

---

# Emu Documentation

*Release 0.6*

**Birdhouse**

**Jan 16, 2018**



---

## Contents

---

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Non-default installation . . . . .	4
1.2	Run Emu as Docker container . . . . .	4
<b>2</b>	<b>Configuration</b>	<b>5</b>
<b>3</b>	<b>Developer Guide</b>	<b>7</b>
3.1	Running unit tests . . . . .	7
3.2	Running WPS service in test environment . . . . .	7
<b>4</b>	<b>Tutorials</b>	<b>9</b>
4.1	Tutorial: using Docker . . . . .	9
4.2	Tutorial: using postgres database . . . . .	10
<b>5</b>	<b>Sphinx AutoAPI Index</b>	<b>13</b>
5.1	wsgi . . . . .	13
5.2	demo . . . . .	13
5.3	processes . . . . .	14
5.4	tests . . . . .	16
	<b>Python Module Index</b>	<b>19</b>



**Emu (the bird)** *Emus are curious birds who are known to follow and watch other animals and humans. Emus do not sleep continuously at night but in several short stints sitting down. [..].* ([Wikipedia](#)).

Emu is a Python package with some test proccess for Web Processing Services (WPS). Currently it is using the [PyWPS-4](#) server.

Emu is part of the [Birdhouse](#) project.



# CHAPTER 1

---

## Installation

---

The installation is using the Python distribution system [Anaconda](#) to maintain software dependencies. Anaconda will be installed during the installation process in your home directory `~/anaconda`.

The installation process setups a conda environment named `emu` with all dependent conda (and pip) packages. The installation folder (for configuration files etc) is by default `~/birdhouse`. Configuration options can be overridden in the buildout `custom.cfg` file. The `ANACONDA_HOME` and `CONDA_ENVS_DIR` locations can be changed in the `Makefile.config` file.

The default installation *does not need admin rights* and files will only be written into the `$HOME` folder of the installation user. The services are started using [supervisor](#) and run as the installation user.

Now, check out the `emu` code from GitHub and start the installation:

```
$ git clone https://github.com/bird-house/emu.git
$ cd emu
$ make clean install
```

After successful installation you need to start the services:

```
$ make start # starts supervisor services
$ make status # shows supervisor status
```

The depolyed WPS service is by default available on <http://localhost:8094/wps?service=WPS&version=1.0.0&request=GetCapabilities>.

Check the log files for errors:

```
$ tail -f ~/birdhouse/var/log/pywps/emu.log
$ tail -f ~/birdhouse/var/log/supervisor/emu.log
```

You will find more information about the installation in the [Makefile documentation](#).

## 1.1 Non-default installation

You can customize the installation to use different ports, locations and run user.

To change the anaconda location edit the `Makefile.config`, for example:

```
ANACONDA_HOME ?= /opt/anaconda
CONDA_ENVS_DIR ?= /opt/anaconda/envs
```

You can install emu as root and run it as unprivileged user like `www-data`:

```
root$ mkdir -p /opt/birdhouse/src
root$ cd /opt/birdhouse/src
root$ git clone https://github.com/bird-house/emu.git
root$ cd emu
```

Edit `custom.cfg`:

```
[buildout]
extends = buildout.cfg

[settings]
hostname = emu
http-port = 80
output-port = 500
log-level = WARN

# deployment options
prefix = /opt/birdhouse
user = www-data
etc-user = root
```

Run the installtion and start the services:

```
root$ make clean install
root$ make start          # stop or restart
root$ make status
```

## 1.2 Run Emu as Docker container

You can also run Emu as a Docker container, see the *Tutorial*.



