Malleefowl Documentation

Release 0.6

Birdhouse

Jan 15, 2018

Contents

Malleefowl (the bird) *Malleefowl are shy, wary, solitary birds that usually fly only to escape danger or reach a tree to roost in. Although very active, they are seldom seen [..]* (Wikipedia).

Malleefowl is a Web Processing Service with a collection of processes to access climate data (ESGF, Thredds Catalogs, . . .).

Malleefowl is part of the Birdhouse project.

Contents:

CHAPTER 1

Installation

Check out code from the malleefowl github repo and start the installation:

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

For other install options run `make help` and read the documention of the Makefile. All installation files are going by default into the folder `~/birdhouse`.

After successful installation you need to start the services:

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

The depolyed WPS service is available at:

http://localhost:8091/wps?service=WPS&version=1.0.0&request=GetCapabilities.

Check the log files for errors:

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

CHAPTER 2

Configuration

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

```
$ cd malleefowl
$ vim custom.cfg
$ cat custom.cfg
[settings]
hostname = localhost
http-port = 8091
```

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
$ make status
```

CHAPTER 3

Developer Guide

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/malleefowl.git
$ cd malleefowl

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

```
# activate conda environment
$ source activate malleefowl

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

# start the WPS service
$ malleefowl

# 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 `malleefowl` service command-line has more options:

```
$ malleefowl -h
```

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

```
$ malleefowl --debug
```

Or you can overwrite the default PyWPS configuration by providing your own PyWPS configuration file (just modifiy 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
$ malleefowl -c mydev.cfg
```

CHAPTER 4

Using Docker

To run Malleefowl Web Processing Service you can also use the Docker image:

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

Check the docker logs:

```
$ docker logs malleefowl
```

Show running docker containers:

```
$ docker ps
```

Open your browser and enter the url of the supervisor service:

> http://localhost:9001/

Run a GetCapabilites WPS request:

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

4.1 Using docker-compose

Start malleefowl with docker-compose (docker-compose version > 1.7):

```
$ docker-compose up
```

By default the WPS is available on port 8080: http://localhost:8080/wps?service=WPS&version=1.0.0&request=GetCapabilities.

You can change the ports and hostname with environment variables:

```
$ HOSTNAME=malleefowl HTTP_PORT=8091 SUPERVISOR_PORT=48091 docker-compose up
```

Now the WPS is available on port 8091: http://malleefowl:8091/wps?service=WPS&version=1.0.0&request=GetCapabilities.

CHAPTER 5

Tutorials

5.1 Using the *download* Process

Go through this tutorial step by step.

5.1.1 Step 0: Install malleefowl with defaults

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

# run the installation
$ make clean install
```

```
# start the service
$ make start

# open the capabilities document
$ firefox http://localhost:8091/wps?service=WPS&request=GetCapabilities
```

5.1.2 Step 1: Install birdy

We are using birdy in the examples, a WPS command line client.

```
# install it via conda
$ conda install -c birdhouse birdhouse-birdy
```

5.1.3 Step 2: Check if birdy works

```
# point birdy to the malleefowl service url
$ export WPS_SERVICE=http://localhost:8091/wps
# show a list of available command (wps processes)
$ birdy -h
```

5.1.4 Step 3: Run the download process

Make sure birdy works and is pointing to malleefowl . . . see above.

```
# show the description of the download process
$ birdy download -h

# download a netcdf file from a public thredds service
$ birdy download --resource \
    https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis2/
↪surface/mslp.1979.nc
```

5.1.5 Step 4: Install Phoenix

Phoenix is a web client for WPS and comes by default with an WPS security proxy (twitcher).

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

5.1.6 Step 5: Login to Phoenix

```
# login ... by default admin password is "qwerty"
$ firefox https://localhost:8443/account/login
```

5.1.7 Step 6: Copy the twitcher access token in Phoenix

1. Go to your profile.
2. Choose the `Twitcher access token` tab.
3. Copy the access token.

5.1.8 Step 7: Access malleefowl behind the OWS proxy with access token

```
# configure wps service
$ export WPS_SERVICE=https://localhost:8443/ows/proxy/malleefowl

# check if it works
$ birdy -h

# run the download again ... you need the access token
$ birdy \
    --token 3d8c24eeebb143b3a199ba8a0e045f93 \
    download --resource \
    https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis2/
↪surface/mslp.1979.nc
```

5.1.9 Step 8: Get a ESGF certificate using Phoenix

1. Go to your profile.
2. Choose the `ESGF credentials` tab.
3. Use the green button `Update credentials`.
4. Choose your ESGF provider, enter your account details and press `Submit`.

