AutoCoconut, a workflow recorder.

AutoCoconut is a tool that tracks simple mouse and keyboard events (such as clicks, moves, key presses, etc.) and interprets them into *meta events* (mouse drags, double clicks, combo presses). While taking the events, it also makes screenshots that are connected to actual events. The recording can be then saved as a json file or converted into a *written procedure* (a workflow report) formatted as *adoc*, *html*, or *pm* (for OpenQA). Also, it can be used to open a previously created *json* file and convert it into the procedure later.

Such workflow reports can be helpful when creating bug reproducers, tutorials, or test cases for GUI testing frameworks, such as OpenQA and others.

Currently, the application only works correctly under **X11**. On Wayland, it only works for applications using the *XWayland* protocol. There is intention to make it **Wayland** compatible, too, once there are ways to get the necessary information from the system.

Development

Currently, the development has reached **Phase 3**.

Which means that the script is able:

- record various events, mouse buttons and actions (click, double click, drag, vertical scroll), keyboard events (press and release)
- identify various types of keys (modifiers, special keys, character keys, etc.)
- find pre-defined patterns in single events and interpret them
- take screenshots to illustrate the workflow (or create needles for OpenQA)
- produce various output - **raw** file, **json** file, or a workflow description in **adoc**, **html** and **pm**.
- it has a GUI to make it easier to use

How does AutoCoconut work?

When the application starts, it is ready to record the mouse and keyboard events. The recording is started using a **stop key** or a **Start** button (GUI). Then mouse and keyboard events are recorded until the **stop key** or the **Stop** button are pressed again. The list of all recognized mouse and keyboard events (in chronological order) forms a *raw json*.

Since this only contains single events without any relations between them, the data is not very useful. Therefore, they are interpreted to find certain logical patterns and therefore merge some of the single events into *meta events*.

These meta events bare the real meaning, such as mouse *double clicks*, *mouse drags*, *wheel scrolls*, *key combination*, or even *mouse and keyboard combinations*. The meta events, also stored in a json format, are then converted into a *procedure* in a selected format.

When recording events, the application also takes pictures of screens (screenshots) to capture the situation on the screen at the time of the event. For most of the actions, two screenshots are taken: a **primary** one and an **alternative** one. The primary screenshot is taken in the moment of the event, the alternative screenshot is taken earlier (or later) according to a **time_offset** that a user can set (CLI application only). By default the `time_offset` is **1 second**.

How to use the application

AutoCoconut can be started as a *CLI application* or a *GUI application*.

As a CLI application, **AutoCoconut** makes a single recording pass and produces the chosen output.

As a GUI applications, it allows for more functionality. For example, you can:

- repeat the recording,
- review the recorded events,
- edit the events,
- create new events.

However, using the GUI will always produce some extra events at the beginning or at the end, when users need to navigate to the application they want to test, so some later editting might be needed.

There is also an option to record the events using the CLI application and then use the GUI to make the edits.

Using AutoCoconut as a CLI application.

This procedure shows the general use of the AutoCoconut CLI.

1. Start the script on a terminal.
2. Switch to a different application and prepare for the recording.
3. Press the **stop key** to start recording (**F10** by default).
4. Perform your desired task.
5. Press the **stop key** again to stop recording.
6. You will receive the output according to your choice.

CLI arguments and their explanation

The script accepts various arguments to control the flow and the output:

`-s, --stopkey`
: The *stop key* is used to start and stop the recording. By default, it is **F10**. Using this option, you can choose a stop key to your likings.

> **WARNING** If you choose a *stop key* that you want to use as a *regular key* later in the process, pressing that key will terminate the recording. If the **F10** key does not fit into your procedure, you can try the **esc** key.

`-e, --offset`
: Defines a time (in seconds) that the script uses as an offset time correction to take the alternative screenshot. Usually, the application takes an earlier screenshot for mouse actions to make sure the screenshot avoids *hover-on* changes of the clicked item, and a later screenshot for keyboard actions to show the result of such action.

> **NOTE** Recording applications with slower response, such as web pages that need to load, it might be better to make the offset higher, to give the later screenshot some time to wait for the application to come to a desired state. The default is **1 second**.

`-o, --output`
: You can choose one of several outputs. The **raw** output returns a json file with all single events without interpretation. In this json file, all key presses and releases are recorded separately, including the combinations. The **json** output provides an interpreted list of super events organized in a json file. The **adoc**, **html**, and **openqa** outputs produce a list of steps in that chosen format. The **openqa** format lists the OpenQA test commands that can be used for OpenQA scripts and also creates the *needle files* for the screenshots.

`-f, --file`
: If a filename is given, the output (including the screenshots) will be saved into a file and all recording data will be moved to a new directory to protect them from deleting when the scrit is run again. Without this option, the output is only shown on a command line.

