

# Geospatial Web Services & Web Mapping in 2D / 3D

Munich Technical University || Chair of Geoinformatics Murat Kendir, Prof. Thomas Kolbe || [murat.kendir@tum.de](mailto:murat.kendir@tum.de)



- In this exercise, you will learn how to use geospatial web services in your applications or web views and how to use the operations supported by the web service.
- In the previous exercise, we selected a few web map services from the given Spatial Data Infrastructures (SDIs) and examined the responses using GIS software.
- Now you will learn how to connect to and interact with geospatial web services with the help of some popular libraries. Additionally, you will learn how to use 2D and 3D JavaScript libraries which supports and visualize 2D or 3D datasets or web services.

## Table of Contents

- 1) Accessing to the WMS service with Python
  - 1.1) Examine a WMS service with OWSLib
  - 1.2) Add a WMS layer to Jupyter Notebook using ipyleaflet
- 2) Accessing WMS services from JavaScript libraries
  - 2.1) Adding WMS services to OpenLayers
  - 2.2) Adding WMS services to CesiumJS
  - 2.3) Using OGC Feature API as GeoJSON source in Cesium
- 3) Fundamentals of 3D Visualization in CesiumJS
  - 3.1) Adding 3D Buildings to the Cesium viewer
  - 3.2) Adding terrain to the Cesium viewer
  - 3.3) Adding custom terrain models, 3DTiles and WMS services to Cesium

## 1) Accessing WMS service with Python

OWSLib is one of the OGC compliant web service client and it supports OGC interface standards. Before using the OWSLib consider the following steps:

- If you are not working whole tale.org, please install OWSLib to your current python environment using : **pip install OWSLib**
- Then you need to import the OWSLib library into your code.
- Check the following site to get more information about the OWSLib:
  - <https://owslib.readthedocs.io/en/latest/>

```
In [1]: import owslib
```

- ☐ Check available contents (packages + classes) in the library with following code:  
`help(owslib)`

You can specify a class or package to reduce the loaded library content in your code. There are multiple ways to do so:

- First Option: `from owslib import wms`
  - Imports only wms module from owslib package
  - Type: `help(wms)`
- Second Option: `import owslib.wms`
  - Again, it calls only wms module from owslib package
  - Type: `help(owslib.wms)`
- Another option to import only the relevant function: `from owslib.wms import WebMapService as any_shorten_form`
  - Imports only WebMapService function in the wms module and assigns a custom alias to it.
  - Type: `help(any_shorten_form)`

## 1.1) Examine a WMS service with OWSLib

Select any Web Map Service (WMS) from your previous work, or select a random WMS that covers the city of Hamburg in whole or in part. (If you want to add a random WMS, revisit the [Hamburg TransparenzPortal](#) or [geoportal.de](#) or [INSPIRE Geoportal](#) websites to search for the appropriate web service.)

```
In [1]: # Decide a way to import the library.
from owslib.wms import WebMapService as mywms
# help(mywms)
```

If you type `help(mywms)`, you will see an explanation of the function as below.

`WebMapService(url, version='1.1.1', xml=None, username=None, ...)`

Notice that only "url" parameter is not assigned to a value. This means that the only mandatory parameter is the "url" and all other parameters will be accepted with their default values unless their values are specified.

Now, test your selected WMS with the imported WebMapService function.

```
In [2]: # mytest = mywms('https://geodienste.hamburg.de/HH_WMS_DGM1')
# The website serves the metadata of the WMS highlights that
# the recommended version is "1.3.0". So we can specify that:
mytest = mywms('https://geodienste.hamburg.de/HH_WMS_DGM1', version='1.3.
```

Now, you can use "the built-in functions" to know more about the returned object:

`print(mytest) / type(mytest) / help(mytest) / dir(mytest)` If you are

## Exercise4 - Geospatial Web Services & Web Mapping in 2D/3D

sure about the data is an iterable object (tuple, set, list, dictionary) then you can use also "list" function.

```
In [3]: list(mytest.contents)
# Alternatively: print(mytest.contents.keys())
```

```
Out[3]: ['HH_WMS_DGM1', 'WMS_DGM1_HAMBURG', 'WMS_DGM1_farbig', 'WMS_DGM1_farbig_10000']
```

- Check the available methods by typing `dir(mytest)`.
  - You may notice the "operations" method in the list.

```
In [4]: list(mytest.operations)
```

```
Out[4]: [<owslib.map.wms130.OperationMetadata at 0x745d0c2c70e0>,
<owslib.map.wms130.OperationMetadata at 0x745d0c2c7980>,
<owslib.map.wms130.OperationMetadata at 0x745d0c2c7ad0>,
<owslib.map.wms130.OperationMetadata at 0x745d0c2c7b60>,
<owslib.map.wms130.OperationMetadata at 0x745d0c2c7bf0>,
<owslib.map.wms130.OperationMetadata at 0x745d0c2c7c80>]
```

- As operations stored as objects in a list (array), we can access them using list index (For Exp. `mylist[0]`). Then you can check the available methods or attributes in the class instance by typing `dir(mytest.operations[0])`.

```
In [5]: print( type(mytest.operations[0]) )
print(mytest.operations[0].name)
print(mytest.operations[1].name)
print(mytest.operations[2].name)
print(mytest.operations[3].name)
print(mytest.operations[4].name)
print(mytest.operations[5].name)
# print(mytest.operations[0].formatOptions)
# print(mytest.operations[0].methods)
```

```
<class 'owslib.map.wms130.OperationMetadata'>
GetCapabilities
GetMap
GetFeatureInfo
DescribeLayer
GetLegendGraphic
GetStyles
```

- Alternatively, you can inspect the webservice using the "getServiceXML" method. Please note that the result will be returned as a binary file in XML format. This means that you have to consider the binary format if you want to store or read the returned object as a file. For example, we specified the file mode as "wb" (Write + Binary) in the following code, because the object "`mytest.getServiceXML()`" was in bytes format, not a string.

```
In [6]: # type(mytest.getServiceXML())
# Specify your own relative or full path to save the XML file, if it is r
with open("responses/exr4/my_wms_servicexml.xml", "wb") as my_wms_xml:
```

```
my_wms_xml.write(mytest.getServiceXML())
```

- Check the saved XML file and try to find where the layers, bounding boxes or coordinate reference systems (CRSs) are defined.