## CHAPTER 2

---

### Configuration

---

If you want to run on a different hostname or port then change the default values in `custom.cfg`:

```
$ cd emu
$ vim custom.cfg
$ cat custom.cfg
[settings]
hostname = localhost
http-port = 8094
```

After any change to your `custom.cfg` you **need** to run `make update` again and restart the supervisor service:

```
$ make update    # or install
$ make restart
```



- *Running unit tests*
- *Running WPS service in test environment*

### 3.1 Running unit tests

Run quick tests:

```
$ make test
```

Run all tests (slow, online):

```
$ make testall
```

Check pep8:

```
$ make pep8
```

### 3.2 Running WPS service in test environment

For development purposes you can run the WPS service without nginx and supervisor. Use the following instructions:

```
# get the source code
$ git clone https://github.com/bird-house/emu.git
$ cd emu

# create conda environment
$ conda env create -f environment.yml
```

```
# activate conda environment
$ source activate emu

# install emu code into conda environment
$ python setup.py develop

# start the WPS service
$ emu

# open your browser on the default service url
$ firefox http://localhost:5000/wps

# ... and service capabilities url
$ firefox http://localhost:5000/wps?service=WPS&request=GetCapabilities
```

The emu service command-line has more options:

```
$ emu -h
```

For example you can start the WPS with enabled debug logging mode:

```
$ emu --debug
```

Or you can overwrite the default **PyWPS** configuration by providing your own PyWPS configuration file (just modify the options you want to change):

```
# edit your local pywps configuration file
$ cat mydev.cfg
[logging]
level = WARN
file = /tmp/mydev.log

# start the service with this configuration
$ emu -c mydev.cfg
```

### 4.1 Tutorial: using Docker

Emu WPS is available as docker image. You can download the docker image from [DockerHub](#) or build it from the provided Dockerfile.

Start the container with the following command:

```
$ docker run -i -d -p 8080:8080 -p 8000:8000 -p 9001:9001 --name=emu birdhouse/emu
```

The ports are:

- PyWPS port: 8080
- NGINX file service port for the outputs: 8000
- Supervisor port: 9001 (optional)

You can map the container port also to another port on your machine, for example: `-p 8094:8080` (your machine port=8094, container port=8080).

Check the docker logs:

```
$ docker logs emu
```

Show running docker containers:

```
$ docker ps
```

Run a GetCapabilities WPS request:

<http://localhost:8080/wps?service=WPS&version=1.0.0&request=getcapabilities>

Run DescribeProcess WPS request for *Hello*:

<http://localhost:8080/wps?service=WPS&version=1.0.0&request=describeprocess&identifier=hello>

Execute *Hello* process with you user name:

```
http://localhost:8080/wps?service=WPS&version=1.0.0&request=execute&identifier=hello&DataInputs=name=Pingu
```

Install *Birdy* WPS command line tool from Anaconda (Anaconda needs to be installed and in your PATH):

```
$ conda install -c birdhouse birdhouse-birdy
```

Use Birdy to access Emu WPS service:

```
$ export WPS_SERVICE=http://localhost:8080/wps
$ birdy -h
$ birdy hello -h
$ birdy hello --name Pingu
```

Stop and remove docker container:

```
$ docker stop emu_wps
$ docker rm emu_wps
```

### 4.1.1 Using docker-compose

Use `docker-compose` (you need a recent version > 1.7) to start the container:

```
$ git clone https://github.com/bird-house/emu.git
$ cd emu
$ docker-compose up -d
$ docker-compose logs emu
```

You can change the ports and hostname with environment variables:

```
$ HOSTNAME=emu HTTP_PORT=8094 SUPERVISOR_PORT=48094 docker-compose up
```

Now the WPS is available on port 8094: <http://emu:8094/wps?service=WPS&version=1.0.0&request=GetCapabilities>.

You can also customize the `docker-compose.yml` file. See the [docker-compose documentation](#).

## 4.2 Tutorial: using postgres database

You can use a postgres database for PyWPS, the default is sqlite. PyWPS is using [SQLAlchemy](#), see the [PYWPS documentation](#) for details.

First run the Emu default installation:

```
$ git clone https://github.com/bird-house/emu.git
$ cd emu
$ make clean install
```

The default installation is using sqlite. We now need a postgres database. If you don't have one yet you can use a [postgres docker container](#).

```
$ docker pull postgres
$ docker run --name postgres -e POSTGRES_PASSWORD=postgres -p 5432:5432 -d postgres
```

The postgres database is now available on default port 5432.

