Automa API Documentation

2.7.8

BugFree Software

27/04/2018

The automa.api module contains the implementation and API of Automa. It is a simple Python API that makes specifying GUI automation cases as simple as describing them to someone looking over their shoulder at a screen.

The public functions and classes of Automa are listed below. If you wish to use Automa functions in your Python scripts you can import them from the `automa.api` module:

```
from automa.api import *
```

If you wish to get an introduction to these functions, do try the interactive tutorial in the Automa console. You can start it via the command:

```
>>> start_tutorial()
```

`click`(**elements*)

Parameters ***elements** – Comma-separated list of GUI elements, labels (strings) or points.

Clicks on the given elements in sequence. Common examples are:

```
click("Close")
click("File", "Save")
click("File", MenuItem("Save"))
click(Button("OK"))
click(Point(200, 300))
click(ComboBox("File type").center + (50, 0))
```

Any object with a property called 'clickable_point' of type `automa.api.Point` (page 12) can be passed as a parameter. The only exception are strings, which are automatically wrapped as `Text` (page 9) objects (so `click("Close")` is equivalent to `click(Text("Close"))`).

`doubleclick`(*element*)

Parameters **element** – A GUI element, label (string) or point.

Performs a double-click on the given element. For example:

```
doubleclick("New folder")
doubleclick(ListItem("Directories"))
doubleclick(Point(200, 300))
doubleclick(TextField("File name").center - (0, 20))
```

Any object with a property called 'clickable_point' of type `automa.api.Point` (page 12) can be passed as a parameter. The only exception are strings, which are automatically wrapped as `Text` (page 9) objects (so `doubleclick("Automa")` is equivalent to `doubleclick(Text("Automa"))`).

`drag`(*gui_element*, *to*, *duration_secs=0*)

Parameters

- **gui_element** – The GUI element to drag.
- **to** – The GUI element or Point to drag to.
- **duration_secs** – the number of seconds, to simulate a slower drag.

Drags the given GUI element to the given location. For example:

```
drag("File", to="Folder")
```

Both parameters "gui_element" and "to" can be of any type that you can pass to `automa.api.click()` (page 1).

The dragging is performed by hovering the mouse cursor over "gui_element", pressing and holding the left mouse button, moving the mouse cursor over "to", and then releasing the left mouse button again.

The drag by default happens instantaneously, that is the cursor jumps from "gui_element" to "to" without any intermediate steps. This causes some applications to not notice the drag. For these cases, the parameter "duration_secs" makes it possible to simulate a more human-like drag. For instance:

```
drag("File", to="Folder", duration_secs=1)
```

`find_all` (*predicate*)

Lets you find all occurrences of the given GUI element predicate. For instance, the following statement returns a list of all buttons with label "Open":

```
find_all(Button("Open"))
```

In a typical usage scenario, you want to pick out one of the occurrences returned by `find_all()` (page 2). In such cases, `list.sort()` can be very useful. For example, to find the leftmost "Open" button, you can write:

```
buttons = find_all(Button("Open"))
leftmost_button = sorted(buttons, key=lambda button: button.x)[0]
```

`hover` (**elements*)

Parameters ***elements** – Comma-separated list of GUI elements, labels (strings) or points.

Consecutively hovers the mouse cursor over the given elements. For example:

```
hover("Close")
hover(Button("OK"))
hover("Home", "Download", "Start")
hover("Home", MenuItem("Download"), "Start")
hover(Point(200, 300))
hover(ComboBox("File type").center + (50, 0))
```

Any object with a property called 'clickable_point' of type `automa.api.Point` (page 12) can be passed as a parameter. The only exception are strings, which are automatically wrapped as `Text` (page 9) objects (so `hover("Close")` is equivalent to `hover(Text("Close"))`).

`kill` (*application*)

Parameters **application** – The application to kill, specified by name (string) or a `automa.api.Application` (page 5) instance.

Kills the given application and all windows belonging to it. For instance:

```
kill("Notepad")
```

You can also pass an `Application` (page 5) object as parameter. This is usually used to clean up after a test or automation case:

```
notepad = start("Notepad")
try:
    # Perform GUI automation...
finally:
    kill(notepad)
```

`press` (**keys*)

Parameters ***keys** – Key or comma separated list of keys to be pressed.

