Birdhouse Documentation

Release 0.7.0

Birdhouse

Feb 18, 2021

CONTENTS

Birdhouse is a GitHub organization comprised of Python projects related to Web Processing Services to support climate data analysis.

The full documentation is available on ReadTheDocs and in the *docs/* folder.

CHAPTER

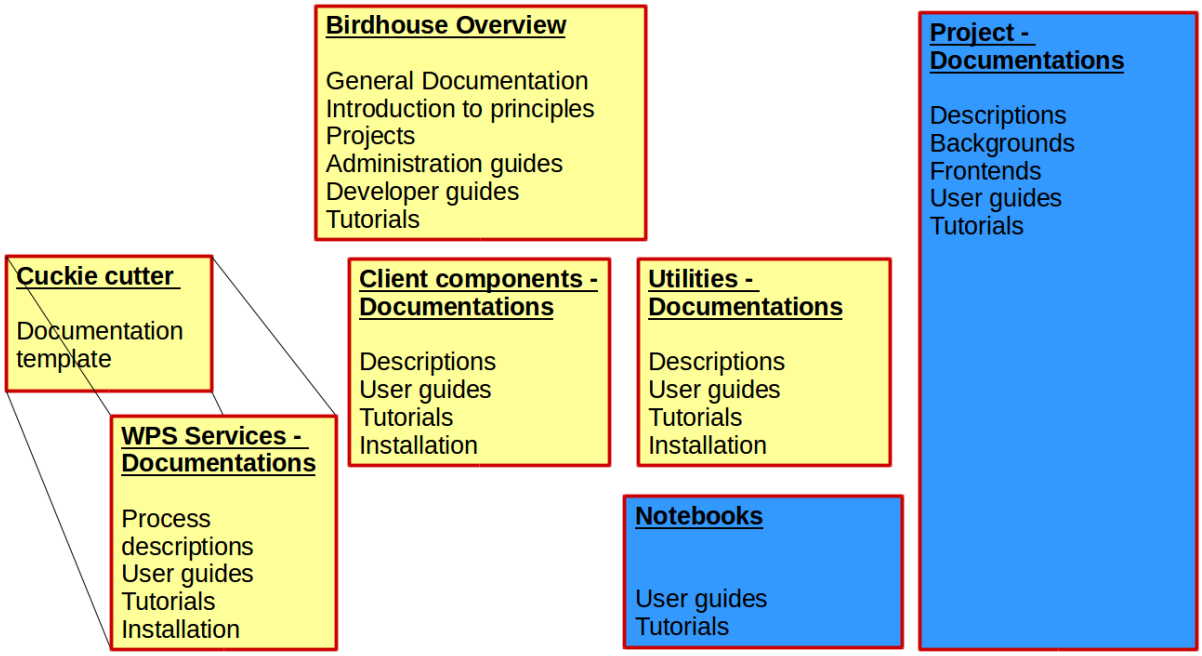
ONE

OVERVIEW

Birdhouse is a collaborative project open for the community to participate. It is a software framework containing a collection of Web Processing Service (WPS). The deployed algorithms are focusing on Earth Systems and environmental data processing with the philosophy of streamlining the software development and deployment. By supporting climate, earth observation and biodiversity data and processes, Birdhouse can be used in a wide array of Earth sciences projects and workflows. The core benefit of this project is to allow the seamless use of climate services developed by a diverse network of national meteorological offices, regional climate service providers, academics, not-for-profit research centers and private industry. As governments move toward open-data policies, there will be a need for analytical services that extract value out of the deluge of information. Using an interoperable software architecture, institutions can provide both data and services allowing users to process the data remotely from a laptop, instead of having to acquire and maintain large storage infrastructures.

1.1 Documentation structure

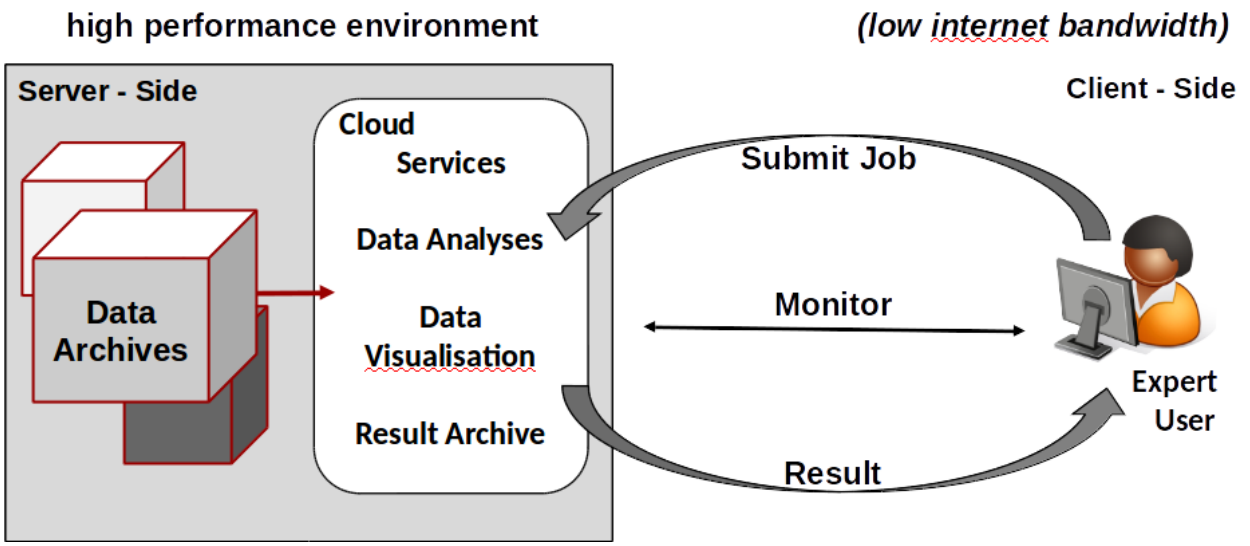
The birdhouse documentation reflects the fact that it is an assemblage of independent software components. It's therefore organized according to the *birdhouse framework structure*. Birdhouse is being used by international working groups who deploy subsets of components tailored to their user base. The following graph shows an overview of the documentation's organization:



1.2 What is WPS?

Geographic Information Processing for the Web *The Web Processing Service (WPS) offers a simple web-based method of finding, accessing, and using all kinds of calculations and models.*

A WPS is a technical solution (WPS Concepts) in which processes are hosted on a server and accessed over the web (Fig. 1). These processes con- form to a standardized format, ensuring that they follow the principle of reusable design: they can be instantiated multiple times for different input arguments or data sources, customized following the same structure to handle new inputs, and are modular, hence can be combined to form new processes. In addition, a WPS can be installed close to the data to enable processing directly out of the archive. A WPS can also be linked to a theoretically limit- less combination of several other WPSs, or generally OpenGIS Web Services (OWS). Our understanding of process is used in the same sense as in the OGC standard: 'for any algorithm, calculation or model that either generates new data or trans- forms some input data into output data'. A submitted process is a job. A service provides a collection of processes containing scientific methods that focus on climate impact and extreme weather events. A combination of processes is called a workflow, and a collection of WPS-related software compartments is a framework. WPS divides the operation into server and client side, with appropriate security in between to avoid misuse.

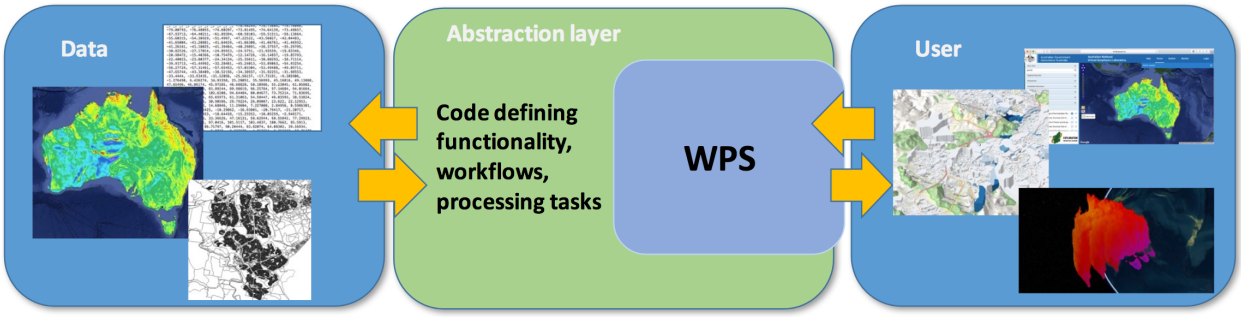


Note: Read the documentation on Geographic Information Processing for the Web

1.3 WPS Use Case

Todo: needs to be updated.

A user runs WPS processes *remotely* on a machine with direct access to climate data archives.



CHAPTER

TWO

ARCHITECTURE

Birdhouse is organized in separate stand-alone software components. Most components are named after birds, which gives the project its name birdhouse. The components can be categorized into *Client Side Components*, i.e. tools for end-users, and *Server Side Components*, i.e. back-end elements of the architecture.

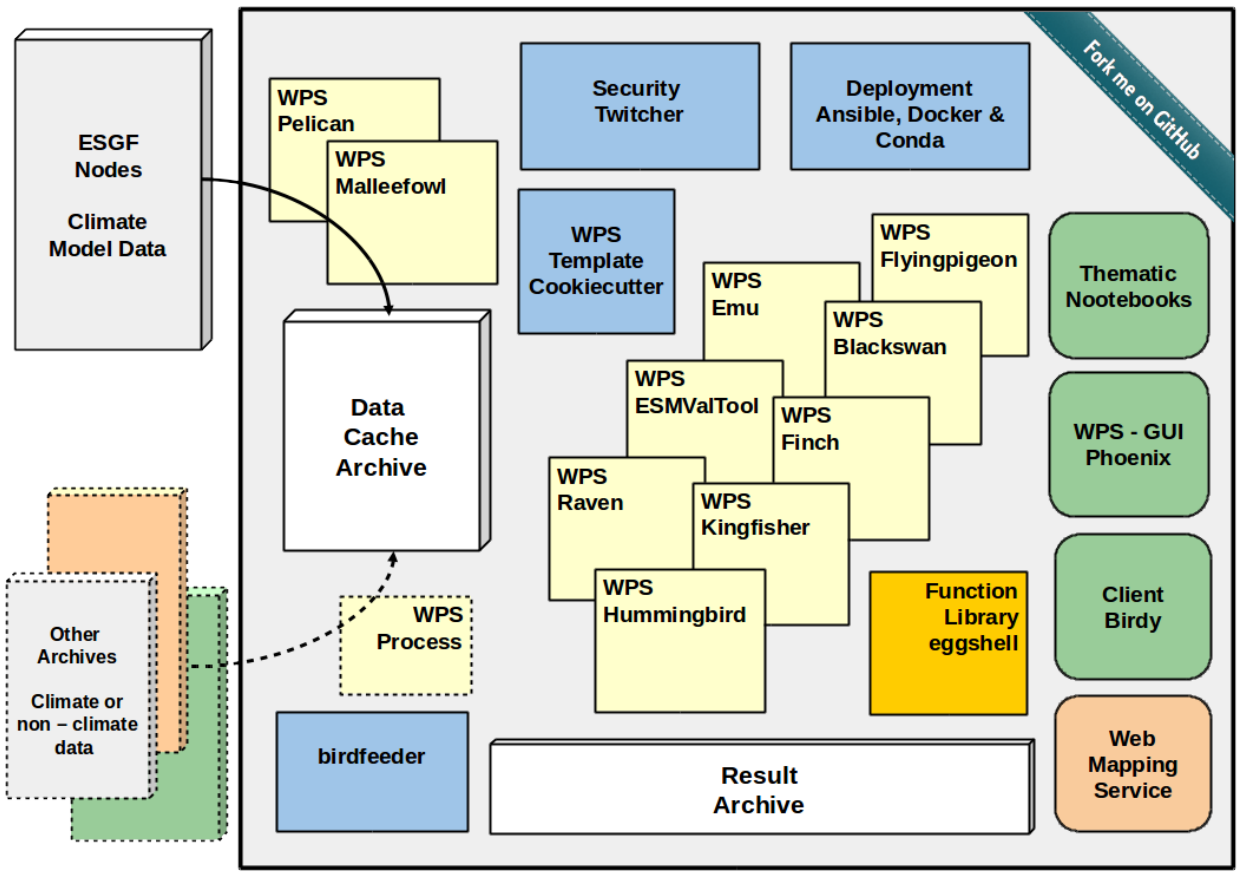
2.1 Framework structure

There are several WPS services. Malleefowl is the main one for the Phoenix client. Malleefowl is used to search, download (with caching) ESGF data and to retrieve certificates. Malleefowl has also a workflow engine (dispel4py) to chain WPS processes.

The results of the WPS processes are stored on the file system and are accessible via URL (with a token id). Results can be shown on a Map using a Web Mapping Service (ncWMS, adagucserver). The PyCSW Catalog Service is used to register WPS services and also to publish WPS outputs. Published results in the PyCSW can also used as input source for processes again.

ESGF is currently the main climate data resource (but more resources are possible). ESGF Solr-index is used to find ESGF data. The ESGF identity provider with OpenIDs and X509 certificate is used for authentication.

WPS serivces can be accessed through web-applications like Phoenix or from scripts.



Note: See also the *Publications and Presentations* for more information and details.

2.2 Client Side Components

- Phoenix: a web-based WPS client with ESGF data access
- Birdy: a WPS command line client and native library

2.3 Server Side Components

WPS services for climate data analysis:

- Emu: some example WPS processes for demo
- Flyingpigeon: Testbed for new process development
- Black Swan: services for the extreme weather event assessments
- Hummingbird: provides cdo and compliance-checker as a service
- Finch: services for climate indices calculation

- Pelican: Supporting ESGF compute API
- Kingfisher: Services for Earth-Observation data analysis

Many climate analysis operations are implemented using OpenClimateGIS including the python package icclim.

Supporting Services and libraries:

- Twitcher: an OWS Security Proxy
- Malleefowl: access to climate data (ESGF, ...) as a service
- Eggshell: provides common functionallity for Birdhouse WPS services

You can find the source code of all birdhouse components on GitHub. Docker images with birdhouse components are on Docker Hub

2.4 Files and Folders

This is an overview of folder structure and important files for administration of a server-side birdhouse ecosystem.

It is recommended to clone the separated WPS services (birds) into one top level folder like:

```
$ ~/birdhouse/emu
$ ~/birdhouse/pyramid-pheonix
$ ~/birdhouse/finch
$ ~/birdhouse/malleefowl
...
```

The dependencies of each bird is deployed as *conda environment* and per default located at:

```
$ ~/.conda/envs/
```

The environment of a bird is defined in *./{birdname}/environment.yml*.

Process descriptions are placed in *./{birdname}/{birdname}/processes/* while modules designed and used for the service are situated in *./{birdname}/{birdname}/*. Here are also static data like shapefiles, templates or additional data used by the processes.

```
$ ./{birdname}/{birdname}/data/shapefiles
$ ./{birdname}/{birdname}/templates
```

Each birdhouse compartment has a documentation build with *Sphinx* and the corresponding files are situated in

```
$ ./{birdname}/docs
```

When running a service, files and folders for input data, result storage, file cache of simply logfiles are defined in the *./{birdname}/.config.cfg*. Default configuration is defined in *./{birdname}/{birdname}/default.cfg* as well as an example can be found in *~./{birdname}/etc*. For more options of configuration see the pywps configuration instructions

For development and deployment testing the installations be checked running tests (*make test*). Test descriptions testdata are situated in:

```
$ ./{birdname}/tests
$ ./{birdname}/tests/testdata
```

CHAPTER

THREE

GUIDELINES

Note: **Code of Conduct:** Before we start please be aware that contributors to this project are expected to act respectfully toward others in accordance with the OSGeo Code of Conduct.