SQLAlchemy needs the `psycopg2` postgres adapter. This was installed by the Emu installation process. You can also install it manually via conda:

```
$ conda install psycopg2
```

The SQLAlchemy connection string for this database is:

```
# postgresql+psycopg2://user:password@host:port/dbname
postgresql+psycopg2://postgres:postgres@localhost:5432/postgres
```

Configure this connection string in `custom.cfg`, `pywps` section, database option:

```
$ vim custom.cfg
[settings]
hostname = localhost
# http-port = 8094
# output-port = 8090

[pywps]
database = postgresql+psycopg2://postgres:postgres@localhost:5432/postgres
```

Update the pywps configuration:

```
$ make update
```

Check the updated pywps configuration (optional):

```
$ less $HOME/birdhouse/etc/pywps/emu.cfg
[logging]
database=postgresql+psycopg2://postgres:postgres@localhost:5432/postgres
```

Start the emu service:

```
$ make restart
```

Your Emu WPS service should be available at the following URL:

```
$ firefox http://localhost:8094/wps?request=GetCapabilities&service=WPS
```





This page is the top-level of your generated API documentation. Below is a list of all items that are documented here.

## 5.1 wsgi

### 5.1.1 Module Contents

`wsgi.application` (*environ*, *start\_response*)

`wsgi.create_app` (*cfgfiles=None*)

## 5.2 demo

### 5.2.1 Module Contents

`demo.get_host` ()

`demo._run` (*application*, *bind\_host=None*, *daemon=False*)

`demo.main` ()

## 5.3 processes

### 5.3.1 Submodules

`processes.wps_bbox`

#### Module Contents

```
class processes.wps_bbox.Box
```

```
    _handler (request, response)
```

`processes.wps_binaryoperator`

#### Module Contents

```
class processes.wps_binaryoperator.BinaryOperator
```

```
    _handler (request, response)
```

`processes.wps_chomsky`

#### Module Contents

```
class processes.wps_chomsky.Chomsky
```

```
    Generates a random chomsky text: http://code.activestate.com/recipes/440546-chomsky-random-text-generator/
```

CHOMSKY is an aid to writing linguistic papers in the style of the great master. It is based on selected phrases taken from actual books and articles written by Noam Chomsky. Upon request, it assembles the phrases in the elegant stylistic patterns that Chomsky is noted for. To generate n sentences of linguistic wisdom, type

(CHOMSKY n) – for example (CHOMSKY 5) generates half a screen of linguistic truth.

```
    _handler (request, response)
```

`processes.wps_dummy`

DummyProcess to check the WPS structure

Author: Jorge de Jesus ([jorge.jesus@gmail.com](mailto:jorge.jesus@gmail.com)) as suggested by Kor de Jong

#### Module Contents

```
class processes.wps_dummy.Dummy
```

```
    _handler (request, response)
```

`processes.wps_error`

### Module Contents

`class processes.wps_error.ShowError`

`_handler (request, response)`

`processes.wps_esgf`

### Module Contents

`class processes.wps_esgf.ESGFDemo`

`_handler (request, response)`

`processes.wps_inout`

### Module Contents

`class processes.wps_inout.InOut`

    This process defines several types of literal type of in- and outputs.

