htmlPy Documentation

Release 2.0.0

Amol Mandhane

December 14, 2016

Contents

HTML5-CSS3-Javascript based GUI libary in Python

htmlPy is a wrapper around PySide's QtWebKit library. It helps with creating beautiful GUIs using **HTML5, CSS3 and Javascript** for standalone Python applications. It is built on Qt which makes it highly **customizable and cross-platform**. htmlPy is compatible with both **Python2 and Python3**. It can be used with any python library or environment like django, flask, scipy, virtualenv etc. You can use front-end libraries and frameworks like bootstrap, jQuery, jQuery UI etc. and create GUIs for your applications in no time.

To start using htmlPy, please read *installation instructions*.

You can get the source at GitHub repository. Versions older than 2.0.0 are not recommended for use in production.

Contents:

CHAPTER 1

Installation

1.1 Installing htmlPy

Note: htmlPy is dependent on PySide which is not included in dependencies. Please refer the installation instructions for PySide.

You can install htmlPy with **pip**:

```
$ [sudo] pip install htmlPy
```

Or with **easy_install**:

```
$ [sudo] easy_install htmlPy
```

Or download the compressed archive from PyPI, extract it, and inside it, run:

```
$ [sudo] python setup.py install
```

Note: Superuser access may be required if you are trying to install htmlPy globally. Please use **sudo** before the above commands in such case.

1.2 Installing PySide

Note: For detailed installation instructions, refer PySide documentation.

htmlPy is dependent on PySide which is not included in the dependencies of htmlPy installation. PySide has to be installed manually. Following are the ways to do this.

1. **On Windows** PySide can be installed with **pip**:

   ```
   $ pip install PySide
   ```

   Or with **easy_install**:

   ```
   $ easy_install PySide
   ```

2. On Mac OS X

   You need to install or build Qt 4.8 first. You can use Homebrew and install Qt with:

```
$ brew install qt
```

After this, PySide can be installed with **pip**:

```
$ pip install PySide
```

Or with **easy_install**:

```
$ easy_install PySide
```

2. **On Linux (refer this for detailed instructions)** Installing PySide on linux with python installers is very slow as it requires compiling PySide from scratch. Hence, it is not included in the dependencies.

   A faster way is to use package managers of the operating system. On Ubuntu, you can install it with:

   ```
   $ sudo apt-get install python-pyside
   ```

   or:

   ```
   $ sudo apt-get install python3-pyside
   ```

   depending on the python version.

   If you are installing in a virtualenv or if you want to build PySide from scratch, you can install it with python installers. (This will take quite some time to compile):

   ```
   $ [sudo] pip install PySide
   $ [sudo] easy_install PySide
   ```

   Or refer to this link.

CHAPTER 2

Quickstart tutorial of htmlPy

2.1 Standalone application

Following code assumes that there is a file `index.html` in the same directory as the file and the script is being executed fromt the same directory.

```
import htmlPy
import os

app = htmlPy.AppGUI(title=u"htmlPy Quickstart", maximized=True)

app.template_path = os.path.abspath(".")
app.static_path = os.path.abspath(".")

app.template = ("index.html", {"username": "htmlPy_user"})

app.start()
```

The above code instantiates an application, renders `index.html` file in that application using Jinja2 templating engine and starts it.

2.2 Web based application

Following code is a web-based application displaying Python website. You can change the URL to a local or network server to display any website as an application.

```
import htmlPy

web_app = htmlPy.WebAppGUI(title=u"Python Website", maximized=True)
web_app.url = u"http://python.org/"

web_app.start()
```

CHAPTER 3

Important instructions for application development with htmlPy

3.1 Use a driver file

Keep your application modularized. Use a separate file for initialization, configuration and execution of htmlPy GUI. Do not include any back-end functionalities in this file. The driver file structure should be

1. Initial configurations
2. htmlPy GUI initialization
3. htmlPy GUI configuration
4. **Binding of back-end functionalities with GUI**
    (a) Import back-end functionalities
    (b) Bind imported functionalities
5. Instructions for running front-end in `if __name__ == "__main__":` conditional. **Always keep the GUI starter code in the "if __name__ == "__main__":" conditional**. The GUI has to be started only when the driver file is running, not when it is being imported