3.1 General Guidelines:

3.1.1 Best practices in FOSS development

- It is agreed that the development guidelines should remain very simple. The typical contribution workflow is to start a branch from the current master (in a fork for external developers) and the pull request is also made to master.
- New features must have documentation and tests. Some PEP8 requirements (with exceptions, which are described by project) must be followed.
- A need for a tutorial or template on how to properly log events inside a WPS and how to write WPS tests is identified.
- The release cycle for birdhouse is roughly 2-3 months, coinciding with the video conference meetings.
- There is a suggestion to clean up repositories that have a lot of obsolete branches. Deleted branches still maintain their commits in the history if they were merged at some point.
- It is strongly suggested that before creating a feature branch to work on, there should be an issue created to explain & track what is being done for that feature.

3.1.2 Contribution Workflow

The Birdhouse project openly welcomes contributions (bug reports, bug fixes, code enhancements/features, etc.). This document will outline some guidelines on contributing to birdhouse. As well, the birdhouse *Communication* is a great place to get an idea of how to connect and participate in birdhouse community and development where everybody is welcome to rise questions and discussions.

3.2 FAIR Guiding Principles

Climate datasets rapidly grow in volume and complexity and creating climate products requires high bandwidth, massive storage and large compute resources. For some regions, low bandwidth constitutes a real obstacle to developing climate services. Data volume also hinders reproducibility because very few institutions have the means to archive original data sets over the long term. Moreover, typical climate products often aggregate multiple sources of information, yet mechanisms to systematically track and document the provenance of all these data are only emerging. So although there is a general consensus that climate information should follow the **FAIR Principles** [WDA+16][MNV+17], that is be *findable, accessible, interoperable, and reusable*, a number of obstacles hinder progress. The following principles can help set up efficient climate services information systems, and show how the four FAIR Principles not only apply to data, but also to analytical processes.

3.2.1 Findable

Findable is the basic requirement for data and product usage and already an difficult obstacle with time intensive work for the data provider ensuring find-able data. On the production level finding algorithms requires open source software with intensive documentation.

Finding data:

Finding data requires a structured data repository and if possible an assigning of a globally unique and eternally persistent identifier (like a DOI or Handle), describing the data with rich metadata, and making sure it is find-able through discovery portals of search clients. It is recommended to establish data repository collecting and managing core input and output data enabling coordinated provisioning and sharing of data focusing on sustainable storage and management of core data collections. Depending on data importance a certified long-term archive can be managed. The identification of core data collections to be managed in centralized repositories might be realized with e.g the Research Data Management Organiser (RDMO) tool. https://rdmorganiser.github.io/

Finding algorithms:

In free and open source for geospatial (FOSS4G) developments workflows, independent developer groups are collaborating in a win-win situation and ensuring a high-quality software product Bahamdain2015. Public repositories enabling a work efficient participating and knowledge sharing approach [Tho10]. A high certainty and quality of scientific evidence is needed for information in a juridical context to regulate the conflict between economic development and environmental protection Brown2019. Therefor backend solutions to provide climate information for decision makers, need to be as much as possible 'error free'. The challenge of high-quality software solutions is illustrated with Linus's law that "given enough eyeballs, all bugs are shallow". Raymond2001.

3.2.2 Accessible

Access to data:

For data users, the prevailing *modus operandi* has traditionally been to download raw data locally to conduct analyses. As data volume grows, bandwidth and local storage capacity limits the type of science that individual scientists can perform.

Access to algorithms:

A high certainty and quality of scientific evidence is needed for information in a juridical context to regulate the conflict between economic development and environmental protection Brown2019. Therefor backend solutions to provide climate information for decision makers, need to be as much as possible 'error free'. The challenge of high-quality software solutions is illustrated with Linus's law that "given enough eyeballs, all bugs are shallow". Raymond2001. In free and open source for geospatial (FOSS4G) developments workflows, independent developer groups are collaborating in a win-win situation and ensuring a high-quality software product Bahamdain2015. Public repositories enabling a work efficient participating and knowledge sharing approach [Tho10].

3.2.3 Interoperable

Following the UNGGIM recommendations (2020) about 'Implementation and adoption of standards for the global geospatial information community' climate data should be organized following this UNGIM recommendations. (http://ggim.un.org/meetings/GGIM-committee/10th-Session/documents/E-C.20-2020-33-Add_1-Implementation-and-Adoption-of-Standards-21Jul2020.pdf) Interoperabillity needs to be respected on two levels:

Interoperable data :

following the conventions regarding metadata . . .

Interoperable structures:

The OGC standardisation also enables communication between climate services information systems services.

3.2.4 Reusabillity

Reusabillity is a major aspect to avoid duplication of work and to foster the dynamique of providing high quality products.

Reusable data:

The data should maintain its initial richness. The description of essential, recommended, and optional metadata elements should be machine processable and verifiable, use should be easy and data should be citable to sustain data sharing and recognize the value of data. Result output data from one service can be post-processed by another service where other component are provided.

Reusable algorithms:

Contrary to running analysis code on a local machine, it is recommended to use remote services have no direct control on the software they are running. The server's maintainer essentially decides when software and services are upgraded, meaning that within the time a scientist performs initial exploration and produces the final version of a figure for a paper, remote-services might have slightly changed or have been retired.

Reproducabillity:

This implies that reproducabillity results might not be easily reproducible if earlier versions of services are not available anymore. This puts an additional burden on scientists to carefully monitor the version of all the remote services used in the analysis to be able to explain discrepancies between results. Similar issues occur with data versions. If a scientist used version 1 for an analysis, there is no guarantee the source data will be archived over the long term if it has been superseded by version 2. In practice, climate services use ensembles of simulations, meaning that typical climate products aggregate hundreds or thousands of files, whose versions should ideally be tracked up until the final graphic or table. This capability to uniquely identify simulation files, errata and updates is available in CMIP6 [SL17][WLTK13], but it is the responsibility of climate service providers to embed this information into the products they develop.

3.3 Software development

- *Release Notes and Versions*

Here are some basic guides to smoothly contribute to birdhouse:

3.3.1 Source code

The source code of all birdhouse components is available on GitHub. Respecting the git mechanisms you can fork, clone and pull source-code into your repositories for modification and enhancement. Once your improvement is ready, make a pull request to integrate your work into the origin birdhouse repositories.

Note: Please keep your forks close to the origin repositories and don't forget the pull requests.

3.3.2 Git contribution

Note: Please find the git contribution guide in the Wiki.

3.3.3 Issue tracker

To keep track on the contribution and development, please use the issue tracker on GitHub for the corresponding birdhouse component.

3.3.4 Code Style

A good start to contribute is an enhancement of existing code with better or new functions. To respect a common coding style, Birdhouse uses PEP8 checks to ensure a consistent coding style. Currently the following PEP8 rules are enabled in `setup.cfg`:

```
[flake8]
ignore=F401,E402
max-line-length=120
exclude=tests
```

See the flake8 documentation on how to configure further options.

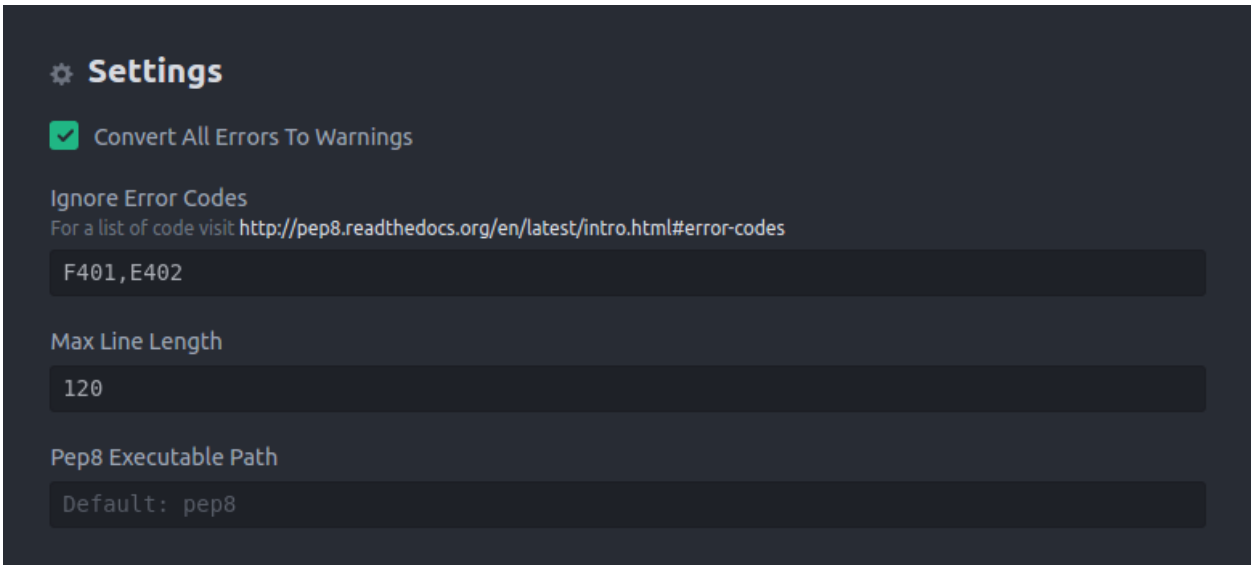
To check the coding style run `flake8`:

```
$ flake8 emu    # emu is the folder with python code
# or
$ make pep8     # make calls flake8
```

To make it easier to write code according to the PEP8 rules enable PEP8 checking in your editor. In the following we give examples how to enable code checking for different editors.

Atom

- Homepage: https://atom.io/
- PEP8 Atom Plugin: https://github.com/AtomLinter/linter-pep8



Sublime

- Install package control if you don't already have it: https://packagecontrol.io/installation
- Follow the instructions here to install Python PEP8 Autoformat: https://packagecontrol.io/packages/Python%20PEP8%20Autoformat
- Edit the settings to conform to the values used in birdhouse, if necessary
- To show the ruler and make wordwrap default, open Preferences → Settings—User and use the following rules

```
{
 // set vertical rulers in specified columns.
 "rulers": [79],

 // turn on word wrap for source and text
 // default value is "auto", which means off for source and on for text
 "word_wrap": true,

 // set word wrapping at this column
 // default value is 0, meaning wrapping occurs at window width
 "wrap_width": 79
 }
```

Todo: Add PEP8 instructions for more editors: PyCharm, Kate, Emacs, Vim, Spyder.

3.3.5 Release Notes and Versions

The development of birdhouse is following a release cycle of around three month. Updates of modules are coordinated by the developers over the communication channels (gitter chat or Video Conference). New releases are documented in the release notes and communicated over the mailing list. A release of a birdhouse module is taged with a version number and appropriate git repository version branch.

For an orientation of when to release a new version:

- Full version (v1.0) with scientific publication in a reviewed journal
- subversion (v1.1) by major changes
- subsub versions (v1.1.1) by minor changes

out of the release cycles bug fix patches can be released every time (communication is not mandatory)

- patch v1.1.1_patch1 bugfix

3.4 Setting up a new WPS

If you are familiar with all the upper chapters you are ready to create your own WPS. The WPS in birdhouse are named after birds, so this section is giving you a guidline of how to make your own bird. Birds are sorted thematically, so before setting up a new one, make sure it is not already covered and just missing some processes and be clear in the new thematic you would like to provide.

There is a Cookiecutter template to create a new bird (PyWPS application). It is the recommended and fastest way to create your own bird:

3.5 WPS design

3.6 Setting up a new WPS

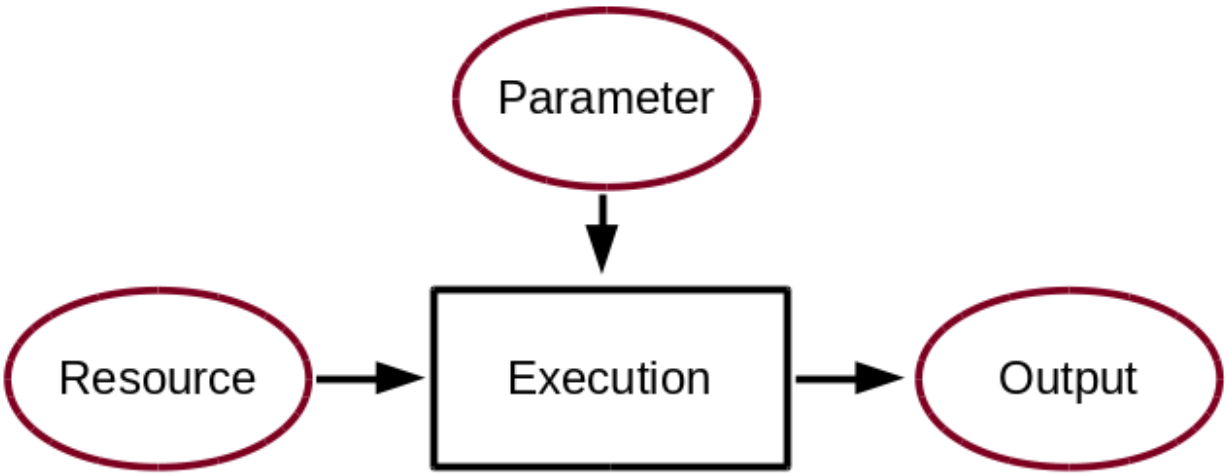
If you are familiar with all the upper chapters you are ready to create your own WPS. The WPS in birdhouse are named after birds, so this section is giving you a guidline of how to make your own bird. Birds are sorted thematically, so before setting up a new one, make sure it is not already covered and just missing some processes and be clear in the new thematic you would like to provide.

There is a Cookiecutter template to create a new bird (PyWPS application). It is the recommended and fastest way to create your own bird:

3.6.1 Writing a WPS process

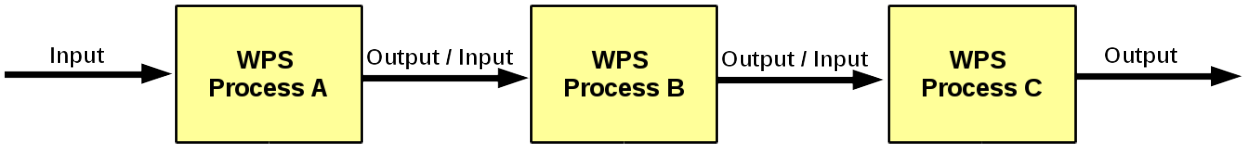
In birdhouse, we are using the PyWPS implementation of a *Web Processing Service*. Please read the PyWPS documentation on how to implement a WPS process.

Note: To get started quickly, you can try the Emu WPS with some example processes for PyWPS.