Presses the given keys in sequence. To press a normal letter key such as 'a' simply call *press* for that character:

```
press('a')
```

You can also simulate the pressing of upper case characters that way:

```
press('A')
```

To press a special key such as ENTER look it up in the list below, then call *press* for it:

```
press(ENTER)
```

To press multiple keys at the same time, concatenate them with +. For example, to press CTRL + a, call:

```
press(CTRL + 'a')
```

To press multiple key combinations in sequence, separate them by commas:

```
press(ALT + 'f', 's')
```

List of non-letter keys:

- **Common keys:** `ENTER`, `SPACE`, `TAB`, `ESC`
- **Modifiers:** `CTRL`, `ALT`, `SHIFT`
- **Insertion / Deletion:** `INS`, `BKSP`, `DEL`, `END`
- **Navigation:** `HOME`, `END`, `PGUP`, `PGDN`
- **Arrows:** `LEFT`, `DOWN`, `RIGHT`, `UP`
- **Other keys above the arrow keys:** `BREAK`, `PAUSE`, `PRTSC`, `SCROLLLOCK`
- **Other keys close to SPACE:** `LWIN`, `CAPSLOCK`, `MENU`, `RWIN`
- **Function keys:** `F1`, `F2`, ..., `F24`
- **Numpad operations:** `ADD`, `SUBTRACT`, `MULTIPLY`, `DIVIDE`,
- **Numpad numbers:** `NUMPAD0`, `NUMPAD1`, ..., `NUMPAD9`
- **Other numpad keys:** `NUMLOCK`, `DECIMAL`
- **Left/right variants of other keys:** `LSHIFT`, `RSHIFT`, `LMENU`, `RMENU`, `LCTRL`, `RCTRL`
- **IME keys:** `ACCEPT`, `CONVERT`, `MODECHANGE`, `NONCONVERT`, `FINAL`, `PROCESSKEY`, `HANGUL`, `HANJA`, `JUNJA`, `KANJI`
- **Others:** `APPS`, `ATTN`, `CLEAR`, `CRSEL`, `EREOF`, `EXECUTE`, `EXSEL`, `HELP`, `OEM_CLEAR`, `PLAY`, `PA1`, `SELECT`, `SEPARATOR`, `ZOOM`

`press_and_hold`(*key*)

Parameters key – The key to be pressed. For possible values, please see the documentation of `automa.api.press()` (page 2).

Presses - and holds - the given key. This can for example be used for zooming:

```
start("Chrome")
press_and_hold(CTRL)
scroll_up()
release(CTRL)
```

To release the pressed key, use `automa.api.release()` (page 3).

`release`(*key*)

Parameters key – The key to be released. For possible values, please see the documentation of `automa.api.press()` (page 2).

Releases a key that was previously held with `automa.api.press_and_hold()` (page 3).

`rightclick`(*element*, *select=None*, *then_select=None*, **finally_select*)

Parameters

- **element** – The GUI element, label (string) or point to right-click.
- **select** – GUI element, label (string) or point to left-click immediately after performing the right-click.
- **then_select** – GUI element, label (string) or point to left-click after clicking the `select` element.

- ***finally_select** – Comma-separated list of GUI elements, labels (strings) and points to left-click after clicking the `select` and `then_select` elements

Performs a right click on the given element, optionally clicking on the specified sequence of context menu elements afterwards. This can for example be used to empty the Recycle Bin:

```
rightclick("Recycle Bin", select="Empty Recycle Bin")
```

Any object with a property called ‘clickable_point’ of type `automa.api.Point` (page 12) can be passed as a parameter. The only exception are strings, which are automatically wrapped as `Text` (page 9) objects (so `rightclick("New folder")` is equivalent to `rightclick(Text("New folder"))`).

`save_screenshot`(*png_file_path*)

Parameters **png_file_path** – The file path (string) where the screenshot should be saved.

Saves a screenshot of the entire screen to the given file path. For instance:

```
save_screenshot(r"C:\screenshot.png")
```

`scroll_down`(*steps=1*)
Scrolls down the mouse wheel given number of steps.

`scroll_left`(*steps=1*)
Scrolls left the mouse wheel given number of steps.

`scroll_right`(*steps=1*)
Scrolls right the mouse wheel given number of steps.

`scroll_up`(*steps=1*)
Scrolls up the mouse wheel given number of steps.

`start`(*application*, **args*)

Parameters

- **application** (*str*) – The name or path to the application to be launched.
- ***args** – Arguments to be passed to the application.

