ahk Documentation

Release 0.8.1

Spencer Phillip Young

Sep 09, 2022

CONTENTS:

GitHub

CHAPTER

ONE

QUICKSTART

This document assumes you have Python 3.6 or newer installed

1.1 Installing AHK

AHK requires the AutoHotkey software in addition to the Python package

1. Download and install AutoHotkey (1.1.x). It can be downloaded from the autohotkey website.
2. Install the Python `ahk` package

```
py -m pip install ahk
```

3. Write your first script:

```
from ahk import AHK
ahk = AHK()
ahk.run_script('Run Notepad')
notepad_window = ahk.find_window_by_title(b'Untitled - Notepad')
notepad_window.send('Hello World')
```

Run the script!

If you get an *ExecutableNotFoundError* it's because AutoHotkey was installed to a location that is not on PATH or the default location (`C:\Program Files\AutoHotkey\AutoHotkey.exe`). You can either place the executable on PATH, in the default location, or specify the location manually in code:

```
ahk = AHK(executable_path='C:\\Path\\To\\AutoHotkey.exe')
```

CHAPTER

TWO

API

This part of the documentation is intended for developers looking to contribute to this project or discover more about the programming interface. A lot of this is auto-generated documentation from the docstrings.

2.1 Autohotkey

class ahk.autohotkey.**AHK**(**args*, ***kwargs*)

Inherits its methods from the following classes:

ahk.window.WindowMixin
ahk.mouse.MouseMixin
ahk.keyboard.KeyboardMixin
ahk.screen.ScreenMixin
ahk.sound.SoundMixin
ahk.registery.RegisteryMixin
ahk.gui.GUIMixin

class ahk.autohotkey.**ActionChain**(**args*, ***kwargs*)

Reusable action chain to execute various actions in order

__init__(**args*, ***kwargs*)

Parameters

- **mouse_speed** – default mouse speed
- **mode** –
- **kwargs** –

perform()

run_script(**args*, ***kwargs*)

override run_script to defer to queue

sleep(*n*)

Parameters **n** – how long (in seconds) to sleep

Returns

class ahk.autohotkey.**AsyncAHK**(*mouse_speed=2*, *mode=None*, ***kwargs*)

2.2 Directives

Contains directive classes

class ahk.directives.**AllowSameLineComments**(***kwargs*)

class ahk.directives.**ClipboardTimeout**(*milliseconds=0*, ***kwargs*)

__init__(*milliseconds=0*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class ahk.directives.**Directive**(***kwargs*)
Simple directive class They are designed to be hashable and comparable with string equivalent of AHK directive. Directives that don't require arguments do not need to be instantiated.

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class ahk.directives.**DirectiveMeta**
Overrides __str__ so directives with no arguments can be used without instantiation Overrides __hash__ to make objects 'unique' based upon a hash of the str representation

class ahk.directives.**ErrorStdOut**(***kwargs*)

class ahk.directives.**HotKeyInterval**(*milliseconds=0*, ***kwargs*)

class ahk.directives.**HotKeyModifierTimeout**(*milliseconds=0*, ***kwargs*)

class ahk.directives.**Include**(*include_name*, ***kwargs*)

__init__(*include_name*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class ahk.directives.**IncludeAgain**(*include_name*, ***kwargs*)

class ahk.directives.**InputLevel**(*level*, ***kwargs*)

__init__(*level*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class ahk.directives.**InstallKeybdHook**(***kwargs*)

class ahk.directives.**InstallMouseHook**(***kwargs*)

class ahk.directives.**KeyHistory**(*limit=40*, ***kwargs*)

__init__(*limit=40*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class ahk.directives.**MaxHotkeysPerInterval**(*value*, ***kwargs*)

__init__(*value*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class ahk.directives.**MaxMem**(*megabytes*, ***kwargs*)

__init__(*megabytes*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class ahk.directives.**MaxThreads**

__init__()
Initialize self. See help(type(self)) for accurate signature.

class ahk.directives.**MaxThreadsBuffer**

__init__()
Initialize self. See help(type(self)) for accurate signature.

class ahk.directives.**MaxThreadsPerHotkey**

__init__()
Initialize self. See help(type(self)) for accurate signature.

class ahk.directives.**MenuMaskKey**

__init__()
Initialize self. See help(type(self)) for accurate signature.

class ahk.directives.**NoEnv**(***kwargs*)

class ahk.directives.**NoTrayIcon**(***kwargs*)

class ahk.directives.**Persistent**(***kwargs*)

class ahk.directives.**SingleInstance**(***kwargs*)

class ahk.directives.**UseHook**(***kwargs*)

class ahk.directives.**Warn**(***kwargs*)

class ahk.directives.**WinActivateForce**(***kwargs*)

2.3 Keyboard

class ahk.keyboard.**AsyncKeyboardMixin**(*executable_path=''*, *directives=None*, ***kwargs*)

async key_press(*key*, *release=True*, *blocking=True*)
Press and (optionally) release a single key

Parameters

- **key** –
- **release** –

Returns

async key_state(*key_name*, *mode=None*)

Return type bool

async key_wait(*key_name*, *timeout=None*, *logical_state=False*, *released=False*)
Wait for key to be pressed or released (default is pressed; specify `released=True` to wait for key release).

https://autohotkey.com/docs/commands/KeyWait.htm

Parameters

- **key_name** – The name of the key
- **timeout** (Optional[int]) – how long (in seconds) to wait for the key. If not specified, waits indefinitely
- **logical_state** – Check the logical state of the key, which is the state that the OS and the active window believe the key to be in (not necessarily the same as the physical state). This option is ignored for joystick buttons.
- **released** – Set to True to wait for the key to be released rather than pressed

Return type None

Returns None

Raises **TimeoutError** – if the key was not pressed (or released, if specified) within timeout

class ahk.keyboard.**Hotkey**(*engine*, *hotkey*, *script*)

__init__(*engine*, *hotkey*, *script*)

Parameters

- **engine** (*ScriptEngine*) – an AHK instance
- **hotkey** (str) – The hotkey to use (AutoHotkey syntax)
- **script** (str) – The script to execute when the hotkey is activated (AutoHotkey code as a string)

property running

start()
Starts an AutoHotkey process with the hotkey script

stop()
Stops the process if it is running

class ahk.keyboard.**KeyboardMixin**(*executable_path=''*, *directives=None*, ***kwargs*)

hotkey(**args*, ***kwargs*)
Convenience function for creating Hotkey instance using current engine.