Another point to think about when designing a process is the possibility of chaining processes together. The result of a process can be a final result or be used as an input for another process. Chaining processes is a common practice but depends on the user you are designing the service for. Technically, for the development of WPS process chaining, here are a few summary points:

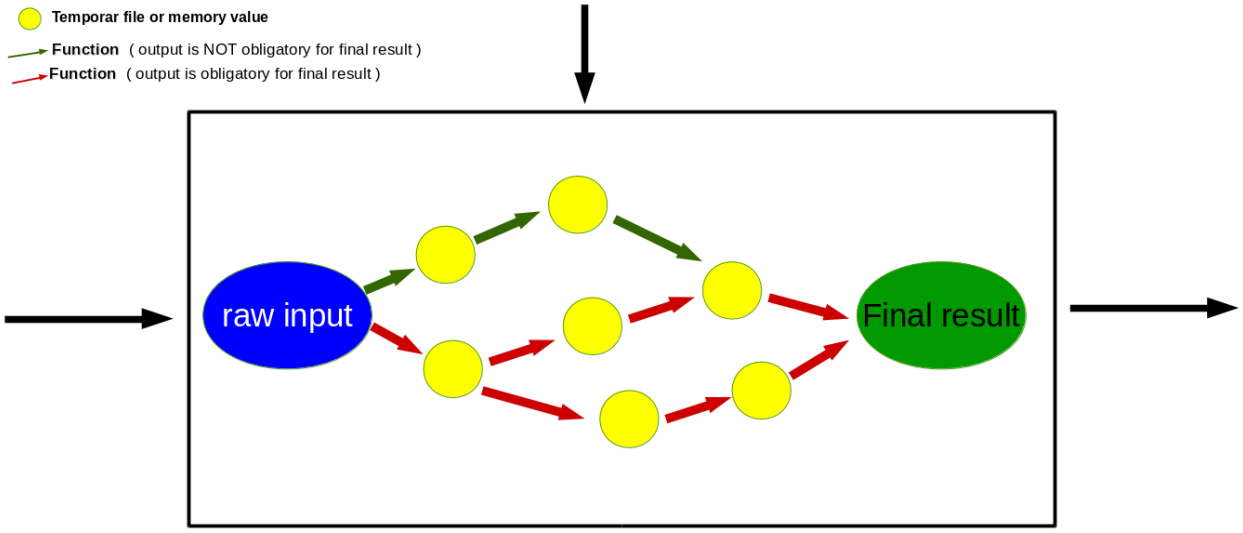
- the functional code should be modular and provide an interface/method for each single task
- provide a wps process for each task
- wps processes can be chained, manually or within the code, to run a complete workflow
- wps chaining can be done manually, with workflow tools, direct wps chaining or with code scripts
- a complete workflow chain could also be started by a wps process.



3.6.2 Writing functions

A Process is calling several functions during the performance. Since WPS is a autonom running process several eventualities needs to be taken into account. If irregularities are occurring, it is a question of the process design if the performance should stop and return an error or continue with may be an modified result.

In practice, the functions should be encapsulated in **try** and **except** calls and appropriate information given to the logfile or shown as a status message. The logger has several options to to influence the running code and the information writing to the logfile:




```python
# the following two line needs to be in the beginning of the *.py file.
# The ._handler will find the appropriate logfile and include timestemps
# and module information into the log.

import logging
LOGGER = logging.getLogger("PYWPS")

# set a status message
per = 5  # 5 will be 5% in the status line
response.update_status('execution started at : {}'.fromat(dt.now()), per)

try:
    response.update_status('the process is doing something: {}'.fromat(dt.now()),10)
    result = 42
    LOGGER.info('found the answer of life')
except Exception as ex:
    msg = 'This failed but is obligatory for the output. The process stops now, because: {} '.format(ex)
    LOGGER.error(msg)

try:
    response.update_status('the process is doing something else : {}'.fromat(dt.now()), 20)
    interesting = True
    LOGGER.info(' Thanks for reading the guidelines ')
    LOGGER.debug(' I need to know some details of the process: {} '.format(interesting)
except Exception as ex:
    msg = 'This failed but is not obligatory for the output. The process will continue. Reason for the failure: {} '.format(ex)
    LOGGER.exception(msg)
```

3.6.3 Writing documentation

Last but not least, a very very important point is to write a good documentation about your work! Each WPS (bird) has a docs folder for this where the documentation is written in reStructuredText and generated with Sphinx.

- http://sphinx-doc.org/tutorial.html
- http://quick-sphinx-tutorial.readthedocs.io/en/latest/

The documentation is automatically published to ReadTheDocs with GitHub webhooks. It is important to keep the *Code Style* and write explanations to your functions. There is an auto-api for documentation of functions.

Todo: explanation of enabling spinx automatic api documentation.

The main **`documentation`_** (which you are reading now) is the starting point to get an overview of birdhouse. Each birdhouse component comes with its own Sphinx documentation and is referenced by the main birdhouse document. Projects using birdhouse components like **PAVICS_** or **`COPERNICUS Data Store`_** generally have their own documentation as well. To include documentation from external repository here, two custom made sphinx directives can be used. The *gittoctree* directive behaves like a normal table of content directive (*toctree*), but takes as an argument the URL to the git repo and refers to files inside this directory through their full path. The *gitinclude* directive acts like an normal *include* directive, but takes as a first argument the URL to the git repo this file belongs to. For example:

Note: Look at the Emu to see examples.

3.7 Server setup

3.7.1 Data repository

3.7.2 Running tests

3.8 References

CHAPTER

FOUR

TUTORIALS

4.1 Climate Data with Phyton

Introduction to basic processing of climate data

Note: Coming soon, planed for Dec 2020.

4.1.1 Climate data tutorials

Here are examples of basic climate data processing for sustainable development.

To make sure all required dependencies are installed run *conda env create* in the root folder of this repository, than *conda activate climdat*.

An running installation of mini-conda or Anacona is required.

Access to Data

This tutorial is showing some options to access data, stored in accessable data archives.

ESGF Archive

Acess to CMIP6 data with subset selection for the coordinates of Paris

The code is oriented from https://esgf-pyclient.readthedocs.io/en/latest/notebooks/demo/subset-cmip6.html

Search data

[1]:
```
# ESGF can be directly connected with the python client pyesgf
# loading the client:
from pyesgf.search import SearchConnection

# connection to one of the ESGF Nodes
conn = SearchConnection('https://esgf-node.ipsl.upmc.fr/esg-search', distrib=False) #␣
↪set distrib=True if you want to search all nodes

# other nodes are:
# https://esgf-data.dkrz.de/esg-search
```

[2]:
```
# search of a set of temperature files
ctx = conn.new_context(project='CMIP6', query='tas', frequency='day', experiment_id=
↪'ssp585')

print('Number of data sets found: {}'.format(ctx.hit_count))

# other querry options might be:
# experiment_id = ['ssp126', 'ssp245', 'ssp370', 'ssp460','ssp585', 'historical',
↪'ssp119', 'ssp434']
# query = [ 'tas', 'tasmin', 'tasmax', 'pr', 'sfcWind' ] #
```

```
Number of data sets found: 20
```

[4]:
```
# getting some more infos of the datasets found:
for result in ctx.search():
    print(result.dataset_id)
```

```
CMIP6.ScenarioMIP.IPSL.IPSL-CM6A-LR.ssp585.r1i1p1f1.day.tas.gr.v20190614|vesg.ipsl.
↪upmc.fr
CMIP6.ScenarioMIP.IPSL.IPSL-CM6A-LR.ssp585.r2i1p1f1.day.tas.gr.v20191121|vesg.ipsl.
↪upmc.fr
CMIP6.ScenarioMIP.IPSL.IPSL-CM6A-LR.ssp585.r14i1p1f1.day.tas.gr.v20191121|vesg.ipsl.
↪upmc.fr
CMIP6.ScenarioMIP.IPSL.IPSL-CM6A-LR.ssp585.r4i1p1f1.day.tas.gr.v20191122|vesg.ipsl.
↪upmc.fr
CMIP6.ScenarioMIP.IPSL.IPSL-CM6A-LR.ssp585.r6i1p1f1.day.tas.gr.v20191121|vesg.ipsl.
↪upmc.fr
CMIP6.ScenarioMIP.IPSL.IPSL-CM6A-LR.ssp585.r1i1p1f1.day.tas.gr.v20190903|vesg.ipsl.
↪upmc.fr
CMIP6.ScenarioMIP.IPSL.IPSL-CM6A-LR.ssp585.r3i1p1f1.day.tas.gr.v20191121|vesg.ipsl.
↪upmc.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-CM6-1.ssp585.r1i1p1f2.day.tas.gr.v20190219|esg1.
↪umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-ESM2-1.ssp585.r5i1p1f2.day.tas.gr.v20190410|esg1.
↪umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-ESM2-1.ssp585.r4i1p1f2.day.tas.gr.v20190410|esg1.
↪umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-ESM2-1.ssp585.r1i1p1f2.day.tas.gr.v20190328|esg1.
↪umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-CM6-1.ssp585.r4i1p1f2.day.tas.gr.v20190410|esg1.
↪umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-CM6-1.ssp585.r6i1p1f2.day.tas.gr.v20190410|esg1.
↪umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-CM6-1.ssp585.r2i1p1f2.day.tas.gr.v20190410|esg1.
↪umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-CM6-1.ssp585.r3i1p1f2.day.tas.gr.v20190410|esg1.
↪umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-CM6-1.ssp585.r5i1p1f2.day.tas.gr.v20190410|esg1.
↪umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-ESM2-1.ssp585.r1i1p1f2.day.tas.gr.v20191021|esg1.
↪umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-CM6-1-HR.ssp585.r1i1p1f2.day.tas.gr.
↪v20191202|esg1.umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-ESM2-1.ssp585.r3i1p1f2.day.tas.gr.v20190410|esg1.
↪umr-cnrm.fr
CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-ESM2-1.ssp585.r2i1p1f2.day.tas.gr.v20190410|esg1.
↪umr-cnrm.fr
```

Data sets can consist of multiple files. Each file have an own url to be findable and accessable

[5]:
```
# print out the url locations
for i in ctx.search():
    files = i.file_context().search()
    for file in files:
        print(file.opendap_url)
```

```
http://vesg.ipsl.upmc.fr/thredds/dodsC/cmip6/ScenarioMIP/IPSL/IPSL-CM6A-LR/ssp585/
↪r2i1p1f1/day/tas/gr/v20191121/tas_day_IPSL-CM6A-LR_ssp585_r2i1p1f1_gr_20150101-
↪21001231.nc
http://vesg.ipsl.upmc.fr/thredds/dodsC/cmip6/ScenarioMIP/IPSL/IPSL-CM6A-LR/ssp585/
↪r14i1p1f1/day/tas/gr/v20191121/tas_day_IPSL-CM6A-LR_ssp585_r14i1p1f1_gr_20150101-
↪21001231.nc
http://vesg.ipsl.upmc.fr/thredds/dodsC/cmip6/ScenarioMIP/IPSL/IPSL-CM6A-LR/ssp585/
↪r4i1p1f1/day/tas/gr/v20191122/tas_day_IPSL-CM6A-LR_ssp585_r4i1p1f1_gr_20150101-
↪21001231.nc
http://vesg.ipsl.upmc.fr/thredds/dodsC/cmip6/ScenarioMIP/IPSL/IPSL-CM6A-LR/ssp585/
↪r6i1p1f1/day/tas/gr/v20191121/tas_day_IPSL-CM6A-LR_ssp585_r6i1p1f1_gr_20150101-
↪21001231.nc
http://vesg.ipsl.upmc.fr/thredds/dodsC/cmip6/ScenarioMIP/IPSL/IPSL-CM6A-LR/ssp585/
↪r1i1p1f1/day/tas/gr/v20190903/tas_day_IPSL-CM6A-LR_ssp585_r1i1p1f1_gr_20150101-
↪21001231.nc
http://vesg.ipsl.upmc.fr/thredds/dodsC/cmip6/ScenarioMIP/IPSL/IPSL-CM6A-LR/ssp585/
↪r1i1p1f1/day/tas/gr/v20190903/tas_day_IPSL-CM6A-LR_ssp585_r1i1p1f1_gr_21010101-
↪23001231.nc
http://vesg.ipsl.upmc.fr/thredds/dodsC/cmip6/ScenarioMIP/IPSL/IPSL-CM6A-LR/ssp585/
↪r3i1p1f1/day/tas/gr/v20191121/tas_day_IPSL-CM6A-LR_ssp585_r3i1p1f1_gr_20150101-
↪21001231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-CM6-1/
↪ssp585/r1i1p1f2/day/tas/gr/v20190219/tas_day_CNRM-CM6-1_ssp585_r1i1p1f2_gr_20150101-
↪21001231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-ESM2-1/
↪ssp585/r5i1p1f2/day/tas/gr/v20190410/tas_day_CNRM-ESM2-1_ssp585_r5i1p1f2_gr_
↪20150101-21001231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-ESM2-1/
↪ssp585/r4i1p1f2/day/tas/gr/v20190410/tas_day_CNRM-ESM2-1_ssp585_r4i1p1f2_gr_
↪20150101-21001231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-CM6-1/
↪ssp585/r4i1p1f2/day/tas/gr/v20190410/tas_day_CNRM-CM6-1_ssp585_r4i1p1f2_gr_20150101-
↪21001231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-CM6-1/
↪ssp585/r6i1p1f2/day/tas/gr/v20190410/tas_day_CNRM-CM6-1_ssp585_r6i1p1f2_gr_20150101-
↪21001231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-CM6-1/
↪ssp585/r2i1p1f2/day/tas/gr/v20190410/tas_day_CNRM-CM6-1_ssp585_r2i1p1f2_gr_20150101-
↪21001231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-CM6-1/
↪ssp585/r3i1p1f2/day/tas/gr/v20190410/tas_day_CNRM-CM6-1_ssp585_r3i1p1f2_gr_20150101-
↪21001231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-CM6-1/
↪ssp585/r5i1p1f2/day/tas/gr/v20190410/tas_day_CNRM-CM6-1_ssp585_r5i1p1f2_gr_20150101-
↪21001231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-ESM2-1/
↪ssp585/r1i1p1f2/day/tas/gr/v20191021/tas_day_CNRM-ESM2-1_ssp585_r1i1p1f2_gr_
↪20150101-21001231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-CM6-1-
↪HR/ssp585/r1i1p1f2/day/tas/gr/v20191202/tas_day_CNRM-CM6-1-HR_ssp585_r1i1p1f2_gr_
↪20150101-20641231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-CM6-1-
↪HR/ssp585/r1i1p1f2/day/tas/gr/v20191202/tas_day_CNRM-CM6-1-HR_ssp585_r1i1p1f2_gr_
↪20650101-21001231.nc
```

(continues on next page)

(continued from previous page)

```
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-ESM2-1/
↪ssp585/r3i1p1f2/day/tas/gr/v20190410/tas_day_CNRM-ESM2-1_ssp585_r3i1p1f2_gr_
↪20150101-21001231.nc
http://esg1.umr-cnrm.fr/thredds/dodsC/CMIP6_CNRM/ScenarioMIP/CNRM-CERFACS/CNRM-ESM2-1/
↪ssp585/r2i1p1f2/day/tas/gr/v20190410/tas_day_CNRM-ESM2-1_ssp585_r2i1p1f2_gr_
↪20150101-21001231.nc
```

