
TagUI

Release 6.46.0

Jun 20, 2022

Contents:

1	Installation	3
2	Main concepts	5
3	Advanced concepts	13
4	Reference	19
5	Tools	41
6	Frequently Asked Questions	43



A free, open-source, cross-platform RPA tool that helps you automate your desktop, web, mouse and keyboard actions easily. Here's what a simple TagUI flow looks like:

```
// below is an example to login to Xero accounting website
https://login.xero.com/identity/user/login

type email as user@gmail.com
type password as 12345678
click Log in
```

```
// besides web identifiers, images of UI elements can be used

type email_box.png as user@gmail.com
type password_box.png as 12345678
click login_button.png
```

```
// (x,y) coordinates of user-interface elements can also be used

type (720,400) as user@gmail.com
type (720,440) as 12345678
click (720,500)
```


1.1 Windows

You are recommended to download [TagUI Windows Installer](#), instead of the advanced steps below. If you use the installer, specify an installation folder which you have write permission, for example `C:\RPA` or a RPA folder on your Windows Desktop. After your installation, continue from step 6.

1. Read the above, or download zip file [TagUI v6.46 for Windows](#)
2. Unzip the contents to `C:\`
3. [Install OpenJDK for Windows](#)
4. [Install Chrome web browser](#)
5. Add `c:\tagui\src` to start of path
6. Open *Command Prompt*
7. Copy, paste and run this command:

```
tagui c:\tagui\flows\samples\1_google.tag
```

8. Finally, run this command to pull the [latest features and bug fixes](#)

```
tagui update
```

9. Having problems? [Click here](#). You have run your first TagUI flow!

To install TagUI Microsoft Word Plug-in (a full-featured app to make RPA very easy), download the [MS Word Plug-in installer](#), unzip the file and double-click Setup.exe. See this [video by RPA Learners](#) on installing and running your first MS Word RPA bot. Download [MS Excel Plug-in installer](#) to define RPA data parameters in Excel and run TagUI from Excel. See this [video by RPA Learners](#) on installing and creating visualisations with the Excel plug-in.

1.2 macOS / Linux

1. Download TagUI v6.46 for [macOS](#) or [Linux](#)
2. Unzip the contents to your desktop on macOS, or /home/your_userid on Linux
3. Install OpenJDK for [macOS](#) or [Linux](#)
4. [Install Chrome](#) web browser
5. Open *Terminal*
6. Copy, paste and run these commands, replacing your_tagui_path accordingly:

```
sudo ln -sf /your_tagui_path/tagui/src/tagui /usr/local/bin/tagui  
tagui /your_tagui_path/tagui/flows/samples/1_google.tag
```

7. Finally, run this command to pull the [latest features and bug fixes](#)

```
tagui update
```

8. Having problems? [Click here](#). You have run your first TagUI flow!

2.1 Flows

TagUI automates your actions by running *flows*, which are text files with `.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:

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

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

2.1.1 Run by double-click

You can create a shortcut file with `-deploy`:

```
tagui my_flow.tag -deploy
```

or using the shortcut `-d`:

```
tagui my_flow.tag -d
```

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.

If you want to create the shortcut with options like `-headless (-h)`, you can include them:

```
tagui my_flow.tag -h -d
```

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/flows/samples/1_  
↪google.tag
```

2.1.3 Hide the browser

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

```
tagui my_flow.tag -headless
```

or using the shortcut `-h`:

```
tagui my_flow.tag -h
```

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 it 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 the full list of steps in the [steps 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 its “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. For example:

```
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 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
```

2.2.5 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')
```

2.3 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/Edge. 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. Within a TagUI flow, you can also use `live` step to pause execution and enter live mode.

2.5 If conditions

You may want your flow to do something different depending on some factors. You can use an if condition 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 that the other lines are *indented*, ie. there are spaces (or tabs) in front of them. This means that they are in the *if block*. The steps in the *if block* will only be run if the condition is met, ie. the url contains the word “success”.

Note: Before v6, you need to use `{` and `}` to surround your *if block*.