Parameters

- **hotkey** – The hotkey to use (AutoHotkey syntax)
- **script** – The script to execute when the hotkey is activated (AutoHotkey code as a string)

Returns an *Hotkey* instance

key_down(*key*, *blocking=True*)
Press down a key (without releasing it)

Parameters

- **key** –
- **blocking** –

Returns

key_press(*key*, *release=True*, *blocking=True*)
Press and (optionally) release a single key

Parameters

- **key** –
- **release** –

Returns

key_release(*key*, *blocking=True*)
Release a key that is currently in pressed down state

Parameters **key** –

Returns

key_state(*key_name*, *mode=None*)

Return type `bool`

key_up(*key*, *blocking=True*)
Alias for :meth:~`KeyboardMixin.key_release`

key_wait(*key_name*, *timeout=None*, *logical_state=False*, *released=False*)
Wait for key to be pressed or released (default is pressed; specify `released=True` to wait for key release).

https://autohotkey.com/docs/commands/KeyWait.htm

Parameters

- **key_name** – The name of the key
- **timeout** (`Optional[int]`) – how long (in seconds) to wait for the key. If not specified, waits indefinitely
- **logical_state** – Check the logical state of the key, which is the state that the OS and the active window believe the key to be in (not necessarily the same as the physical state). This option is ignored for joystick buttons.
- **released** – Set to True to wait for the key to be released rather than pressed

Return type `None`

Returns None

Raises **TimeoutError** – if the key was not pressed (or released, if specified) within timeout

send(*s*, *raw=False*, *delay=None*, *blocking=True*)
https://autohotkey.com/docs/commands/Send.htm

Parameters

- **s** –
- **raw** –
- **delay** –
- **blocking** – if `True`, waits until script finishes, else returns immediately.

Returns

send_event(*s*, *delay=None*)
https://autohotkey.com/docs/commands/Send.htm

Parameters

- **s** –
- **delay** –

Returns

send_input(*s*, *blocking=True*)

send_play(*s*)
https://autohotkey.com/docs/commands/Send.htm

Parameters **s** –

Returns

send_raw(*s*, *delay=None*)
https://autohotkey.com/docs/commands/Send.htm

Parameters

- **s** –
- **delay** –

Returns

set_capslock_state(*state=None*)
https://www.autohotkey.com/docs/commands/SetNumScrollCapsLockState.htm

Sets capslock state

Parameters **state** (*str*) – the desired state ("on" or "off"). Can also be True/False. If omitted, toggles capslock state.

Returns

type(*s*, *blocking=True*)
Sends keystrokes using send_input, also escaping the string for use in AHK.

Parameters

- **s** – the string to type
- **blocking** – if `True`, waits until script finishes, else returns immediately.

2.4 Keys

The ahk.keys module contains some useful constants and classes for working with keys.

class ahk.keys.**KEYS**
KEYS constants REF: https://autohotkey.com/docs/KeyList.htm

ALT = KeyModifier(key_name='Alt')

Alt = KeyModifier(key_name='Alt')

BACKSPACE = Key(key_name='Backspace')

Backspace = Key(key_name='Backspace')

CAPS_LOCK = Key(key_name='CapsLock')

CONTROL = KeyModifier(key_name='Control')

```
CTRL = KeyModifier(key_name='Control')
CapsLock = Key(key_name='CapsLock')
Control = KeyModifier(key_name='Control')
Ctrl = KeyModifier(key_name='Control')
DEL = Key(key_name='Delete')
DELETE = Key(key_name='Delete')
DOWN = Key(key_name='Down')
Del = Key(key_name='Delete')
Delete = Key(key_name='Delete')
Down = Key(key_name='Down')
ENTER = Key(key_name='Enter')
ESCAPE = Key(key_name='Escape')
Enter = Key(key_name='Enter')
F1 = Key(key_name='F1')
F10 = Key(key_name='F10')
F11 = Key(key_name='F11')
F12 = Key(key_name='F12')
F13 = Key(key_name='F13')
F14 = Key(key_name='F14')
F15 = Key(key_name='F15')
F16 = Key(key_name='F16')
F17 = Key(key_name='F17')
F18 = Key(key_name='F18')
F19 = Key(key_name='F19')
F2 = Key(key_name='F2')
F20 = Key(key_name='F20')
F21 = Key(key_name='F21')
F22 = Key(key_name='F22')
F23 = Key(key_name='F23')
F24 = Key(key_name='F24')
F3 = Key(key_name='F3')
F4 = Key(key_name='F4')
F5 = Key(key_name='F5')
F6 = Key(key_name='F6')
F7 = Key(key_name='F7')
F8 = Key(key_name='F8')
```

```
F9 = Key(key_name='F9')
JOY1 = Key(key_name='Joy1')
JOY10 = Key(key_name='Joy10')
JOY11 = Key(key_name='Joy11')
JOY12 = Key(key_name='Joy12')
JOY13 = Key(key_name='Joy13')
JOY14 = Key(key_name='Joy14')
JOY15 = Key(key_name='Joy15')
JOY16 = Key(key_name='Joy16')
JOY17 = Key(key_name='Joy17')
JOY18 = Key(key_name='Joy18')
JOY19 = Key(key_name='Joy19')
JOY2 = Key(key_name='Joy2')
JOY20 = Key(key_name='Joy20')
JOY21 = Key(key_name='Joy21')
JOY22 = Key(key_name='Joy22')
JOY23 = Key(key_name='Joy23')
JOY24 = Key(key_name='Joy24')
JOY25 = Key(key_name='Joy25')
JOY26 = Key(key_name='Joy26')
JOY27 = Key(key_name='Joy27')
JOY28 = Key(key_name='Joy28')
JOY29 = Key(key_name='Joy29')
JOY3 = Key(key_name='Joy3')
JOY30 = Key(key_name='Joy30')
JOY31 = Key(key_name='Joy31')
JOY32 = Key(key_name='Joy32')
JOY4 = Key(key_name='Joy4')
JOY5 = Key(key_name='Joy5')
JOY6 = Key(key_name='Joy6')
JOY7 = Key(key_name='Joy7')
JOY8 = Key(key_name='Joy8')
JOY9 = Key(key_name='Joy9')
Joy1 = Key(key_name='Joy1')
Joy10 = Key(key_name='Joy10')
Joy11 = Key(key_name='Joy11')
```

```
Joy12 = Key(key_name='Joy12')
Joy13 = Key(key_name='Joy13')
Joy14 = Key(key_name='Joy14')
Joy15 = Key(key_name='Joy15')
Joy16 = Key(key_name='Joy16')
Joy17 = Key(key_name='Joy17')
Joy18 = Key(key_name='Joy18')
Joy19 = Key(key_name='Joy19')
Joy2 = Key(key_name='Joy2')
Joy20 = Key(key_name='Joy20')
Joy21 = Key(key_name='Joy21')
Joy22 = Key(key_name='Joy22')
Joy23 = Key(key_name='Joy23')
Joy24 = Key(key_name='Joy24')
Joy25 = Key(key_name='Joy25')
Joy26 = Key(key_name='Joy26')
Joy27 = Key(key_name='Joy27')
Joy28 = Key(key_name='Joy28')
Joy29 = Key(key_name='Joy29')
Joy3 = Key(key_name='Joy3')
Joy30 = Key(key_name='Joy30')
Joy31 = Key(key_name='Joy31')
Joy32 = Key(key_name='Joy32')
Joy4 = Key(key_name='Joy4')
Joy5 = Key(key_name='Joy5')
Joy6 = Key(key_name='Joy6')
Joy7 = Key(key_name='Joy7')
Joy8 = Key(key_name='Joy8')
Joy9 = Key(key_name='Joy9')
LAlt = KeyModifier(key_name='LAlt')
LControl = KeyModifier(key_name='LControl')
LCtrl = KeyModifier(key_name='LControl')
LEFT = Key(key_name='Left')
LEFT_ALT = KeyModifier(key_name='LAlt')
LEFT_CONTROL = KeyModifier(key_name='LControl')
LEFT_SHIFT = KeyModifier(key_name='LShift')
```

```
LEFT_WIN = KeyModifier(key_name='LWin')
LShift = KeyModifier(key_name='LShift')
LWin = KeyModifier(key_name='LWin')
Left = Key(key_name='Left')
NUMPAD0 = Key(key_name='Numpad0')
NUMPAD1 = Key(key_name='Numpad1')
NUMPAD2 = Key(key_name='Numpad2')
NUMPAD3 = Key(key_name='Numpad3')
NUMPAD4 = Key(key_name='Numpad4')
NUMPAD5 = Key(key_name='Numpad5')
NUMPAD6 = Key(key_name='Numpad6')
NUMPAD7 = Key(key_name='Numpad7')
NUMPAD8 = Key(key_name='Numpad8')
NUMPAD9 = Key(key_name='Numpad9')
NUMPAD_ADD = Key(key_name='NumpadAdd')
NUMPAD_DEL = Key(key_name='NumpadDel')
NUMPAD_DIV = Key(key_name='NumpadDiv')
NUMPAD_DOT = Key(key_name='NumpadDot')
NUMPAD_ENTER = Key(key_name='NumpadEnter')
NUMPAD_MULT = Key(key_name='NumpadMult')
NUMPAD_SUB = Key(key_name='NumpadSub')
NUM_LOCK = Key(key_name='NumLock')
NumLock = Key(key_name='NumLock')
Numpad0 = Key(key_name='Numpad0')
Numpad1 = Key(key_name='Numpad1')
Numpad2 = Key(key_name='Numpad2')
Numpad3 = Key(key_name='Numpad3')
Numpad4 = Key(key_name='Numpad4')
Numpad5 = Key(key_name='Numpad5')
Numpad6 = Key(key_name='Numpad6')
Numpad7 = Key(key_name='Numpad7')
Numpad8 = Key(key_name='Numpad8')
Numpad9 = Key(key_name='Numpad9')
NumpadAdd = Key(key_name='NumpadAdd')
NumpadDel = Key(key_name='NumpadDel')
NumpadDiv = Key(key_name='NumpadDiv')
```

```
NumpadDot = Key(key_name='NumpadDot')
NumpadEnter = Key(key_name='NumpadEnter')
NumpadMult = Key(key_name='NumpadMult')
NumpadSub = Key(key_name='NumpadSub')
RAlt = KeyModifier(key_name='RAlt')
RControl = KeyModifier(key_name='RControl')
RCtrl = KeyModifier(key_name='RControl')
RIGHT = Key(key_name='Right')
RIGHT_ALT = KeyModifier(key_name='RAlt')
RIGHT_CONTROL = KeyModifier(key_name='RControl')
RIGHT_SHIFT = KeyModifier(key_name='RShift')
RIGHT_WIN = KeyModifier(key_name='RWin')
RShift = KeyModifier(key_name='RShift')
RWin = KeyModifier(key_name='RWin')
Right = Key(key_name='Right')
SCROLL_LOCK = Key(key_name='ScrollLock')
SHIFT = KeyModifier(key_name='Shift')
SPACE = Key(key_name='Space')
ScrollLock = Key(key_name='ScrollLock')
Shift = KeyModifier(key_name='Shift')
TAB = Key(key_name='Tab')
Tab = Key(key_name='Tab')
UP = Key(key_name='Up')
Up = Key(key_name='Up')
WIN = KeyModifier(key_name='Win')
Win = KeyModifier(key_name='Win')
```