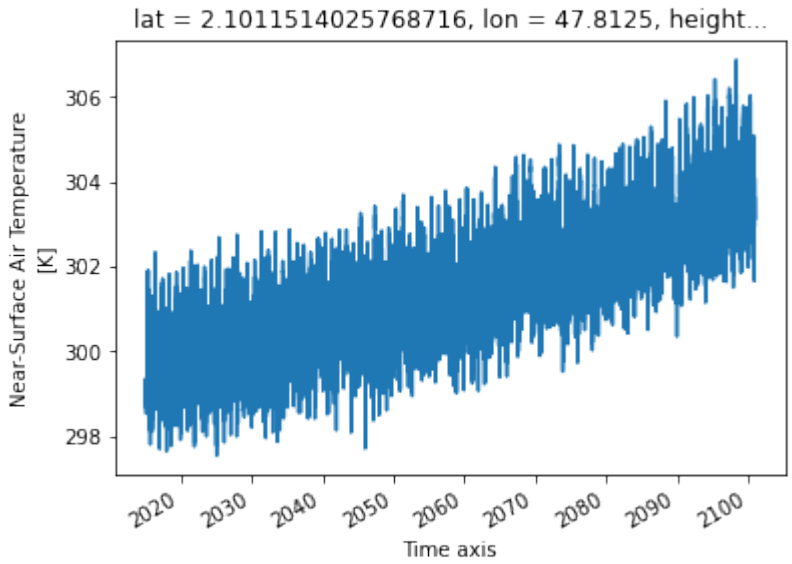
Access Data

[ ]:
```
# access the metadata
import xarray as xr
ds = xr.open_dataset(files[0].opendap_url, chunks={'time': 120})
print(ds)
```

[6]:
```
# select the area of Paris, due to the low resolution of CMIP6 it will result in just␣
↪one grid-point
da = ds['tas']
# da = da.isel(time=slice(0, 1000)) # fetching just the first 1000 days
da = da.sel(lat=slice(2, 3), lon=slice(47, 49))
```

[7]:
```
# Plot the timeseries
%matplotlib inline
da.plot()
```

[7]:
```
[<matplotlib.lines.Line2D at 0x7f0e950be580>]
```

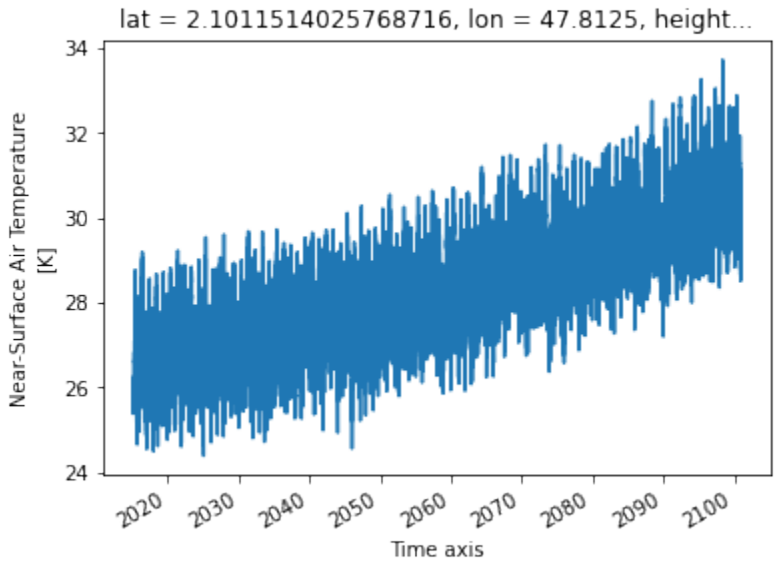


[10]:
```
# convert from Kelvin to Celcius
tem_C = da[:] - 273.15
```

[9]:
```
# import a plotting library
from matplotlib import pyplot as plt
```

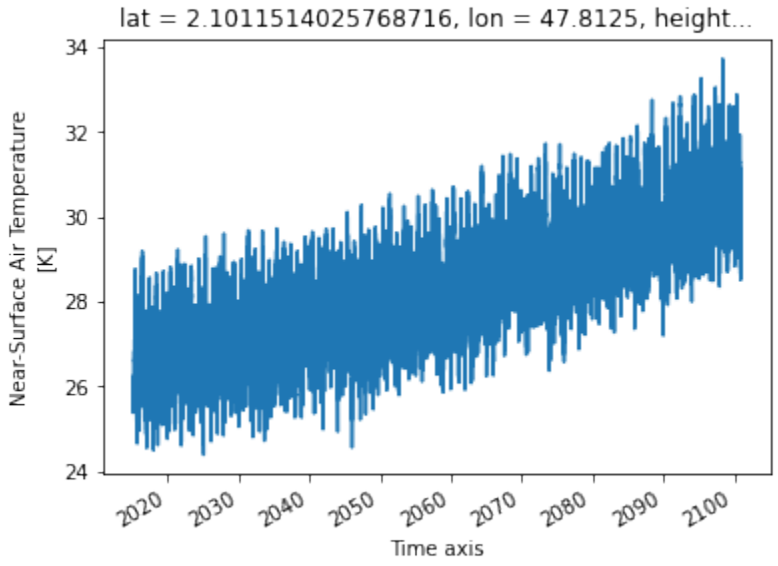
[26]:
```
# from numpy import squeeze
# squeeze(da.values)
da.values = da.values - 273.15
# plot the temperature in celsius
# plt.plot(tem_C)
```

[26]:
```
[<matplotlib.lines.Line2D at 0x7f0e8cc93070>]
```



[27]:
```
da.plot()
```

[27]:
```
[<matplotlib.lines.Line2D at 0x7f0e8cdb0220>]
```



[31]:
```
# to explore the data according to the choosen query:
da
```

[31]:
```
<xarray.DataArray 'tas' (time: 31411, lat: 1, lon: 1)>
dask.array<getitem, shape=(31411, 1, 1), dtype=float32, chunksize=(120, 1, 1),␣
↪chunktype=numpy.ndarray>
```

(continues on next page)

(continued from previous page)

```
Coordinates:
  * lat      (lat) float64 2.101
  * lon      (lon) float64 47.81
    height   float64 ...
  * time     (time) datetime64[ns] 2015-01-01T12:00:00 ... 2100-12-31T12:00:00
Attributes:
    online_operation:    average
    cell_methods:        area: time: mean
    interval_operation:  900 s
    interval_write:      1 d
    standard_name:       air_temperature
    description:         Near-Surface Air Temperature
    long_name:           Near-Surface Air Temperature
    history:             none
    units:               K
    cell_measures:       area: areacella
    _ChunkSizes:         [  1 128 256]
```

[15]:
```
da.to_netcdf('~/data/testforparis_2015-2100.nc')
```

[16]:
```
files[0].file_id
```

[16]:
```
'CMIP6.ScenarioMIP.CNRM-CERFACS.CNRM-ESM2-1.ssp585.r2i1p1f2.day.tas.gr.v20190410.tas_
↪day_CNRM-ESM2-1_ssp585_r2i1p1f2_gr_20150101-21001231.nc|esg1.umr-cnrm.fr'
```

[17]:
```
import xclim
```

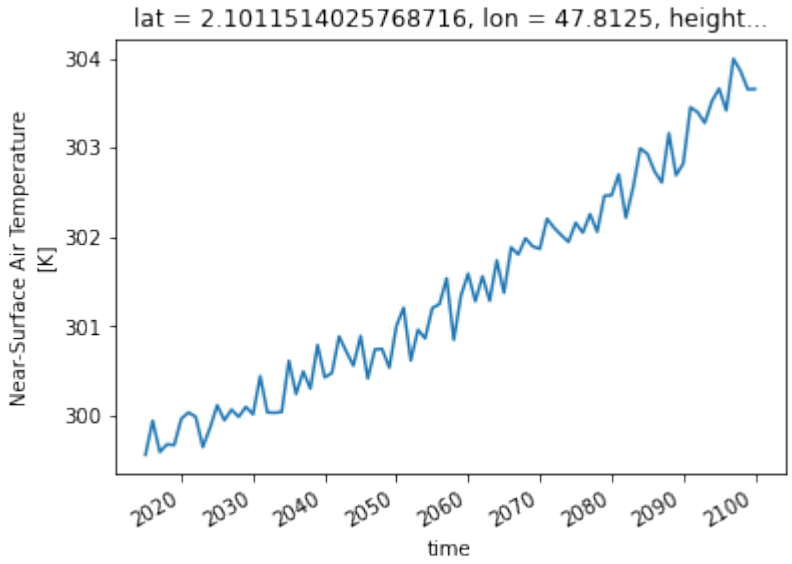
```
WARNING:pint.util:Redefining 'delta_degC' (<class 'pint.definitions.UnitDefinition'>)
WARNING:pint.util:Redefining 'celsius' (<class 'pint.definitions.UnitDefinition'>)
WARNING:pint.util:Redefining 'degC' (<class 'pint.definitions.UnitDefinition'>)
WARNING:pint.util:Redefining 'celsius' (<class 'pint.definitions.UnitDefinition'>)
WARNING:pint.util:Redefining 'C' (<class 'pint.definitions.UnitDefinition'>)
WARNING:pint.util:Redefining 'd' (<class 'pint.definitions.UnitDefinition'>)
WARNING:py.warnings:/home/testuser/anaconda3/envs/climatedata/lib/python3.8/site-
↪packages/xarray/core/options.py:48: FutureWarning: The enable_cftimeindex option is␣
↪now a no-op and will be removed in a future version of xarray.
  warnings.warn(
```

[37]:
```
meanTemp = xclim.indices.tg_mean(da)
```

```
WARNING:py.warnings:/home/testuser/anaconda3/envs/climatedata/lib/python3.8/site-
↪packages/xarray/core/common.py:978: FutureWarning: 'base' in .resample() and in␣
↪Grouper() is deprecated.
The new arguments that you should use are 'offset' or 'origin'.

>>> df.resample(freq="3s", base=2)

becomes:

>>> df.resample(freq="3s", offset="2s")

  grouper = pd.Grouper(
```

[38]:
```
# yearly mean
meanTemp.plot()
```

[38]:
```
[<matplotlib.lines.Line2D at 0x7f24d1860850>]
```



[ ]:
```
```

Resolution of Climate Model Data

This module of the tutorial is focussing on the different resolutions for global and regional climate model data. Test data are available in the VirtualMashine `~/data/orog/` or to be fetched directly from the ESGF archive (see Module 1)

load necessary libraries for plotting maps

[2]:
```
print('Hello FAZO')
```

```
Hello FAZO
```

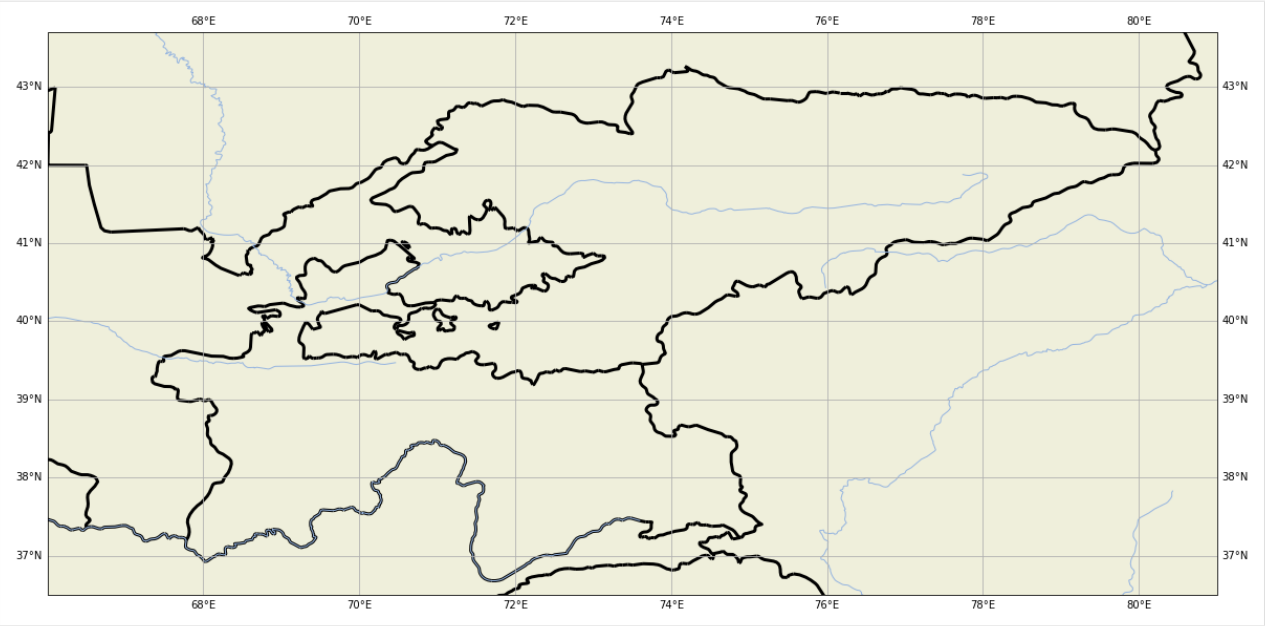
[4]:
```
from matplotlib import pyplot as plt
import cartopy.crs as ccrs
from numpy import meshgrid
import cartopy.feature as cfeature

# additional features
# from cartopy import config
# from cartopy.util import add_cyclic_point

# to show the plots inline (this is only necessary when running the code in a␣
↪notebook)
%matplotlib inline
```

configuration of the map focussing on Kirgistan and Tadjikistan

The ploting of maps is realised with the python libraies matplotlib and cartopy Further code ideas can be found here: https://scitools.org.uk/cartopy/docs/v0.15/index.html

[5]:
```
# NaturalEarthFeature is an open data source providing general data like catchment␣
↪areas

land_50m = cfeature.NaturalEarthFeature('physical', 'land', '50m', # Valid scales are
↪"110m", "50m", and "10m".
                                        edgecolor='k',
                                        facecolor=cfeature.COLORS['land'])

# defining a colorsceem
cmap='gist_earth_r'

# extent of the map (lat0, lat1, lon0, lon1)
extent=(66.0 , 81 , 36.5, 43.7)
linewidth=0.001
```

[6]:
```
# defining a new graphic
fig = plt.figure( figsize=(20,20 ), facecolor='w', edgecolor='k')
# fig.tight_layout()

# setting projections and extent of the map
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent(extent)

# ax = plt.axes(projection=ccrs.Orthographic(central_longitude=70.0, central_
↪latitude=40.0, globe=None))


# defining some features to be seen in the map:
ax.add_feature(land_50m)
# ax.add_feature(cfeature.LAKES)
ax.add_feature(cfeature.OCEAN)
ax.add_feature(cfeature.BORDERS, linewidth=3,)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.RIVERS)

ax.gridlines(draw_labels=True)
# ax.stock_img()
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

# save the graphic as a png:
plt.savefig(fname='/home/testuser/Pictures/map_country_overview.png')

# show the result
plt.show()
```



open a climate model file

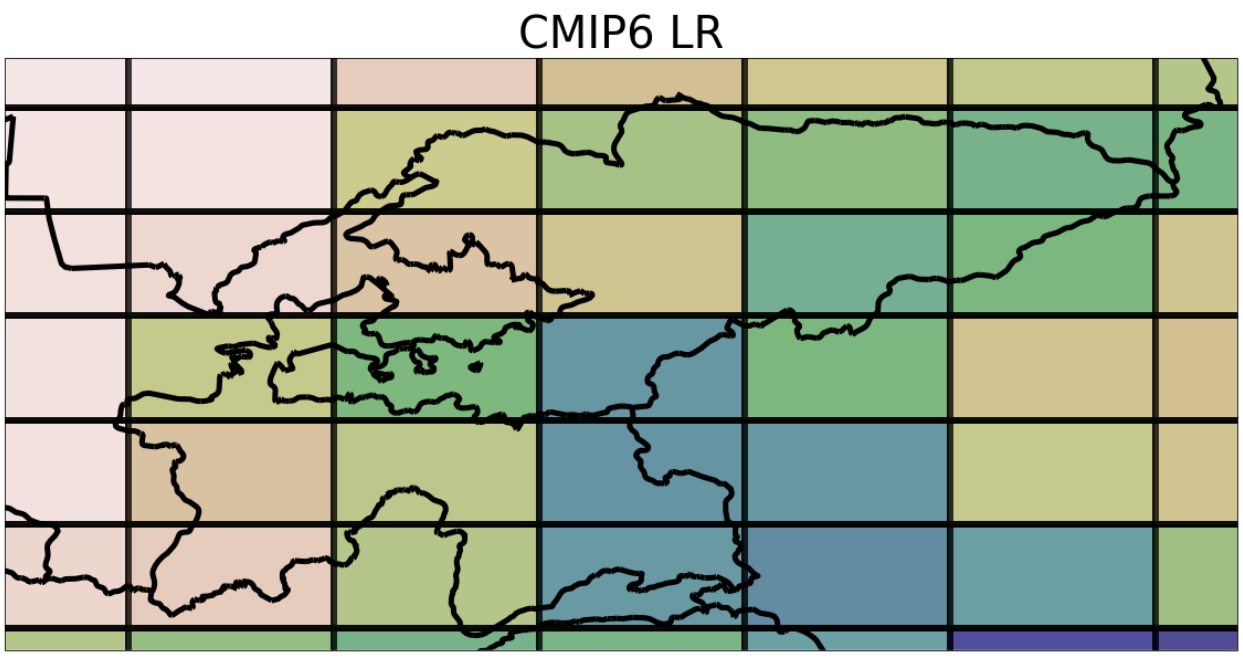
In the Tutorial Virtual Mashine there are some test data in the folder `/home/testuser/data/orog`.