Starts an application using the given parameters. Often, this works with just the name of the application, for instance:

```
start("Firefox")
```

Absolute paths and command line arguments can also be used. For example:

```
start("c:\Tools\Eclipse\eclipse.exe", "-data", "c:\workspace")
```

When absolute paths with backslashes are used, it is recommended that the strings containing the backslashes be prefixed with ‘r’, as in:

```
start(r"c:\Tools\Eclipse\eclipse.exe", "-data", r"c:\workspace")
```

The reason for this is that some characters have special meaning when prefixed by ‘’ - eg. ‘n’ stands for the newline character. The ‘r’ prefix disables the special treatment.

`switch_to`(*window_or_application*)

Parameters **window_or_application** – The title (string) of a window on screen, a `Window` (page 11) object or an `Application` (page 5) object as returned by `automa.api.start()` (page 4).

Switches to the given window. For example:

```
switch_to("Notepad")
```

This searches for a window whose title contains “Notepad”, and activates it.

You can also switch to an Application object returned by `automa.api.start()` (page 4):

```
notepad_1 = start("Notepad")
notepad_2 = start("Notepad")
switch_to(notepad_1)
write("Hello World!")
press(CTRL + 'a', CTRL + 'c')
switch_to(notepad_2)
press(CTRL + 'v')
```

This starts two instances of Notepad, writes “Hello World!” into the first, copies that text via CTRL + ‘c’, and pastes it into the second instance.

`wait_until`(*condition_fn*, *timeout_secs=None*, *interval_secs=1.0*)

Waits until the given condition function evaluates to true. This is most commonly used to wait for a GUI element to exist:

```
wait_until(Text("Update complete").exists)
```

When the optional parameter `timeout_secs` is given and not `None`, `wait_until` raises `TimeoutExpired` (page 13) if the condition is not satisfied within the given number of seconds. The parameter `interval_secs` specifies the number of seconds Automa waits between evaluating the condition function.

`write`(*text*, *into=None*)

Parameters

- **text** (*str*) – The text to be written.
- **into** (*str*) – Name or label of a text field to write into.

Types the given text into the active window of the process last started by `automa.api.start()` (page 4). If parameter ‘into’ is given, first sets the focus to the text field with name or label specified by this parameter. Common examples of ‘write’ are:

```
write("Hello World!")
write("test.txt", into="File name")
```

class `Application`(*title=None*)

Bases: `automa.api.APIElement`

A running instance of a program. This class is returned by `automa.api.start()` (page 4) when a program is started and can be passed to functions `switch_to()` (page 4) and `kill()` (page 2). Eg.:

```
notepad = start("Notepad")
# Perform some actions...
kill(notepad)
```

The `Application` objects roughly correspond to the entries you see in the “Applications” tab of the Windows task manager.

One difference between an `Application` and a `Window` is that an application can have any number of windows. For instance, when you press CTRL + s in Notepad, you get a new window (the “Save As” dialog). This window still belongs to the Notepad application. When you do:

```
kill("Notepad")
```

you kill the whole application, and thus all windows associated with it. This is useful for tearing down an application with possibly several open windows at the end of a script or test case:

```
my_app = start("MyApp")
try:
        # Perform GUI automation
finally:
        # Ensure the application is closed no matter what state
        # the previous steps left it in:
        kill(my_app)
```

exists()
Evaluates to true if this GUI element exists.

main_window
Returns the main window of this application.

title
Returns the title of this application. This is usually the title of the application's main window.

class Button(*name=None*, *below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*)
Bases: `automa.api.GUIElement` (page 7)

Lets you identify a button by its name and read its properties. An example usage of 'Button' is:

```
Button("Open file").exists()
```

This will look for a button with the 'Open file' label and will return True if found, False otherwise.

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `GUIElement` (page 7).

center
The center of this GUI element, as a `automa.api.Point` (page 12).

exists()
Evaluates to true if this GUI element exists.

height
The height of this GUI element.

is_enabled()
Returns true if this GUI element can currently be interacted with.

width
The width of this GUI element.

x
The x-coordinate of the top-left point of this GUI element.

y
The y-coordinate of the top-left point of this GUI element.

class CheckBox(*name_or_label=None*, *below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*)
Bases: `automa.api.GUIElement` (page 7)

Lets you identify a check box by its name or label and read its properties:

```
CheckBox("Show Navigator").exists()
```

This looks for a check box with label 'Show Navigator' and returns True if found, False otherwise.