Reminder: Firefox and Google Chrome browsers have some extensions to visualize XML files in a "pretty" way. Otherwise you can open the XML file with your favorite text editor and search for a "pretty print" solution. (For Example: XMLTools extension is available in the Notepad++ plugin repository.)

- You can also send queries to get single images with predefined properties. To do that, you may need to expose more information about the WMS service using following commands:

```
In [7]: print(mytest['WMS_DGM1_HAMBURG'].crsOptions, '\n')
print(mytest['WMS_DGM1_HAMBURG'].boundingBox, '\n')
print(mytest['WMS_DGM1_HAMBURG'].styles, '\n')
print(mytest.getOperationByName('GetMap').formatOptions)

['EPSG:25833', 'EPSG:4647', 'EPSG:4258', 'EPSG:31467', 'EPSG:3044', 'EPSG:4326', 'EPSG:3857', 'EPSG:25832', 'CRS:84']

(8.482047, 53.385591, 10.355638, 53.941383)

{'default': {'title': 'default', 'legend': 'https://geodienste.hamburg.de/HH_WMS_DGM1?language=ger&version=1.3.0&service=WMS&request=GetLegendGraphic&sld_version=1.1.0&layer=WMS_DGM1_HAMBURG&format=image/png&STYLE=default', 'legend_width': '298', 'legend_height': '865', 'legend_format': 'image/png'}}
```

```
['image/png', 'image/jpeg', 'image/png; mode=8bit', 'image/vnd.jpeg-png', 'image/vnd.jpeg-png8', 'application/x-pdf', 'image/svg+xml', 'image/tiff', 'application/vnd.google-earth.kml+xml', 'application/vnd.google-earth.kmz', 'application/vnd.mapbox-vector-tile', 'application/x-protobuf', 'application/json']
```

- So, you are aware of capability of the WMS service and you can use this data to get data from the server:

Note that if you are using a different WMS service, you need to change the parameters given in the next example.

```
In [8]: from IPython.display import Image

img = mytest.getmap(
    layers=['WMS_DGM1_HAMBURG'],
    size=[300, 300],
    srs="EPSG:4326",
    bbox=[9.5, 53.4, 10, 53.7],
    format="image/jpeg")

Image(img.read())
```

Out[8]:



## 1.2) Add a WMS layer to Jupyter Notebook using ipyleaflet

As you may know that leaflet is well known and simple-to-use web map library based on JavaScript. However, leaflet has also a widget implementation with Python Notebook which lets users to visualize their leaflet maps in notebook documents. To start using ipyleaflet, type `pip install ipyleaflet` in your python environment or follow the instructions to install ipyleaflet which are available in [this website](#).

```
In [10]: from ipyleaflet import Map

center = (53.547668, 9.985685)
map = Map(center=center, zoom=13)
display(map)
```

```
Map(center=[53.547668, 9.985685], controls=(ZoomControl(options=['position', 'zoom_in_text', 'zoom_in_title', ...
```

You can change the basemap by importing "basemaps" class into your code.

```
In [12]: from ipyleaflet import Map, basemaps
from ipywidgets.embed import embed_minimal_html
import IPython

center = (53.547668, 9.985685)
map = Map(basemap=basemaps.CartoDB.Positron, center=center, zoom=13)
#display(map)
src = 'raw_codes/exr4/ipyleaflet_basemap.html'
embed_minimal_html(src, views=[map])
IPython.display.IFrame(src, width='800px', height='400px')
```

Out[12]:



Now, we can add our WMS service to the leaflet widget by specifying the connection details.

```
In [13]: from ipyleaflet import Map, basemaps, WMSLayer
center = (53.547668, 9.985685)
map = Map(basemap=basemaps.CartoDB.Positron, center=center, zoom=13)
wms = WMSLayer(
    url='https://geodienste.hamburg.de/HH_WMS_DGM1',
    layers='WMS_DGM1_HAMBURG',
    format='image/png',
    transparent=True
)
map.add(wms)
map
```

```
Out[13]: Map(center=[53.547668, 9.985685], controls=(ZoomControl(options=['position', 'zoom_in_text', 'zoom_in_title', ...
```

Alternatively, you can call any WMS parameter including URL from the former OWSLib object.

```
In [14]: center = (53.547668, 9.985685)
map = Map(basemap=basemaps.CartoDB.Positron, center=center, zoom=13)

# print(mytest.url) # Shows the URL of WMS Server
# list(mytest.items()) # Lists the available layers
# print(mytest.items()[0][0]) # Exposes one of the layers in WMS Server
# print(mytest['HH_WMS_DGM1'].name) # Takes only the name of the specific
# print(mytest.getOperationByName('GetMap').formatOptions[0])
# Gives one of the possible image format

wms = WMSLayer(
    url= mytest.url,
```