Examples

- To use the *Esc* key as a *stop key* use the `-s` argument.

```
$ ./autococonut.py -s esc
```

- To create the workflow in *html* format, use the `-o` argument.

```
$ ./autococonut.py -o html
```

- To save the workflow in an *AsciiDoc* file, use the `-f` and the `-o` arguments.

```
$ ./autococonut.py -o adoc -f <filename.adoc>
```

Using the GUI version of AutoCoconut

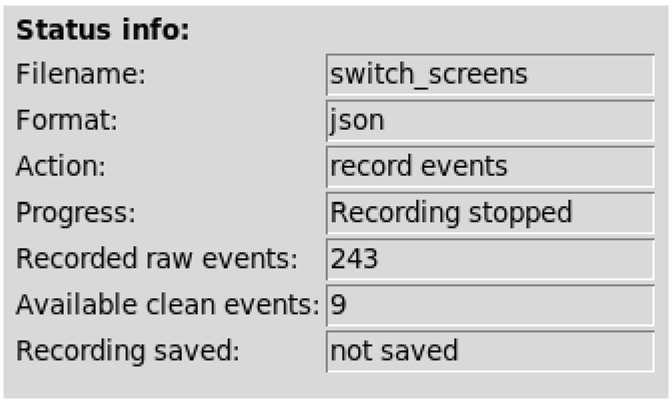
Using the GUI version of **AutoCoconut** has some advantages when compared to the CLI version, however the GUI version does not allow users to set the amount of *offset* (see the CLI chapter) which is always **1 second**.

Use this command to start the GUI version of **AutoCoconut**:

```
$ ./autococonut-gui.py
```

The Status Info frame

AutoCoconut starts in the recording mode. In this mode, you can start and stop recording the events, as well as inspect the recorded events.



Status info:

Filename:	switch_screens
Format:	json
Action:	record events
Progress:	Recording stopped
Recorded raw events:	243
Available clean events:	9
Recording saved:	not saved

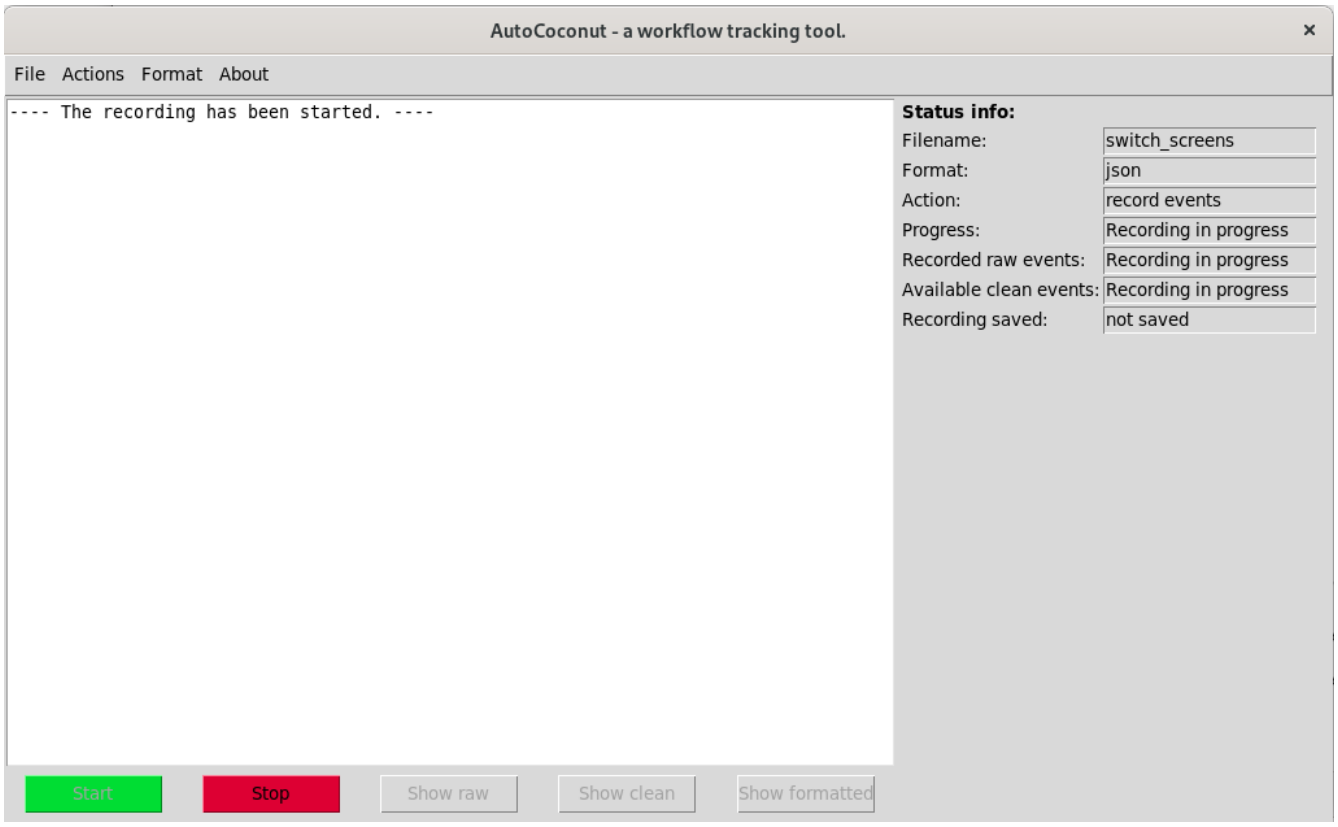
Figure 1. AutoCoconut - Status info panel