    TODO: add literal input with value range[(0,100)] ... see pywps doc

`_handler (request, response)`

`processes.wps_multiple_outputs`

### Module Contents

`class processes.wps_multiple_outputs.MultipleOutputs`

`_handler (request, response)`

`processes.wps_nap`

### Module Contents

`class processes.wps_nap.Nap`

`_handler (request, response)`

`processes.wps_say_hello`

#### Module Contents

`class processes.wps_say_hello.SayHello`

`_handler (request, response)`

`processes.wps_sleep`

#### Module Contents

`class processes.wps_sleep.Sleep`

`_handler (request, response)`

`processes.wps_ultimate_question`

#### Module Contents

`class processes.wps_ultimate_question.UltimateQuestion`

`_handler (request, response)`

`processes.wps_wordcounter`

#### Module Contents

`class processes.wps_wordcounter.WordCounter`

`_handler (request, response)`

## 5.4 tests

### 5.4.1 Submodules

`tests.common`

#### Module Contents

`class tests.common.WpsTestClient`

`get (*args, **kwargs)`

`tests.common.client_for (service)`

`tests.test_wps_bbox`

### Module Contents

`tests.test_wps_bbox.test_wps_bbox()`

`tests.test_wps_caps`

### Module Contents

`tests.test_wps_caps.test_wps_caps()`

`tests.test_wps_chomsky`

### Module Contents

`tests.test_wps_chomsky.test_wps_chomsky()`

`tests.test_wps_dummy`

### Module Contents

`tests.test_wps_dummy.test_wps_dummy()`

`tests.test_wps_hello`

### Module Contents

`tests.test_wps_hello.test_wps_hello()`

`tests.test_wps_hello.test_wps_hello_again()`

Example of how to debug this process, running outside a PyWPS instance.

`tests.test_wps_inout`

### Module Contents

`tests.test_wps_inout.test_wps_inout()`

`tests.test_wps_ultimate_question`

### Module Contents

`tests.test_wps_ultimate_question.test_wps_ultimate_question()`

`tests.test_wps_wordcounter`

## Module Contents

`tests.test_wps_wordcounter.test_wps_wordcount()`

### d

[demo](#), [13](#)

### p

[processes](#), [14](#)  
[processes.wps\\_bbox](#), [14](#)  
[processes.wps\\_binaryoperator](#), [14](#)  
[processes.wps\\_chomsky](#), [14](#)  
[processes.wps\\_dummy](#), [14](#)  
[processes.wps\\_error](#), [15](#)  
[processes.wps\\_esgf](#), [15](#)  
[processes.wps\\_inout](#), [15](#)  
[processes.wps\\_multiple\\_outputs](#), [15](#)  
[processes.wps\\_nap](#), [15](#)  
[processes.wps\\_say\\_hello](#), [16](#)  
[processes.wps\\_sleep](#), [16](#)  
[processes.wps\\_ultimate\\_question](#), [16](#)  
[processes.wps\\_wordcounter](#), [16](#)

### t

[tests](#), [16](#)  
[tests.common](#), [16](#)  
[tests.test\\_wps\\_bbox](#), [17](#)  
[tests.test\\_wps\\_caps](#), [17](#)  
[tests.test\\_wps\\_chomsky](#), [17](#)  
[tests.test\\_wps\\_dummy](#), [17](#)  
[tests.test\\_wps\\_hello](#), [17](#)  
[tests.test\\_wps\\_inout](#), [17](#)  
[tests.test\\_wps\\_ultimate\\_question](#), [17](#)  
[tests.test\\_wps\\_wordcounter](#), [18](#)

### w

[wsgi](#), [13](#)





## Symbols

[\\_handler\(\)](#) (processes.wps\_bbox.Box method), 14  
[\\_handler\(\)](#) (processes.wps\_binaryoperator.BinaryOperator method), 14  
[\\_handler\(\)](#) (processes.wps\_chomsky.Chomsky method), 14  
[\\_handler\(\)](#) (processes.wps\_dummy.Dummy method), 14  
[\\_handler\(\)](#) (processes.wps\_error.ShowError method), 15  
[\\_handler\(\)](#) (processes.wps\_esgf.ESGFDemo method), 15  
[\\_handler\(\)](#) (processes.wps\_inout.InOut method), 15  
[\\_handler\(\)](#) (processes.wps\_multiple\_outputs.MultipleOutputs method), 15  
[\\_handler\(\)](#) (processes.wps\_nap.Nap method), 15  
[\\_handler\(\)](#) (processes.wps\_say\_hello.SayHello method), 16  
[\\_handler\(\)](#) (processes.wps\_sleep.Sleep method), 16  
[\\_handler\(\)](#) (processes.wps\_ultimate\_question.UltimateQuestion method), 16  
[\\_handler\(\)](#) (processes.wps\_wordcounter.WordCounter method), 16  
[\\_run\(\)](#) (in module demo), 13