From v6 onwards, the curly braces { } are optional.

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

```
if row_count equals to 5
    some steps
```

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

```
if row_count more than 5
    some steps
```

```
if row_count less than 5
    some steps
```

2.6 For 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 block for a total of 20 times:

```
for n from 1 to 20
    some step to take
    some other step
    some more step
```

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 the full list of helper functions in the *helper functions 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
keyboard [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()`
```


3.1 Saving run results of flow

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

```
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, the link to the log file for that run, and the user's workgroup\userid.

3.2 Handling exceptions and errors

There are 3 ways to handle exceptions in TagUI when things do not go as planned.

The first way is **local error handling**. This means using if conditions to check specifically for certain scenarios and handling the scenarios accordingly. For example, check if some UI element is missing, then do xyz steps. Using this way, a workflow can have multiple fine-grain exception handling.

The second way is **workflow error handling**. A workflow can be chained as follows to handle error or success accordingly. The workflow `error.tag` will run only if `flow.tag` errors out. The workflow `success.tag` will run only if `flow.tag` runs successfully. TagUI will automatically throw error when it detects an expected UI element missing (and autosave screenshot) or some other unknown errors.

Windows example from the command prompt:

```
call tagui flow.tag || tagui error.tag  
call tagui flow.tag && tagui success.tag
```

macOS / Linux example from the terminal:

```
tagui flow.tag || tagui error.tag  
tagui flow.tag && tagui success.tag
```

The third way is **global error handling**. Configuration can be done for TagUI such that after every run, special handling is done to send data or files generated from the report option to some target folder or API endpoint for error / success handling. For example, syncing all automation runs to central storage for auditing purpose. The special handling applies to all TagUI flows that are run.

3.3 Datatables for batch automation

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 for 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.

To know which iteration your flow is in you can use the `iteration` variable:

```
echo current iteration: `iteration`  
if iteration equals to 1  
  // go to login URL and do the login steps  
  www.xero.com  
  
// do rest of the steps for every iteration
```

3.4 Object repositories for reusability

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
```

3.5 Running flows within a flow

You can modularise your RPA workflows by breaking a large workflow file into many subflow files. For more complex RPA scenarios, you can even let a subflow run other subflows.

Some common reasons for doing that include the convenience of reusing the same subflow in other flows, doing something specific which is easier to organise by keeping the sequence of steps in a subflow, or storing your Python or JavaScript code and functions in separate subflows (using py begin and py finish code blocks for example).

A flow can run another flow, like this:

```
tagui login_crm.tag
```

Flows can also be stored in subfolders:

```
// Windows example
tagui CRM\login.tag

// Mac/Linux example
tagui CRM/login.tag
```

Variables in the parent flow are accessible in the child flow and vice versa:

```
// in this case, username and password variables are available in login.tag
username = 'jennifer'; password = '12345678';
tagui login.tag

// you can also define variables on separate lines instead of all in 1 line
username = 'jennifer'
password = '12345678'
tagui login.tag

// in login.tag you can define and return variables for its parent to use
echo `login_result`
```

You can even combine multiple sequences of steps into one subflow as follows. By designing a subflow this way, you can assign the variable `action = 'login'` in the parent flow to determine which sequence of steps gets executed when the subflow is called with `tagui` step:

```
// crm_steps.tag
if action equals to 'login'
  do some steps
  do some more steps

else if action equals to 'report'
  do some steps
  do some more steps

else if action equals to 'logout'
  do some steps
  do some more steps
```

(continues on next page)

(continued from previous page)

```
else
  echo ERROR - action undefined
```

3.6 Turbo mode to run 10X faster

To run TagUI with turbo option (use with caution):