On the right side of the window, the **Status info** frame displays some useful information about the current session:

Filename	Shows the selected name of the output file. To set the file name, use the **File** → **New file** menu item.
Format	Shows the selected format. The format is selected when using the **File** → **New file** menu item, but it can be overridden using one of the **Format** menu items.
Action	Shows the currently selected action.
Progress	Shows if any changes have been made to the data.

Recorded raw events	Shows the number of simple events recorded during the session. This will only apply when you have actually recorded some data. The field will not show any manual changes to the events.
Available clean events	Shows the number of interpreted *meta* events in the recording or the number of events in the current data.
Recording saved	Shows if the current recording has been saved. If **not saved** is shown, you need to save it if you want to keep the current data.

The Record screen



```
---- The recording has been started. ----
```

Status info:

Filename:	switch_screens
Format:	json
Action:	record events
Progress:	Recording in progress
Recorded raw events:	Recording in progress
Available clean events:	Recording in progress
Recording saved:	not saved

Figure 2. AutoCoconut - Record screen

The **Record** screen is the **AutoCoconut**'s default application screen. When on a different screen, you can come back to this one using **Actions** → **Record** menu items.

Making the recording

In the GUI version of AutoCoconut, you can make any number of recordings without a need to restart it. To make a recording:

1. Press the **Start** button.

NOTE	When the **Start** button is pressed, any following events will be recorded until you press the **Stop** button.

2. Perform a procedure you want to record.

3. Press the **Stop** button.

Reviewing the recording

After you have made the recording, you can review the recorded data as *raw json* or *json* to see the particular events. Alternatively, you can have the data translated into the formatted output which will respect the selected format. buttons below the text field, where the data appear:

1. The **Show raw** button shows the *raw* json with all recorded events.

   NOTE When no recording has been made, because you have opened a previously saved **json** file or you have created all events manually, the *raw* data will not be available for reviewing.

2. The **Show clean** button shows the *interpreted* json with the *meta* events.
3. The **Show formatted** button shows the *formatted* workflow description in the selected format.

   NOTE If no format is selected (using the **Format** menu for instance), the *clean json* file will be shown similarly to using the **Show clean** button.

Saving the recording

When you are satisfied with the recorded data, you can save the output:

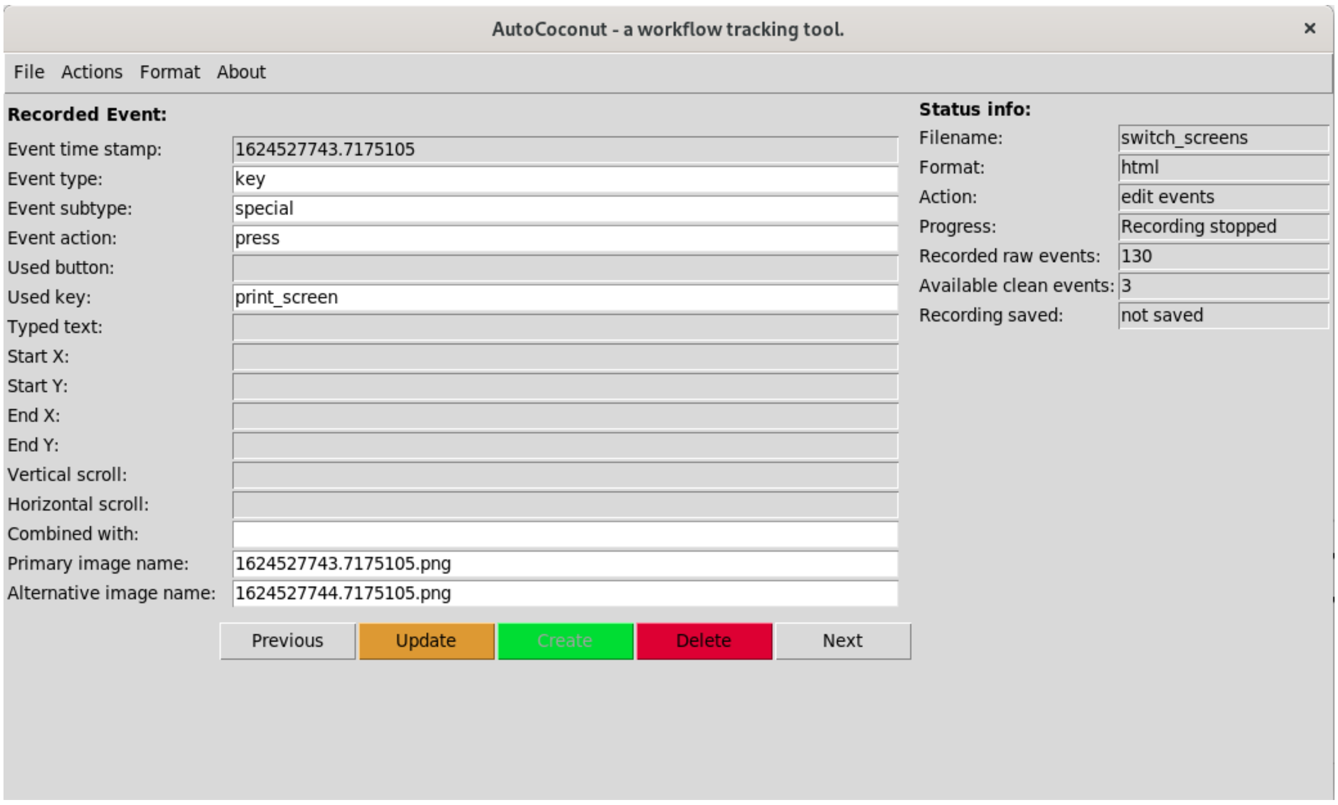
1. Use **File** → **New file** to select the location and the output format. The selection will be indicated in the **Status Info**.