Here's a sample driver file

```
import os
import htmlPy
from PyQt4 import QtGui


# Initial confiurations
BASE_DIR = os.path.abspath(os.path.dirname(__file__))


# GUI initializations
app = htmlPy.AppGUI(title=u"Application", maximized=True, plugins=True)


# GUI configurations
app.static_path = os.path.join(BASE_DIR, "static/")
app.template_path = os.path.join(BASE_DIR, "templates/")

app.web_app.setMinimumWidth(1024)
app.web_app.setMinimumHeight(768)
app.window.setWindowIcon(QtGui.QIcon(BASE_DIR + "/static/img/icon.png"))
```

```
# Binding of back-end functionalities with GUI

# Import back-end functionalities
from html_to_python import ClassName

# Register back-end functionalities
app.bind(ClassName())


# Instructions for running application
if __name__ == "__main__":
    # The driver file will have to be imported everywhere in back-end.
    # So, always keep app.start() in if __name__ == "__main__" conditional
    app.start()
```

3.2 Set `static_path` and `template_path`

When using `htmlPy.AppGUI`, always set `static_path` and `template_path` right after instantiating GUI. Set `BASE_DIR` variable as the absolute path to the directory of the driver file and set `static_path` and `template_path` with respect to `BASE_DIR`. Refer to the driver file section for example.

htmlPy uses Jinja2 for templating which is inspired by Django's templating system but extends with powerful tools. Jinja2 requires a base directory to be set. This can be done by setting the `template_path` as displayed in driver file section. You can use your own templating system as `htmlPy.AppGUI` allows setting the attribute `html`.

`static_path` is where all the static files including images, stylesheets and javascripts are stored. The static files will have to be present on the user computer and using htmlPy static filter, their links can be generated dynamically. For example,

```
<script src="{{ 'js/jquery.min.js'|staticfile }}"></script>
<link rel="stylesheet" href="{{ 'css/bootstrap.min.css'|staticfile }}">
```

CHAPTER 4

Tutorials for common tasks

Following are some basic instructions for performing most common GUI and Python tasks with htmlPy.

4.1 GUI to Python calls

These calls work only for `htmlPy.AppGUI` applications.

An essential aspect of GUI is to attach back-end calls to GUI events. htmlPy needs the corresponding back-end functions to be selectively exposed to GUI. The calls from GUI can be done in very HTML way.

The back-end functions that have to be attached to GUI events are defined as follows

```
import htmlPy
import json
from sample_app import app as htmlPy_app


class ClassName(htmlPy.Object):
    # GUI callable functions have to be inside a class.
    # The class should be inherited from htmlPy.Object.

    def __init__(self):
        super(ClassName. self).__init__()
        # Initialize the class here, if required.
        return

    @htmlPy.Slot()
    def function_name(self):
        # This is the function exposed to GUI events.
        # You can change app HTML from here.
        # Or, you can do pretty much any python from here.
        #
        # NOTE: @htmlPy.Slot decorater needs argument and return data-types.
        # Refer to API documentation.
        return

    @htmlPy.Slot(str, result=str)
    def form_function_name(self, json_data):
        # @htmlPy.Slot(arg1_type, arg2_type, ..., result=return_type)
        # This function can be used for GUI forms.
        #
        form_data = json.loads(json_data)
```

```
        return json.dumps(form_data)

    @htmlPy.Slot()
    def javascript_function(self):
        # Any function decorated with @htmlPy.Slot decorater can be called
        # using javascript in GUI
        return


## You have to bind the class instance to the AppGUI instance to be
## callable from GUI
htmlPy_app.bind(ClassName())
```

After exposing the class methods to GUI, they can be called from HTML as follows

```
<a href="ClassName.function_name" id="link" data-bind="true">GUI clickable link</a>
<!-- The "a" tag needs to have unique ID and data-bind attribute set to "true"
The ClassName and function_name have to be set in href attribute as displayed above.
The "a" tag can be styled using CSS with other HTML elements inside it -->


<form action="ClassName.form_function_name" id="form" data-bind="true">
    <input type="text" id="form_input" name="name">
    <input type="submit" value="Submit" id="form_submit">
</form>
<!-- The "form" tag needs to have unique ID and data-bind attribute set to "true".
The ClassName and form_function_name have to be set in action attribute as
displayed above. The argument given to ClassName.form_function_name on form submit
will be a json string of the form data. -->

<script>
ClassName.javascript_function();
// You can treat the class inherited from htmlPy.Object (in this case, ClassName)
// as a javascript object.
</script>
```