`orog` is the variable for the orography underlying in the climate model.

[7]:
```
# loading necessary librabries
from netCDF4 import Dataset
import numpy as np
from os.path import join

# defining the folder where the data are stored
p = '/home/testuser/data/orog/'

# defining the file pathes as python variables:

# CMIP5_LR = join(p, 'orog_fx_MPI-ESM-LR_historical_r0i0p0.nc')
# CMIP5_MR = join(p, 'orog_fx_MPI-ESM-MR_historical_r0i0p0.nc')
# CMIP5_P  = join(p, 'orog_fx_MPI-ESM-P_historical_r0i0p0.nc')

CMIP6_HR = join(p, 'orog_fx_CNRM-CM6-1-HR_historical_r1i1p1f2_gr.nc')
CMIP6_LR = join(p, 'orog_fx_IPSL-CM6A-LR_ssp126_r2i1p1f1_gr.nc')

CAS22 = join(p, 'orog_CAS-22_MPI-M-MPI-ESM-LR_historical_r0i0p0_GERICS-REMO2015_v1_fx.
↪nc')
CAS44 = join(p, 'orog_CAS-44_ECMWF-ERAINT_evaluation_r0i0p0_MOHC-HadRM3P_v1_fx.nc')



# orog_fx_CNRM-CM6-1_historical_r1i1p1f2_gr.nc
```

[ ]:

[10]:
```
##############################
# CMIP6
ds = Dataset(CMIP6_LR)
orogC6_LR = ds.variables['orog']
latC6_LR = ds.variables['lat']
lonC6_LR = ds.variables['lon']


# fig , ax = plt.subplots(nrows=1, ncols=3, figsize=(20,20 ), facecolor='w',␣
↪edgecolor='k')
fig = plt.figure( figsize=(20,20 ), facecolor='w', edgecolor='k')

# fig.tight_layout()
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent(extent)
ax.add_feature(cfeature.BORDERS, linewidth=5)
ax.add_feature(cfeature.COASTLINE, linewidth=7)

lonsC6_LR, latsC6_LR = meshgrid(lonC6_LR, latC6_LR)


cs = ax.pcolormesh(lonsC6_LR, latsC6_LR, orogC6_LR, transform=ccrs.PlateCarree(),␣
↪cmap=cmap,
                   edgecolor='black', linewidth=5, alpha=0.7) #, vmin=0, vmax=3800)

plt.title('CMIP6 LR', fontsize=40)

plt.savefig(fname='/home/testuser/Pictures/CIMP6_LR.png')
```

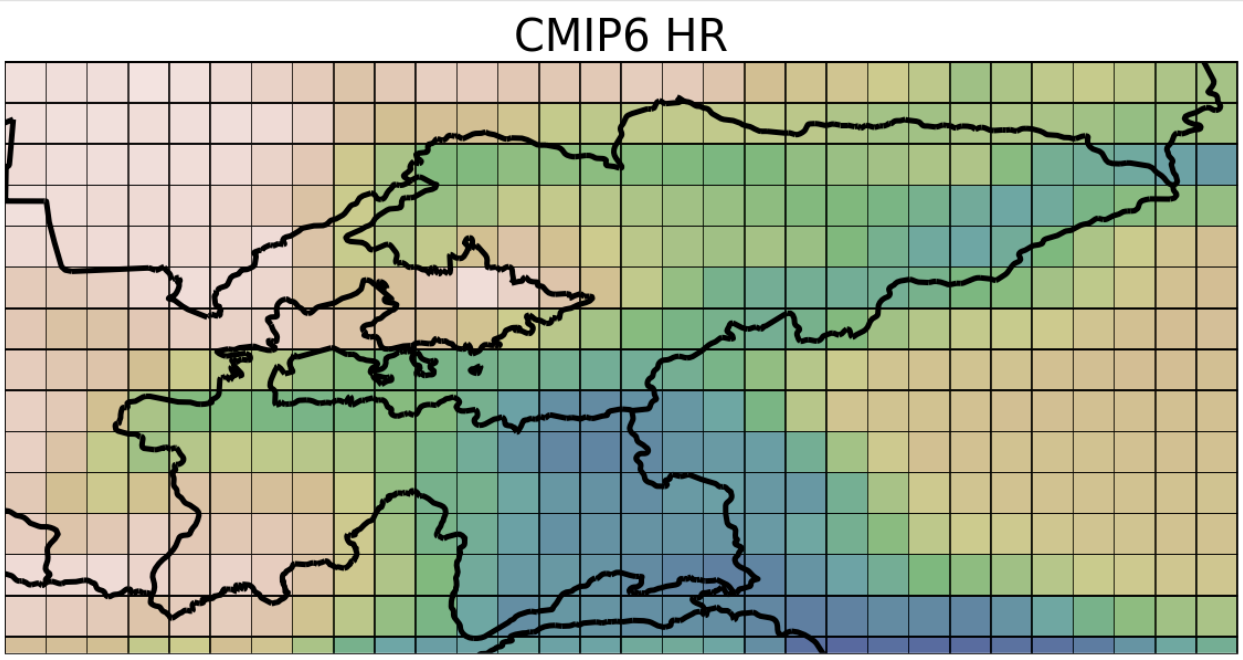


[11]:
```
ds = Dataset(CMIP6_HR)
orogC6_HR = ds.variables['orog']
latC6_HR = ds.variables['lat']
```

(continues on next page)

(continued from previous page)

```
lonC6_HR = ds.variables['lon']

# fig , ax = plt.subplots(nrows=1, ncols=3, figsize=(20,20 ), facecolor='w',␣
↪edgecolor='k')
fig = plt.figure( figsize=(20,20 ), facecolor='w', edgecolor='k')

# fig.tight_layout()
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent(extent)
ax.add_feature(cfeature.BORDERS, linewidth=5)
ax.add_feature(cfeature.COASTLINE, linewidth=7)

lonsC6_HR, latsC6_HR = meshgrid(lonC6_HR, latC6_HR)

cs = ax.pcolormesh(lonsC6_HR, latsC6_HR, orogC6_HR, transform=ccrs.PlateCarree(),␣
↪cmap=cmap,
                   edgecolor='black', linewidth=0.5, alpha=0.7) #, vmin=0, vmax=3800)

plt.title('CMIP6 HR', fontsize=40)

plt.savefig(fname='/home/testuser/Pictures/CIMP6_HR.png')
```



[ ]:

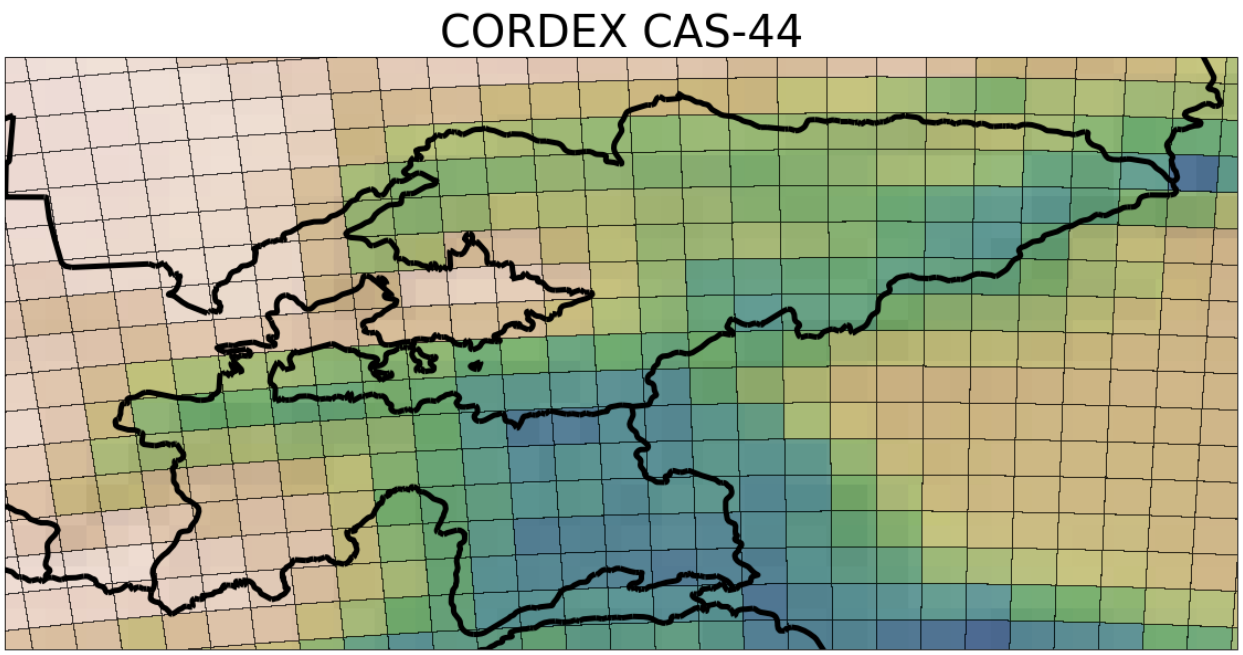
[8]:
```
#
#
#
```

[9]:
```
################
# CORDEX CAS-44

ds = Dataset(CAS44)
```

(continues on next page)

(continued from previous page)

```
orog44 = ds.variables['orog']
lats44 = ds.variables['lat']
lons44 = ds.variables['lon']
lon44 = lons44[0,:]
lat44 = lats44[:,0]

fig = plt.figure( figsize=(20,20 ), facecolor='w', edgecolor='k')

# fig.tight_layout()
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent(extent)
ax.add_feature(cfeature.BORDERS, linewidth=5)
ax.add_feature(cfeature.COASTLINE, linewidth=7)

ax.stock_img()
cs = ax.pcolormesh(lons44, lats44, orog44, transform=ccrs.PlateCarree(), cmap=cmap,
                   edgecolor='black', linewidth=0.01, alpha=0.7) # visible=True,␣
↪hatch = '|', linestyle='-' ) #, vmin=0, vmax=3800)

plt.title('CORDEX CAS-44', fontsize=40)
plt.savefig(fname='/home/testuser/Pictures/CAS-44.png')
```

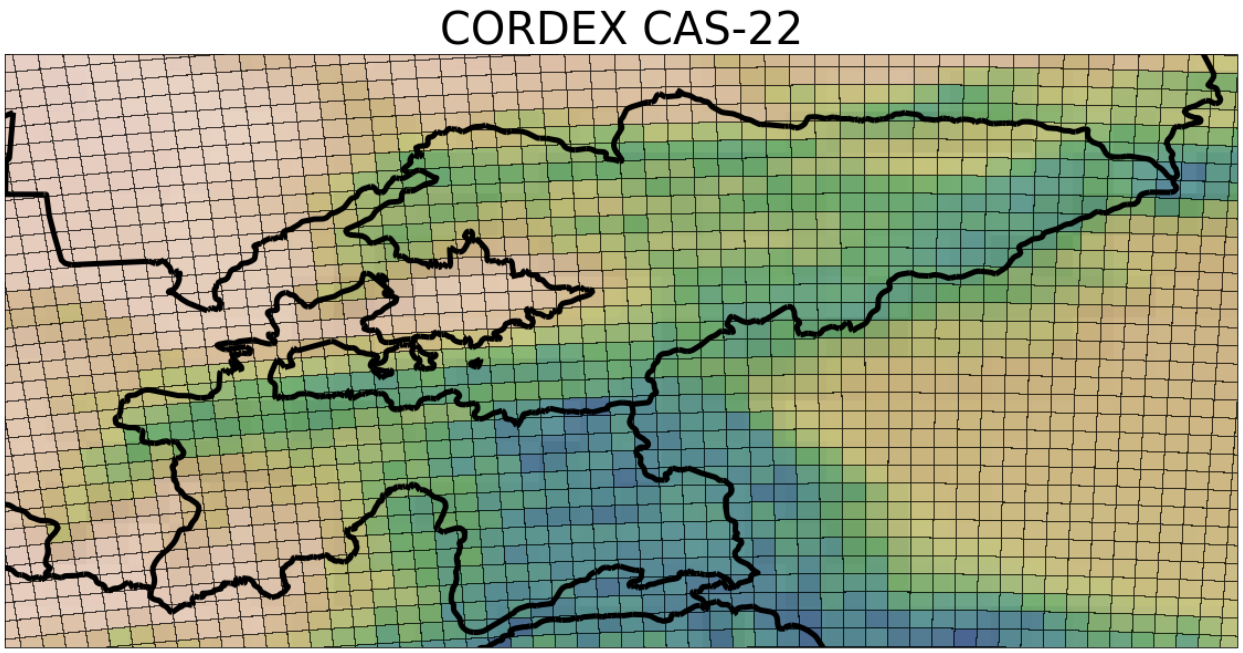


[12]:
```
ds = Dataset(CAS22)
orog22 = ds.variables['orog']
lats22 = ds.variables['lat']
lons22 = ds.variables['lon']
lon22 = lons22[0,:]
lat22 = lats22[:,0]

fig = plt.figure( figsize=(20,20 ), facecolor='w', edgecolor='k')

# fig.tight_layout()
ax = plt.axes(projection=ccrs.PlateCarree())
```

(continues on next page)

(continued from previous page)

```
ax.set_extent(extent)
ax.add_feature(cfeature.BORDERS, linewidth=5)
ax.add_feature(cfeature.COASTLINE, linewidth=7)

ax.stock_img()
cs = ax.pcolormesh(lons22, lats22, orog22, transform=ccrs.PlateCarree(), cmap=cmap ,
                   edgecolor='black', linewidth=0.01, alpha=0.7) # visible=True,␣
↪hatch = '|', linestyle='-' ) #, vmin=0, vmax=3800)

ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
# plt.colorbar(cs)

plt.title('CORDEX CAS-22', fontsize=40)
plt.savefig(fname='/home/testuser/Pictures/CAS-22.png')
```