   WARNING Selecting the new file will **not** actually save it. This action will only select the output file and the file format.

2. Use **File** → **Save file** to save the output data. The data will be formatted according to the selected format. Also, the screenshots will be renamed to their final names, e.g. `stepX_needle.png`, and moved to the location where the file being saved is placed.

   WARNING If the application status is **not saved** then all current screenshot files will be deleted when the **Start** button is pressed to start a new recording in order not to mix screenshots between recordings.

The Edit screen



Recorded Event:

Event time stamp:	1624527743.7175105
Event type:	key
Event subtype:	special
Event action:	press
Used button:	
Used key:	print_screen
Typed text:	
Start X:	
Start Y:	
End X:	
End Y:	
Vertical scroll:	
Horizontal scroll:	
Combined with:	
Primary image name:	1624527743.7175105.png
Alternative image name:	1624527744.7175105.png

Status info:

Filename:	switch_screens
Format:	html
Action:	edit events
Progress:	Recording stopped
Recorded raw events:	130
Available clean events:	3
Recording saved:	not saved

Figure 3. AutoCoconut - Edit screen

The **Edit** screen allows you to edit or delete any existing *meta* event in the recording.

You can switch to this screen using the **Actions** → **Edit** menu items.

When you switch to this screen, the very first event will be displayed for editting, if such event exists. **AutoCoconut** will only allow to edit fields that are used by the certain event and will not allow to add any other values to it in order to protect the recorded event from destroying its inner logic and keeping coherence with other events.

NOTE

Such behaviour has a good reason. If, for example, you want to edit a mouse click, there is no need why you should add values for a *key* or a *text* event and vice versa.

Also, editting the time stamp is never allowed to maintain the chronological continuity of the data.

Following buttons can be used on this screen:

Previous	Moves backward in the sequence of events. If you reach the beginning of the sequence, pressing the button will not have any effect.
Next	Moves forward in the sequence of events. If you reach the end of the sequence, pressing the button will not have any effect.
Update	Update the current event with the suggested changes.

Delete Deletes the current event from the sequence of events. Note that deleting an event is irreversible.

NOTE

Deleting an event may be actually very useful in situations when you need to perform several steps after you have started the recording and you do not want to keep them in your workflow.