```
tagui flow.tag -turbo

or

tagui flow.tag -t
```

Most websites and desktop apps are not designed for the super-human speed user. If your RPA runs at a speed beyond what those websites are designed and tested for, you are surely going to run into problems with some apps. Problems could be fields and data not filling up properly, not triggering expected validations, form submissions with missing data, account being blocked etc.

And the problems might happen randomly, including working on your PC but not working on another PC due to difference in CPU speed. Because of this, using turbo mode option is not recommended. You may save some cheap computer time, but if something is broken or does not work, you may end up spending expensive human time (your time) to troubleshoot or fix.

However, this is very useful for some users for some specific scenarios. For eg, data collection from apps, data entry in web applications that can handle super-human speed reliably, as part of a chatbot doing backend RPA for user, for fast and rapid prototyping, perhaps taking part in RPA competitions and hackathons etc. Thoroughly test for your use case before using!

3.7 Visual automation tricks

For many steps, you can end the step with `using ocr` or `using OCR` to tell TagUI to interact on some UI element on the screen using OCR (optical character recognition). See the examples below. Steps which this can be done: `click`, `relick`, `dclick`, `hover`, `type`, `select`, `read`, `snap`, `exist()`, `present()`.

```
click Submit using ocr

if exist('Special Offer using ocr')
  click Add To Cart using OCR

// various usage combinations for select step
select Dress Color using OCR as Dark Blue using OCR
select dress_color.png as Bright Pink using ocr
select Dress Color using OCR as dark_black.png
select dress_color.png as bright_white.png
```

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.

3.8 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_step('phone = ' + phone)
py print(phone)
echo `py_result`

name = 'Donald'
py_step('name = "' + name + '"')
py print(name)
echo `py_result`
```

To pass and return more complex data, for example multiple variables, you can use JavaScript and Python JSON libraries to send and receive back JSON strings. [See an example here](#) of passing 2 variables, doing some processing, and returning 2 variables.

3.9 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 step-by-step 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.

3.10 Running TagUI on the cloud

For cloud lovers, you can run TagUI on your web browser or phone using [free Google Cloud](#). You can run up to 5 sessions concurrently on different tabs of your browser.

For more control running on the cloud, you can run this [Docker image](#) (use edge tag for the latest version) or [Docker file](#) on your preferred cloud vendor.

Or run on [free Node-RED instance](#) on OpenFlow. The Docker image, Docker file and OpenFlow cloud are maintained and sponsored by [Allan Zimmermann](#).

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:

- [shift] [ctrl] [alt] [win] [cmd] [enter]
- [space] [tab] [esc] [backspace] [delete] [clear]
- [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 or text]
```

Examples

```
select variant as blue
```

table

Saves table data to a csv file, base on the table number on webpage or its *XPath* identifier.

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

Examples

```
table 1 to regional_exchange_rates.csv
table (//table)[2] to global_exchange_rates.csv
table //table[@name='report'] to report.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 to

Specifies a location to store file downloads. The default location is the folder of the TagUI flow.

```
download to [folder location]
```

upload

Uploads file to a website.

Only *DOM* identifier can be used.

```
upload [DOM of upload element] as [filename]
```

Examples

```
upload #element_id as report.csv
```

api

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

If the response is in JSON, `api_json` will automatically be created.

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

Examples

```
api https://api.github.com/repos/kelaberetiv/TagUI/releases
echo `api_result`
author = api_json[0].author.login
```

For an advanced example of using api step, setting POST/GET method, header and body, [see this example from aito.ai](#) - a web-based machine learning solution for no-coders and RPA developers. In the example, `api` step is used to make a machine-learning inference to generate the account code of an invoice item, based on its description and price. aito.ai's free tier comes with 2000 API calls/month and it works perfectly with TagUI.

4.1.3 Excel

Perform read, write, copy, delete actions on Excel files using standard Excel formula like this one `[workbook]sheet!range`. This feature works with both Windows and Mac Excel apps. [See this link](#) for notes of passed test cases and known limitations for this feature. To access a password-protected Excel file, use `excel_password = 'password'`.

variables

You can use variables in your Excel formula, for eg `range` or `sheet`. Various Excel file formats are supported, be sure to put the file's `.xlsx` extension as part of the formula so that TagUI can recognise that the instruction is an Excel step instead of some JavaScript code.

```
[`workbook`.xlsx]`sheet`!`range` = 123
data = [`workbook`.xlsx]`sheet`!`range`
```

visibility

By default, the Excel app will be opened and run in the background. If you want the automated actions on Excel to be in focus in foreground, you can set it with `excel_focus = true` in your workflow. Use `excel_focus = false` to set it off again in your workflow.