4.2 Python to GUI calls

```
from sample_app import app
# app imported from sample_app file is an instance of htmlPy.AppGUI class.


## Change HTML of the app
app.html = u"<html></html>"

## Change HTML of the app using Jinja2 templates
app.template = ("./index.html", {"template_variable_name": "value"})

## Execute javascript on currently displayed HTML in the app
app.evaluate_javascript("alert('Hello from back-end')")
```

4.3 General structure of htmlPy applications

Following should be a general directory structure for htmlPy applications

```
back_end_codes/
static/
    css/
        style.css
        .
        .
        .
    js/
        script.js
        .
        .
        .
    img/
        logo.img
        .
        .
        .
templates/
    first_template_directory/
        template1.html
        template2.html
        .
        .
    another_template_directory/
        another_template.html
    base_layout.html
main.py
```

`main.py` is the driver file for the applications. In the driver file, you should initialize GUI, import functionalities from `back_end_codes` and bind it to GUI as explained on the first section of this page. Refer this section for sample driver file.

In the `back_end_codes`, you can import the `app` from `main.py` and perform operations on in as explained in this section.

4.4 Integration with django

Django can be used for standalone application development using htmlPy. The integration can be done easily. In the previous section, the django application and projects can be kept in `back_end_codes` directory. In the GUI driver file, include this code before initializing GUI for loading django settings.

```
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "<project_name>.settings")
```

Note: **TODO**: Add sample application developed using django and htmlPy

4.5 Using file input

htmlPy replaces the HTML file input code with PyQt's file dialog. To use file input, write the file input tag with an additional data attribute for filtering extensions, if required. The filter attribute string should be of the form "`[{'title': 'title for extension', 'extensions': 'space separated extensions'}, {'title': 'title for another extension', 'extensions': 'space separated extensions'}]`". For example

```
<input type="file" name="file" id="file" data-filter="[{'title': 'Images', 'extensions': '*.png *.xpm
```

The json returned to the python backend for the form will have the absolute path of the selected file as the input value.

CHAPTER 5

API Reference

5.1 Class htmlPy.AppGUI (htmlPy.BaseGUI)

class htmlPy.**AppGUI**(**args*, ***kwargs*)

GUI class for creating apps using PySide's QtWebkit.

The class AppGUI can be used to create standalone applications with HTML GUI. It uses Jinja2 templating engine for generating HTML which can be overridden.

Note: Arguments and Attributes of this class come from the parent class htmlPy.BaseGUI. Please refer to its documentation for more details.

Keyword Arguments

- **title** (*Optional[unicode]*) – The title of the window. Defaults to u"Application".
- **width** (*Optional[int]*) – Width of the window in pixels. Defaults to 800 px. Redundant if maximized is True.
- **height** (*Optional[int]*) – Height of the window in pixels. Defaults to 600 px. Redundant if maximized is True.
- **x_pos** (*Optional[int]*) – The X-coordinate for top-left corner of the window in pixels. Defaults to 10 px. Redundant if maximized is True.
- **y_pos** (*Optional[int]*) – The Y-coordinate for top-left corner of the window in pixels. Defaults to 10 px. Redundant if maximized is True.
- **maximized** (*Optional[bool]*) – window is maximized when set to True. Defaults to False.
- **plugins** (*Optional[bool]*) – Enables plugins like flash when set as True. Defaults to False.
- **developer_mode** (*Optional[bool]*) – Enables developer mode when set as True. Defaults to False. The developer mode gives access to web inspector and other development tools and enables right-click on the webpage.
- **allow_overwrite** (*Optional[bool]*) – PySide.QtGui.QApplication can be instantiated only once. If it is already instantiated, then setting allow_overwrite to True overwrites the QApplication's window with window of this class instance. If False, RuntimeError is raised. If QApplication is not instantiated, this is irrelevent.