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `GUIElement` (page 7).

center
The center of this GUI element, as a `automa.api.Point` (page 12).

exists()
Evaluates to true if this GUI element exists.

height
The height of this GUI element.

is_checked()
Returns true if this GUI element is checked (selected).

is_enabled()
Returns true if this GUI element can currently be interacted with.

width
The width of this GUI element.

x
The x-coordinate of the top-left point of this GUI element.

y
The y-coordinate of the top-left point of this GUI element.

class ComboBox(*name_or_label=None*, *below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*)
Bases: `automa.api.GUIElement` (page 7)

Lets you identify a combo box and read its properties and options. Eg.:

```
ComboBox("Language").exists()
```

This looks for a combo box with label 'Language' and returns True if found, False otherwise.

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `GUIElement` (page 7).

center
The center of this GUI element, as a `automa.api.Point` (page 12).

exists()
Evaluates to true if this GUI element exists.

height
The height of this GUI element.

is_enabled()
Returns true if this GUI element can currently be interacted with.

is_readonly()
Returns true if the value of this GUI element can be modified.

value
Returns the value of this GUI element.

width
The width of this GUI element.

x
The x-coordinate of the top-left point of this GUI element.

y
The y-coordinate of the top-left point of this GUI element.

class GUIElement(*below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*)
Bases: `automa.api.APIElement`

This class defines properties that are available for all of Automa's GUI elements.

One feature that all of Automa's GUI elements support are the optional arguments `below=`, `to_right_of=`, `above=` and `to_left_of=`. These arguments specify where a particular GUI element is to be searched for. For example:

```
Button("OK", to_left_of="Cancel")
```

This identifies the Button to the left of text "Cancel". Being able to restrict the search region for a GUI element like this can be useful for disambiguating multiple occurrences of a GUI element, especially when the occurrences are arranged in a table.

Relative GUI element searches can be nested and combined arbitrarily with Automa's other functions. For example:

```
click(Button("Open", to_right_of=TextField("File name")))
```

This clicks on the button with text “Open” to the right of text field “File name”.

center
The center of this GUI element, as a `automa.api.Point` (page 12).

exists()
Evaluates to true if this GUI element exists.

height
The height of this GUI element.

width
The width of this GUI element.

x
The x-coordinate of the top-left point of this GUI element.

y
The y-coordinate of the top-left point of this GUI element.

class **ListItem**(*name=None*, *below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*)
Bases: `automa.api.GUIElement` (page 7)

Lets you identify a list item by its name. This is most useful for distinguishing GUI elements with similar labels: In a file dialog, you often have a tree of folders on the left, and a list of elements of the currently selected folder on the right. Depending on which folder is selected, it may happen that a folder appears both in the tree on the left and in the list on the right. ListItem lets you pick out the item in the list. To select the item from the tree, use TreeItem.

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `GUIElement` (page 7).

center
The center of this GUI element, as a `automa.api.Point` (page 12).

exists()
Evaluates to true if this GUI element exists.

height
The height of this GUI element.

width
The width of this GUI element.

x
The x-coordinate of the top-left point of this GUI element.

y
The y-coordinate of the top-left point of this GUI element.

class **MenuItem**(*name=None*, *below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*)
Bases: `automa.api.GUIElement` (page 7)

Lets you identify a menu item by its name. This can be useful for distinguishing GUI elements with the same name. For instance, you could have a menu item “Close” in the “File” menu, and a button “Close”. To distinguish between the two, you can use:

```
MenuItem("Close")
```

and:

```
Button("Close")
```

respectively.

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `GUIElement` (page 7).

center
The center of this GUI element, as a `automa.api.Point` (page 12).

exists()
Evaluates to true if this GUI element exists.

height
The height of this GUI element.

width
The width of this GUI element.

x
The x-coordinate of the top-left point of this GUI element.

y
The y-coordinate of the top-left point of this GUI element.

class **RadioButton**(*name_or_label=None*, *below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*)
Bases: `automa.api.GUIElement` (page 7)

Lets you identify a radio button by name or label and read its properties. To for instance check whether a radio button is currently selected, use:

```
RadioButton("Option 2").is_selected()
```

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `GUIElement` (page 7).

center
The center of this GUI element, as a `automa.api.Point` (page 12).