For some usage scenarios, you might not even want the Excel app to be visible in the background. In that case, you can set `excel_visible = false` in your workflow to run Excel invisibly. Use `excel_visible = true` to make Excel visible again in your RPA workflow.

read

Read data from Excel files. Both relative and absolute file paths supported. Error will be shown if the specified file or sheet does not exist. In below line, range can be a cell or range in Excel.

```
variable = [workbook]sheet!range
```

Reading columns and rows can be done using standard Excel formula for range, for example A:A (column A), B:D (columns B to D), 2:2 (row 2), 3:5 (rows 3 to 5). There is no standard Excel formula for selecting the entire range of a sheet, so you will have to provide the actual range required.

Examples

```
top_profit = [Monthly Report.xlsx]August!E10
top_salesman = [Monthly Report.xlsx]August!E11
data_array = [Quarterly Metrics.xlsx]Main!B3:G100
data_array = [C:\Reports\June.xls]Sheet1!A1:C2

data_array = [C:\Reports\June.xls]Sheet1!A:A
data_array = [C:\Reports\June.xls]Sheet1!B:D
data_array = [C:\Reports\June.xls]Sheet1!2:2
data_array = [C:\Reports\June.xls]Sheet1!3:5
```

TagUI's backend language is JavaScript, thus `data_array` can be used just like a JavaScript array.

```
// to work on data in data_array cell by cell
for row from 0 to data_array.length-1
  for col from 0 to data_array[row].length-1
    echo `data_array[row][col]`
```

Note - There was a limitation on reading multiple rows and columns, for eg B:D and 3:5 (data array returned will be a 1 x N array instead of the correct row x column array). This is now fixed in v6.87. Get your copy with `tagui update` command or from MS Word Plug-in Update TagUI button.

write

Write data to Excel files. Both relative and absolute file paths supported. If the specified file does not exist, a new file will be created. If the sheet does not exist, a new sheet will be created. If the data is an array, the given cell will be used as the top-left cell to write the range of cells.

```
[workbook]sheet!cell = variable
```

Examples

```
[Monthly Report.xlsx]August!E10 = 12345
[Monthly Report.xlsx]August!E11 = "Alan"
[Monthly Report.xlsx]August!E12 = variable
[Quarterly Metrics.xlsx]Main!B3 = data_array
```

TagUI's backend language is JavaScript, thus range data can be defined just like a JavaScript array.

```
// to assign a set of range data with 2 rows of 3 columns
[C:\Reports\June.xls]Sheet1!A1 = [[1, 2, 3], [4, 5, 6]]
[C:\Reports\June.xls]Sheet1!A1 = [[variable_1, variable_2, variable_3], [4, 5, 6]]

// example spreadsheet data with #, name and country
[Participants.xlsx]Sheet1!A1 = [['1', 'John', 'USA'], [2, 'Jenny', 'Russia'], [3,
→ 'Javier', 'Serbia']]

// get the next row count for the example spreadsheet
column_A = [Participants.xlsx]Sheet1!A:A
next_row = column_A.length + 1

// write a new row accordingly to example spreadsheet
[Participants.xlsx]Sheet1!A`next_row` = [[next_row, 'Janice', 'Brazil']]
```

copy

Copy data across Excel files. Both relative and absolute file paths supported. Error will be shown if the specified source file or sheet does not exist. If the specified destination file does not exist, a new file will be created. If the destination sheet does not exist, a new sheet will be created. If the data is an array, the given cell will be used as the top-left cell to write the range of cells.

```
[workbook]sheet!cell = [workbook]sheet!range
```

