# Birdy Documentation

*Release 0.6.8*

**Birdhouse**

**Jan 13, 2021**

# CONTENTS

**Birdy (the bird)** *Birdy is not a bird but likes to play with them.*

Birdy is a Python library to work with Web Processing Services (WPS). It is using *OWSLib* from the *GeoPython* project.

You can try Birdy online using Binder (just click on the binder link below), or view the notebooks on NBViewer.

Birdy is part of the Birdhouse project.

Full documentation is on ReadTheDocs.

CHAPTER

ONE

# INSTALLATION

## 1.1 Install from Anaconda

```
$ conda install -c conda-forge birdy
```

## 1.2 Install from GitHub

Check out code from the birdy GitHub repo and start the installation:

```
$ git clone https://github.com/bird-house/birdy.git
$ cd birdy
$ conda env create -f environment.yml
$ python setup.py install
```

CHAPTER

TWO

# EXAMPLES

You can try these notebook online using Binder, or view the notebooks on NBViewer.

## 2.1 Basic Usage

### 2.1.1 Birdy WPSClient example with Emu WPS

[ ]:
```
from birdy import WPSClient
```

#### Use Emu WPS

https://github.com/bird-house/emu

[ ]:
```
emu = WPSClient(url='http://localhost:5000/wps')
emu_i = WPSClient(url='http://localhost:5000/wps', progress=True)
```

#### Get Infos about `hello`

[ ]:
```
emu.hello?
```

#### Run `hello`

[ ]:
```
emu.hello(name='Birdy').get()[0]
```

### Run a long running process

```
[ ]: result = emu_i.sleep(delay='1.0')
```

```
[ ]: result.get()[0]
```

### Run a process returning a reference to a text document

```
[ ]: emu.chomsky(times='5').get()[0]
```

### Pass a local file to a remote process

The client can look up local files on this machine and embed their content in the WPS request to the server. Just set the path to the file or an opened file-like object.

```
[ ]: fn = '/tmp/text.txt'
with open(fn, 'w') as f:
    f.write('Just an example')
emu.wordcounter(text=fn).get(asobj=True)
```

### Automatically convert the output to a Python object

The client is able to convert input objects into strings to create requests, and also convert output strings into python objects. This can be demonstrated with the `inout` process, which simply takes a variety of `LiteralInputs` of different data types and directs them to the output without any change.

```
[ ]: emu.inout?
```

```
[ ]: import datetime as dt
result = emu.inout(string='test', int=1, float=5.6, boolean=True, time='15:45',␣
↪datetime=dt.datetime(2018, 12, 12), text=None, dataset=None)
```

Get result as object

```
[ ]: result.get(asobj=True).text
```

### Example with multiple_outputs

Similarly, the `multiple_outputs` function returns a `text/plain` file. The converter will automatically convert the text file into a string.

```
[ ]: out = emu.multiple_outputs(1).get(asobj=True)[0]
print(out)
```

... or use the metalink library on the referenced metalink file:

```
[ ]: out = emu.multiple_outputs(1).get(asobj=False)[0]
print(out)
```

```
[ ]: from metalink import download
     download.get(out, path='/tmp', segmented=False)
```

### 2.1.2 Interactive usage of Birdy WPSClient with widgets

```
[ ]: from birdy import WPSClient
     from birdy.client import nb_form
     emu = WPSClient(url='http://localhost:5000/wps')
```

```
[ ]: resp = nb_form(emu, 'binaryoperatorfornumbers')
```

```
[ ]: resp.widget.result.get(asobj=True)
```

```
[ ]: nb_form(emu, 'non.py-id')
```

```
[ ]: nb_form(emu, 'chomsky')
```

### 2.1.3 OWSLib versus Birdy

This notebook shows a side-by-side comparison of `owslib.wps.WebProcessingService` and `birdy.WPSClient`.

```
[ ]: from owslib.wps import WebProcessingService
     from birdy import WPSClient

     url = "https://bovec.dkrz.de/ows/proxy/emu?Service=WPS&Request=GetCapabilities&
     ↪Version=1.0.0"

     wps = WebProcessingService(url)
     cli = WPSClient(url=url)
```

#### Displaying available processes

With `owslib`, `wps.processes` is the list of processes offered by the server. With `birdy`, the client is like a module with functions. So you just write `cli.` and press `Tab` to display a drop-down menu of processes.

```
[ ]: wps.processes
```

#### Documentation about a process

With `owslib`, the process title and abstract can be obtained simply by looking at these attributes. For the process inputs, we need to iterate on the inputs and access their individual attributes. To facilitate this, `owslib.wps` provides the `printInputOuput` function.