exists()
Evaluates to true if this GUI element exists.

height
The height of this GUI element.

is_enabled()
Returns true if this GUI element can currently be interacted with.

is_selected()
Returns true if this radio button is selected.

width
The width of this GUI element.

x
The x-coordinate of the top-left point of this GUI element.

y
The y-coordinate of the top-left point of this GUI element.

class **Text**(*value=None*, *below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*)
Bases: `automa.api.GUIElement` (page 7)

Lets you identify any text or label. This is most useful for checking whether a particular text is shown on the screen:

```
Text("Hello World!").exists()
```

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `GUIElement` (page 7).

center
The center of this GUI element, as a `automa.api.Point` (page 12).

exists()
Evaluates to true if this GUI element exists.

height
The height of this GUI element.

value
Returns the current value of this Text object.

width
The width of this GUI element.

x
The x-coordinate of the top-left point of this GUI element.

y
The y-coordinate of the top-left point of this GUI element.

class **TextField**(*name_or_label=None*, *below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*)
Bases: `automa.api.GUIElement` (page 7)

Lets you identify a text field on the screen. For example:

```
TextField("File name").value
```

This returns the value of the "File name" text field. If it is empty, the empty string '' is returned.

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `GUIElement` (page 7).

center
The center of this GUI element, as a `automa.api.Point` (page 12).

exists()
Evaluates to true if this GUI element exists.

height
The height of this GUI element.

is_enabled()
Returns true if this GUI element can currently be interacted with.

is_readonly()
Returns true if the value of this GUI element can be modified.

value
Returns the current value of this text field. '' if there is no value.

width
The width of this GUI element.

x
The x-coordinate of the top-left point of this GUI element.

y
The y-coordinate of the top-left point of this GUI element.

class **TreeItem**(*name=None*, *below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*)
Bases: `automa.api.GUIElement` (page 7)

Lets you identify a tree item by its name. This is most useful for distinguishing GUI elements with similar labels: In a file dialog, you often have a tree of folders on the left, and a list of elements of the currently selected folder on the right. Depending on which folder is selected, it may happen that a folder appears both in the tree on the left and in the list on the right. TreeItem lets you pick out the item in the tree. To select the item from the list, use ListItem.

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `GUIElement` (page 7).

`center`
The center of this GUI element, as a `automa.api.Point` (page 12).

`exists``()`
Evaluates to true if this GUI element exists.

`height`
The height of this GUI element.

`width`
The width of this GUI element.

`x`
The x-coordinate of the top-left point of this GUI element.

`y`
The y-coordinate of the top-left point of this GUI element.

class **`Window`**`(`*title=None*, *below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*`)`
Bases: `automa.api.GUIElement` (page 7)

Lets you identify a window by its title.

`center`
The center of this GUI element, as a `automa.api.Point` (page 12).

`exists``()`
Evaluates to true if this GUI element exists.

`handle`
Returns the Windows Operating System handle (HWND) assigned to this window.

`height`
The height of this GUI element.

`is_minimized``()`
Returns whether this window is minimized.

`width`
The width of this GUI element.

`x`
The x-coordinate of the top-left point of this GUI element.

`y`
The y-coordinate of the top-left point of this GUI element.

class **`Image`**`(`*image_file_path*, *min_similarity=0.7*, *below=None*, *to_right_of=None*, *above=None*, *to_left_of=None*`)`
Bases: `automa.api.GUIElement` (page 7)

Lets you find an image on the screen. For instance:

```
click(Image("save.png"))
```

This searches the screen for the pattern given by “save.png” and clicks it.

The optional parameter ‘min_similarity’ specifies the minimum similarity (as a number between 0.0 and 1.0) the image on the screen must have with the sought image. The default value 0.7 represents a “main similarity”. A value of 0.9 means “very similar” and values >= 0.99 mean “extremely similar, with full colors and details”. Below 0.99, all image comparisons are performed in grayscale.

You can also use absolute paths to locate image files. For example:

```
hover(Image(r"c:\images\button_ok.png"))
```

The ‘r’ prefix before the first quotation mark is required when inputting paths containing backslashes “\”.

When not using absolute paths, image files are assumed to lie in the current working directory. You can find out what the current working directory is using the following commands:

```
>>> import os
>>> os.getcwd()
```

For an explanation of the parameters `below`, `to_right_of`, `above` and `to_left_of`, please see the documentation of `GUIElement` (page 7).