[ ]:

Bias adjustment

[6]:
```
from xclim import sdba

# dqm.train(ref, hist)
# scen = dqm.adjust(sim)
```

[7]:
```
import xarray as xr
import numpy as np
from os import path, listdir


path_data = '/home/nils/ramboll/paris/data/'
```

(continues on next page)

(continued from previous page)

```
path_adjust = '/home/nils/ramboll/paris/data_adjust/'
path_pics = '/home/nils/Dropbox/Paris_diag_climat - Documents/9_climate/pics/'
path_obs = '/home/nils/Dropbox/Paris_diag_climat - Documents/5_travail/1_phase1/1_
↪1projections/observation/'

tas_files = [path.join(path_data,'tas',f) for f in listdir('data/'+ 'tas' )]
tasmin_files = [path.join(path_data,'tasmin',f) for f in listdir('data/'+ 'tasmin' )]
tasmax_files = [path.join(path_data,'tasmax',f) for f in listdir('data/'+ 'tasmax' )]
pr_files = [path.join(path_data,'pr',f) for f in listdir('data/'+ 'pr' )]
# xr.concat

# Observation:
```

[8]:
```
def sortssp_by_drsname(resource):
    nc_datasets = {}
    tmp_dic = {}

    try:
        for nc in resource:
            # LOGGER.info('file: %s' % nc)
            p, f = path.split(path.abspath(nc.replace('.nc', '')))
            n = f.split('_')
            if len([int(i) for i in n[-1].split('-') if i.isdigit()]) == 2:
                bn = '_'.join(n[0:-1])  # skipping the date information in the␣
↪filename
                nc_datasets[bn] = []  # dictionary containing all datasets names
            elif len([int(i) for i in n[-2].split('-') if i.isdigit()]) == 2:
                bn = '_'.join(n[0:-2])  # skipping the date information in the␣
↪filename
                nc_datasets[bn] = []  # dictionary containing all datasets names
            else:
                print('file is not DRS convention conform!')

            #  select only necessary names
            ssp_datasets = nc_datasets.copy()
            if any("_ssp" in s for s in nc_datasets.keys()):
                for key in nc_datasets.keys():
                    if 'historical' in key:
                        ssp_datasets.pop(key)
                nc_datasets = ssp_datasets.copy()
                print('historical data set names removed from dictionary')
            else:
                print('no SSP dataset names found in dictionary')
        print('Got dataset names for dic keys')
    except Exception as e:
        print('failed to get dataset names for dic keys {}'.format(e))

    # collect the file according to datasets
    for key in nc_datasets:
        try:
#             if historical_concatination is False:
#                 for n in resource:
#                     if '%s_' % key in n:
#                         nc_datasets[key].append(path.abspath(n))  # path.join(p, n))
#    ex = ['ssp119', 'ssp126', 'ssp245', 'ssp370',  'ssp434','ssp460','ssp585',
```

(continues on next page)

(continued from previous page)

```
#             elif historical_concatination is True:
            key_hist = key.replace('ssp119', 'historical').\
                replace('ssp126', 'historical').\
                replace('ssp245', 'historical').\
                replace('ssp370', 'historical').\
                replace('ssp434', 'historical').\
                replace('ssp460', 'historical').\
                replace('ssp585', 'historical')
            for n in resource:
                if '{}_'.format(key_hist) in n:
                    nc_datasets[key].append(path.abspath(n))
                if '{}_'.format(key) in n:
                    nc_datasets[key].append(path.abspath(n))  # path.join(p, n))
#             else:
#                 LOGGER.error('append file paths to dictionary for key %s failed' %␣
↪key)
            nc_datasets[key].sort()
        except Exception as e:
            print('failed for{e}'.fromat(e))
    return nc_datasets
```

[ ]:

[12]:
```
ds = sortssp_by_drsname(pr_files)

var = 'pr'
tas_obs = xr.open_dataset(path.join(path_obs + '{}_day_montsouris-observation.nc'.
↪format(var)))

for key in ds:
    try:
        files = []
        for f in ds[key]:
            files.append(xr.open_dataset(f))

        ts = xr.concat(files, 'time')
        ts.attrs['experiment_id'] = key.split('_')[3]

        dqm = sdba.adjustment.DetrendedQuantileMapping()
        # dqm = sdba.EmpiricalQuantileMapping(nquantiles=20, group='time', kind='+')
        # QM.train(ref, hist)

        dqm.train(tas_obs[var].sel(time=slice("1971-01-01", "2000-12-31")),
                  ts[var].sel(time=slice("1971-01-01", "2000-12-31")))
        scen = dqm.adjust(ts[var])
        ts[var].values = scen.values
        out_file = path.join(path_adjust, var, '{}.nc'.format(key))
        ts.to_netcdf(out_file, unlimited_dims='time')
        ts.close()
    except Exception as e:
        print('failed for {} : {}'.format(key, e))
```

```
historical data set names removed from dictionary
historical data set names removed from dictionary
```

(continues on next page)

(continued from previous page)

```
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
```

(continues on next page)

(continued from previous page)

```
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
```

(continues on next page)

(continued from previous page)

```
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
```

(continues on next page)

(continued from previous page)

```
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
historical data set names removed from dictionary
Got dataset names for dic keys
```

```
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
```

(continues on next page)

(continued from previous page)

```
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
```

```
failed for pr_day_IPSL-CM6A-LR_ssp460_r1i1p1f2_gr : dayofyear must not be empty
```

```
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
```

(continues on next page)

(continued from previous page)

```
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
```

(continues on next page)

(continued from previous page)

```
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
```

(continues on next page)

(continued from previous page)

```
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
```

(continues on next page)

(continued from previous page)

```
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
```

[111]:

```

```

[111]:

```
'ssp126'
```

[ ]:

```
tas_day_CNRM-ESM2-1_ssp126_r4i1p1f2_gr
tas_day_IPSL-CM6A-LR_ssp460_r1i1p1f2_gr
```

[104]:

```

out_file = path.join(path_adjust, var, '{}.nc'.format(key))
out_file
```

[104]:

```
'/home/nils/ramboll/paris/data_ajust/tas/tas_day_CNRM-ESM2-1_ssp245_r4i1p1f2_gr.nc'
```

[92]:
```
scen = dqm.adjust(ts[var])
# ts['tas'].values = scen.values
# ts.to_netcdf(path.join(path_adjust,var , "/{}.nc".format(key)), unlimited_dims='time
↪')
# ts.close()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-92-c8e4051ebff0> in <module>
----> 1 scen = dqm.adjust(ts[var])
      2 # ts['tas'].values = scen.values
      3 # ts.to_netcdf(path.join(path_adjust,var , "/{}.nc".format(key)), unlimited_
↪dims='time')
      4 # ts.close()

~/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xclim/sdba/adjustment.py in␣
↪adjust(self, sim, **kwargs)
    104         if hasattr(self, "group"):
    105             # Right now there is no other way of getting the main adjustment␣
↪dimension
--> 106             _raise_on_multiple_chunk(sim, self.group.dim)
    107
    108             if (

~/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xclim/sdba/adjustment.py in␣
↪_raise_on_multiple_chunk(da, main_dim)
     38     if da.chunks is not None and len(da.chunks[da.get_axis_num(main_dim)]) >␣
↪1:
     39         raise ValueError(
---> 40             f"Multiple chunks along the main adjustment dimension {main_dim}␣
↪is not supported."
     41         )
     42

ValueError: Multiple chunks along the main adjustment dimension time is not supported.
```

[93]:
```
ds[key]
```

[93]:
```
['/home/nils/ramboll/paris/data/tas/tas_day_CNRM-ESM2-1_historical_r4i1p1f2_gr_
↪18500101-20141231_sub.nc',
 '/home/nils/ramboll/paris/data/tas/tas_day_CNRM-ESM2-1_ssp245_r4i1p1f2_gr_20150101-
↪21001231_sub.nc']
```

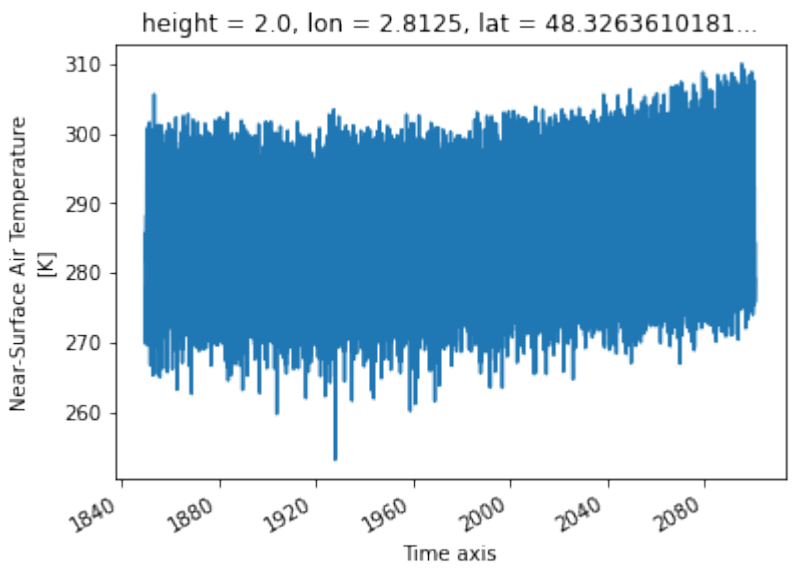
[63]:
```
files = []
for f in enumerate(ds['tas_day_CNRM-CM6-1_ssp370_r5i1p1f2_gr']):
    files.append(xr.open_dataset(f))

ts = xr.concat(files, 'time')
```

[80]:
```
ds['tas_day_CNRM-CM6-1_ssp370_r5i1p1f2_gr']
```

[80]:
```
['/home/nils/ramboll/paris/data/tas/tas_day_CNRM-CM6-1_historical_r5i1p1f2_gr_
↪18500101-20141231_sub.nc',
 '/home/nils/ramboll/paris/data/tas/tas_day_CNRM-CM6-1_ssp370_r5i1p1f2_gr_20150101-
↪21001231_sub.nc']
```

[83]:
```
ts['tas'].plot()
```

[83]:
```
[<matplotlib.lines.Line2D at 0x7f39de15f590>]
```



[32]:
```
# file_hist = '/home/nils/ramboll/paris/data/pr/pr_day_CNRM-CM6-1_historical_
↪r10i1p1f2_gr_19500101-20141231_sub.nc'


hist = xr.open_dataset(file_hist)
ref = xr.open_dataset(file_obs)

ref = xr.DataArray(val_obs, dims=('time',), coords={'time': ts_obs}, attrs={'units':
↪'K'})
hist = xr.DataArray(val_hist, dims=('time',), coords={'time': ts_hist}, attrs={'units
↪': 'K'})
```

[44]:
```

# dqm.train(obs['tas'].sel(time=slice("1999-01-01", "2000-12-31")), hist['tas'].
↪sel(time=slice("1971-01-01", "2000-12-31")))
```

[45]:
```

```

[75]:
```
scen.values
```

[75]:
```
array([[[283.11070111]],

       [[285.09585079]],

       [[282.85011458]],

       ...,

       [[282.16726647]],

       [[279.29240315]],

       [[279.72952784]]])
```

[76]:

[79]:

```
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/xarray/conventions.
↪py:427: UserWarning: Variable 'time' has datetime type and a bounds variable but␣
↪time.encoding does not have units specified. The units encodings for 'time' and
↪'time_bounds' will be determined independently and may not be equal, counter to CF-
↪conventions. If this is a concern, specify a units encoding for 'time' before␣
↪writing to a file.
  UserWarning,
```

[78]:
```
scen.mean()
```

[78]:
```
<xarray.DataArray ()>
array(285.75631892)
Coordinates:
    height   float64 2.0
```

[59]:
```
# ts.close()
# tas_obs.close()
scen.var?
# scen.to_netcdf(path.join(path_adjust, "tas/test.nc"), unlimited_dims='time')
```

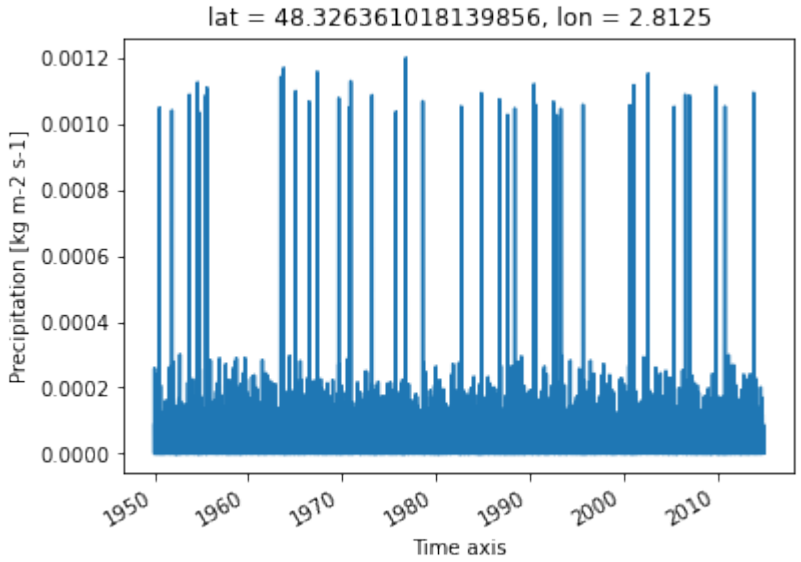
[69]:
```
ds_ad = sdba.processing.adapt_freq(sim=hist['pr'], ref=ref['pr'], thresh=0.05)
QM_ad = sdba.EmpiricalQuantileMapping(nquantiles=15, kind='*', group='time')
QM_ad.train(ref['pr'], ds_ad.sim_ad)
scen_ad = QM_ad.adjust(hist['pr'])
```

[24]:
```
import numpy as np
tas = np.squeeze(obs['tas'])
```

[45]:
```
sl = obs['tas'].sel(time=slice("2000-06-01", "2000-06-10"))
```

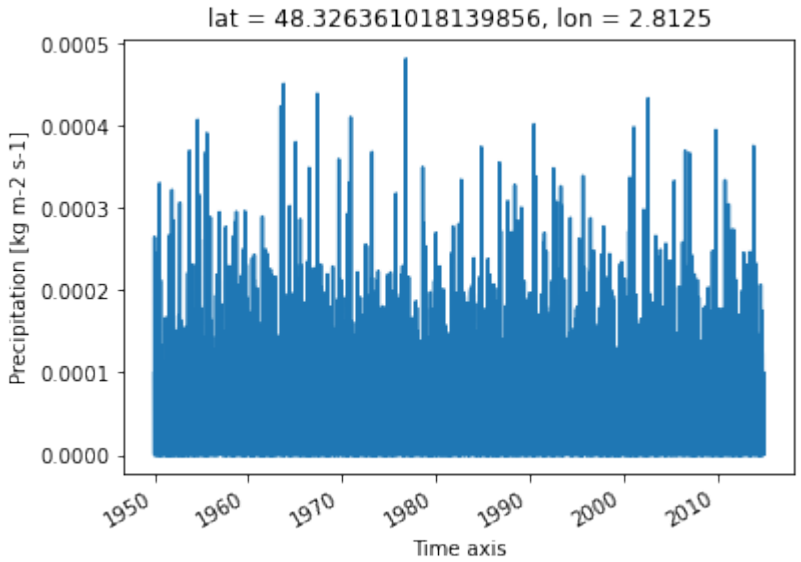
[65]:
```
# dif = hist - scen
scen.plot()
```