Examples

```
[Monthly Report.xlsx]August!A1 = [Jennifer Report.xlsx]August!A1
[Monthly Report.xlsx]August!A1 = [Jennifer Report.xlsx]August!A1:E200
```

delete

Delete data in Excel files. Both relative and absolute file paths supported. Error will be shown if the specified file or sheet does not exist. Delete a range of cells by assigning an empty array to it.

```
[workbook]sheet!cell = ""
```

Examples

```
[Monthly Report.xlsx]August!E10 = ""
[Quarterly Metrics.xlsx]Main!A1 = [["", "", ""], ["", "", ""]]
```

4.1.4 Word

You can read the text contents of a Microsoft Word document simply by assigning its filename to a variable as follows. TagUI will automate Microsoft Word to copy out the text contents and assign to the variable. Note that you need to have Microsoft Word installed on your computer. This feature works for both Windows and Mac.

Examples for Windows

```
word_text = [Research Report.docx]
word_text = [C:\Users\Jennifer\Desktop\Report.docx]
word_text = [FY2021 Reports\Research Report.docx]

filename = 'C:\\Users\\Jennifer\\Desktop\\Report '
word_text = ['filename`.docx]
filename = 'Research Report '
word_text = ['filename`.docx]
```

Examples for Mac

```
word_text = [Research Report.docx]
word_text = [/Users/jennifer/Desktop/Report.docx]
word_text = [FY2021 Reports/Research Report.docx]

filename = '/Users/jennifer/Desktop/Report '
word_text = ['filename`.docx]
filename = 'Research Report '
word_text = ['filename`.docx]
```

After reading the text content into a variable, you can process it using TagUI's helper functions such as `get_text()` and `del_chars()` to retrieve specific information required for your RPA scenario. Standard JavaScript functions can also be used to do string processing, for more information google javascript how to xxxx. After reading the text content from a Word document, TagUI will close Microsoft Word and continue with the rest of the automation steps.

4.1.5 PDF

You can read the text contents of a PDF file simply by assigning its filename to a variable as follows. TagUI will automate the PDF viewer app to copy out the text contents and assign to the variable. On Windows, you will need the free [Adobe Acrobat Reader](#) and set it as your default PDF viewer. On Mac, TagUI will use the default Preview app that can already view PDF files.

Examples for Windows

```
pdf_text = [Research Report.pdf]
pdf_text = [C:\Users\Jennifer\Desktop\Report.pdf]
pdf_text = [FY2021 Reports\Research Report.pdf]

filename = 'C:\\Users\\Jennifer\\Desktop\\Report '
pdf_text = ['filename`.pdf]
filename = 'Research Report '
pdf_text = ['filename`.pdf]
```

Examples for Mac

```
pdf_text = [Research Report.pdf]
pdf_text = [/Users/jennifer/Desktop/Report.pdf]
pdf_text = [FY2021 Reports/Research Report.pdf]

filename = '/Users/jennifer/Desktop/Report '
pdf_text = ['filename`.pdf]
filename = 'Research Report '
pdf_text = ['filename`.pdf]
```

After reading the text content into a variable, you can process it using TagUI's helper functions such as `get_text()` and `del_chars()` to retrieve specific information required for your RPA scenario. Standard JavaScript functions can also be used to do string processing, for more information google `javascript how to xxxx`. After reading the text content from a PDF file, TagUI will close the PDF viewer and continue with the rest of the automation steps.

4.1.6 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, and it is case sensitive.

Examples

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

access

To access the value of a variable in most steps, surround the variable in backticks, like `“my_variable”`. The following is an example of accessing a variable in the echo step.

Examples

```
my_variable = "hello world"
echo `my_variable`
// output: hello world
```

However, in certain steps like [if conditions](#), [for loops](#) and [helper functions](#), the variable can be accessed directly without backticks. For more information, see docs for the relevant step.

Examples

```
a = "hello"
b = "world"
if a equals to b
  echo same
  // output:

my_variable = 3
for n from 1 to my_variable
  echo `n`
  // output: 1
  // 2
  // 3

my_variable = "some text to copy"
clipboard(my_variable)
```

concatenation

To concatenate variables, the syntax varies depending on whether you are doing so within a step. The following examples show the difference between concatenating variables within and outside the echo step.