2.5 Mouse

class ahk.mouse.**AsyncMouseMixin**(*mouse_speed=2*, *mode=None*, ***kwargs*)

async get_mouse_position(*mode=None*)

property mouse_position

class ahk.mouse.**MouseMixin**(*mouse_speed=2*, *mode=None*, ***kwargs*)

Provides mouse functionality for the AHK class

__init__(*mouse_speed=2*, *mode=None*, ***kwargs*)

Parameters

- **mouse_speed** – default mouse speed
- **mode** –
- **kwargs** –

click(**args*, ***kwargs*)
Click mouse button at a specified position. REF: https://www.autohotkey.com/docs/commands/Click.htm

Parameters

- **x** –
- **y** –
- **button** –
- **n** – number of times to click the button
- **direction** –
- **relative** –
- **blocking** –
- **mode** –

Returns

double_click(**args*, ***kwargs*)
Convenience function to double click, equivalent to `click` with `n=2`

Parameters

- **args** –
- **kwargs** –

Returns

get_mouse_position(*mode=None*)

mouse_drag(**args*, ***kwargs*)
Click and drag the mouse

Parameters

- **x** –
- **y** –
- **from_position** – (x,y) tuple of an optional starting position. Current position is used if omitted
- **speed** –
- **button** – The button the click and drag; defaults to left mouse button
- **relative** – click and drag to a relative position rather than an absolute position
- **blocking** –
- **mode** –

Returns

mouse_move(**args*, ***kwargs*)
REF: https://www.autohotkey.com/docs/commands/MouseMove.htm

Parameters

- **x** – the x coordinate to move to. If omitted, current position is used
- **y** – the y coordinate to move to. If omitted, current position is used
- **speed** – 0 (fastest) to 100 (slowest). Can be a callable or string AHK expression
- **relative** – Move the mouse realtive to current position rather than absolute x,y coordinates
- **mode** –
- **blocking** –

Returns

property mouse_position

property mouse_speed

mouse_wheel(*direction*, **args*, ***kwargs*)

Convenience function for 'clicking' the mouse wheel in a given direction.

Parameters

- **direction** – the string 'up' or 'down'
- **args** – args passed to `click`
- **kwargs** – keyword args passed to `click`

Returns

right_click(**args*, ***kwargs*)

Convenience function clicking right mouse button. Equivalent to `click` with `button='R'`

Parameters

- **args** –
- **kwargs** –

Returns

wheel_down(**args*, ***kwargs*)

Convenience function for `click` with wheel down button

Parameters

- **args** –
- **kwargs** –

Returns

wheel_up(**args*, ***kwargs*)

Convenience function for `click` with wheel up button

Parameters

- **args** –
- **kwargs** –

Returns

ahk.mouse.**resolve_button**(*button*)