[65]:
```
[<matplotlib.lines.Line2D at 0x7f08aa698450>]
```



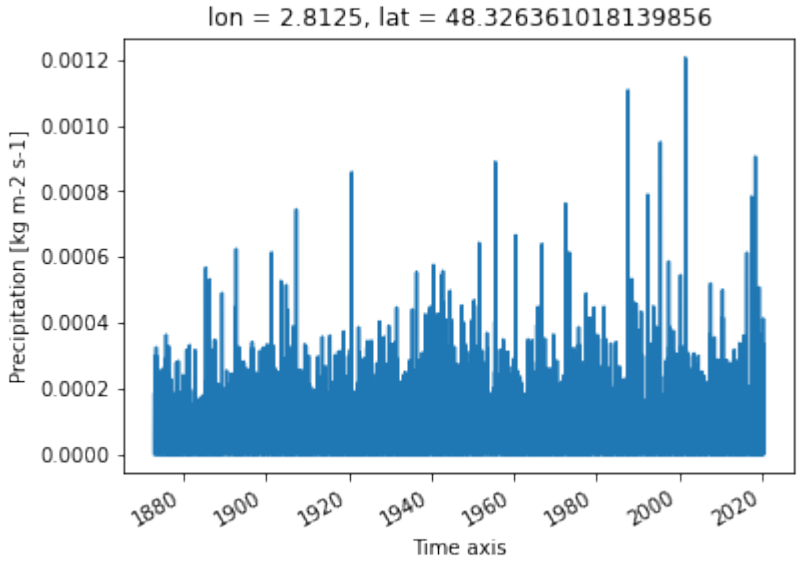
[59]:
```
hist['pr'].plot()
```

[59]:
```
[<matplotlib.lines.Line2D at 0x7f08aa6b25d0>]
```



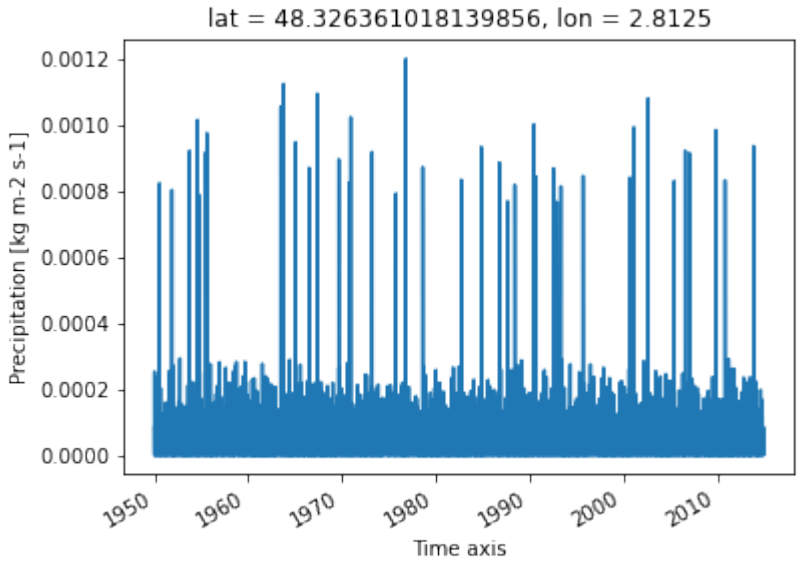
[63]:
```
obs['pr'].plot()
```

[63]:
```
[<matplotlib.lines.Line2D at 0x7f08aa7e1310>]
```



[71]:
```
scen_ad.plot()
```

[71]:
```
[<matplotlib.lines.Line2D at 0x7f08aa407e50>]
```



[ ]:

Basic Visusalisation

[1]:
```
from birdy import WPSClient
from os import listdir, path

from shutil import move
from os.path import join
```

[2]:
```
# To display Images from an url
from IPython.core.display import HTML
from IPython.display import Image
```

[10]:
```
from matplotlib import pyplot as plt
from matplotlib import colors
# from matplotlib.patches import Polygon
# import matplotlib.patches as mpatches
# import cartopy.feature as cfeature

# import cartopy.crs as ccrs
# from cartopy.util import add_cyclic_point

# from flyingpigeon.nc_statistic import fieldmean
# from flyingpigeon.nc_utils import get_variable, get_coordinates

from flyingpigeon.nc_utils import get_time, get_values # sort_by_filename,
from flyingpigeon.plt_utils import fig2plot

# from numpy import meshgrid
# from netCDF4 import Dataset
import numpy as np
import pandas as pd
from datetime import datetime as dt
from tempfile import mkstemp
```

[ ]:
```
```

[11]:
```
SSP_colors = {'ssp585':'#830b22',
              'ssp460':'#e78731',
              'ssp370':'#f11111',
              'ssp245':'#e9dc3d',
              'ssp126':'#1d3354',
              'ssp119':'#1e9583'}
```

[12]:
```
indices = ['tg-mean',  'tx-mean' , 'tn-mean' , 'tn-min' , 'tx-max',
           'prcptot', 'wetdays', 'rx1day', 'rx5day', 'dry-days', 'cdd', 'cwd',
           'ice-days', 'frost-days',
           'jours-chaud', 'jours-tres-chaud', 'tropical-nights']

delta = [-273.15, -273.15, -273.15, -273.15, -273.15,
        0, 0,0,0,0,0,
        0,0,
        0,0,
        0]

titles = ['Température moyenne quotidienne (°C/an)',
```

(continues on next page)

(continued from previous page)

```
            'Température maximale quotidienne (°C/an)',
            'Température minimale quotidienne (°C/an)',
            'Température minimale pour le jour le plus froid (°C/an)',
            'Température maximale pour le jour le plus chaud (°C/an)',

            'Cumul pluviométrique annuel (mm/an)',
            'Nombre de jours humide par an',
#             'Cumul pluviométrique hivernal : octobre à mars (mm/an)',
#             'Cumul pluviométrique estival : avril à septembre (mm/an)',

#             'Nombre de jours avec fortes pluies : > 10 mm (j/an)',
#             'Nombre de jours sur 30 ans avec précipitations intenses : > 50 mm (j/an)
↪',
#              Evolution des précipitations> 20 mm en 24h en hiver (%),
#              Evolution des précipitations> 20 mm en 24h en été (%),


            'Précipitation del jour le plus pluvieux par an (mm/j)',
            'Somme précipitations max. sur 5 jours consécutifs',
            'Nombre de jours sans précipitation (j/an)',
            'Journées consécutives de sécheresse',
            'Jours humides consécutifs',

            'Nombre de jours de gel : (j/an)',  # t_max 0°C
            'Nombre gelées nocturnes : (j/an)',  # t_min 0°C

            'Nombre de jours chauds  25 °C (j/an)',
            'Nombre de jours très chauds :  30 °C (j/an)',

            'Nombre de nuits tropicales :  20 °C (j/an)',

            "Index d'intensité de précipitations",
            ]

# Nombre de DJU période estivale avec T° référence 24°C (°C/an)
# Nombre de DJU période hivernale avec T° référence 18°C (°C/an)

# Nombre de jours faisant suite à une période de 15 jours consécutifs sans␣
↪précipitation - indicateur de sécheresse météorologique simple (j/an).

# Nombre de jours sur 30 ans avec fortes rafales : > 100km/h
# Evolution du vent moyen hivernal à 10m (%)
# Evolution du vent moyen estival à 10m (%)
```

[13]:
```
path_pics = '/home/nils/Dropbox/Paris_diag_climat - Documents/5_travail/1_phase1/1_
↪1projections/pics/'
path_csv = '/home/nils/Dropbox/Paris_diag_climat - Documents/5_travail/1_phase1/1_
↪1projections/csv/'
path_indices = '/home/nils/Dropbox/Paris_diag_climat - Documents/5_travail/1_phase1/1_
↪1projections/indices_adjust/'
path_obs = '/home/nils/Dropbox/Paris_diag_climat - Documents/5_travail/1_phase1/1_
↪1projections/observation/indices/'
```

[67]:
```
def plot_ssp_uncertainty_anormalie(resource, variable, ylim=None, title=None,␣
↪observation=None, decode_cf=True,
                                   file_extension='png', delta=0, window=None, dir_
↪output=None,
```

(continues on next page)

(continued from previous page)

```
                                    figsize=(10, 10)):
    """
    creates a png file containing the appropriate uncertainty plot.

    :param resource: list of files containing the same variable
    :param delta: set a delta for the values e.g. -273.15 to convert Kelvin to Celsius
    :param variable: variable to be visualised. If None (default), variable will be␣
↪detected
    :param ylim: Y-axis limitations: tuple(min,max)
    :param title: string to be used as title
    :param observation: optional data of observations
    :param figsize: figure size defult=(10,10)
    :param decode_cf: decode of netCDF values according cf convention
    :param window: windowsize of the rolling mean

    :returns str: path/to/file.png
    """
    from flyingpigeon.plt_ncdata import ts_data
    from flyingpigeon.nc_utils import sortssp_by_drsname

    try:
        fig = plt.figure(figsize=figsize, facecolor='w', edgecolor='k')
        ax = fig.add_subplot(111)
    #         plt.subplots_adjust( wspace=0, hspace=0.2) #

    #         fig = plt.figure(figsize=figsize, dpi=600, facecolor='w', edgecolor='k')
    #         LOGGER.debug('Start visualisation spaghetti plot')
    #         === prepare invironment
        if type(resource) != list:
            resource = [resource]
    #         var = get_variable(nc)
    #         if variable is None:
    #             variable = get_variable(resource[0])
    #         LOGGER.info('plot values preparation done')
    except Exception as ex:
        print("plot values preparation failed {}".format(ex))
    #         LOGGER.exception(msg)
    #         raise Exception(msg)
    try:
        dic = sortssp_by_drsname(resource)  # sort_by_filename(resource, historical_
↪concatination=True)
        df = ts_data(dic, delta=delta)
    except Exception as ex:
        print("failed to sort data".format(ex))


    ###################################
    # serach datasets according to scenario
    try:
        ssp126 = [ds for ds in df.columns if 'ssp126' in ds]
        ssp245 = [ds for ds in df.columns if 'ssp245' in ds]
        ssp585 = [ds for ds in df.columns if 'ssp585' in ds]

        print('all scenarios are seperated')
    except Exception as e:
        print('failed to split scenarios {}'.format(e))
```

(continues on next page)

(continued from previous page)

```
    window = 30 # 30 years

    if len(df.index.values) >= window * 2:
        # TODO: calculate windowsize according to timestapms (day,mon,yr ... with get_
↪frequency)
        df_smooth = df.rolling(window=window, center=True, min_periods=2).mean()
        print('rolling mean calculated for all input data')
    else:
        df_smooth = df.copy()
        fig.text(0.95, 0.05, '!!! timeseries too short for moving mean over 30years !!
↪!',
                 fontsize=20, color='red',
                 ha='right', va='bottom', alpha=0.5)


    ###################################
    # normalisation

    x = pd.to_datetime(df_smooth.index.values)
    ts_ref = x.get_loc(dt.strptime("1985", "%Y"), method='nearest')
    ref_val = np.nanmean(df_smooth.values, axis=1)[ts_ref]


    try:
        df_ssp126 = df[ssp126].rolling(window=window, center=True, min_periods=2).
↪mean() - ref_val
        df_ssp245 = df[ssp245].rolling(window=window, center=True, min_periods=2).
↪mean() - ref_val
        df_ssp585 = df[ssp585].rolling(window=window, center=True, min_periods=2).
↪mean() - ref_val
    except Exception as e:
        print('failed to group scenarios {}'.format(e))


    ####################
    # calculation of mean and uncertainties

    # ssp126
    try:
        ssp126_rmean = np.squeeze(df_ssp126.quantile([0.5], axis=1,).values)
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp126_q05 = np.squeeze(df_ssp126.quantile([0.05], axis=1,).values)
        ssp126_q33 = np.squeeze(df_ssp126.quantile([0.33], axis=1,).values)
        ssp126_q66 = np.squeeze(df_ssp126.quantile([0.66], axis=1,).values)
        ssp126_q95 = np.squeeze(df_ssp126.quantile([0.95], axis=1,).values)
        print('quantile calculated for all input ssp126 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))

    # ssp245
    try:
        ssp245_rmean = np.squeeze(df_ssp245.quantile([0.5], axis=1,).values)
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp245_q05 = np.squeeze(df_ssp245.quantile([0.05], axis=1,).values)
        ssp245_q33 = np.squeeze(df_ssp245.quantile([0.33], axis=1,).values)
        ssp245_q66 = np.squeeze(df_ssp245.quantile([0.66], axis=1,).values)
```

(continues on next page)

(continued from previous page)

```
        ssp245_q95 = np.squeeze(df_ssp245.quantile([0.95], axis=1,).values)
        print('quantile calculated for all input ssp245 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))


    # ssp585
    try:
        ssp585_rmean = np.squeeze(df_ssp585.quantile([0.5], axis=1,).values)
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp585_q05 = np.squeeze(df_ssp585.quantile([0.05], axis=1,).values)
        ssp585_q33 = np.squeeze(df_ssp585.quantile([0.33], axis=1,).values)
        ssp585_q66 = np.squeeze(df_ssp585.quantile([0.66], axis=1,).values)
        ssp585_q95 = np.squeeze(df_ssp585.quantile([0.95], axis=1,).values)
        print('quantile calculated for all input ssp585 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))


    ###################
    # plot

    try:
        x = pd.to_datetime(df.index.values)
#               df[(df['date'] > '2000-6-1') & (df['date'] <= '2000-6-10')]
        x1 = x[x <= dt.strptime('2015-12-31', "%Y-%m-%d")]
        x2 = x[len(x1)-1:]  # -1 to catch up with the last historical value

#         plt.fill_between(x, ssp126_q05, ssp126_q95, alpha=1, color='lightgrey')
#         plt.fill_between(x, ssp245_q05, ssp245_q95, alpha=1, color='lightgrey')
#         plt.fill_between(x, ssp585_q05, ssp585_q95, alpha=1, color='lightgrey')

        plt.fill_between(x1, ssp126_q05[:len(x1)], ssp126_q95[:len(x1)], alpha=1,␣
↪color='lightgrey')
        plt.fill_between(x1, ssp245_q05[:len(x1)], ssp245_q95[:len(x1)], alpha=1,␣
↪color='lightgrey')
        plt.fill_between(x1, ssp585_q05[:len(x1)], ssp585_q95[:len(x1)], alpha=1,␣
↪color='lightgrey')


        plt.fill_between(x2, ssp126_q05[len(x1)-1:], ssp126_q95[len(x1)-1:], alpha=0.
↪3, color=SSP_colors['ssp126'])
        plt.fill_between(x2, ssp245_q05[len(x1)-1:], ssp245_q95[len(x1)-1:], alpha=0.
↪3, color=SSP_colors['ssp245'])
        plt.fill_between(x2, ssp585_q05[len(x1)-1:], ssp585_q95[len(x1)-1:], alpha=0.
↪3, color=SSP_colors['ssp585'])

        mean_hist = np.mean([ssp126_rmean[:len(x1)], ssp245_rmean[:len(x1)] , ssp245_
↪rmean[:len(x1)]],axis=0)

        plt.plot(x1, mean_hist , c='darkgrey', lw=2)
#         plt.plot(x1, ssp245_rmean[:len(x1)], c='dimgrey', lw=2)
#         plt.plot(x1, ssp245_rmean[:len(x1)], c='dimgrey', lw=2)

        plt.plot(x2, ssp585_rmean[len(x1)-1:], c=SSP_colors['ssp585'], lw=2)
        plt.plot(x2, ssp245_rmean[len(x1)-1:], c=SSP_colors['ssp245'], lw=2)
        plt.plot(x2, ssp126_rmean[len(x1)-1:], c=SSP_colors['ssp126'], lw=2)
```