app
PySide.QtGui.QApplication – The singleton Qt application object. This can be instantiated only once in the entire process.

window
PySide.QtGui.QMainWindow – The window being displayed in the `app`.

web_app
PySide.QtWebKit.QWebView – The web view widget which renders and displays HTML in the a `window`.

html
unicode property – The HTML currently rendered in the `web_app`. The HTML in `web_app` can be changed by assigning the new HTML to this property.

static_path
str property – The absolute path relative to which the `staticfile` filter will create links in templating. Changing this creates a function dynamically which replaces current `staticfile` filter in current templating environment.

template_path
str property – The absolute path relative to which jinja2 finds the templates to be rendered. Changing this updates the template loader in current templating environment.

template
tuple(str, dict – The current template being displayed in `web_app`. First element of the tuple is the path of the template file relative to `template_path`. The second element of the tuple is the context dictionary in which it is being rendered.

maximized
bool property – A boolean which describes whether the `window` is maximized or not. Can be set to `True` to maximize the window and set to `False` to restore.

width
int property – Width of the `window` in pixels. Set the value of this property in pixels to change the width.

height
int property – Height of the `window` in pixels. Set the value of this property in pixels to change the height.

x_pos
int property – The X-coordinate for top-left corner of the `window` in pixels. Set the value of this property in pixels to move the `window` horizontally.

y_pos
int property – The Y-coordinate for top-left corner of the `window` in pixels. Set the value of this property in pixels to move the `window` vertically.

title
unicode property – The title of the `window`. Set the value of this property to change the title.

plugins
bool property – A boolean flag which indicates whether plugins like flash are enabled or not. Set the value to `True` or `False` as required.

developer_mode
bool property – A boolean flag which indicated whether developer mode is active or not. The developer mode gives access to web inspector and other development tools and enables right-click on the webpage. Set the value to `True` or `False` as required.

Raises `RuntimeError` – If `PySide.QtGui.QApplication` is already instantiated and `allow_overwrite` is `False`.

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__
x.__hash__() <==> hash(x)

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
x.__repr__() <==> repr(x)

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

__str__
x.__str__() <==> str(x)

auto_resize()
Resizes and relocates the `window` to previous state

If the `window` is not maximized, this function resizes it to the stored dimensions, moves it to the stored location.

bind(*signal_object*, *variable_name=None*)
Binds an object to be called from GUI javascript.

This function binds an object to the javascript window of the page. The `signal_object` should be inherited from `htmlPy.Object`. The methods that should be callable from javascript should be decorated with `htmlPy.Slot`. A variable name can be supplied which will be the name of the variable in javascript corresponding to that object. Otherwise, name of the class of that object will be used as the variable name

Parameters **signal_object** (*htmlPy.Object*) – The object that has to be bound to GUI javascript.

Keyword Arguments **variable_name** (*str*) – The name of the javascript variable the object should be attached to. Defaults to None. If None, `signal_object.__class__.__name__` is used.

Raises

- `TypeError` – If `signal_object` is not of type `htmlPy.Object`.
- `NameError` – If `variable_name` is "GUIHelper" or name of the class of `signal_object` is "GUIHelper"

evaluate_javascript(*javascript_string*)
Evaluates javascript in web page currently displayed

Parameters **javascript_string** (*str*) – The string of javascript code that has to be evaluated.

execute()
Executes the application without ending the process on its end.

DO NOT execute this process directly. Use only when `htmlPy.BaseGUI.stop()` is connected to some signal.

right_click_setting(*value*)
Javascript based setting for right click on the application.

This function changes the web page's behaviour on right click. Normal behaviour is to open a context menu. Enabling right click exhibits that behaviour. Right click is enabled by default. Disabling right click suppresses context menu for entire page. Enabling right click for only inputs suppresses context menu for all elements excepts inputs and textarea, which is the recommended option. The arguments provided should be from `htmlPy.settings` module as explained further.

Parameters value (*int*) – should be either `htmlPy.settings.ENABLE` (default) or `htmlPy.settings.DISABLE` or `htmlPy.settings.INPUTS_ONLY` (recommended)

start()
Starts the application.

