TomSchimansky / TkinterMapView

Code Issues 68 Pull requests 16 Actions Projects Security Insights

A python Tkinter widget to display tile based maps like OpenStreetMap or Google Satellite Images.

CC0-1.0 license

644 stars 87 forks 13 watching Branches Activity
Tags

Public repository

main 1 Branch 0 Tags Go to file t Go to file Add file Code

TomSchimansky update version 14a51d1 · 11 months ago

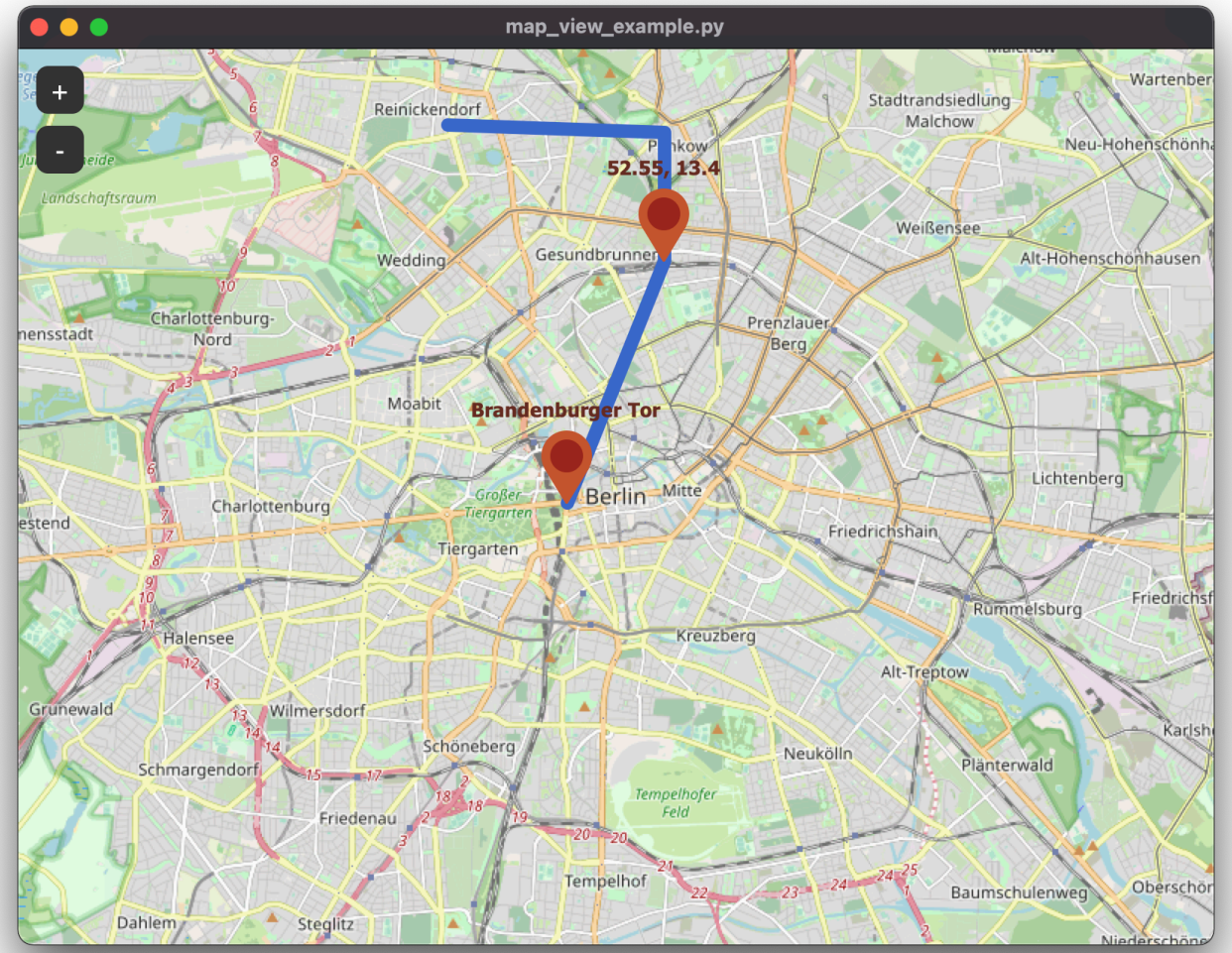
documentation_images	added polygons, added data attribute for mark...	2 years ago
examples	update version	11 months ago
tkintermapview	update version	11 months ago
.gitignore	readme updates	3 years ago
LICENSE.txt	minor bug fixes and pypi files	3 years ago
README.md	Update README.md	2 years ago
project.toml	minor bug fixes and pypi files	3 years ago
requirements.txt	upgrade pillow version to 9.3.0	2 years ago
setup.py	update version	11 months ago