Resolve a string of a button name to a canonical name used for AHK script

Parameters **button** (*str*) –

Returns

2.6 Registery

2.7 Screen

class ahk.screen.**AsyncScreenMixin**(*executable_path=''*, *directives=None*, ***kwargs*)

async image_search(**args*, ***kwargs*)
AutoHotkey ImageSearch reference

Parameters

- **image_path** – path to the image file e.g. C:locationofcats.png
- **upper_bound** – a two-tuple of X,Y coordinates for the upper-left corner of the search area e.g. (200, 400) defaults to (0,0)
- **lower_bound** – like `upper_bound` but for the lower-righthand corner of the search area e.g. (400, 800) defaults to screen width and height (lower right-hand corner; `%A_ScreenWidth%`, `%A_ScreenHeight%`).
- **color_variation** – Shades of variation (up or down) for the intensity of RGB for each pixel. Equivalent of `*n` option. Defaults to 0.
- **coord_mode** – the Pixel CoordMode to use. Default is 'Screen'
- **scale_height** – Scale height in pixels. Equivalent of `*hn` option
- **scale_width** – Scale width in pixels. Equivalent of `*wn` option
- **transparent** – Specific color in the image that will be ignored during the search. Pixels with the exact color given will match any color. Can be used with color names e.g. (Black, Purple, Yellow) found at https://https://www.autohotkey.com/docs/commands/Progress.htm#colors or hexadecimal values e.g. (0xFFFFAA, 05FA15, 632511). Equivalent of `*TransN` option
- **icon** – Number of the icon group to use. Equivalent of `*Icon` option

Returns coordinates of the upper-left pixel of where the image was found on the screen; `None` if the image was not found

Return type Union[Tuple[int, int], None]

Note: when only scale_height or only scale_width are provided, aspect ratio is maintained by default.

async pixel_search(**args*, ***kwargs*)
AutoHotkey PixelSearch reference

Parameters

- **color** –
- **variation** –
- **upper_bound** –
- **lower_bound** –
- **coord_mode** –

- **fast** –
- **rgb** –

Returns the coordinates of the pixel; None if the pixel is not found

class ahk.screen.**ScreenMixin**(*executable_path=''*, *directives=None*, ***kwargs*)

image_search(*image_path*, *upper_bound=(0, 0)*, *lower_bound=None*, *color_variation=None*, *coord_mode='Screen'*, *scale_height=None*, *scale_width=None*, *transparent=None*, *icon=None*)

AutoHotkey ImageSearch reference

Parameters

- **image_path** (`str`) – path to the image file e.g. C:locationofcats.png
- **upper_bound** (`Tuple[int, int]`) – a two-tuple of X,Y coordinates for the upper-left corner of the search area e.g. (200, 400) defaults to (0,0)
- **lower_bound** (`Optional[Tuple[int, int]]`) – like `upper_bound` but for the lower-righthand corner of the search area e.g. (400, 800) defaults to screen width and height (lower right-hand corner; `%A_ScreenWidth%`, `%A_ScreenHeight%`).
- **color_variation** (`Optional[int]`) – Shades of variation (up or down) for the intensity of RGB for each pixel. Equivalent of `*n` option. Defaults to 0.
- **coord_mode** (`str`) – the Pixel CoordMode to use. Default is 'Screen'
- **scale_height** (`Optional[int]`) – Scale height in pixels. Equivalent of `*hn` option
- **scale_width** (`Optional[int]`) – Scale width in pixels. Equivalent of `*wn` option
- **transparent** (`Optional[str]`) – Specific color in the image that will be ignored during the search. Pixels with the exact color given will match any color. Can be used with color names e.g. (Black, Purple, Yellow) found at https://https://www.autohotkey.com/docs/commands/Progress.htm#colors or hexadecimal values e.g. (0xFFFFAA, 05FA15, 632511). Equivalent of `*TransN` option
- **icon** (`Optional[int]`) – Number of the icon group to use. Equivalent of `*Icon` option

Returns coordinates of the upper-left pixel of where the image was found on the screen; `None` if the image was not found

Return type Union[Tuple[int, int], None]

Note: when only scale_height or only scale_width are provided, aspect ratio is maintained by default.

pixel_get_color(**args*, ***kwargs*)

AutoHotkey PixelGetColor reference

Parameters

- **x** – x coordinate
- **y** – y coordinate
- **coord_mode** –
- **alt** –
- **slow** –
- **rgb** – returns

Returns the color as an RGB hexidecimal string;

pixel_search(**args*, ***kwargs*)
AutoHotkey PixelSearch reference

Parameters

- **color** –
- **variation** –
- **upper_bound** –
- **lower_bound** –
- **coord_mode** –
- **fast** –
- **rgb** –

Returns the coordinates of the pixel; None if the pixel is not found

2.8 The Script Engine

The *script* module, most essentially, houses the `ScriptEngine` class.

The `ScriptEngine` is responsible for rendering autohotkey code from jinja templates and executing that code. This is the heart of how this package works. Every other major component either inherits from this class or utilizes an instance of this class.

The current implementation of how autohotkey code is executed is by calling the autohotkey executable with `subprocess`.

`ahk.script.`**DEFAULT_EXECUTABLE_PATH = C:\Program Files\AutoHotkey\AutoHotkey.exe**
The deafult path to look for AutoHotkey, if not specified some other way

class `ahk.script.`**AsyncScriptEngine**(*executable_path=''*, *directives=None*, ***kwargs*)

async run_script(*script_text*, *decode=True*, *blocking=True*, ***runkwargs*)
async version of `run_script`

Parameters

- **script_text** (`str`) – a string containing AutoHotkey code
- **decode** – If `True`, attempt to decode the stdout of the completed process. If `False`, returns the completed process. Only has effect when `blocking=True`
- **blocking** – If `True`, script must finish before returning. If `False`, function returns a `asyncio.process` object immediately without blocking
- **runkwargs** – keyword arguments passed to `subprocess.Popen` or `subprocess.run`

Returns

A string of the decoded stdout if `blocking` and `decode` are True.
A bytes object of stdout if `blocking` is True and `decode` is False.
`asyncio.subprocess.Process` object if `blocking` is False.

ahk.script.**DEFAULT_EXECUTABLE_PATH = 'C:\\Program Files\\AutoHotkey\\AutoHotkey.exe'**
The deafult path to look for AutoHotkey, if not specified some other way

exception ahk.script.**ExecutableNotFoundError**

class ahk.script.**ScriptEngine**(*executable_path=''*, *directives=None*, ***kwargs*)

__init__(*executable_path=''*, *directives=None*, ***kwargs*)
This class is typically not used directly. AHK components inherit from this class and the arguments for this class should usually be passed in to AHK.