Examples

```
echo `a` `b`
// output: hello world

a = "hello"
b = "world"
c = a + " " + b
echo `c`
// output: hello world
```

4.1.7 File saving/loading**dump**

Saves text to a new file.

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



```
// creates blank CSV file with header
dump First Name,Last Name to names.csv
```

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
```

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
snap (0,0)-(100,100) to image.png
```

4.1.8 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"
```

4.1.9 Custom code

js

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

```
js [JavaScript statement]

js begin
[JavaScript statements]
js finish
```

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 finish
dump `obj` to result.json

// declare and initilise variable to use it inside/outside js code block
```

(continues on next page)

(continued from previous page)

```
a = ""
js begin
a = "some string"
js finish
echo `a`
```

py

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

```
py [Python statement]

py begin
[Python statements]
py finish
```

Examples

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

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

See [this link](#) for more examples and usage patterns on running Python code.

run

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

```
run [shell command]
```

Examples

```
run cmd /c mkdir new_directory
```

vision

Runs Sikuli code.

```
vision [Sikuli statement]

vision begin
[Sikuli statements]
vision finish
```

Examples

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

dom

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

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

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

Examples

```
// goes back to previous page
dom window.history.back()

// returns text of an element
dom return document.querySelector('#some_id').textContent
```

r

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

```
r [R statement]

r begin
[R statements]
r finish
```

4.1.10 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 (default 10 seconds).

```
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

Run steps interactively and immediately see the output. 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. If you are inside a code block, for example an if condition or for loop, be sure to indent your comment accordingly to let TagUI run correctly after it converts into JavaScript code.

```
// [comment]
```

Examples

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

telegram

Sends a Telegram notification, for example, to update on automation completion or exception.

First, message [@taguibot](#) to authorise it to send messages to your Telegram. Then in TagUI:

```
telegram [id] [message]
```

Examples

```
// this line sends message to Telegram user with ID 1234567890, \n means a new line
telegram 1234567890 Hello Alena,\n\nYour HR onboarding bot has completed successfully.

// show telegram_result variable - 'success' means sent, 'fail' means sending failed
echo Telegram message - `telegram_result`

// if condition to check telegram_result 'success' or 'fail' and handle accordingly
if telegram_result equals to 'success'
    echo Message sent successfully.
else
    echo Message sending failed.
```

Note that the telegram step requires an internet connection. This feature is being hosted at <https://tebel.org>, but the [source code](#) is on GitHub if you wish to host this feature on your own cloud or server. The implementation is in pure PHP without any dependencies.

The only info logged is chat_id, length of the message, datetime stamp (to prevent abuse). If you wish to host on your own, first read through this link to learn more about Telegram Bot API, creating your bot API token and setting up the webhook - <https://core.telegram.org/bots>

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 -deploy or -d

Deploys a flow, creating a shortcut which can be double-clicked to run the flow. If the flow file is moved, a new shortcut must be created. The flow will be run with all the options used when creating the shortcut.

4.2.2 -headless or -h

Runs the flow with an invisible Chrome web browser (does not work for visual automation).

4.2.3 -nobrowser or -n

Runs without any web browser, for example to perform automation only with visual automation.

4.2.4 -report or -r

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

4.2.5 -turbo or -t

Run automation at 10X the speed of normal human user. Read caveats at Advanced concepts!

4.2.6 -quiet or -q

Runs without output to command prompt except for explicit output (echo, show, check steps and errors etc). To have fine-grained control on showing and hiding output during execution (eg hiding password from showing up), use `quiet_mode = true` and `quiet_mode = false` in your flow.

4.2.7 -edge or -e

Runs using Microsoft Edge browser instead of Chrome (can be used with `-headless` option).

4.2.8 my_datatable.csv

Uses the specified csv file as the datatable for batch automation. See [datatables](#).

4.2.9 input(s)

Add your own parameter(s) to be used in your automation flow as variables p1 to p8.

For example, from the command prompt, below line runs `register_attendance.tag` workflow using Microsoft Edge browser and with various student names as inputs.