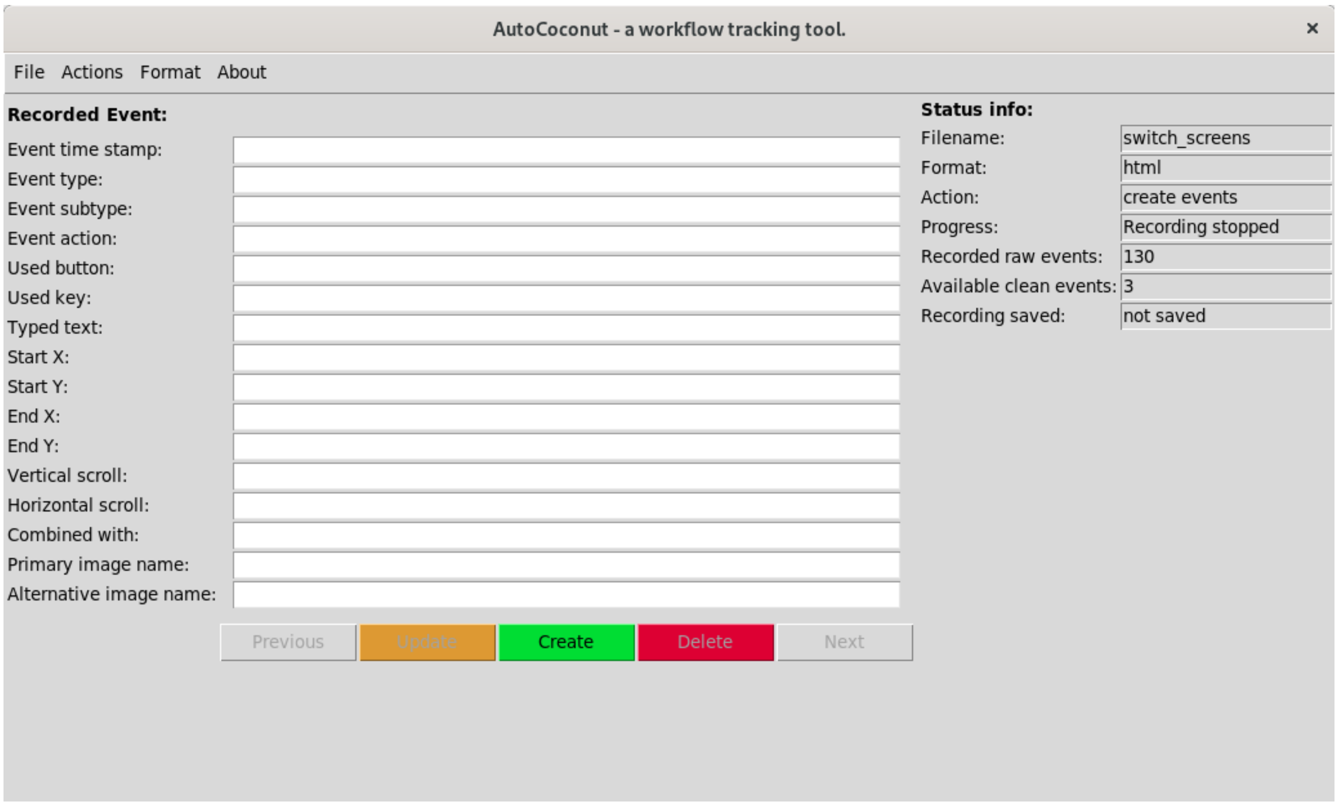
When you press **Record** any events will be recorded, also these that take you to the application you actually want to record and those taking you back to the AutoCoconut's window. If you do not want to have them in your recording, you can delete them from it.

WARNING

Updating or deleting an event will update the event in the memory but **will not** save the changes to the disk. Save the file, if you want to keep the changes.

Some actions on the **Edit screen** will also update some of the values in the **Status Info** frame.

The Create screen



Status info:

Filename:	switch_screens
Format:	html
Action:	create events
Progress:	Recording stopped
Recorded raw events:	130
Available clean events:	3
Recording saved:	not saved

Figure 4. AutoCoconut - Create screen

The **Create** screen allows you to create a new event, or a new sequence of events. Although you could use it to create the whole recording manually, the purpose to have this option is to enable corrections to the recording when an extra event would be useful.

You can get to this screen using the **Actions** → **Create** menu items.

Available entry fields

Event time stamp

Each event should have a time stamp which is used to maintain chronological order among the data. You should be very careful if you want to add an event that will fit in between two existing events. In that case, the new time stamp should be bigger than that of the first event and smaller then of the second event. You are free to enter decimal values, too. If left out, the current time will be filled in automatically, which makes the created event being the last in the data. Time stamps provided by the *event collector* will always be in the *Unix epoch* format.

Event type

Each event must have this set. Refer to Quick overview of event features for available event types.

Event subtype

Each event must have this set. Refer to Quick overview of event features for available event subtypes.

Event action

Each event represents an action. Refer to Quick overview of event features for available event actions.

Used button

Defines a mouse button clicked during this event. Refer to Quick overview of event features for available buttons.

Used key

Defines a key that was pressed during this event. Refer to Keys for a list of key names.

Typed text

If this event should be a typed text, the *string* value is given here.

Start X and Y

Define the X and Y coordinates for any simple mouse click, but also the starting point coordinates for a mouse drag.

End X and Y

Define the ending point coordinates for a mouse drag.

Vertical scroll

Defines the number of steps scrolled in the vertical direction. A positive value represents the **northern** direction (up the screen) while the negative value represents the **southern** direction (down the screen).

Horizontal scroll

Defines the number of steps scrolled in the horizontal direction. A positive value represents the **eastern** direction (right) while the negative value represents the **western** direction (left). This is not available on normal mice.

Combined with

Defines combination keys pressed during this event, such as the modifier keys (*shift*, *ctrl*, *alt*, etc.).

Primary and secondary images

Define the names of the primary and secondary screenshots for this event.

WARNING

The application will not perform any checks whether the screenshots exist or not. Usually, the recorded screenshots are placed in the working directory and they are moved into a new location, when the project is saved for the first time. The target location will correspond with the location of the project file.

If you want to generate correct output from the saved files, make sure, you have the screenshots in the same directory as the project file and that the names do match.

When you are satisfied with the entered event, you can create it using the **Create** button.