With `birdy`, just type `help(cli.hello)` and the docstring will show up in your console. With the IPython console or a Jupyter Notebook, `cli.hello?` would do as well. The docstring follows the NumPy convention.

[ ]:
```
from owslib.wps import printInputOutput
p = wps.describeprocess('hello')
print("Title: ", p.title)
print("Abstract: ", p.abstract)

for inpt in p.dataInputs:
    printInputOutput(inpt)
```

[ ]:
```
help(cli.hello)
```

### Launching a process and retrieving literal outputs

With `owslib`, processes are launched using the `execute` method. Inputs are an an argument to `execute` and defined by a list of key-value tuples. These keys are the input names, and the values are string representations. The `execute` method returns a `WPSExecution` object, which defines a number of methods and attributes, including `isComplete` and `isSucceeded`. The process outputs are stored in the `processOutputs` list, whose content is stored in the `data` attribute. Note that this `data` is a list of strings, so we may have to convert it to a `float` to use it.

[ ]:
```
resp = wps.execute('binaryoperatorfornumbers', inputs=[('inputa', '1.0'), ('inputb',
↪'2.0'), ('operator', 'add')])
if resp.isSucceeded:
    output, = resp.processOutputs
    print(output.data)
```

With `birdy`, inputs are just typical keyword arguments, and outputs are already converted into python objects. Since some processes may have multiple outputs, processes always return a `namedtuple`, even in the case where there is only a single output.

[ ]:
```
z = cli.binaryoperatorfornumbers(1, 2, operator='add').get()[0]
z
```

[ ]:
```
out = cli.inout().get()
out.date
```

### Retrieving outputs by references

For `ComplexData` objects, WPS servers often return a reference to the output (an http link) instead of the actual data. This is useful if that output is to serve as an input to another process, so as to avoid passing back and forth large files for nothing.

With `owslib`, that means that the `data` attribute of the output is empty, and we instead access the `reference` attribute. The referenced file can be written to the local disk using the `writeToDisk` method.

With `birdy`, the outputs are by default the references themselves, but it's also possible to download these references in the background and convert them into python objects. To trigger this automatic conversion, set `convert_objects` to `True` when instantating the client `WPSClient(url, convert_objects=True)`. Ini the example below, the first output is a plain text file, and the second output is a json file. The text file is converted into a string, and the json file into a dictionary.

[ ]:
```
resp = wps.execute('multiple_outputs', inputs=[('count', '1')])
output, ref = resp.processOutputs
print(output.reference)
print(ref.reference)
output.writeToDisk('/tmp/output.txt')
```

[ ]:
```
output = cli.multiple_outputs(1).get()[0]
print(output)
# as reference
output = cli.multiple_outputs(1).get(asobj=True)[0]
print(output)
```

## 2.2 Demo

### 2.2.1 AGU 2018 Demo

This notebook shows how to use `birdy`'s high-level interface to WPS processes.

Here we access a test server called `Emu` offering a dozen or so dummy processes.