`any_times` (*axis=1*)

Returns an `Image` that matches zero or more occurrences of this image on the screen.

By default, `any_times` means "any number of times horizontally". To identify any number of occurrences of an image stacked on top of each other, use `any_times(..., VERTICALLY)`.

`center`

The center of this GUI element, as a `automa.api.Point` (page 12).

`exists`()

Evaluates to true if this GUI element exists.

`followed_by` (*image*, *axis=1*)

Returns an `Image` that represents this image followed by the given image.

By default, `followed_by` means "followed by horizontally". To identify an image below another image, use `followed_by(..., VERTICALLY)`.

`height`

The height of this GUI element.

`maybe_once`()

Returns an `Image` that matches zero or one occurrences of this image on the screen.

`once_or_more` (*axis=1*)

Returns an `Image` that matches one or more occurrences of this image on the screen.

By default, `once_or_more` means "once or more horizontally". To identify one or more occurrences of an image stacked on top of each other, use `once_or_more(..., VERTICALLY)`.

`width`

The width of this GUI element.

`x`

The x-coordinate of the top-left point of this GUI element.

`y`

The y-coordinate of the top-left point of this GUI element.

class `Point` (*x=0*, *y=0*)

A clickable point. To create a `Point` at an offset of an existing point, use + and -:

```
>>> point = Point(x=10, y=25)
>>> point + (10, 0)
Point(x=20, y=25)
>>> point - (0, 10)
Point(x=10, y=15)
```

`x`

The x coordinate of the point.

`y`

The y coordinate of the point.

exception `NoWorkWindowError` (*message=None*)

Automa performs all searches for GUI elements and images in the window it last operated with. This window is called the "work window". Work windows are typically registered with Automa as the result of the start(...) and switch_to(...) commands. If there is no work window and an operation is performed that requires searching for a GUI element, such as for instance the command:

```
>>> click("File")
```

then a NoWorkWindowError is raised, asking you to start(...) or switch_to(...) an existing window first.

exception `TimeoutExpired`(*message=None*)

Exception raised when Automa runs into a (predictable) timeout, such as during execution of command `wait_until()` (page 5).

class `Config`

This class contains Automa's run-time configuration. To modify Automa's behaviour, simply assign to the properties of this class. For instance:

```
Config.auto_wait_enabled = False
```

`auto_wait_enabled` = True

When auto-wait is enabled, Automa automatically waits after performing GUI actions, such as `start()` (page 4), `click()` (page 1) or `write()` (page 5). The amount of time waited is calculated dynamically, using criteria such as the current CPU usage. This can sometimes lead to wait intervals that are too long. To speed up the execution of your scripts, the property `auto_wait_enabled` can therefore be used to disable Automa's automatic wait facility.

You can disable or enable Automa's auto-wait at any point in your script. For example:

```
>>> Config.auto_wait_enabled = False
>>> write("John", into="Name")
>>> press(TAB)
>>> write("Smith")
>>> Config.auto_wait_enabled = True
>>> click("Submit")
```

For a way of speeding up Automa's auto-wait without completely disabling it, see `automa.api.Config.wait_interval_secs` (page 13).

`search_timeout_secs` = 30

Amount of time (in seconds) Automa attempts to find a GUI element for. If the search does not succeed within the time defined by this configuration value, Automa raises `TimeoutExpired` (page 13). Increasing the property of this value can be useful on lower-spec computers.

`search_max_depth` = 100

The maximum number of UI elements Automa investigates when looking for a given element. Increasing this value makes Automa run slower, however the results returned are more accurate.

`wait_interval_secs` = 0.3

Determines the minimum time (in seconds) Automa waits for actions to complete when `automa.api.Config.auto_wait_enabled` (page 13) is set to True. This configuration parameter also specifies how long Automa waits between every two consecutive measurements when probing an applications' activity to check whether the action has completed or not.

The higher the value of this configuration parameter, the more stable but slower script runs become. The value must always be greater than zero.

`delay_after_mouse_move_secs` = 0.01

Determines the time (in seconds) Automa waits between moving the mouse and performing a mouse click when one of the commands `click()` (page 1), `rightclick()` (page 3) and `doubleclick()` (page 1) is executed. The value should always be strictly greater than zero to ensure stability of automation scripts.

classmethod `reset`()

Resets the config to default values.