NOTE

On this page, events only can be created! If you need to edit or delete them later, switch to the **Edit** screen using **Actions** → **Edit**.

Key and event library

AutoCoconut recognizes presses and releases of most of the keyboard keys as well as the most common mouse actions. Certain patterns are recognized in the sequence of events and merged into *meta events* which are then stored in a json file. They inherit some of the features from the single events, such as the coordinates of the mouse click point and so on.

When you want to manually enter events, you should not deliberately features, but you should only add those that would be naturally present in the event created by the *interpreter*. Also, when you want to constitute an event where a key is involved, you should use the correct name for the key.

This part is a collection of key names and event features for your reference.

Keys

Keys are divided into **three** main groups:

1. Modifiers
2. Special keys
3. Alpha-numeric keys

Modifiers

Modifiers are used to create key combinations. Usually, when a modifier is pressed for a first time, it is remembered and all events that happen until the same modifier is released, they are recorded as **modified**, i.e. they have this modifier in the **Combined with** field. You can use more than one modifier at a time.

Shift is never recorded as a modifier when it comes with an alpha-numeric key. The capitalized version is recorded instead. Because of this behaviour, it is not possible to recognize **Shift** as a modifier unless it is used with another modifier preceding it. If, for example, you want to record the **Shift-Ctrl-G** combination, you should record it as **Ctrl-Shift-G** combination to avoid possible recognition problems.

Right Alt is also a problematic key, because **pynput** (the Python library to record events) returns an empty string when it is pressed on my set-up. Similarly, some other keys might behave the same way. Luckily, there are corner cases, so AutoCoconut will try to make a guess which key might have been pressed based on the context, if it sees an empty value. However, you should expect discrepancies when using these keys.

AutoCoconut has been tested on *American English* and *Czech* keyboard layouts. Expect some deviations on different layouts, especially on non-latin ones.

Table 1. Recognized modifiers

Key	Key type	Key code	AutoCoconut’s use
Alt	modifier	alt	combination
Left Ctrl	modifier	ctrl	combination
Right Ctrl	modifier	ctrl_r	combination
Shift	modifier	shift	combinations or capitalization
Win key	modifier	cmd	combinations or standalone
Alt Gr	modifier	None	combinations or switching

Special keys

Non alpha-numeric keys are taken as **special keys**. They mostly have the same function as they would have in the system, but they might have some effects on the interpretation when converting *single events* into *meta events*. For example, a special key can be combined with a modifier, but cannot be combined with other keys. When there is a typing sequence (a sequence of plain alpha-numeric characters), this sequence is **always** interrupted when a *special key* is pressed.

For example, if you typed `Hello<TAB>World`, it would be split into the following events:

1. Type `Hello`.
2. Press `TAB`.
3. Type `World`.

Backspace is never recorded. It is always used to correct previously typed characters.

F10 is by default used as a stop key that *starts and stops* the recording when using the CLI version of AutoCoconut. If your workflow requires pressing this key, you have to redefine it using the `-s` or `--stopkey` option. For the GUI version, **F10** works normally as the recording is started and stopped from the **AutoCoconut**'s recording window.

Table 2. Recognized special keys

Key	Key type	Key code
Menu key	special key	menu
Esc	special key	esc
Enter	special key	enter
Backspace	special key	backspace
Insert	special key	insert
Home	special key	home
Page up	special key	page_up
Page down	special key	page_down
End	special key	end
Delete	special key	delete
Print Screen	special key	print_screen
Scroll Lock	special key	scroll_lock
Caps Lock	special key	caps_lock
Pause	special key	pause
Up	special key	up
Down	special key	down
Left	special key	left
Right	special key	right
F1 - F9	special keys	f1 - f9
F10	stop key (CLI), special key (GUI)	f10
F11 - F12	special keys	f11 - f12

NOTE To reduce the number of created pictures and save system resources, screenshots are only taken when modifiers and special keys are pressed.

Mouse events

There are several mouse events recognized by **AutoCoconut**. Each of the events may be described using certain features.

Mouse click

The mouse click occurs when a mouse button is clicked and released and when this event is not followed by another click happening right after the first one and in the same location.

The typical mouse event looks like this:

Key word	Value	More info
timestamp	positive (decimal) number	usually an epoch number, e.g. 1623421318.394444
type	mouse	-
action	click	-
button	left, right, middle	indicates which button to press
start X	positive number	indicates the X coordinate of the click location
start Y	positive number	indicates the Y coordinate of the click location
primary image	file name	taken one second before the event
alternative image	file name	taken in the moment of event happening
combined with	empty or key name(s)	indicates if the event is modified

Table 3. Example: Right mouse button click at 1000, 450 (not modified)

timestamp	1623421318.394444
type	mouse
action	click
button	right
start X	1000
start Y	450
primary image	before_click_start_button.png
alternative image	click_start_button.png
combined with	*empty field*

Mouse scroll

The mouse scroll is a situation when when a mouse wheel is scrolled in a certain direction.