#### The shell interface

[1]:
```
%%bash
export WPS_SERVICE="http://localhost:5000/wps?Service=WPS&Request=GetCapabilities&
↪Version=1.0.0"
birdy -h
```

```
Usage: birdy [OPTIONS] COMMAND [ARGS]...

  Birdy is a command line client for Web Processing Services.

  Documentation is available on readthedocs:
  http://birdy.readthedocs.org/en/latest/

Options:
  --version             Show the version and exit.
  --cert TEXT           Client side certificate containing both certificate
                        and private key.

  -S, --send            Send client side certificate to WPS. Default: false
  -s, --sync            Execute process in sync mode. Default: async mode.
  -t, --token TEXT      Token to access the WPS service.
  -l, --language TEXT   Set the accepted language to send to the WPS service.
  -L, --show-languages  Show a list of accepted languages for the WPS service.
  -h, --help            Show this message and exit.

Commands:
  ultimate_question         Answer to the ultimate question: This process...
  sleep                     Sleep Process: Testing a long running process,...
  nap                       Afternoon Nap (supports sync calls only): This...
  bbox                      Bounding box in- and out: Give bounding box,...
  hello                     Say Hello: Just says a friendly Hello.Returns a...
  dummyprocess              Dummy Process: DummyProcess to check the WPS...
  wordcounter               Word Counter: Counts words in a given text.
  chomsky                   Chomsky text generator: Generates a random...
  inout                     In and Out: Testing all WPS input and output...
  binaryoperatorfornumbers  Binary Operator for Numbers: Performs operation...
  show_error                Show a WPS Error: This process will fail...
  multiple_outputs          Multiple Outputs: Produces multiple files and...
```

(continues on next page)

(continued from previous page)

```
  esgf_demo                 ESGF Demo: Shows how to use WPS metadata for...
  output_formats            Return different output formats.: Dummy process...
  poly_centroid             Approximate centroid of a polygon.: Return the...
  ncmeta                    Return NetCDF Metadata: Return metadata from a...
  non.py-id                 Dummy process including non-pythonic...
  simple_dry_run            Simple Dry Run: A dummy download as simple...
  ncml                      Test NcML THREDDS capability: Return links to
                            an...

  translation               Translated process: Process including...
```

[2]:
```
%%bash
export WPS_SERVICE="http://localhost:5000/wps?Service=WPS&Request=GetCapabilities&
↪Version=1.0.0"
birdy hello -h
```

```
Usage: birdy hello [OPTIONS]

  Say Hello: Just says a friendly Hello.Returns a literal string output with
  Hello plus the inputed name.

Options:
  --version                 Show the version and exit.
  --name TEXT               Your name
  --output_formats TEXT...  Modify output format (optional). Takes three
                            arguments, output name, as_reference (True, False,
                            or None for process default), and mimetype(None
                            for process default).

  -h, --help                Show this message and exit.
```

[3]:
```
%%bash
export WPS_SERVICE="http://localhost:5000/wps?Service=WPS&Request=GetCapabilities&
↪Version=1.0.0"
birdy hello --name stranger
```

```
Output:
output=['Hello stranger']
```

### The python interface

The `WPSClient` function creates a *mock* python module whose functions actually call a remote WPS process. The docstring and signature of the function are dynamically created from the remote's process description. If you type `wps.` and then press `Tab`, you should see a drop-down list of available processes. Simply call `help` on each process of type `?` after the process to print the docstring for that process.

[4]:
```
from birdy import WPSClient
url = "http://localhost:5000/wps?Service=WPS&Request=GetCapabilities&Version=1.0.0"
wps = WPSClient(url, verify=False)
help(wps.binaryoperatorfornumbers)
```

```
Help on method binaryoperatorfornumbers in module birdy.client.base:

binaryoperatorfornumbers(inputa=2.0, inputb=3.0, operator='add', output_formats=None)␣
↪method of birdy.client.base.WPSClient instance
    Performs operation on two numbers and returns the answer. This example process is␣
↪taken from Climate4Impact.
```

(continues on next page)

(continued from previous page)

```
    Parameters
    ----------
    inputa : float
        Enter Input 1
    inputb : float
        Enter Input 2
    operator : {'add', 'substract', 'divide', 'multiply'}string
        Choose a binary Operator

    Returns
    -------
    output : float
        Binary operator result
```

Type `wps.` and the press `Tab`, you should see a drop-down list of available processes.

[5]:
```
# wps.
```

### Process execution

Processes are executed by calling the function. Each process instantaneoulsy returns a `WPSExecute` object. The actual output values of the process are obtained by calling the `get` method. This `get` method returns a namedtuple storing the process outputs as native python objects.