Parameters

- **executable_path** (str) – the path to the AHK executable. If not provided explicitly in this argument, the path to the AHK executable is resolved in the following order:
  - The AHK_PATH environment variable, if present
  - *DEFAULT_EXECUTABLE_PATH* if the file exists

  If environment variable not present, tries to look for 'AutoHotkey.exe' or 'AutoHotkeyA32.exe' with shutil.which
- **directives** (Optional[Set]) – a set of directives to apply to all generated AHK scripts

Raises **ExecutableNotFound** – if AHK executable cannot be found or the specified file does not exist

async a_run_script(*script_text*, *decode=True*, *blocking=True*, ***runkwargs*)
async version of run_script

Parameters

- **script_text** (str) – a string containing AutoHotkey code
- **decode** – If True, attempt to decode the stdout of the completed process. If False, returns the completed process. Only has effect when blocking=True
- **blocking** – If True, script must finish before returning. If False, function returns a asyncio.process object immediately without blocking
- **runkwargs** – keyword arguments passed to subprocess.Popen or subprocess.run

Returns

A string of the decoded stdout if blocking and decode are True.
A bytes object of stdout if blocking is True and decode is False.
asyncio.subprocess.Process object if blocking is False.

static escape_sequence_replace(**args*, ***kwargs*)

render_template(*template_name*, *directives=None*, *blocking=True*, ***kwargs*)
Renders a given jinja template and returns a string of script text

Parameters

- **template_name** – the name of the jinja template to render
- **directives** – additional AHK directives to add to the resulting script

- **blocking** – whether the template should be rendered to block (use #Persistent directive)
- **kwargs** – keywords passed to template rendering

Returns An AutoHotkey script as a string

```
>>> from ahk import AHK
>>> ahk = AHK()
>>> ahk.render_template('keyboard/send_input.ahk', s='Hello')
'#NoEnv\n#Persistent\n\n\nSendInput Hello\n\nExitApp\n'
```

run_script(*script_text*, *decode=True*, *blocking=True*, ***runkwargs*)

Given an AutoHotkey script as a string, execute it

Parameters

- **script_text** (str) – a string containing AutoHotkey code
- **decode** – If True, attempt to decode the stdout of the completed process. If False, returns the completed process. Only has effect when blocking=True
- **blocking** – If True, script must finish before returning. If False, function returns a subprocess.Popen object immediately without blocking
- **runkwargs** – keyword arguments passed to subprocess.Popen or subprocess.run

Returns

A string of the decoded stdout if blocking and decode are True.
subprocess.CompletedProcess if blocking is True and decode is False.
subprocess.Popen object if blocking is False.

```
>>> from ahk import AHK
>>> ahk = AHK()
>>> ahk.run_script('FileAppend, Hello World, *')
'Hello World'
>>> ahk.run_script('FileAppend, Hello World, *', decode=False)
CompletedProcess(args=['C:\\pathto\\AutoHotkey.exe', '/ErrorStdOut', '*'],␣
↪returncode=0, stdout=b'Hello World', stderr=b'')
>>> ahk.run_script('FileAppend, Hello World, *', blocking=False)
<subprocess.Popen at 0x18a599cde10>
```

2.9 Sound

class ahk.sound.**AsyncSoundMixin**(*executable_path=''*, *directives=None*, ***kwargs*)

class ahk.sound.**SoundMixin**(*executable_path=''*, *directives=None*, ***kwargs*)

get_volume(*device_number=1*)

REF: https://autohotkey.com/docs/commands/SoundGetWaveVolume.htm

Parameters **device_number** –

Returns

set_volume(*value*, *device_number=1*)
REF: https://autohotkey.com/docs/commands/SoundSetWaveVolume.htm

Parameters

- **value** – percent volume to set volume to
- **device_number** –

Returns

sound_beep(*frequency=523*, *duration=150*)
REF: https://autohotkey.com/docs/commands/SoundBeep.htm

Parameters

- **frequency** – number between 37 and 32767
- **duration** – how long in milliseconds to play the beep

Returns None

sound_get(*device_number=1*, *component_type='MASTER'*, *control_type='VOLUME'*)
REF: https://autohotkey.com/docs/commands/SoundGet.htm

Parameters

- **device_number** –
- **component_type** –
- **control_type** –

Returns

sound_play(*filename*, *blocking=True*)
REF: https://autohotkey.com/docs/commands/SoundPlay.htm

Parameters

- **filename** –
- **blocking** –
- **wait** –

Returns

sound_set(*value*, *device_number=1*, *component_type='MASTER'*, *control_type='VOLUME'*)
REF: https://autohotkey.com/docs/commands/SoundSet.htm

Parameters

- **value** –
- **device_number** –
- **component_type** –
- **control_type** –

Returns

2.10 Utils

ahk.utils.**async_filter**(*async_pred*, *iterable*)

ahk.utils.**escape_sequence_replace**(*s*)

Replace Python escape sequences with AHK equivalent escape sequences Additionally escapes some other characters for AHK escape sequences. Intended for use with AHK Send command functions.

Note: This DOES NOT provide ANY assurances against accidental or malicious injection. Does NOT escape quotes.

```
>>> escape_sequence_replace('Hello, World!')
'Hello`, World{!}'
```

ahk.utils.**make_logger**(*name*)

2.11 Window

class ahk.window.**AsyncWindow**(*engine*, *ahk_id*, *encoding=None*)

property always_on_top

Return type bool

async classmethod from_mouse_position(*engine*, ***kwargs*)

async classmethod from_pid(*engine*, *pid*, ***kwargs*)

async get_pos(*info=None*)

Returns

async get_transparency()

Return type int

property height

async is_always_on_top()

async is_maximized()

async is_minimized()

async is_minmax()

property non_max_non_min

property position

property rect

async set_transparency(*value*)

property title

property transparent

Return type int

property width

class ahk.window.**AsyncWindowMixin**(**args*, ***kwargs*)

async find_window(*func=None*, ***kwargs*)
Like `find_windows` but only returns the first found window

Parameters

- **func** –
- **kwargs** –

Returns a *Window* object or `None` if no matching window is found

async find_window_by_class(**args*, ***kwargs*)

Parameters

- **args** –
- **kwargs** –

Returns a *Window* object or `None` if no matching window is found

async find_window_by_text(**args*, ***kwargs*)

Parameters

- **args** –
- **kwargs** –

Returns a *Window* object or `None` if no matching window is found

async find_window_by_title(*title*)
Like `find_windows_by_title` but only returns the first result.

Returns a *Window* object or `None` if no matching window is found

find_windows(*func=None*, ***kwargs*)
Find all matching windows

Parameters

- **func** – a callable to filter windows
- **exact** (*bool*) – if False (the default) partial matches are found. If True, only exact matches are returned
- **kwargs** – keywords of attributes of the window (has no effect if `func` is provided)

Returns a generator containing any matching *Window* objects.

find_windows_by_class(*class_name*, *exact=False*)

Parameters

- **class_name** –
- **exact** –

Returns a generator containing any matching *Window* objects.

find_windows_by_text(*text*, *exact=False*)

Parameters

- **text** –
- **exact** –

Returns a generator containing any matching *Window* objects.

find_windows_by_title(*title*, *exact=False*)
Equivalent to `find_windows(title=title)`

Note that `title` is a `bytes` object

Parameters

- **title** (*bytes*) –
- **exact** –

Returns

async win_get(**args*, ***kwargs*)

async win_wait(*, *title=''*, *text=''*, *exclude_title=''*, *timeout=0.5*, *exclude_text=''*, *encoding=None*)
WinWait Wait for a window. If found within the timeout (in seconds), returns a Window object. If not found, raises a TimeoutError

ref: https://www.autohotkey.com/docs/commands/WinWait.htm

async windows()
Returns a list of windows

Returns

class ahk.window.**Control**

__init__()
Initialize self. See help(type(self)) for accurate signature.

click()
REF: https://www.autohotkey.com/docs/commands/ControlClick.htm

Returns

focus()
REF: https://www.autohotkey.com/docs/commands/ControlFocus.htm

Returns

get(*key*)
REF: https://www.autohotkey.com/docs/commands/ControlGet.htm

Parameters **key** –

Returns

has_focus()

move()
REF: https://www.autohotkey.com/docs/commands/ControlMove.htm

Returns

property position
REF: https://www.autohotkey.com/docs/commands/ControlGetPos.htm

Returns

send(*raw=False*)
REF: https://www.autohotkey.com/docs/commands/ControlSend.htm

Parameters **raw** –

Returns

property text
REF: https://www.autohotkey.com/docs/commands/ControlGetText.htm

Returns

class ahk.window.**Window**(*engine*, *ahk_id*, *encoding=None*)

MAXIMIZED = '1'

MINIMIZED = '-1'

NON_MIN_NON_MAX = '0'

__init__(*engine*, *ahk_id*, *encoding=None*)
Initialize self. See help(type(self)) for accurate signature.

activate()
Activate the window.

