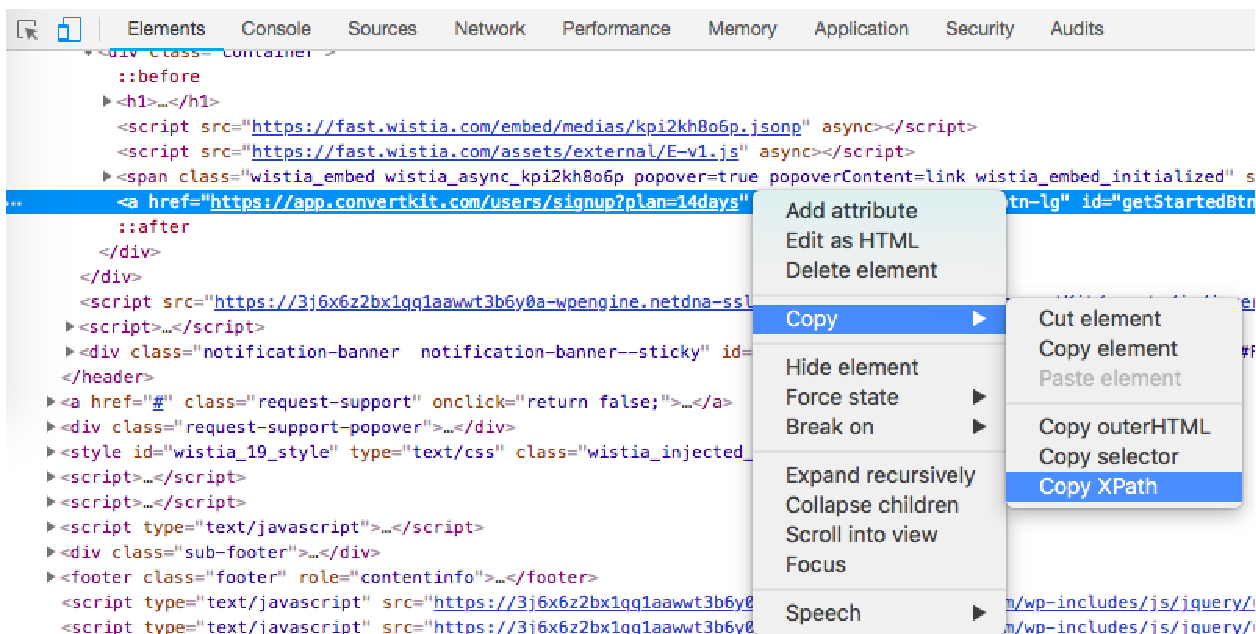
FREQUENTLY ASKED QUESTIONS

6.1 How is TagUI licensed?

TagUI is open-source software released under the Apache 2.0 license.

6.2 How do I find the XPath of a web element?

In Chrome, right-click on the element, click Inspect, right-click on the highlighted HTML, then:



For some web pages, the XPath of an element can change. To combat this, you can find a stable element in the web page and writing a custom XPath relative to that stable element.

XPath is very powerful and can allow you to select web elements in many ways. Learn more about XPath at w3schools.

6.3 How do I use the Command Prompt?

Hold the Windows key and press R. Then type `cmd` and press Enter to enter the Command Prompt.

From here, you can run a command by typing it and pressing Enter.

6.4 How do I use the Terminal?

Hold Command and press spacebar, then type `Terminal` and press Enter.

From here, you can run a command by typing it and pressing Enter.

6.5 How do I find the id, name, class or other attributes of a web element?

In Chrome, right-click on the element, click Inspect. There will be some highlighted HTML, like this:

```
<div class="chat-line__trash" data-behavior=
"delete_chat_line"></div>
▶<span class="chat-line__meta">…</span>
... ▶<div class="chat-line__body">…</div> == $0
</div>
</article>
▶<article class="chat-line chat-line--rich-text " data-
```

This highlighted element has a class attribute of "chat-line__body". It doesn't have any `id` or `name` attribute.

6.6 How do I use the cutting edge version of TagUI?

1. Download the latest stable version at *the installation page*.
2. Download the cutting edge version.
3. Unzip and overwrite the files in your `tagui/src/` folder.

6.7 What are csv files?

CSV files are files which stores data in a table form. They can be opened with Microsoft Excel and Google Sheets.

Each line is a row of values. The values are split into different columns by commas `,`, which is why CSV stands for Comma Separated Values.

6.8 Running flows on a fixed schedule

It is often useful to run flows automatically on a fixed schedule: monthly; weekly; daily or even every 5 minutes.

On Windows, use the Task Scheduler.

On macOS/Linux, use crontab.

6.9 Is TagUI safe to use?

As TagUI and the foundation it's built on is open-source software, it means users can read the source code of TagUI and all its dependencies to check if there is a security flaw or malicious code. This is an advantage compared to using commercial software that is closed-source, as users cannot see what is the code behind the software.

Following are links to the source code for TagUI and its open-source dependencies. You can dig through the source code for the other open-source dependencies below, or make the fair assumption that security issues would have been spotted by users and fixed, as these projects are mature and have large user bases.

- TagUI - https://github.com/kelaberetiv/TagUI
- SikuliX - https://github.com/RaiMan/SikuliX1
- CasperJS - https://github.com/casperjs/casperjs
- PhantomJS - https://github.com/ariya/phantomjs
- SlimerJS - https://github.com/laurentj/slimerjs
- Python - https://github.com/python/cpython
- R - https://github.com/wch/r-source
- PHP - https://github.com/php/php-src

6.10 Does TagUI track what I automate?

No. TagUI does not send outgoing web traffic or outgoing data, other than what the user is automating on, for example visiting a website.

6.11 Why doesn't my visual automation work?

On macOS, it may be due to how the image was captured.

On Linux, you may need to set up dependencies.

Get it here.