README CC0-1.0 license

pypi v1.29 pip downloads 5.1k/month license Creative Commons Zero v1.0 Universal

TkinterMapView - simple Tkinter map component

1.00

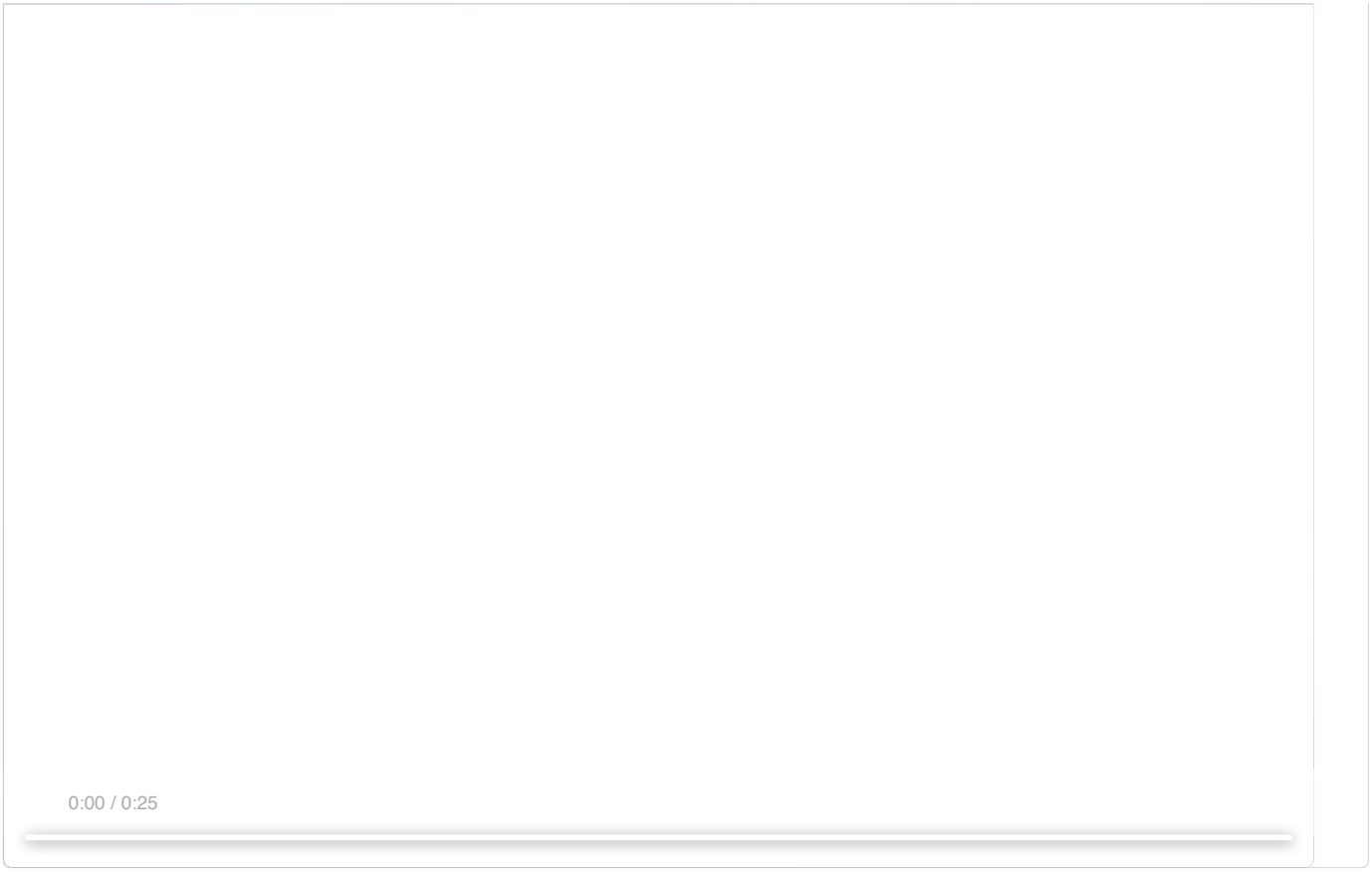


TkinterMapView is a tile based interactive map renderer widget for the python Tkinter library. By default, it displays the OpenStreetMap map, but you can change the tile server to whatever you like, and it also supports a second tile server for overlays like OpenSeaMap. You can set the current focus of the widget by a position or address, and place markers or a path on the map.