See also: WinActivate

Returns

activate_bottom()
Calls WinActivateBottom on the window

Returns

property active

property always_on_top

Return type bool

property class_name

click(**args*, ***kwargs*)
Click at an x/y location on the screen. Uses ControlClick https://autohotkey.com/docs/commands/ControlClick.htm

x/y position params may also be specified as a 2-item sequence

Parameters

- **x** – x offset relative to topleft corner of the window
- **y** – y offset relative to the top of the window
- **button** – the button to press (default is left mouse)
- **n** – number of times to click
- **options** – per ControlClick documentation
- **blocking** –

Returns

close(*seconds_to_wait=''*)
Closes the Window. See also: WinClose

Parameters **seconds_to_wait** –

Returns

disable()
Distable the window

Returns

enable()
Enable the window

Returns

property exist

exists()

classmethod from_mouse_position(*engine*, ***kwargs*)

classmethod from_pid(*engine*, *pid*, ***kwargs*)

get(*subcommand*)

get_class_name()

get_pos(*info=None*)

Returns

get_text()

get_title()

get_transparency()

Return type int

property height

hide()
Hides the window. See also: WinHide

Returns

is_active()

is_always_on_top()

Return type bool

is_maximized()

is_minimized()

is_minmax()

kill(*seconds_to_wait=''*)

maximize()
maximize the window

Returns

property maximized

minimize()
minimize the window

Returns

property minimized

move(*x=''*, *y=''*, *width=None*, *height=None*)
Move the window to a position and/or change its geometry

Parameters

- **x** –
- **y** –
- **width** –
- **height** –

Returns

property non_max_non_min

property position

property rect

redraw()

restore()
restore the window

Returns

send(*keys*, *delay=10*, *raw=False*, *blocking=True*, *escape=False*, *press_duration=-1*)
Send keystrokes directly to the window. Uses ControlSend https://autohotkey.com/docs/commands/Send.htm

set(*subcommand*, *value*)

set_always_on_top(*value*)

set_position(*new_position*)

set_title(*value*)

set_transparency(*value*)

show()
show the window

Returns

property text

property title

to_bottom()
Send window to bottom (behind other windows) :return:

to_top()
Bring the window to the foreground (above other windows)

Returns

property transparent

Return type int

wait_active(*seconds_to_wait=''*)

Parameters **seconds_to_wait** –

Returns

wait_close(*seconds_to_wait=''*)

Parameters **seconds_to_wait** –

Returns

wait_not_active(*seconds_to_wait=''*)

Parameters **seconds_to_wait** –

Returns

property width

class ahk.window.**WindowMixin**(**args*, ***kwargs*)

__init__(**args*, ***kwargs*)

This class is typically not used directly. AHK components inherit from this class and the arguments for this class should usually be passed in to `AHK`.

Parameters

- **executable_path** – the path to the AHK executable. If not provided explicitly in this argument, the path to the AHK executable is resolved in the following order:
  - The `AHK_PATH` environment variable, if present
  - *`DEFAULT_EXECUTABLE_PATH`* if the file exists

  If environment variable not present, tries to look for 'AutoHotkey.exe' or 'AutoHotkeyA32.exe' with shutil.which
- **directives** – a set of directives to apply to all generated AHK scripts

Raises **ExecutableNotFound** – if AHK executable cannot be found or the specified file does not exist

property active_window

find_window(*func=None*, ***kwargs*)

Like `find_windows` but only returns the first found window

Parameters

- **func** –
- **kwargs** –

Returns a *`Window`* object or `None` if no matching window is found

find_window_by_class(**args*, ***kwargs*)

Parameters

- **args** –
- **kwargs** –

Returns a *`Window`* object or `None` if no matching window is found

find_window_by_text(**args*, ***kwargs*)

Parameters

- **args** –
- **kwargs** –

Returns a *Window* object or `None` if no matching window is found

find_window_by_title(**args*, ***kwargs*)
Like `find_windows_by_title` but only returns the first result.

Returns a *Window* object or `None` if no matching window is found

find_windows(*func=None*, ***kwargs*)
Find all matching windows

Parameters

- **func** – a callable to filter windows
- **exact** (*bool*) – if False (the default) partial matches are found. If True, only exact matches are returned
- **kwargs** – keywords of attributes of the window (has no effect if `func` is provided)

Returns a generator containing any matching *Window* objects.

find_windows_by_class(*class_name*, *exact=False*)

Parameters

- **class_name** –
- **exact** –

Returns a generator containing any matching *Window* objects.

find_windows_by_text(*text*, *exact=False*)

Parameters

- **text** –
- **exact** –

Returns a generator containing any matching *Window* objects.

find_windows_by_title(*title*, *exact=False*)
Equivalent to `find_windows(title=title)` `

Note that `title` is a `bytes` object

Parameters

- **title** (*bytes*) –
- **exact** –

Returns

get_active_window()

win_get(*title=''*, *text=''*, *exclude_title=''*, *exclude_text=''*, *encoding=None*)

win_set(*subcommand*, **args*, *blocking=True*)

win_wait(***, *title=''*, *text=''*, *exclude_title=''*, *timeout=0.5*, *exclude_text=''*, *encoding=None*)
WinWait Wait for a window. If found within the timeout (in seconds), returns a Window object. If not found, raises a TimeoutError

ref: https://www.autohotkey.com/docs/commands/WinWait.htm

windows()
Returns a list of windows

Returns

exception ahk.window.**WindowNotFoundError**

CHAPTER

THREE

AHK

A Python wrapper around AHK.

CHAPTER

FOUR

INSTALLATION

```
pip install ahk
```

Requires Python 3.6+

Async API requires Python 3.8+

See also *Non-Python dependencies*

CHAPTER

FIVE

USAGE

```
from ahk import AHK

ahk = AHK()

ahk.mouse_move(x=100, y=100, blocking=True)  # Blocks until mouse finishes moving␣
↪(the default)
ahk.mouse_move(x=150, y=150, speed=10, blocking=True) # Moves the mouse to x, y␣
↪taking 'speed' seconds to move
print(ahk.mouse_position)  #  (150, 150)
```

CHAPTER

SIX

EXAMPLES

Non-exhaustive examples of some of the functions available with this package. Full documentation coming soon!