5.1.10 Step 9: Download a file from ESGF

Make sure birdy works and points to the proxy url of malleefowl . . . see above.

Choose a file from the ESGF archive you would like to download and make sure you have dowload permissions.

You can choose the ESGF search browser in Phoenix or an ESGF portal.

```
# try the download ... in this example with a CORDEX file.
# make sure your twitcher token and your ESGF cert are still valid.
$ birdy \
    --token 3d8c24eeebb143b3a199ba8a0e045f93 \
    download --resource \
    http://esgf1.dkrz.de/thredds/fileServer/cordex/cordex/output/EUR-44/MPI-CSC/MPI-M-
↪MPI-ESM-LR/historical/r1i1p1/MPI-CSC-REMO2009/v1/mon/tas/v20150609/tas_EUR-44_MPI-M-
↪MPI-ESM-LR_historical_r1i1p1_MPI-CSC-REMO2009_v1_mon_200101-200512.nc
```

5.2 Debugging the *download* Process

Go through this tutorial step by step.

5.2.1 Step 0: Install malleefowl in debug mode

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

# create conda env
$ conda env create

# activate malleefowl env
$ source activate malleefowl

# install malleefowl package in develop mode
$ python setup.py develop

# check if the demo service is available
$ malleefowl -h
```

5.2.2 Step 1: Start the malleefowl demo service

You might do this more often when debugging. Make sure you are in the malleefowl conda env.

```
# start service
$ malleefowl

# open the capabilities document
$ firefox http://localhost:5000/wps?service=WPS&request=GetCapabilities
```

The service is started in debug mode. See the Werkzeug documenation how to work with this.

You can stop the service with `CTRL-c`. The service is automatically restarted on source changes.

5.2.3 Step 2: Install birdy

We are using birdy in the examples, a WPS command line client.

```
# install it via conda
$ conda install -c birdhouse birdhouse-birdy
```

5.2.4 Step 3: Check if birdy works

```
# point birdy to the malleefowl service url
$ export WPS_SERVICE=http://localhost:5000/wps
# show a list of available command (wps processes)
$ birdy -h
```

5.2.5 Step 4: Run the download process

Make sure birdy works and is pointing to malleefowl . . . see above.

```
# show the description of the download process
$ birdy download -h

# download a netcdf file from a public thredds service
$ birdy download --resource \
    https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis2/
↪surface/mslp.1979.nc
```

5.2.6 Step 5: Install Phoenix

Phoenix is a web client for WPS and comes by default with an WPS security proxy (twitcher).

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

5.2.7 Step 6: Login to Phoenix

```
# login ... by default admin password is "qwerty"
$ firefox https://localhost:8443/account/login
```

5.2.8 Step 7: Register your WPS demo service

Go to the registration page: https://localhost:8443/services/register

Register your service with the following parameters:

- Service URL: http://localhost:5000/wps
- Service Name: demo

5.2.9 Step 8: Copy the twitcher access token in Phoenix

1. Go to your profile.
2. Choose the `Twitcher access token` tab.
3. Copy the access token.

5.2.10 Step 9: Access demo service behind the OWS proxy with access token

```
# configure wps service
$ export WPS_SERVICE=https://localhost:8443/ows/proxy/demo

# check if it works
$ birdy -h

# run the download again ... you need the access token
$ birdy \
    --token 3d8c24eeebb143b3a199ba8a0e045f93 \
    download --resource \
    https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis2/
↪surface/mslp.1979.nc
```

5.2.11 Step 10: Get an ESGF certificate using Phoenix

1. Go to your profile.
2. Choose the `ESGF credentials` tab.
3. Use the green button `Update credentials`.
4. Choose your ESGF provider, enter your account details and press `Submit`.

5.2.12 Step 11: Download a file from ESGF

Make sure birdy works and points to the proxy url of demo service ... see above.

Choose a file from the ESGF archive you would like to download and make sure you have dowload permissions.

You can choose the ESGF search browser in Phoenix or an ESGF portal.