## A

[application\(\)](#) (in module wsgi), 13

## B

[BinaryOperator](#) (class in processes.wps\_binaryoperator), 14  
[Box](#) (class in processes.wps\_bbox), 14

## C

[Chomsky](#) (class in processes.wps\_chomsky), 14  
[client\\_for\(\)](#) (in module tests.common), 16  
[create\\_app\(\)](#) (in module wsgi), 13

## D

[demo](#) (module), 13  
[Dummy](#) (class in processes.wps\_dummy), 14

## E

[ESGFDemo](#) (class in processes.wps\_esgf), 15

## G

[get\(\)](#) (tests.common.WpsTestClient method), 16  
[get\\_host\(\)](#) (in module demo), 13

## I

[InOut](#) (class in processes.wps\_inout), 15

## M

[main\(\)](#) (in module demo), 13  
[MultipleOutputs](#) (class in processes.wps\_multiple\_outputs), 15

## N

[Nap](#) (class in processes.wps\_nap), 15

## P

[processes](#) (module), 14  
[processes.wps\\_bbox](#) (module), 14  
[processes.wps\\_binaryoperator](#) (module), 14  
[processes.wps\\_chomsky](#) (module), 14  
[processes.wps\\_dummy](#) (module), 14  
[processes.wps\\_error](#) (module), 15  
[processes.wps\\_esgf](#) (module), 15  
[processes.wps\\_inout](#) (module), 15  
[processes.wps\\_multiple\\_outputs](#) (module), 15  
[processes.wps\\_nap](#) (module), 15  
[processes.wps\\_say\\_hello](#) (module), 16  
[processes.wps\\_sleep](#) (module), 16  
[processes.wps\\_ultimate\\_question](#) (module), 16  
[processes.wps\\_wordcounter](#) (module), 16

## S

[SayHello](#) (class in processes.wps\_say\_hello), 16  
[ShowError](#) (class in processes.wps\_error), 15  
[Sleep](#) (class in processes.wps\_sleep), 16

## T

`test_wps_bbox()` (in module `tests.test_wps_bbox`), [17](#)  
`test_wps_caps()` (in module `tests.test_wps_caps`), [17](#)  
`test_wps_chomsky()` (in module `tests.test_wps_chomsky`), [17](#)  
`test_wps_dummy()` (in module `tests.test_wps_dummy`), [17](#)  
`test_wps_hello()` (in module `tests.test_wps_hello`), [17](#)  
`test_wps_hello_again()` (in module `tests.test_wps_hello`), [17](#)  
`test_wps_inout()` (in module `tests.test_wps_inout`), [17](#)  
`test_wps_ultimate_question()` (in module `tests.test_wps_ultimate_question`), [17](#)  
`test_wps_wordcount()` (in module `tests.test_wps_wordcounter`), [18](#)  
`tests` (module), [16](#)  
`tests.common` (module), [16](#)  
`tests.test_wps_bbox` (module), [17](#)  
`tests.test_wps_caps` (module), [17](#)  
`tests.test_wps_chomsky` (module), [17](#)  
`tests.test_wps_dummy` (module), [17](#)  
`tests.test_wps_hello` (module), [17](#)  
`tests.test_wps_inout` (module), [17](#)  
`tests.test_wps_ultimate_question` (module), [17](#)  
`tests.test_wps_wordcounter` (module), [18](#)

## U

`UltimateQuestion` (class in `processes.wps_ultimate_question`), [16](#)

## W

`WordCounter` (class in `processes.wps_wordcounter`), [16](#)  
`WpsTestClient` (class in `tests.common`), [16](#)  
`wsgi` (module), [13](#)