The above image program is produced by the code example `map_view_simple_example.py` .

But you can also embed the widget into a program like the following image shows. For example by using the CustomTkinter library, which provides rounded buttons and frames in a light and dark mode:

tkintermapview_macos.mp4



| `examples/map_with_customtkinter.py` on macOS

Installation

```
pip3 install tkintermapview
```

Update: `pip3 install tkintermapview --upgrade`

https://pypi.org/project/tkintermapview/

Documentation / Tutorial

- Importing
- Create the widget
- Set coordinate position
- Set address position
- Set position with marker
- Set position and zoom to fit bounding box
- Create position markers
- Create path from position list
- Create polygon from position list
- Mouse events on the map
- Utility methods
- Use other tile servers
- Use offline tiles

Importing

Import tkinter as normal and from tkintermapview import the TkinterMapView widget.

```
import tkinter
import tkintermapview
```

Create the widget

Create the standard tkinter window and place a TkinterMapView in the middle of the window. The first argument must be the widgets master, then you specify the `width`, `height` and `corner_radius` of the widget.

```
# create tkinter window
root_tk = tkinter.Tk()
root_tk.geometry(f"{800}x{600}")
root_tk.title("map_view_example.py")

# create map widget
map_widget = tkintermapview.TkinterMapView(root_tk, width=800, height=600, corner_radius=0)
map_widget.place(relx=0.5, rely=0.5, anchor=tkinter.CENTER)
```

If you also call `root_tk.mainloop()` at the end, this is already a fully working example to test the map widget.

Set coordinate position

The standard position on which the map is focused is Berlin, Germany, but you can change the position and zoom with the following commands. The position must be given in decimal coordinates and the zoom ranges from 0 to 19 where 19 is the highest zoom level.

```
# set current widget position and zoom
map_widget.set_position(48.860381, 2.338594)  # Paris, France
map_widget.set_zoom(15)
```

Set address position

But you can not only set the position by decimal coordinates, but also by an address string like `"colosseo, rome, italy"` or `"3 St Margaret St, London, United Kingdom"`. The address is converted to a position by the OpenStreetMap geocode service Nomatim.

```
# set current widget position by address
map_widget.set_address("colosseo, rome, italy")
```

Set position with marker

If you also want a red marker at the set position with a text of the current location, then you can pass the `marker=True` argument to the `set_position` or `set_address` funtions. You get back a PositionMarker object, so that you can modify or delete the marker later:

```
# set current widget position by address
marker_1 = map_widget.set_address("colosseo, rome, italy", marker=True)

print(marker_1.position, marker_1.text)  # get position and text

marker_1.set_text("Colosseo in Rome")  # set new text
# marker_1.set_position(48.860381, 2.338594)  # change position
# marker_1.delete()
```

Set position and zoom to fit bounding box

If you have two decimal coordinates (,) and (,), that define a box, so that the first coordinate is the top-left corner and the second coordinate is the bottom-right corner. Then you can use the `fit_bounding_box` method of the map widget to fit this box into the map widget:

```
map_widget.fit_bounding_box((<lat1>, <long1>), (<lat2>, <long2>))
```

Create position markers

You can also set a position marker without focusing the widget on it. You can pass a `text` argument to the function and get back the marker object, so that you can store the marker and modify or delete it later.

```
# set a position marker
marker_2 = map_widget.set_marker(52.516268, 13.377695, text="Brandenburger Tor")
marker_3 = map_widget.set_marker(52.55, 13.4, text="52.55, 13.4")

# methods
marker_3.set_position(...)
marker_3.set_text(...)
marker_3.change_icon(new_icon)
marker_3.hide_image(True)  # or False
marker_3.delete()
```

A marker can be also customized by passing the following arguments to .set_marker(), .set_address() or .set_position(): `text, font, icon, icon_anchor, image (PhotoImage), image_zoom_visibility, marker_color_circle, marker_color_outside, text_color, command` .

The command function will be called when the marker is clicked and will pass the clicked marker as an argument to the functin which gets called.