[6]:
```
resp = wps.binaryoperatorfornumbers(1, 2, operator='add')
print(resp)
resp.get()
```

```
<birdy.client.utils.WPSExecution object at 0x108237d30>
```

[6]:
```
binaryoperatorfornumbersResponse(
    output=3.0
)
```

For instance, the `inout` function returns a wide variety of data types (float, integers, dates, etc) all of which are converted into a corresponding python type.

[7]:
```
wps.inout().get()
```

[7]:
```
inoutResponse(
    string='This is just a string',
    int=7,
    float=3.14,
    boolean=True,
    angle=90.0,
    time=datetime.time(12, 0),
    date=datetime.date(2012, 5, 1),
    datetime=datetime.datetime(2016, 9, 2, 12, 0, tzinfo=tzutc()),
    string_choice='scissor',
    string_multiple_choice='gentle albatros',
    int_range=1,
    any_value='any value',
    ref_value='Scotland',
    text='http://localhost:5000/outputs/e7700e9c-559c-11eb-bcba-784f435e8862/input.txt
↪',
```

(continues on next page)

(continued from previous page)

```
    dataset='http://localhost:5000/outputs/e7700e9c-559c-11eb-bcba-784f435e8862/input_
↪pd0bgv88.txt',
    bbox=BoundingBox(minx='0.0', miny='0.0', maxx='10.0', maxy='10.0', crs=Crs(id=
↪'epsg:4326', naming_authority=None, category=None, type=None, authority='EPSG',␣
↪version=None, code=4326, axisorder='yx', encoding='code'), dimensions=2)
)
```

### Retrieving outputs by references

For `ComplexData` objects, WPS servers often return a reference to the output (an http link) instead of the actual data. This is useful if that output is to serve as an input to another process, so as to avoid passing back and forth large files for nothing.

With `birdy`, the outputs are by default return values are the references themselves, but it's also possible to download these references in the background and convert them into python objects. To trigger this automatic conversion, set `asobj` to `True` when calling the `get` method. In the example below, we're using a dummy process called `output_formats`, whose first output is a netCDF file, and second output is a json file. With `asobj=True`, the netCDF file is opened and returned as a `netcdf4.Dataset` instance, and the json file into a dictionary.

[8]:
```
# NBVAL_SKIP
# This cell is failing due to an unautheticated SSL certificate
out = wps.output_formats()
nc, json = out.get()
print(out.get())
ds, json = out.get(asobj=True)
print(json)
ds
```

```
output_formatsResponse(
    netcdf='http://localhost:5000/outputs/e78722ee-559c-11eb-8bc2-784f435e8862/dummy.
↪nc',
    json='http://localhost:5000/outputs/e78722ee-559c-11eb-8bc2-784f435e8862/dummy.
↪json'
)
{'testing': [1, 2]}
```

[8]:
```
<xarray.Dataset>
Dimensions:  (time: 1)
Coordinates:
  * time     (time) float64 42.0
Data variables:
    *empty*
Attributes:
    title:    Test dataset
```

### Progress bar

It's possible to display a progress bar when calling a process. The interface to do so at the moment goes like this. Note that the cancel button does not do much here, as the WPS server does not support interruption requests.

[9]:
```
wps = WPSClient(
    'http://localhost:5000/wps',
    progress=True)
resp = wps.sleep()
```

```
HBox(children=(IntProgress(value=0, bar_style='info', description='Processing:'),␣
↪Button(button_style='danger'...
```

CHAPTER

THREE

# DEVELOPMENT

## 3.1 Get Started!

Check out code from the birdy GitHub repo and start the installation:

```
$ git clone https://github.com/bird-house/birdy.git
$ cd birdy
$ conda env create -f environment.yml
$ python setup.py develop
```

Install additional dependencies:

```
$ pip install -r requirements_dev.txt
```

When you're done making changes, check that your changes pass *flake8* and the tests:

```
 $ flake8
 $ pytest

Or use the Makefile::

  $ make lint
  $ make test
  $ make test-all
```

## 3.2 Write Documentation

You can find the documentation in the *docs/source* folder. To generate the Sphinx documentation locally you can use the *Makefile*:

```
$ make docs
```

## 3.3 Bump a new version

Make a new version of Birdy in the following steps:

- Make sure everything is commit to GitHub.
- Update `CHANGES.rst` with the next version.
- Dry Run: `bumpversion --dry-run --verbose --new-version 0.3.1 patch`
- Do it: `bumpversion --new-version 0.3.1 patch`
- ... or: `bumpversion --new-version 0.4.0 minor`
- Push it: `git push --tags`

See the bumpversion documentation for details.