6.1 Mouse

```
from ahk import AHK

ahk = AHK()

ahk.mouse_position  # Returns a tuple of mouse coordinates (x, y)
ahk.mouse_move(100, 100, speed=10, relative=True)  # Moves the mouse reletave to the␣
↪current position
ahk.mouse_position = (100, 100)  # Moves the mouse instantly to absolute screen␣
↪position
ahk.click()  # Click the primary mouse button
ahk.double_click() # Clicks the primary mouse button twice
ahk.click(200, 200)  # Moves the mouse to a particular position and clicks
ahk.right_click() # Clicks the secondary mouse button
ahk.mouse_drag(100, 100, relative=True) # Holds down primary button and moves the␣
↪mouse
```

6.2 Keyboard

```
from ahk import AHK

ahk = AHK()

ahk.type('hello, world!')  # Send keys, as if typed (performs ahk string escapes)
ahk.send_input('Hello`, World{!}')  # Like AHK SendInput, must escape strings␣
↪yourself!
ahk.key_state('Control')  # Return True or False based on whether Control key is␣
↪pressed down
ahk.key_state('CapsLock', mode='T')  # Check toggle state of a key (like for NumLock,␣
↪CapsLock, etc)
ahk.key_press('a')  # Press and release a key
ahk.key_down('Control')  # Press down (but do not release) Control key
ahk.key_up('Control')  # Release the key
ahk.key_wait('a', timeout=3)  # Wait up to 3 seconds for the "a" key to be pressed.␣
↪NOTE: This throws
```

(continues on next page)

(continued from previous page)

```
                                # a TimeoutError if the key isn't pressed within the␣
↪timeout window
ahk.set_capslock_state("on")  # Turn CapsLock on
```

6.3 Windows

You can do stuff with windows, too.

6.3.1 Getting windows

```
from ahk import AHK
from ahk.window import Window

ahk = AHK()

win = ahk.active_window                          # Get the active window
win = ahk.win_get(title='Untitled - Notepad')    # by title
win = list(ahk.windows())                        # list of all windows
win = Window(ahk, ahk_id='0xabc123')             # by ahk_id
win = Window.from_mouse_position(ahk)            # the window under the mouse cursor
win = Window.from_pid(ahk, pid='20366')                  # by process ID

# Wait for a window
try:
    # wait up to 5 seconds for notepad
    win = ahk.win_wait(title='Untitled - Notepad', timeout=5)
except TimeoutError:
    print('Notepad was not found!')
```

6.3.2 Working with windows

```
from ahk import AHK

ahk = AHK()

ahk.run_script('Run Notepad') # Open notepad
win = ahk.find_window(title=b'Untitled - Notepad') # Find the opened window

win.send('hello')  # Send keys directly to the window (does not need focus!)
win.move(x=200, y=300, width=500, height=800)

win.activate()           # Give the window focus
win.activate_bottom()    # Give the window focus
win.close()              # Close the window
win.hide()               # Hide the windwow
win.kill()               # Kill the window
win.maximize()           # Maximize the window
win.minimize()           # Minimize the window
win.restore()            # Restore the window
win.show()               # Show the window
```

(continues on next page)

(continued from previous page)

```
win.disable()             # Make the window non-interactable
win.enable()              # Enable it again
win.to_top()              # Move the window on top of other windows
win.to_bottom()           # Move the window to the bottom of the other windows

win.always_on_top = True # Make the window always on top

for window in ahk.windows():
    print(window.title)

    # Some more attributes
    print(window.text)
    print(window.rect)    # (x, y, width, height)
    print(window.id)      # ahk_id
    print(window.pid)
    print(window.process)


if window.active:         # Check if window active
    window.minimize()

if window.exist:          # Check if window exist
    window.maximize()
```

6.4 Screen

```
from ahk import AHK

ahk = AHK()

ahk.image_search('C:\\path\\to\\image.jpg')  # Find an image on screen

# Find an image within a boundary on screen
ahk.image_search('C:\\path\\to\\image.jpg', upper_bound=(100, 100),  # upper-left␣
↪corner of search area
                                            lower_bound=(400, 400))  # lower-right␣
↪corner of search area
ahk.pixel_get_color(100, 100)  # Get color of pixel located at coords (100, 100)
ahk.pixel_search('0x9d6346')  # Get coords of the first pixel with specified color
```

6.5 Sound

```
from ahk import AHK

ahk = AHK()

ahk.sound_play('C:\\path\\to\\sound.wav')  # Play an audio file
ahk.sound_beep(frequency=440, duration=1000)  # Play a beep for 1 second (duration in␣
↪microseconds)
ahk.get_volume(device_number=1)  # Get volume of a device
ahk.set_volume(50, device_number=1)  # Set volume of a device
```

(continues on next page)

(continued from previous page)

```
ahk.sound_get(device_number=1, component_type='MASTER', control_type='VOLUME') # Get␣
↪sound device property
ahk.sound_set(50, device_number=1, component_type='MASTER', control_type='VOLUME') #␣
↪Set sound device property
```

6.6 GUI

```
from ahk import AHK

ahk = AHK()
ahk.show_tooltip("hello4", second=2, x=10, y=10)                                    #␣
↪ToolTip
ahk.show_info_traytip("Info", "It's also info", silent=False, blocking=True)   #␣
↪Default info traytip
ahk.show_warning_traytip("Warning", "It's warning")                                 #␣
↪Warning traytip
ahk.show_error_traytip("Error", "It's error")                                       # Error␣
↪trytip
```

6.7 non-blocking modes

For some functions, you can also opt for a non-blocking interface, so you can do other stuff while AHK scripts run.

```
import time

from ahk import AHK

ahk = AHK()

ahk.mouse_position = (200, 200)  # Moves the mouse instantly to the start position
start = time.time()
ahk.mouse_move(x=100, y=100, speed=30, blocking=False)
while True:  #  report mouse position while it moves
    t = round(time.time() - start, 4)
    position = ahk.mouse_position
    print(t, position)
    if position == (100, 100):
        break
```

You should see an output something like

```
0.032 (187, 187)
0.094 (173, 173)
0.137 (164, 164)
...
0.788 (100, 103)
0.831 (100, 101)
0.873 (100, 100)
```

6.8 Add directives

You can add directives that will be added to all generated scripts. For example, to prevent the AHK trayicon from appearing, you can add the NoTrayIcon directive.

```
from ahk import AHK
from ahk.directives import NoTrayIcon

ahk = AHK(directives=[NoTrayIcon])
```

By default, some directives are automatically added to ensure functionality and are merged with any user-provided directives.

6.9 Run arbitrary AutoHotkey scripts

```
from ahk import AHK

ahk = AHK()

ahk_script = 'Run Notepad'
ahk.run_script(ahk_script, blocking=False)
```

6.9.1 Communicating data from ahk to Python

If you're writing your own ahk scripts to use with this library, you can use `FileAppend` with the `*` parameter to get data from your ahk script into Python.