The given image will be visible above the marker when the zoom level is in the range specified by `image_zoom_visibility` , which is tuple like the following (min_zoom, max_zoom). `image_zoom_visibility=(0, float('inf'))` means the image will be visible alle the time. The image can also be hidden by calling: `marker.hide_image(True)` or `marker.hide_image(False)` . To check if the image is currently hidden you can access: `marker.image_hidden` which is True or False.

You can also store an object or some reference inside the marker with the `data` attribute, which can be set when creating a marker, and accessed or modified with `marker.data` . This data attribute also exists for path and polygons.

With the `icon` attribute you can pass a PIL.ImageTk.PhotoImage object to a marker, which will be displayed instead of the standard location icon. With `icon_anchor` you can specify the anchor for the icon image (center, n, nw, w, sw, s, ew, e, ne), corresponding to the position of the marker, standard is center, where the icon image is centered over the marker position. With the `.change_icon(new_icon)` method you can change the icon image later, but only if the marker already has an icon image from the beginning. In `examples/map_view_marker_icon_images.py` you can find example code for the `icon` attributes.

With `map_widget.delete_all_marker()` all marker on the map will be deleted.

Create path from position list

You can also create a path which connects multiple markers or completely new positions. You pass a list with position tuples to the function `set_path` and get back a path object. The path object can be modified by adding a new position or remove a specific position. The `set_path` method accepts the following arguments: `position_list`, `color`, `command` (on click), `name` (string for identification), `width`, `data (anything can be stored in the path object).

```
# set a path
path_1 = map_widget.set_path([marker_2.position, marker_3.position, (52.57, 13.4), (52.55, 13.35)])

# methods
path_1.set_position_list(new_position_list)
path_1.add_position(position)
path_1.remove_position(position)
path_1.delete()
```

With `map_widget.delete_all_path()` all path on the map will be deleted.

Create polygon from position list

To create a polygon on the map call the `map_widget.set_polygon()` function and pass a list of coordinate tuples from which the polygon will be created. You can edit the appearance with the following arguments: `fill_color, outline_color, border_width`. You can also set a command function which will be called when the polygon gets clicked and which will get the polygon object as an argument.

```
def polygon_click(polygon):
    print(f"polygon clicked - text: {polygon.name}")

polygon_1 = map_widget.set_polygon([(46.0732306, 6.0095215),
                                    ...
                                    (46.3772542, 6.4160156)],
                                   # fill_color=None,
                                   # outline_color="red",
                                   # border_width=12,
                                   command=polygon_click,
                                   name="switzerland_polygon")

# methods
polygon_1.remove_position(46.3772542, 6.4160156)
polygon_1.add_position(0, 0, index=5)
polygon_1.delete()
```

In `examples/map_view_polygon_example.py` you can find the full example program, which results in the following:

With `map_widget.delete_all_polygon()` all polygons on the map will be deleted.

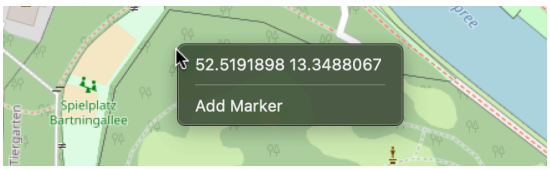
Mouse events on the map

When you click on the map with the right mouse button, a menu pops up, where you can view the current decimal coordinates and copy them into the clipboard by clicking them. You can also add other options to this menu, with the `map_widget.add_right_click_menu_command` method:

```
def add_marker_event(coords):
    print("Add marker:", coords)
    new_marker = map_widget.set_marker(coords[0], coords[1], text="new marker")


map_widget.add_right_click_menu_command(label="Add Marker",
                                        command=add_marker_event,
                                        pass_coords=True)
```

With the `label` argument you set the text inside the menu, and if `pass_coords` is True, the clicked coordinates will be passed to the command function as a tuple.



You can also assign a callback function for a left click event on the map with:

```
def left_click_event(coordinates_tuple):
    print("Left click event with coordinates:", coordinates_tuple)

map_widget.add_left_click_map_command(left_click_event)
```

The callback function will get the decimal coordinates of the clicked location as a tuple.

Releases

No releases published

Packages

No packages published

Used by 386

 + 378

Contributors 3

 **TomSchimansky** Tom Schimansky

 **davidoesch** David Oesch

 **arnehaak** Arne

Languages

- **Python** 100.0%