```
# try the download ... in this example with a CORDEX file.
# make sure your twitcher token and your ESGF cert are still valid.
$ birdy \
    --token 3d8c24eeebb143b3a199ba8a0e045f93 \
    download --resource \
    http://esgf1.dkrz.de/thredds/fileServer/cordex/cordex/output/EUR-44/MPI-CSC/MPI-M-
↪MPI-ESM-LR/historical/r1i1p1/MPI-CSC-REMO2009/v1/mon/tas/v20150609/tas_EUR-44_MPI-M-
↪MPI-ESM-LR_historical_r1i1p1_MPI-CSC-REMO2009_v1_mon_200101-200512.nc
```

You can also try this in WPS synchronous mode when your process is not long running:

```
$ birdy \
    --sync \
    --token 3d8c24eeebb143b3a199ba8a0e045f93 \
```

```
    download --resource \
    http://esgf1.dkrz.de/thredds/fileServer/cordex/cordex/output/EUR-44/MPI-CSC/MPI-M-
↪MPI-ESM-LR/historical/r1i1p1/MPI-CSC-REMO2009/v1/mon/tas/v20150609/tas_EUR-44_MPI-M-
↪MPI-ESM-LR_historical_r1i1p1_MPI-CSC-REMO2009_v1_mon_200101-200512.nc
```

... and with `debug` option to see more log message:

```
$ birdy \
    --sync \
    --debug \
    --token 3d8c24eeebb143b3a199ba8a0e045f93 \
    download --resource \
    http://esgf1.dkrz.de/thredds/fileServer/cordex/cordex/output/EUR-44/MPI-CSC/MPI-M-
↪MPI-ESM-LR/historical/r1i1p1/MPI-CSC-REMO2009/v1/mon/tas/v20150609/tas_EUR-44_MPI-M-
↪MPI-ESM-LR_historical_r1i1p1_MPI-CSC-REMO2009_v1_mon_200101-200512.nc
```

CHAPTER 6

Sphinx AutoAPI Index

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

6.1 wsgi

6.1.1 Module Contents

wsgi.**application**(*environ*, *start_response*)

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

6.2 exceptions

6.2.1 Module Contents

class ProcessFailed

6.3 demo

6.3.1 Module Contents

demo.**get_host**()

demo.**_run**(*application*, *daemon=False*)

demo.**main**()

6.4 workflow

6.4.1 Module Contents

class workflow.**MonitorPE**(*output=None*)

set_monitor(*monitor*, *start_progress=0*, *end_progress=100*)

class workflow.**GenericWPS**(*url*, *identifier*, *resource="resource"*, *inputs=list*, *output=None*, *headers=None*)

progress(*execution*)

monitor_execution(*execution*)

_build_wps_inputs()

_build_wps_outputs()

execute()

_set_inputs(*inputs*)

process(*inputs*)

_process(*inputs*)

class workflow.**EsgSearch**(*url*, *search_url="https://esgf-data.dkrz.de/esg-search"*, *constraints="project:CORDEX"*, *query=None*, *limit=100*, *search_type="File"*, *distrib=False*, *replica=False*, *latest=True*, *temporal=False*, *start=None*, *end=None*)

_process(*inputs*)

class workflow.**SolrSearch**(*url*, *query*, *filter_query=None*)
Run search against birdhouse solr index and return a list of download urls.

process(*inputs*)

class workflow.**Download**(*url*, *headers=None*)

_process(*inputs*)

class workflow.**ThreddsDownload**(*url*, *catalog_url*, *headers=None*)

_process(*inputs*)

workflow.**esgf_workflow**(*source*, *worker*, *monitor=None*, *headers=None*)

workflow.**thredds_workflow**(*source*, *worker*, *monitor=None*, *headers=None*)

workflow.**solr_workflow**(*source*, *worker*, *monitor=None*, *headers=None*)

workflow.**run**(*workflow*, *monitor=None*, *headers=None*)

6.5 download

TODO: handle parallel downloads

6.5.1 Module Contents

download.**download_with_archive**(*url*, *credentials=None*)
Downloads file. Checks before downloading if file is already in local esgf archive.

download.**download**(*url*, *use_file_url=False*, *credentials=None*)
Downloads url and returns local filename.

Parameters

- **url** – url of file
- **use_file_url** – True if result should be a file url "file://", otherwise use system path.
- **credentials** – path to credentials if security is needed to download file

Returns downloaded file with either file:// or system path

download.**wget**(*url*, *use_file_url=False*, *credentials=None*)
Downloads url and returns local filename.

TODO: refactor cache handling.

Parameters

- **url** – url of file
- **use_file_url** – True if result should be a file url "file://", otherwise use system path.
- **credentials** – path to credentials if security is needed to download file

Returns downloaded file with either file:// or system path

download.**download_files**(*urls=list*, *credentials=None*, *monitor=None*)

download.**download_files_from_thredds**(*url*, *recursive=False*, *monitor=None*)

class download.**DownloadManager**(*monitor=None*)

show_status(*message*, *progress*)

threader()

download_job(*url*, *credentials*)

download(*urls*, *credentials=None*)

6.6 config

6.6.1 Module Contents

config.**wps_url**()

config.**cache_path**()

config.**archive_root**()

config.**archive_node**()

6.7 utils

Utility functions for WPS processes.

6.7.1 Module Contents

utils.**esgf_archive_path**(*url*)

utils.**dupname**(*path*, *filename*)
avoid dupliate filenames TODO: needs to be improved

utils.**user_id**(*openid*)
generate user_id from openid

utils.**within_date_range**(*timesteps*, *start=None*, *end=None*)

utils.**filter_timesteps**(*timesteps*, *aggregation="monthly"*, *start=None*, *end=None*)

utils.**nc_copy**(*source*, *target*, *overwrite=True*, *time_dimname="time"*, *nchunk=10*, *istart=0*, *istop=-1*, *format="NETCDF3_64BIT"*)
copy netcdf file from opendap to netcdf3 file

Parameters

- **overwrite** – Overwite destination file (default is to raise an error if output file already exists).
- **format** – netcdf3 format to use (NETCDF3_64BIT by default, can be set to NETCDF3_CLASSIC)
- **chunk** – number of records along unlimited dimension to write at once. Default 10. Ignored if there is no unlimited dimension. chunk=0 means write all the data at once.
- **istart** – number of record to start at along unlimited dimension. Default 0. Ignored if there is no unlimited dimension.
- **istop** – number of record to stop at along unlimited dimension. Default -1. Ignored if there is no unlimited dimension.

6.8 esgf

6.8.1 Submodules

esgf.logon

This module is used to get esgf logon credentials. There are two choices:

- a proxy certificate from a myproxy server with an ESGF openid.
- OpenID login as used in browsers.

Some of the code is taken from esgf-pyclient: https://github.com/ESGF/esgf-pyclient

See also:

- open climate workbench: https://github.com/apache/climate
- MyProxyLogon: https://github.com/cedadev/MyProxyClient

Module Contents

esgf.logon.**myproxy_logon_with_openid**(*openid*, *password=None*, *interactive=False*, *outdir=None*)