This is not asynchronous. Starting the application will halt the further processes. DO NOT start outside the `if __name__ == "__main__":` conditional

stop()
Stops the application. Use only to bind with signals.

The Qt application does not have to be manually stopped. Also, after starting the application is stuck in the execution loop and will not go further until it is stopped. Calling this function manually is redundant. This function exits only to be binded with QSignals to stop the application when that signal is emitted.

template
tuple(str, dict) – The current template being displayed in `web_app`. First element of the tuple is the path of the template file relative to `template_path`. The second element of the tuple is the context dictionary in which it is being rendered.

text_selection_setting(*value*)
Javascript based setting for text selection in the application.

This function changes the web page's behaviour on selection of text. Normal behaviour is to highlight the selected text. Enabling text selection exhibits that behaviour. Text selection is enabled by default. Disabling text selection disallows user to select any text on the page except for inputs and disables the I-beam cursor for text selection. The arguments provided should be from `htmlPy.settings` module as explained further.

Parameters value (*int*) – should be either `htmlPy.settings.ENABLE` (default) or `htmlPy.settings.DISABLE` or

5.2 Class htmlPy.WebAppGUI (htmlPy.BaseGUI)

class htmlPy.**WebAppGUI**(**args*, ***kwargs*)
GUI class for creating web apps using PySide's Qt.

The class WebAppGUI can be used to create web based applications in a QtWebKit based browser running on user side. The server for the web app can be remote or local. This can be used for quick desktop deployment of existing websites. However, for a standalone application, it is recommended to used `htmlPy.AppGUI` class.

Note: Arguments and Attributes of this class come from the parent class `htmlPy.BaseGUI`. Please refer to its documentation for more details.

Keyword Arguments

- **title** (*Optional[unicode]*) – The title of the `window`. Defaults to u"Application".
- **width** (*Optional[int]*) – Width of the `window` in pixels. Defaults to 800 px. Redundant if `maximized` is `True`.
- **height** (*Optional[int]*) – Height of the `window` in pixels. Defaults to 600 px. Redundant if `maximized` is `True`.
- **x_pos** (*Optional[int]*) – The X-coordinate for top-left corner of the `window` in pixels. Defaults to 10 px. Redundant if `maximized` is `True`.
- **y_pos** (*Optional[int]*) – The Y-coordinate for top-left corner of the `window` in pixels. Defaults to 10 px. Redundant if `maximized` is `True`.
- **maximized** (*Optional[bool]*) – `window` is maximized when set to `True`. Defaults to `False`.
- **plugins** (*Optional[bool]*) – Enables plugins like flash when set as `True`. Defaults to `False`.
- **developer_mode** (*Optional[bool]*) – Enables developer mode when set as `True`. Defaults to `False`. The developer mode gives access to web inspector and other development tools and enables right-click on the webpage.
- **allow_overwrite** (*Optional[bool]*) – `PySide.QtGui.QApplication` can be instantiated only once. If it is already instantiated, then setting `allow_overwrite` to `True` overwrites the `QApplication`'s window with window of this class instance. If `False`, RuntimeError is raised. If `QApplication` is not instantiated, this is irrelevent.

app
PySide.QtGui.QApplication – The singleton Qt application object. This can be instantiated only once in the entire process.

window
PySide.QtGui.QMainWindow – The window being displayed in the `app`.

web_app
PySide.QtWebKit.QWebView – The web view widget which renders and displays HTML in the a `window`.

url
unicode property – The URL currently being displayed in `window`. Set the property to a URL unicode string to change the URL being displayed.

html
unicode property – The HTML currently rendered in the `web_app`. This is a readonly property.

maximized
bool property – A boolean which describes whether the `window` is maximized or not. Can be set to `True` to maximize the window and set to `False` to restore.

width
int property – Width of the `window` in pixels. Set the value of this property in pixels to change the width.

height
int property – Height of the `window` in pixels. Set the value of this property in pixels to change the height.

x_pos
int property – The X-coordinate for top-left corner of the `window` in pixels. Set the value of this property in pixels to move the `window` horizontally.

y_pos
int property – The Y-coordinate for top-left corner of the `window` in pixels. Set the value of this property in pixels to move the `window` vertically.

title
unicode property – The title of the `window`. Set the value of this property to change the title.

plugins
bool property – A boolean flag which indicates whether plugins like flash are enabled or not. Set the value to `True` or `False` as required.

developer_mode
bool property – A boolean flag which indicated whether developer mode is active or not. The developer mode gives access to web inspector and other development tools and enables right-click on the webpage. Set the value to `True` or `False` as required.