The typical mouse scroll has the following features:

Key word	Value	More info
timestamp	positive (decimal) number	usually an epoch number, e.g. 1623421318.394444
type	mouse	-
action	scroll	-
start X	positive number	indicates the X coordinate of the mouse position
start Y	positive number	indicates the Y coordinate of the mouse position

Key word	Value	More info
horizontal	number	shows how many scroll steps were made in the horizontal direction
vertical	number	shows how many scroll steps were made in the vertical direction
primary image	file name	taken right after the event ends
alternative image	file name	taken one second later
combined with	empty or key name(s)	indicates if the event is modified

Table 4. Example: Mouse scroll 28 steps southbound (down), not modified

timestamp	1623421318.394444
type	mouse
action	scroll
start X	1000
start Y	450
horizontal	0
vertical	-28
primary image	after_mouse_scroll.png
alternative image	after_after_mouse_scroll.png
combined with	*empty field*

Mouse double click

The mouse double click occurs when a mouse button is clicked and released twice in a very short time and in the same or a very near location.

The typical mouse double click like this:

Key word	Value	More info
timestamp	positive (decimal) number	usually an epoch number, e.g. 1623421318.394444
type	mouse	-
action	doubleclick	-
button	left, right, middle	indicates which button to press
start X	positive number	indicates the X coordinate of the click location
start Y	positive number	indicates the Y coordinate of the click location
primary image	file name	taken before the first click appears
alternative image	file name	taken after the event

Key word	Value	More info
combined with	empty or key name(s)	indicates if the event is modified

Table 5. Example: Left mouse button doubleclick at 1000, 450 (not modified)

timestamp	1623421318.394444
type	mouse
action	doubleclick
button	left
start X	1000
start Y	450
primary image	before_double_click.png
alternative image	after_double_click.png
combined with	*empty field*

Mouse drag

The mouse drag occurs when a mouse button is clicked and held, the mouse is moved and then the button is released.

The typical mouse drag like this:

Key word	Value	More info
timestamp	positive (decimal) number	usually an epoch number, e.g. 1623421318.394444
type	mouse	-
action	drag	-
button	left, right, middle	indicates which button to press
start X	positive number	indicates the X coordinate of the click location
start Y	positive number	indicates the Y coordinate of the click location
end X	positive number	indicates the X coordinate of the release location
end Y	positive number	indicates the Y coordinate of the release location
primary image	file name	taken at the moment when the drag ends
alternative image	file name	taken one second after the drag
combined with	empty or key name(s)	indicates if the event is modified

Table 6. Example: Left mouse button drag from 1000, 450 to 1500, 650

timestamp	1623421318.394444
type	mouse

action	drag
button	left
start X	1000
start Y	450
end X	1500
end Y	650
primary image	right_after_mouse_dragged.png
alternative image	somewhat_later_mouse_dragged.png
combined with	*empty field*

Key events

Anytime a key is pressed, **AutoCoconut** records a key event which is later categorized into one of the groups.

Key press

In **AutoCoconut** if an alpha-numeric key is pressed, it will be regarded as *typing a text* and not a single key **press**. These are only reserved for *special* keys.

Special key press

A *special key* press is recorded any time such a key is pressed. The event uses the following features.

Key word	Value	More info
timestamp	positive (decimal) number	usually an epoch number, e.g. 1623421318.394444
type	key	-
subtype	special	-
action	press	-
key	name of key	indicates which key is pressed
primary image	file name	taken one second after the key is pressed
alternative image	file name	taken in the moment of the press
combined with	empty or key name(s)	indicates if the event is modified

Table 7. Example: F5 key press (not modified)

timestamp	1623421318.394444
type	key
subtype	special

action	press
key	f5
primary image	after_f5_pressed.png
alternative image	when_f5_pressed_alt.png
combined with	*empty field*

Modifier press

A *modifier* press is recorded any time a modifier (see Modifiers) is pressed. Modifiers are mostly used with other keys so each time a modifier is pressed, **AutoCoconut** records a **key combination** even if the modifier is pressed alone. The behaviour of the modifiers also differs from the other keys because they need to be properly released using a **release event** in order to tell the application where to exit the modified context.

Key word	Value	More info
timestamp	positive (decimal) number	usually an epoch number, e.g. 1623421318.394444
type	key combination	-
subtype	modifier	-
action	press	-
key	name of key	indicates which key is pressed
primary image	file name	indicates the image file used for the primary image
alternative image	file name	indicates the image file used for the alternative image
combined with	empty or key name(s)	indicates if the event is modified

Table 8. Example: CTRL key pressed.

timestamp	1623421318.394444
type	key combination
subtype	modifier
action	press
key	ctrl
primary image	modifier_pressed.png
alternative image	modifier_pressed_alt.png
combined with	ctrl

Modifier release

A release is recorded any time a modifier (see Modifiers) is released. Releasing a modifier is a crucial event that is used to exit the modified event and close a key combination. Whenever you use a modifier press, you need to add the modified release as well as a rule of thumb.