```
tagui register_attendance.tag -edge Jenny Jason John Joanne
```

Inside the workflow, the variables p1, p2, p3, p4 will be available for use as part of the automation, for example to fill up student names into a web form for recording attendance. The following lines in the workflow will output various student names given as inputs.

```
echo `p1`
echo `p2`
echo `p3`
echo `p4`
```

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

```
if present('//table')
  click button1
```

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`)
```

4.3.13 get_files()

Returns an array of files and folders in a given folder. Both relative and absolute paths supported.

Examples

```
// list of files in the same folder as the flow file
list = get_files('.')

// list of files in the Desktop folder of user Alan
// note double backslash because of JavaScript string
list = get_files('C:\\Users\\Alan\\Desktop')

// alternatively, use single forward slash instead
list = get_files('C:/Users/Alan/Desktop')

// showing the list of files after retrieving it
// JavaScript array start from 0 for 1st element
for n from 0 to list.length-1
    echo `list[n]`

// checking to process a specific file extension
for n from 0 to list.length-1
    if list[n] contains '.XLSX'
        echo `list[n]`
```

4.3.14 get_text()

Extracts text between 2 provided anchors from given text. Optional 4th parameter for occurrence during multiple matches (for example 3 to tell the function to return the 3rd match found).

Examples

```
pdf_text = 'Name: John State: Texas City: Plano Contact: ...'

name = get_text(pdf_text, 'Name:', 'State:')
state = get_text(pdf_text, 'State:', 'City:')
city = get_text(pdf_text, 'City:', 'Contact:')

echo `name`, `state`, `city`
```

4.3.15 del_chars()

Cleans data by removing provided character(s) from given text and returning the result.

Examples

```
pdf_text = 'Name: John\n State: Texas\t City: Plano\n Contact: ...'
echo `del_chars(pdf_text, '\n\t:')`
```

4.3.16 get_env()

Returns the value of given environment variable from the operating system.

Examples

```
// getting %USERPROFILE% variable for Windows
echo `get_env('USERPROFILE')`
home_dir = get_env('USERPROFILE')

// getting $HOME variable for Mac or Linux
echo `get_env('HOME')`
home_dir = get_env('HOME')
```


These are separate apps which help you in writing TagUI flows.

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 (for Python users)

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 is based on a fork of TagUI optimised for use by the Python package. Created and maintained by TagUI's creator Ken Soh [@kensoh](#).