Raises `RuntimeError` – If `PySide.QtGui.QApplication` is already instantiated and `allow_overwrite` is `False`.

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__
x.__hash__() <==> hash(x)

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__
x.__repr__() <==> repr(x)

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

__str__
x.__str__() <==> str(x)

auto_resize()
Resizes and relocates the `window` to previous state

If the `window` is not maximized, this function resizes it to the stored dimensions, moves it to the stored location.

evaluate_javascript(*javascript_string*)
Evaluates javascript in web page currently displayed

Parameters **javascript_string** (*str*) – The string of javascript code that has to be evaluated.

execute()
Executes the application without ending the process on its end.

DO NOT execute this process directly. Use only when `htmlPy.BaseGUI.stop()` is connected to some signal.

html
unicode – The HTML currently rendered in the window.

This property will return the HTML which is being displayed in the `web_app`. This is not asynchronous. The URL set with htmlPy will not load until the window is in display.

right_click_setting(*value*)
Javascript based setting for right click on the application.

This function changes the web page's behaviour on right click. Normal behaviour is to open a context menu. Enabling right click exhibits that behaviour. Right click is enabled by default. Disabling right click suppresses context menu for entire page. Enabling right click for only inputs suppresses context menu for all elements excepts inputs and textarea, which is the recommended option. The arguments provided should be from `htmlPy.settings` module as explained further.

Parameters value (*int*) – should be either `htmlPy.settings.ENABLE` (default) or `htmlPy.settings.DISABLE` or `htmlPy.settings.INPUTS_ONLY` (recommended)

start()
Starts the application.

This is not asynchronous. Starting the application will halt the further processes. DO NOT start outside the `if __name__ == "__main__":` conditional

stop()
Stops the application. Use only to bind with signals.

The Qt application does not have to be manually stopped. Also, after starting the application is stuck in the execution loop and will not go further until it is stopped. Calling this function manually is redundant. This function exits only to be binded with QSignals to stop the application when that signal is emitted.

text_selection_setting(*value*)
Javascript based setting for text selection in the application.

This function changes the web page's behaviour on selection of text. Normal behaviour is to highlight the selected text. Enabling text selection exhibits that behaviour. Text selection is enabled by default. Disabling text selection disallows user to select any text on the page except for inputs and disables the I-beam cursor for text selection. The arguments provided should be from `htmlPy.settings` module as explained further.

Parameters value (*int*) – should be either `htmlPy.settings.ENABLE` (default) or `htmlPy.settings.DISABLE` or

5.3 Class `htmlPy.Object`

class `htmlPy.Object`
Alias of `PySide.QtCore.QObject`.

For binding python functionalities to GUI, the classes being bound should inherit `htmlPy.Object`. Its constructor has to be called. The methods of the class that have to be bound to GUI must be decorated with `htmlPy.Slot`.

Example: Refer to the API reference of `htmlPy.Slot` for an example.

5.4 Decorator htmlPy.Slot

class htmlPy.**Slot**
Alias of PySide.QtCore.Slot

This decorator binds the methods of classes which inherit htmlPy.Object to the GUI. The argument types and return type of the method being bound have to be provided as argument to the decorator.

Parameters

- ***args** (*[type]*) – Data types of arguments of the method being decorated
- **result** (*type*) – Data type of return value of the method being decorated

Example:

```
import htmlPy
htmlPy_app = htmlPy.AppGUI()


class BindingClass(htmlPy.Object):

    @htmlPy.Slot(str, int, result=int)
    def binding_method(self, string_arg, int_arg):
        int_return = 1
        return int_return

htmlPy_app.bind(BindingClass())
```

5.5 Module htmlPy.settings

htmlPy.settings has 3 variables.

htmlPy.settings.**ENABLE**
Used for enabling some setting

htmlPy.settings.**DISABLE**
Used for enabling some setting

htmlPy.settings.**INPUTS_ONLY**
Currently used only to DISABLE right clicking on application except for input fields.