Tries to get MyProxy parameters from OpenID and calls `logon()`.

Parameters **openid** – OpenID used to login at ESGF node.

esgf.logon.**parse_openid**(*openid*, *ssl_verify=False*)
parse openid document to get myproxy service

esgf.logon.**cert_infos**(*filename*)

esgf.search

Module Contents

esgf.search.**date_from_filename**(*filename*)
Example cordex: tas_EUR-44i_ECMWF-ERAINT_evaluation_r1i1p1_HMS-ALADIN52_v1_mon_200101-200812.nc

esgf.search.**variable_filter**(*constraints*, *variables*)
return True if variable fulfills contraints

esgf.search.**temporal_filter**(*filename*, *start_date=None*, *end_date=None*)
return True if file is in timerange start/end

class esgf.search.**ESGSearch**(*url="http://localhost:8081/esg-search"*, *distrib=False*, *replica=False*, *latest=True*, *monitor=None*)
wrapper for esg search.

TODO: bbox constraint for datasets

show_status(*message*, *progress*)

search(*constraints=list*, *query=None*, *start=None*, *end=None*, *limit=1*, *offset=0*, *search_type="Dataset"*, *temporal=False*)

_index(*datasets*, *limit*, *offset*)

_file_context(*dataset*)

_aggregation_context(*dataset*)

threader()

_file_search_job(*f_ctx*, *start_date*, *end_date*)

_file_search(*datasets*, *constraints*, *start_date*, *end_date*)

_aggregation_search(*datasets*, *constraints*)

6.9 processes

6.9.1 Submodules

processes.wps_download

Module Contents

class processes.wps_download.**Download**

The download process gets as input a list of URLs pointing to NetCDF files which should be downloaded.

The downloader first checks if the file is available in the local ESGF archive or cache. If not then the file will be downloaded and stored in a local cache. As a result it provides a list of local `file://` paths to the requested files.

The downloader does not download files if they are already in the ESGF archive or in the local cache.

_handler(*request*, *response*)

processes.wps_esgsearch

Module Contents

class processes.wps_esgsearch.**ESGSearchProcess**

The ESGF search process runs a ESGF search request with constraints (project, experiment, ...) to get a list of matching files on ESGF data nodes. It is using esgf-pyclient Python client for the ESGF search API.

In addition to the esgf-pyclient the process checks if local replicas are available and would return the replica files instead of the original one.

The result is a JSON document with a list of `http://` URLs to files on ESGF data nodes.

TODO: bbox constraint for datasets

_handler(*request*, *response*)

processes.wps_thredds

Module Contents

class processes.wps_thredds.**ThreddsDownload**

_handler(*request*, *response*)

processes.wps_workflow

Module Contents

class processes.wps_workflow.**DispelWorkflow**

The workflow process is usually called by the Phoenix WPS web client to run WPS process for climate data (like cfchecker, climate indices with ocgis, ...) with a given selection of input data (currently NetCDF files from ESGF data nodes).

Currently the Dispel4Py workflow engine is used.

The Workflow for ESGF input data is as follows:

Search ESGF files -> Download ESGF files -> Run choosen process on local (downloaded) ESGF files.

_handler(*request*, *response*)

6.10 tests

6.10.1 Submodules

tests.common

Module Contents

class tests.common.**WpsTestClient**

get(**args*, ***kwargs*)

tests.common.**client_for**(*service*)

tests.test_download

Module Contents

tests.test_download.**test_download**()

tests.test_download.**test_download_with_file_url**()

tests.test_esgf_logon

Module Contents

tests.test_esgf_logon.**test_parse_openid**()

tests.test_esgf_search

Module Contents

class tests.test_esgf_search.**EsgDistribSearchTestCase**

setUpClass()

test_file_cmip5_with_local_replica()

class tests.test_esgf_search.**EsgSearchTestCase**

setUpClass()

test_dataset()

test_file()

test_file_cmip5_many()

test_file_more_than_one()

test_aggregation()

test_file_cordex()

test_file_cordex_date()

test_file_cordex_many()

test_file_cordex_fx()

test_file_cordex_fly()

test_file_cmip5()

test_file_cmip5_date()

tests.test_utils

Module Contents

tests.test_utils.**test_esgf_archive_path_cordex**()

tests.test_utils.**test_esgf_archive_path_cmip5**()

tests.test_utils.**test_esgf_archive_path_cmip5_noaa**()

tests.test_utils.**test_dupname**()

tests.test_utils.**test_user_id**()

tests.test_utils.**test_within_date_range**()

tests.test_utils.**test_filter_timesteps**()

tests.test_utils.**test_filter_timesteps2**()

tests.test_utils.**test_nc_copy**()

tests.test_wps_caps

Module Contents

tests.test_wps_caps.**test_wps_caps**()

tests.test_wps_download

Module Contents

tests.test_wps_download.**test_wps_download**()

tests.test_wps_esgsearch

Module Contents

tests.test_wps_esgsearch.**test_dataset**()

tests.test_wps_esgsearch.**test_dataset_with_spaces**()

tests.test_wps_esgsearch.**test_dataset_out_of_limit**()

tests.test_wps_esgsearch.**test_dataset_out_of_offset**()

tests.test_wps_esgsearch.**test_dataset_latest**()

tests.test_wps_esgsearch.**test_dataset_query**()

tests.test_wps_esgsearch.**test_aggregation**()

tests.test_wps_esgsearch.**test_file**()

tests.test_wps_thredds

Module Contents

tests.test_wps_thredds.**test_wps_thredds_download**()

tests.test_wps_workflow

Module Contents

tests.test_wps_workflow.**test_wps_thredds_workflow**()

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

Symbols

A

C

D

E

F

G

U

V

W