CHAPTER

FOUR

# API REFERENCE



## 4.1 Using the command line

Birdy has a command line interface to interact with a Web Processing Service.

### 4.1.1 Example

Here is an example with Emu WPS service:

```
$ birdy -h
$ birdy hello -h
$ birdy hello --name stranger
'Hello Stranger'
```

### 4.1.2 Configure WPS service URL

By default Birdy talks to a WPS service on URL http://localhost:5000/wps. You can change this URL by setting the enivronment variable `WPS_SERVICE`:

```
$ export WPS_SERVICE=http://localhost:5000/wps
```

### 4.1.3 Configure SSL verification for HTTPS

In case you have a WPS serive using HTTPS with a self-signed certificate you need to configure the environment variable `WPS_SSL_VERIFY`:

```
# deactivate SSL server validation for a self-signed certificate.
$ export WPS_SSL_VERIFY=false
```

You can also set the path of the service certificate. Read the requests documentation.

### 4.1.4 Use an OAuth2 access token

If the WPS service is secured by an OAuth2 access tokens then you can provide an access token with the `--token` option:

```
$ birdy --token abc123 hello --name stranger
```

### 4.1.5 Use client certificate to access WPS service

If the WPS service is secured by x509 certificates you can add a certificate with the `--cert` option to a request:

```
# run hello with certificate
$ birdy --cert cert.pem hello --name stranger
```

### 4.1.6 Using the output_formats option for a process

Each process also has a default option named *output_formats*. It can be used to override a process' output format's default values.

This option takes three parameters;

The format identifier: the name given to it

The `as reference` parameter: if the output is returned as a link of not. Can be True, False, or None (which uses the process' default value)

The MIME type: of which MIME type is the output. Unless the process has multiple supported mime types, this can be left to None.

Looking at the emu process *output_formats*, the JSON output's default's the `as reference` parameter to False and returns the content directly:

```
$ birdy output_formats
  Output:
  netcdf=http://localhost:5000/outputs/d9abfdc4-08d6-11eb-9334-0800274cd70c/dummy.nc
  json=['{"testing": [1, 2]}']
```

We can then use the output_formats option to redefine it:

```
$ birdy output_formats --output_formats json True None
  Output:
  netcdf=http://localhost:5000/outputs/38e9aefe-08db-11eb-9334-0800274cd70c/dummy.nc
  json=http://localhost:5000/outputs/38e9aefe-08db-11eb-9334-0800274cd70c/dummy.json
```

## 4.2 Using the Python library

The `WPSClient` class aims to make working with WPS servers easy, even without any prior knowledge of WPS.

Calling the `WPSClient` class creates an instance whose methods call WPS processes. These methods are generated at runtime based on the process description provided by the WPS server. Calling a function sends an *execute* request to the server. The server response is parsed and returned as a `WPSExecution` instance, which includes information about the job status, the progress percentage, the starting time, etc. The actual output from the process are obtained by calling the *get* method.

The output is parsed to convert the outputs in native Python whenever possible. *LiteralOutput* objects (string, float, integer, boolean) are automatically converted to their native format. For *ComplexOutput*, the module can either return a link to the output files stored on the server, or try to convert the outputs to a Python object based on their mime type. This conversion will occur with *get(asobj=True)*. So for example, if the mime type is 'application/json', the output would be a *dict*.

Inputs to processes can be native Python types (string, float, int, date, datetime), http links or local files. Local files can be transferred to a remote server by including their content into the WPS request. Simply set the input to a valid path or file object and the client will take care of reading and converting the file.