5.6 Class htmlPy.BaseGUI

class htmlPy.**BaseGUI**(*title=u'Application'*, *width=800*, *height=600*, *x_pos=10*, *y_pos=10*, *maximized=False*, *plugins=False*, *developer_mode=False*, *allow_overwrite=False*)
Abstract GUI class for creating apps using PySide's Qt and HTML.

This class shouldn't be used directly. It serves as a parent to other GUI classes. Use htmlPy.AppGUI and htmlPy.WebAppGUI for developing applications.

Parameters **args** (*No*) – This is an abstract base class. It must not be instantiated.

app
PySide.QtGui.QApplication – The singleton Qt application object. This can be instantiated only once in the entire process.

window
PySide.QtGui.QMainWindow – The window being displayed in the `app`.

web_app
PySide.QtWebKit.QWebView – The web view widget which renders and displays HTML in the a `window`.

maximized
bool property – A boolean which describes whether the `window` is maximized or not. Can be set to `True` to maximize the window and set to `False` to restore.

width
int property – Width of the `window` in pixels. Set the value of this property in pixels to change the width.

height
int property – Height of the `window` in pixels. Set the value of this property in pixels to change the height.

x_pos
int property – The X-coordinate for top-left corner of the `window` in pixels. Set the value of this property in pixels to move the `window` horizontally.

y_pos
int property – The Y-coordinate for top-left corner of the `window` in pixels. Set the value of this property in pixels to move the `window` vertically.

title
unicode property – The title of the `window`. Set the value of this property to change the title.

plugins
bool property – A boolean flag which indicates whether plugins like flash are enabled or not. Set the value to `True` or `False` as required.

developer_mode
bool property – A boolean flag which indicated whether developer mode is active or not. The developer mode gives access to web inspector and other development tools and enables right-click on the webpage. Set the value to `True` or `False` as required.

auto_resize()
Resizes and relocates the `window` to previous state

If the `window` is not maximized, this function resizes it to the stored dimensions, moves it to the stored location.

evaluate_javascript(*javascript_string*)
Evaluates javascript in web page currently displayed

Parameters **javascript_string** (*str*) – The string of javascript code that has to be evaluated.

execute()
Executes the application without ending the process on its end.

DO NOT execute this process directly. Use only when `htmlPy.BaseGUI.stop()` is connected to some signal.

right_click_setting(*value*)
Javascript based setting for right click on the application.

This function changes the web page's behaviour on right click. Normal behaviour is to open a context menu. Enabling right click exhibits that behaviour. Right click is enabled by default. Disabling right click suppresses context menu for entire page. Enabling right click for only inputs suppresses context menu for all elements excepts inputs and textarea, which is the recommended option. The arguments provided should be from `htmlPy.settings` module as explained further.

Parameters **value** (*int*) – should be either `htmlPy.settings.ENABLE` (default) or `htmlPy.settings.DISABLE` or `htmlPy.settings.INPUTS_ONLY` (recommended)

start()
Starts the application.

This is not asynchronous. Starting the application will halt the further processes. DO NOT start outside the `if __name__ == "__main__":` conditional

stop()
Stops the application. Use only to bind with signals.

The Qt application does not have to be manually stopped. Also, after starting the application is stuck in the execution loop and will not go further until it is stopped. Calling this function manually is redundant. This function exits only to be binded with QSignals to stop the application when that signal is emitted.

text_selection_setting(*value*)
Javascript based setting for text selection in the application.

This function changes the web page's behaviour on selection of text. Normal behaviour is to highlight the selected text. Enabling text selection exhibits that behaviour. Text selection is enabled by default. Disabling text selection disallows user to select any text on the page except for inputs and disables the I-beam cursor for text selection. The arguments provided should be from `htmlPy.settings` module as explained further.

Parameters **value** (*int*) – should be either `htmlPy.settings.ENABLE` (default) or `htmlPy.settings.DISABLE` or

CHAPTER 6

Indices and tables

- *genindex*
- *modindex*
- *search*

h

Symbols

A

B

D

E

H

I

M

P

R

S

T

U

W

X

Y