Suppose you have a script like so

```
#Persistent
data := "Hello Data!"
FileAppend, %data%, *, UTF-8 ; send data var to stdout
ExitApp
```

```
result = ahk.run_script(my_script)
print(result)  # Hello Data!
```

If your autohotkey returns something that can't be decoded, add the keyword argument `decode=False` in which case you'll get back a `CompletedProcess` object where stdout (and stderr) will be bytes and you can handle it however you choose.

```
result = ahk.run_script(my_script, decode=False)
print(result.stdout)  # b'Hello Data!'
```

6.10 Preview features

Preview features are experimental features that are may not be fully functional. These features are (even more) likely to have breaking changes without warning.

Github issues are provided for convenience to collect feedback on these features.

6.11 AHKDaemon

Normally, `AHK` works by creating a new subprocess for every command invocation. Because processes are expensive to create in Windows, this can lead to performance issues for some use-cases. `AHKDaemon` allows all AHK commands to be carried out in a single process, as opposed to running each command in a new subprocess, improving performance.

Some other details change in Daemon mode, such as persistence of state (e.g. changes to CoordMode).

```
from ahk.daemon import AHKDaemon
daemon = AHKDaemon()
daemon.start()
daemon.mouse_move(100, 100)
```

For the most part, the AHK Daemon works just like the regular `AHK` class, with a few caveats. Most notably, AHKDaemon does not allow you to run arbitrary AutoHotkey scripts and does not yet support Hotkeys. However, you can always use the normal `AHK` class alongside the daemon for these needs.

`AsyncAHKDaemon` is also available for asyncio support.

In the future, AHKDaemon may become the default implementation.

6.12 Async API

An async API is provided so functions can be called using `async`/`await`. All the same methods from the synchronous API are available in the async API.

```
from ahk import AsyncAHK
import asyncio
ahk = AsyncAHK()

async def main():
    await ahk.mouse_move(100, 100)
    x, y = await ahk.get_mouse_position()
    print(x, y)

asyncio.run(main())
```

For the most part, the async API is identical to that of the normal API, with a few exceptions:

While properties (like `.mouse_position` or `.title` for windows) can be `await`ed, additional methods (like `get_mouse_position()` and `get_title()`) have been added for a more intuitive API.

Property setters have different (probably undesired) behavior in the async API. Instead, you should use a comparable method. If you *do* use the property setters, the invocation is created using `asyncio.create_task()`, which means that the task won't run until control is yielded back to the event loop. For now, this will also raise a warning to the same.

Lastly, while it's possible to pass `blocking=False` in the async API, this sometimes will cause problems with certain functions. For now, a warning is raised in this case.

```
ahk = AsyncAHK()
async def main():
    pos = ahk.mouse_position  # BAD! Does not work!
    pos = await ahk.mouse_position # OK. Works, but looks kind of weird
    pos = await ahk.get_mouse_position() # GOOD!

    # BAD: You probably don't want to do this
    ahk.mouse_position = (100, 100) # won't do anything right away. Raises warning
    print(await ahk.get_mouse_position()) # probably won't be 100,100

    # GOOD: Instead, do this:
    await ahk.mouse_move(100, 100, speed=0)
    assert await ahk.get_mouse_position() == (100, 100)
```

6.12.1 Hotkeys

GH-9

Hotkeys now have a primitive implementation. You give it a hotkey (a string the same as in an ahk script, without the `::`) and the body of an AHK script to execute as a response to the hotkey.

Right now, only AHK code is supported as callbacks for hotkeys. Support for Python callbacks via the Async API is planned.

```
from ahk import AHK, Hotkey

ahk = AHK()

key_combo = '#n' # Define an AutoHotkey key combonation
script = 'Run Notepad' # Define an ahk script
hotkey = Hotkey(ahk, key_combo, script) # Create Hotkey
hotkey.start()  #  Start listening for hotkey
```

At this point, the hotkey is active. If you press + , the script `Run Notepad` will execute.

There is no need to add `return` to the provided script, as it is provided by the template.

To stop the hotkey call the `stop()` method.

```
hotkey.stop()
```

See also the relevant AHK documentation

6.12.2 ActionChain

GH-25

`ActionChains` let you define a set of actions to be performed in order at a later time.

They work just like the `AHK` class, except the actions are deferred until the `perform` method is called.

An additional method `sleep` is provided to allow for waiting between actions.

```
from ahk import ActionChain

ac = ActionChain()

# An Action Chain doesn't perform the actions until perform() is called on the chain

ac.mouse_move(100, 100, speed=10)  # nothing yet
ac.sleep(1)  # still nothing happening
ac.mouse_move(500, 500, speed=10)  # not yet
ac.perform()  # *now* each of the actions run in order
```

Just like anywhere else, scripts running simultaneously may conflict with one another, so using blocking interfaces is generally recommended. Currently, there is limited support for interacting with windows in actionchains, you may want to use `win_set`)

6.12.3 find_window/find_windows methods

GH-26

Right now, these are implemented by iterating over all window handles and filtering with Python. They may be optimized in the future.

`AHK.find_windows` returns a generator filtering results based on attributes provided as keyword arguments. `AHK.find_window` is similar, but returns the first matching window instead of all matching windows.

There are couple convenience functions, but not sure if these will stay around or maybe we'll add more, depending on feedback.

- find_windows_by_title
- find_window_by_title
- find_windows_by_text
- find_window_by_text

6.13 Errors and Debugging

You can enable debug logging, which will output script text before execution, and some other potentially useful debugging information.

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

(See the logging module documentation for more information)

Also note that, for now, errors with running AHK scripts will often pass silently. In the future, better error handling will be added.

6.14 Non-Python dependencies

To use this package, you need the AutoHotkey executable. It's expected to be on PATH by default.

A convenient way to do this is to install the `binary` extra (requires version 0.13 or higher of this package)

```
pip install "ahk[binary]"
```

For versions < 0.13 you can install the ahk-binary package directly:

```
pip install "ahk-binary<2"
```

You can also use the `AHK_PATH` environment variable to specify the executable location.

```
set AHK_PATH=C:\Path\To\AutoHotkey.exe
```

Alternatively, you may provide the path in code

```
from ahk import AHK

ahk = AHK(executable_path='C:\\path\\to\\AutoHotkey.exe')
```

CHAPTER

SEVEN

CONTRIBUTING

All contributions are welcomed and appreciated.

Please feel free to open a GitHub issue or PR for feedback, ideas, feature requests or questions.

There's still some work to be done in the way of implementation. The ideal interfaces are still yet to be determined and *your* help would be invaluable.

The vision is to provide access to the most useful features of the AutoHotkey API in a Pythonic way.

CHAPTER

EIGHT

SIMILAR PROJECTS

These are some similar projects that are commonly used for automation with Python.

- Pyautogui - Al Sweigart's creation for cross-platform automation
- Pywinauto - Automation on Windows platforms with Python.
- keyboard - Pure Python cross-platform keyboard hooks/control and hotkeys!
- mouse - From the creators of `keyboard`, Pure Python *mouse* control!
- pynput - Keyboard and mouse control

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

P

R

S

T

U

W