### 4.2.1 Example

If a WPS server with a simple *hello* process is running on the local host on port 5000:

```
>>> from birdy import WPSClient
>>> emu = WPSClient('http://localhost:5000/')
>>> emu.hello
<bound method hello of <birdy.client.base.WPSClient object>>
>>> print(emu.hello.__doc__)
""
Just says a friendly Hello. Returns a literal string output with Hello plus the␣
↪inputed name.

Parameters
----------
name : string
    Please enter your name.

Returns
-------
output : string
    A friendly Hello from us.

""

# Call the function. The output is a namedtuple
>>> emu.hello('stranger')
hello(output='Hello stranger')
```

### 4.2.2 Authentication

If you want to connect to a server that requires authentication, the `WPSClient` class accepts an *auth* argument that behaves exactly like in the popular *requests* module (see requests Authentication)

The simplest form of authentication is HTTP Basic Auth. Although wps processes are not commonly protected by this authentication method, here is a simple example of how to use it:

```
>>> from birdy import WPSClient
>>> from requests.auth import HTTPBasicAuth
>>> auth = HTTPBasicAuth('user', 'pass')
>>> wps = WPSClient('http://www.example.com/wps', auth=auth)
```

Because any *requests*-compatible class is accepted, custom authentication methods are implemented the same way as in *requests*.

For example, to connect to a magpie protected wps, you can use the requests-magpie module:

```
>>> from birdy import WPSClient
>>> from requests_magpie import MagpieAuth
>>> auth = MagpieAuth('https://www.example.com/magpie', 'user', 'pass')
>>> wps = WPSClient('http://www.example.com/wps', auth=auth)
```

### 4.2.3 Output format

Birdy automatically manages process output to reflect it's default values or Birdy's own defaults.

However, it's possible to customize the output of a process. Each process has an input named ``output_formats`, that takes a dictionary as a parameter:

```
# example format = {
#     'output_identifier': {
#         'as_ref': <True, False or None>
#         'mimetype': <MIME type as a string or None>,
#     },
# }

# A dictionary defining netcdf and json outputs
>>> custom_format = {
>>>     'netcdf': {
>>>         'as_ref': True,
>>>         'mimetype': 'application/json',
>>>     },
>>>     'json': {
>>>         'as_ref': False,
>>>         'mimetype': None
>>>     }
>>> }
```

Utility functions can also be used to create this dictionary:

```
>>> custom_format = create_output_dictionary('netcdf', True, 'application/json')
>>> add_output_format(custom_format, 'json', False, None)
```

The created dictionary can then be used with a process:

```
>>> cli = WPSClient("http://localhost:5000")
>>> z = cli.output_formats(output_formats=custom_format).get()
>>> z
```

CHAPTER

FIVE

# CHANGE HISTORY

## 5.1 0.6.9 (2020-03-10)

Changes:

- Fixed passing Path objects (#169)
- Trying to guess mime type of inputs rather than taking the first value (#171)

## 5.2 0.6.6 (2020-03-03)

Changes:

- Fixed the docs (#150).
- Added outputs to execute in CLI (#151).
- Updated tests (#152).
- Added offline tests (#153).
- Updated conda links (#155).
- Handle Python keywords (#158)
- Fix emu (#159).
- Updated demo notebook tests (#160).
- Added ECMWF demo notebook (#162).
- Added roocs wps demo notebook (#165).
- Added missing files in MANIFEST.in for pypi install (#166).

## 5.3 0.6.5 (2019-08-19)

Changes:

- Fix arguments ordering (#139).
- Fix imports warning (#138).
- Using nbsphinx (#142).
- Fix pip install (#143).

- Add custom authentication methods (#144).
- Use oauth token (#145).
- Skip Python 2.7 (#146).

## 5.4 0.6.4 (2019-07-03)

Changes:

- Fix default converter to return bytes (#137).

## 5.5 0.6.3 (2019-06-21)

Changes:

- Disabled segmented metalink downloads (#132).
- Fix nested conversion (#135).

## 5.6 0.6.2 (2019-06-06)

Changes:

- Added support for passing sequences (list, tuple) as WPS inputs (#128).