Key word	Value	More info
timestamp	positive (decimal) number	usually an epoch number, e.g. 1623421318.394444
type	key	-
subtype	modifier	-
action	release	-
key	name of key	indicates which key is pressed
primary image	file name	taken one second after the modifier is released
alternative image	file name	taken at the moment of releasing the modifier
combined with	empty or key name(s)	indicates if the event is modified

Table 9. Example: CTRL key released.

timestamp	1623421318.394444
type	key
subtype	modifier
action	release
key	ctrl
primary image	after_modifier_released.png
alternative image	when_modifier_released.png
combined with	ctrl

Typing

Typing occurs any time an alpha-numeric key (or a sequence of these keys) is pressed. **AutoCoconut** keeps recording alpha-numeric keys into a typing buffer until a mouse event, a special key event, or a modifier event occurs, which terminates the typing sequence. The typing sequence is than added as a **single event** which you might add, edit, or delete.

Typing sequences do not have any screenshots assigned, because alpha-numeric keys do not take screenshots to save resources as there might be a lot of typing involved.

Key word	Value	More info
timestamp	positive (decimal) number	usually an epoch number, e.g. 1623421318.394444
type	typing	-
subtype	text	-
action	type	-

Key word	Value	More info
key	name of key	indicates which key was the last key of the typing sequence
text	string	indicates the typed in string
primary image	file name	*empty field*
alternative image	file name	*empty field*
combined with	empty or key name(s)	*empty field*

Table 10. Example: A dangerous command typed.

timestamp	1623421318.394444
type	typing
subtype	text
action	type
text	rm -rf *
primary image	-
alternative image	-
combined with	-

Key combo

A **key combo** is a situation when a modifier is pressed, then some other keys are pressed and then the modifier is released again. Most of the time, the modifiers are combined with alpha-numeric keys, but they also can be combined with special keys and mouse events (see Examples of Combined events).

The typical key combo has the following features:

Key word	Value	More info
timestamp	positive (decimal) number	usually an epoch number, e.g. 1623421318.394444
type	key combination	-
subtype	modifier	-
action	press	-
key	name of key	indicates which key is pressed
primary image	file name	taken after the combination was released
alternative image	file name	taken one second earlier than when released
combined with	empty or key name(s)	indicates if the event is modified

Table 11. Example: CTRL-ALT-K key combo pressed.

timestamp	1623421318.394444
type	key combination
subtype	modifier
action	press
key	k
primary image	after_key_combo_pressed.png
alternative image	when_key_combo_pressed_alt.png
combined with	ctrl-alt

NOTE Remember that any key combo event is followed by a modifier release event. The released modifier should be the very first modifier pressed in this combination.

Examples of Combined events

Combined evets are mouse events happening when one or more modifiers are pressed. To create or edit them, do as if you wanted to create a mouse event and indicate the pressed modifier in the **Combined with** field.

Modified click

Table 12. Example: Right mouse button click at 1000, 450 modified with CTRL

timestamp	1623421318.394444
type	mouse
action	click
button	right
start X	1000
start Y	450
primary image	click_start_button.png
alternative image	before_click_start_button.png
combined with	ctrl

Modified double click

Table 13. Example: Left mouse button doubleclick at 1000, 450 modified with ALT.

timestamp	1623421318.394444
type	mouse
action	doubleclick
button	left

start X	1000
start Y	450
primary image	double_clicked.png
alternative image	double_clicked_alt.png
combined with	alt

Modified scroll

Table 14. Example: Mouse scroll 28 steps southbound (down), modified with ALT-SHIFT

timestamp	1623421318.394444
type	mouse
action	scroll
start X	1000
start Y	450
horizontal	0
vertical	-28
primary image	mouse_scroll.png
alternative image	mouse_scroll_alt.png
combined with	alt-shift

Modified drag

Table 15. Example: Left mouse button drag from 1000, 450 to 1500, 650 combined with CTRL-ALT

timestamp	1623421318.394444
type	mouse
action	drag
button	left
start X	1000
start Y	450
end X	1500
end Y	650
primary image	mouse_dragged.png
alternative image	mouse_dragged_alt.png
combined with	ctrl-alt

Quick overview of event features

Feature	Possible values	Comment
type	`mouse`, `key`, `key combination`, `typing`	always required
subtype	`special`, `modifier`, `text`	required for key events
action	`click`, `drag`, `scroll`, `doubleclick` ,`press`, `type`, `release`	always required
button	`left`, `middle`, `right`	required for click, doubleclick and drag
key	see Keys	required for key events
text	string	required for typing events
vertical scroll	number	required for scrolls
horizontal scroll	number	required for scrolls
start X and Y	positive numbers	required for mouse events
end X and Y	positive numbers	required for mouse drags
combined with	modifier(s), see Modifiers	required for key combinations
primary and alternative image	filename	recommended for mouse events, special keys, key combinations

NOTE Do not use any of the features in an event when they are not explicitely required, except for screenshots.