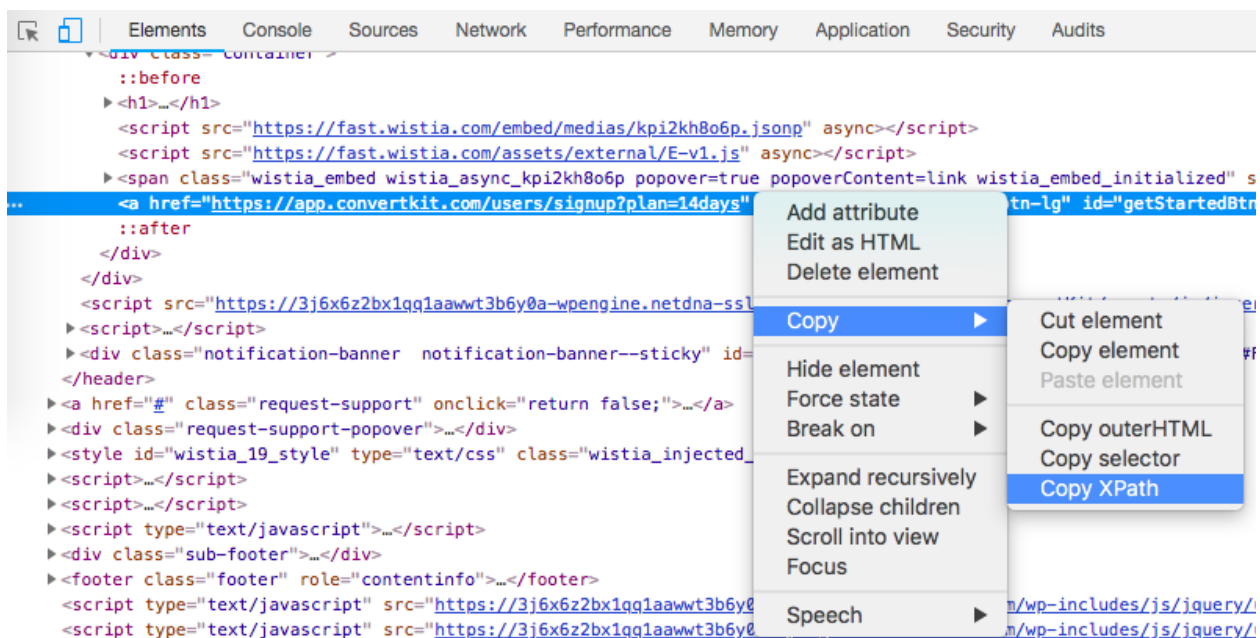
Frequently Asked Questions

6.1 How is TagUI licensed?

TagUI is a free, open-source, cross-platform software released under the Apache 2.0 license.

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

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

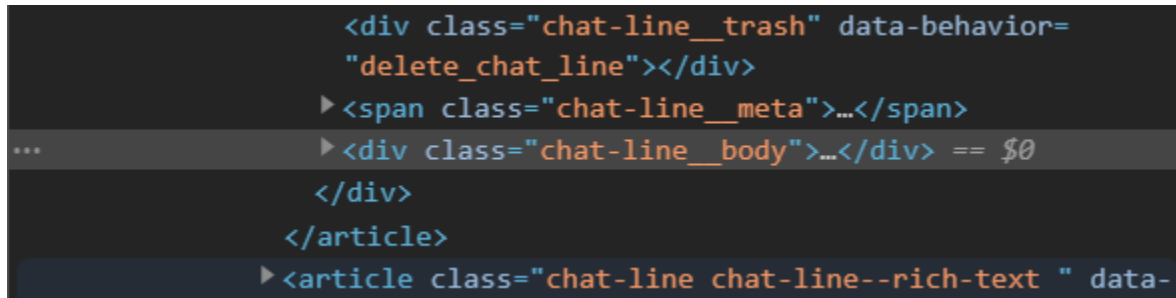


For some web pages, the XPath of an element can change. To combat this, you can find a stable element in the web page and writing a custom XPath relative to that stable element.

XPath is very powerful and can allow you to select web elements in many ways. Learn more about XPath at [W3Schools](#). Alternatively, check out [SelectorsHub Chrome extension](#) tool.

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

In Chrome/Edge, 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

6.4 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.

6.5 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.

6.6 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.

6.7 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` command.

6.8 How do I kill any unfinished TagUI processes?

If you Ctrl+C to break a TagUI automation, you can use `tagui/src/end_processes` (for macOS / Linux) or `end_processes.cmd` (for Windows) to kill any dead processes of TagUI integrations (Chrome, Edge, SikuliX, Python etc).

6.9 Why doesn't TagUI work on zoom levels other than 100%?

TagUI mimics the user mouse-clicks at the (x,y) coordinates of web elements, so using a different zoom level for your web browser will cause clicks to be triggered at wrong coordinates. Make sure TagUI's Chrome/Edge browser is set to 100% zoom for best results.

6.10 Is TagUI safe and secure 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/slimersjs>
- Python - <https://github.com/python/cpython>
- R - <https://github.com/wch/r-source>
- PHP - <https://github.com/php/php-src>

See this section on why TagUI has enterprise security by design - <https://github.com/kelaberetiv/TagUI#enterprise-security-by-design>

6.11 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.

6.12 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 OpenCV and Tesseract](#).

Download TagUI v6.46 [here](#).