## 5.7 0.6.1 (2019-05-27)

Changes:

- Added verify argument when downloading files to disk (#123).
- Bugfixes: #118, #121

## 5.8 0.6.0 (2019-04-04)

Changes:

- Added conversion support for nested outputs (metalink, zip) (#114).
- Added support for Metalink (#113).
- Added support for zip converter (#111).
- Added support for ESGF CWT API (#102).
- Speed up by using *DescribeProcess* with *identifier=all* (#98).
- Added support for passing local files to server as raw data (#97).
- Cleaned up notebooks (#107).
- Various Bugfixes: #83, #91, #99

## 5.9 0.5.1 (2018-12-18)

Changes:

- Added support to launch Jupyter notebooks with birdy examples on binder (#94, #95).

## 5.10 0.5.0 (2018-12-03)

Changes:

- Renamed pythonic WPS client (#63): `birdy.client.base.WPSClient` and `from birdy import WPSClient`.
- Added *WPSResult* for WPS outputs as *namedtuple* (#84, #64).
- Support for Jupter Notebooks (#40): cancel button (work in progress), progress bar, input widget.
- Updated notebooks with examples for *WPSClient*.

## 5.11 0.4.2 (2018-09-26)

Changes:

- Fixed WPS default parameter (#52).
- Using `WPS_SSL_VERIFY` environment variable (#50).

## 5.12 0.4.1 (2018-09-14)

Changes:

- Fixed test-suite (#49).
- Import native client with `import_wps` (#47).
- Fix: using string type when dataType is not provided (#46).
- Updated docs for native client (#43).

## 5.13 0.4.0 (2018-09-06)

Release for Dar Es Salaam.

Changes:

- Conda support on RTD (#42).
- Fix optional input (#41).

## 5.14 0.3.3 (2018-07-18)

Changes:

- Added initial native client (#24, #37).

## 5.15 0.3.2 (2018-06-06)

Changes:

- Fix MANIFEST.in.

## 5.16 0.3.1 (2018-06-06)

Changes:

- Fix bumpversion.

## 5.17 0.3.0 (2018-06-05)

Changes:

- Use bumpversion (#29).
- Use click for CLI (#6).
- Using GitHub templates for issues, PRs and contribution guide.

## 5.18 0.2.2 (2018-05-08)

Fixes:

- Update travis for Python 3.x (#19).
- Fix parsing of WPS capabilities with `%` (#18).

New Features:

- using `mode` for async execution in OWSLib (#22).

## 5.19 0.2.1 (2018-03-14)

Fixes:

- Fixed Sphinx and updated docs: #15.

New Features:

- Fix #14: added `--cert` option to use x509 certificates.

## 5.20 0.2.0 (2017-09-25)

- removed buildout . . . just using conda.
- cleaned up docs.
- updated travis.
- fixed tests.
- added compat module for python 3.x

## 5.21 0.1.9 (2017-04-07)

- updated buildout and Makefile.
- updated conda environment.
- fixed tests.
- replaced nose by pytest.
- pep8.
- fixed travis.
- fixed ComplexData input.
- show status message in log.

## 5.22 0.1.8 (2016-05-02)

- added backward compatibility for owslib.wps without headers and verify parameter.

## 5.23 0.1.7 (2016-05-02)

- added twitcher token parameter.
- using ssl verify option again.

## 5.24 0.1.6 (2016-03-22)

- added support for bbox parameters.

## 5.25 0.1.5 (2016-03-15)

- fixed wps init (using standard owslib).
- update makefile.

## 5.26 0.1.4 (2015-10-29)

- using ssl verify option of WebProcessingSerivce
- moved python requirements to requirements/deploy.txt

## 5.27 0.1.3 (2015-08-20)

- more unit tests.
- fixed unicode error in wps description.
- using latest ComplexDataInput from owslib.wps.

## 5.28 0.1.2 (2015-08-14)

- fixed encoding of input text files.
- more unit tests.

## 5.29 0.1.1 (2015-08-13)

- allow local file path for complex inputs.
- send complex data inline with requet to remote wps service.

## 5.30 0.1.0 (2014-12-02)

- Initial Release.

# PYTHON MODULE INDEX

## b

# INDEX

## B



## M