(continues on next page)

(continued from previous page)

```
    except Exception as e:
        raise Exception('Failed to make plot. {}'.format(e))


    plt.xlim(dt.strptime('1940-01-01', "%Y-%m-%d"), dt.strptime('2100-01-02', "%Y-%m-
↪%d"))

    #         ax1.set_xticks(fontsize=16, rotation=45) # ax1.set_
↪xticklabels(rotation=45, fontsize=12)
    plt.grid(axis='y')  # .grid_line_alpha=0.3

    plt.title(title)


    from matplotlib.offsetbox import TextArea, VPacker, AnnotationBbox
#         from pylab import *
#         fig = figure(1)
#         ax = gca()
    texts = ['SSP 126','SSP 245','SSP 585','historique', 'observation']
    colors = [SSP_colors['ssp126'],SSP_colors['ssp245'],SSP_colors['ssp585'],
↪'darkgrey','black']
    Texts = []
    for t,c in zip(texts,colors):
        Texts.append(TextArea(t,textprops=dict(color=c)))
    texts_vbox = VPacker(children=Texts,pad=0,sep=0)
    ann = AnnotationBbox(texts_vbox,(.02,.8), xycoords=ax.transAxes,
                         box_alignment=(0,.5),
                         bboxprops = dict(facecolor='red',boxstyle='round', alpha=0.5,
↪ color='lightgrey'))
    ann.set_figure(fig)
    fig.artists.append(ann)

    try:
        plt.axvline(dt.strptime('1985-01-01', "%Y-%m-%d"), color='gray', linestyle='-.
↪', alpha=0.5)
        plt.axvline(dt.strptime('2030-01-01', "%Y-%m-%d"), color='gray', linestyle='--
↪', alpha=0.5)
        plt.axvline(dt.strptime('2050-01-01', "%Y-%m-%d"), color='gray', linestyle='--
↪', alpha=0.5)
        plt.axvline(dt.strptime('2085-01-01', "%Y-%m-%d"), color='gray', linestyle='--
↪', alpha=0.5)

    except:
        raise Exception('Failed to make scatters')

    # include Observation
    if observation is not None:
        try:
            import xarray as xr

            ds = xr.open_dataset(observation, drop_variables='height', decode_
↪cf=decode_cf)
            if delta == 0:
                obs = ds.to_dataframe()
            else:
```

(continues on next page)

(continued from previous page)

```
                obs = ds.to_dataframe() + delta

            obs_rm = obs.rolling(window=window, center=True, min_periods=16 ).mean()
↪#   closed='right',

            obs_rollmean = obs_rm - obs_rm.values[ts_ref]

            if decode_cf is True:
                mi = obs.index
                # plt.plot(mi.get_level_values('time'), obs, c='black', lw=1,␣
↪linestyle='--')
                plt.plot(mi.get_level_values('time'), obs_rollmean, c='black', lw=3,␣
↪linestyle='--' )
            else:
                # plt.plot(x, obs, c='black', lw=1, linestyle='--')
                plt.plot(x, obs_rollmean, c='black', lw=3, linestyle='--')

#             plt.scatter(dt.strptime('1985', "%Y"), 12, c='black', s=20)
#             plt.annotate(12, (dt.strptime('1985', "%Y"), 12), c='red')

        except Exception as e:
            raise Exception('Failed to plot observation {}'.format(e))
    try:
        output_png = fig2plot(fig=fig, file_extension=file_extension, dir_output=dir_
↪output)
        plt.close()

    except Exception as e:
        raise Exception('Failed to make boxplots. {}'.format(e))

    try:
        out_csv = "{}{}.csv".format(path_csv, variable)
        arr = [x , obs_rollmean.values, ssp126_rmean, ssp126_q05, ssp126_q95,
                                        ssp245_rmean, ssp245_q05, ssp245_q95,
                                        ssp585_rmean, ssp585_q05, ssp585_q95, ]
        pd.DataFrame(arr).transpose().to_csv(out_csv, header=['Date', 'OBS',
                                                              'ssp126_median',
↪'ssp126_q05', 'ssp126_q95',
                                                              'ssp245_median',
↪'ssp245_q05', 'ssp245_q95',
                                                              'ssp585_median',
↪'ssp585_q05', 'ssp585_q95',
                                                             ])


    except Exception as e:
        raise Exception('failed to write csv file')

    try:
        fig = plt.figure()
        ax = fig.add_subplot(111)
        y = [1, 2, 3, 4, 5, 4, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1]
        col_labels = ['1871-1900','1971-2000','2001-2019','2016-2045', '2036-2065',
↪'2071-2100']
        row_labels = ['observation', 'ssp126', 'ssp245' ,'ssp585']

        ts_ref18 = x.get_loc(dt.strptime("1885", "%Y"), method='nearest')
        ts_ref = x.get_loc(dt.strptime("1985", "%Y"), method='nearest')
```

(continues on next page)

(continued from previous page)

```
        ts_15 = x.get_loc(dt.strptime("2015", "%Y"), method='nearest')
        ts_30 = x.get_loc(dt.strptime("2030", "%Y"), method='nearest')
        ts_50 = x.get_loc(dt.strptime("2050", "%Y"), method='nearest')
        ts_85 = x.get_loc(dt.strptime("2085", "%Y"), method='nearest')

        table_vals = [[np.round(obs_rm.values[ts_ref18],2)[0], np.round(obs_rm.
↪values[ts_ref],2)[0],
                       np.round(obs_rm.values[ts_15],2)[0], '-' , '-', '-'], # [obs_
↪rollmean[ts_ref]
#                       [round(mean_hist[ts_ref],2), '-', '-', '-'],
                      ['-', '-','-', round(ssp126_rmean[ts_30],2), round(ssp126_
↪rmean[ts_50],2),round(ssp126_rmean[ts_85],2)],  # 126
                      ['-', '-','-', round(ssp245_rmean[ts_30],2), round(ssp245_
↪rmean[ts_50],2),round(ssp245_rmean[ts_85],2)],
                      ['-', '-','-', round(ssp585_rmean[ts_30],2), round(ssp585_
↪rmean[ts_50],2),round(ssp585_rmean[ts_85],2)]
                      ] # round(ssp126_rmean[ts_ref],2)

        # Draw table
        the_table = plt.table(cellText=table_vals,
                              colWidths=[0.1] * 30,
                              rowLabels=row_labels,
                              colLabels=col_labels,
                              loc='center')
        the_table.auto_set_font_size(False)
        the_table.set_fontsize(24)
        the_table.scale(4, 4)

        # Removing ticks and spines enables you to get the figure only with table
        plt.tick_params(axis='x', which='both', bottom=False, top=False,␣
↪labelbottom=False)
        plt.tick_params(axis='y', which='both', right=False, left=False,␣
↪labelleft=False)

        for pos in ['right','top','bottom','left']:
            plt.gca().spines[pos].set_visible(False)
        table_png = fig2plot(fig=fig, file_extension=file_extension, dir_output=dir_
↪output)
        plt.close()

    except Exception as e:
        raise Exception('Failed to make table. {}'.format(e))

    return output_png , table_png , out_csv
```

[94]:
```
i = 13 # 13

indices_files = join(path_indices, indices[i])
files = [join(path_indices,indices[i],f) for f in listdir(indices_files)]

observation = join(path_obs, '{}_yr_montsouris-observation.nc'.format(indices[i]))
```

[95]:
```
output_png, table_png, csv_file = plot_ssp_uncertainty_anormalie(resource=files,␣
↪variable=indices[i], title=titles[i],
                                                 observation=observation, decode_cf =␣
↪False,
```

(continues on next page)

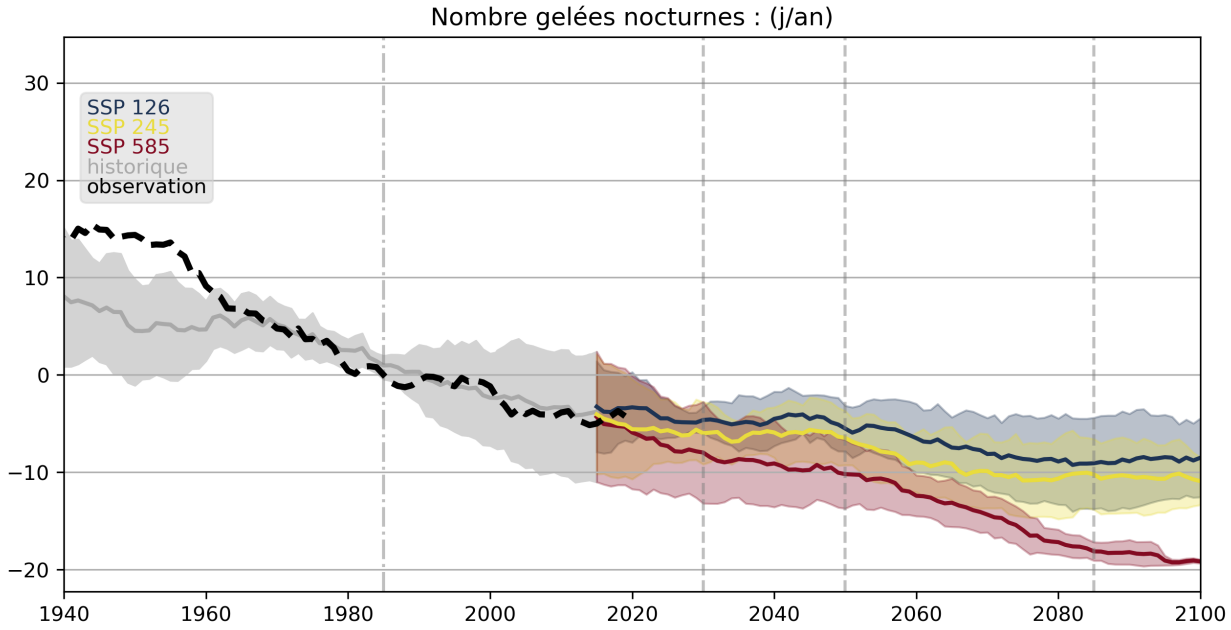
(continued from previous page)

```
                                                figsize=(10, 5),  delta=delta[i],
                                                dir_output=path_pics
                                                )
Image(output_png, width=1000) # , table_png
```

```
/home/nils/anaconda3/envs/flyingpigeon/lib/python3.7/site-packages/pandas/core/
↪indexing.py:671: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/
↪user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_with_indexer(indexer, value)
```

```
all scenarios are seperated
rolling mean calculated for all input data
quantile calculated for all input ssp126 data
quantile calculated for all input ssp245 data
quantile calculated for all input ssp585 data
```

[95]:



[24]:

```
Image(table_png, width=1000)
```

[24]:

	1871-1900	1971-2000	2001-2019	2016-2045	2036-2065	2071-2100
observation	104.04	111.0	108.85	-	-	-
ssp126	-	-	-	-2.63	-3.41	-5.11
ssp245	-	-	-	-5.93	-2.63	-5.71
ssp585	-	-	-	-0.63	-3.39	-5.61

[48]:

```
# df_smooth.plot()
```

(continues on next page)

(continued from previous page)

```
x = pd.to_datetime(df_smooth.index.values)
ts_ref = x.get_loc(dt.strptime("1985", "%Y"), method='nearest')
ref_val = np.nanmean(df_smooth.values, axis=1)[ts_ref]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-48-b6e47fd8fd75> in <module>
      1 # df_smooth.plot()
      2
----> 3 x = pd.to_datetime(df_smooth.index.values)
      4 ts_ref = x.get_loc(dt.strptime("1985", "%Y"), method='nearest')
      5 ref_val = np.nanmean(df_smooth.values, axis=1)[ts_ref]

NameError: name 'df_smooth' is not defined
```

[22]:
```
import numpy
a = numpy.asarray([ [1,2,3], [4,5,6], [7,8,9] ])
numpy.savetxt("foo.csv", a, delimiter=",", header='OBS',)
```

[32]:
```
pd.DataFrame?
```

[75]:
```
# dt = dt.strptime("2016", "%Y")
# if decode_cf is True:
#                 mi = obs.index
#                 # plt.plot(mi.get_level_values('time'), obs, c='black', lw=1,␣
↪linestyle='--')
#                 plt.plot(mi.get_level_values('time'), obs_rollmean, c='black', lw=3,
↪ linestyle='--' )
#             else:
#                 # plt.plot(x, obs, c='black', lw=1, linestyle='--')
#                 plt.plot(x, obs_rollmean, c='black', lw=3, linestyle='--')
```

[44]:
```
ts_ref
```

[44]:
```
135
```

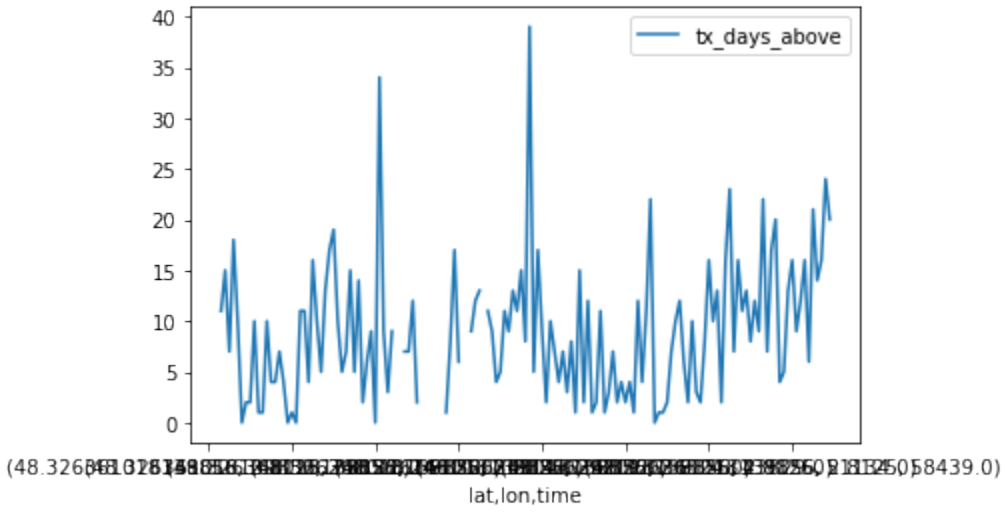
[113]:
```
# ts = dt.strptime("2016", "%Y")
# x.index.get_loc(ts, method='nearest')
```

[47]:
```
x.values[176]
```

[47]:
```
array([nan])
```

[154]:
```
i = 11 # 10  9   8 7  6  5
observation = join(path, '{}_yr_montsouris-observation.nc'.format(indices[i]))
ds = xr.open_dataset(observation, drop_variables='height',  decode_cf=False)
df_obs = ds.to_dataframe()
df_obs.plot()
```

[154]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8de8d8aed0>
```



[51]:
```
from birdy import WPSClient
from os import listdir, path

from shutil import move
from os.path import join
fp_url = 'http://localhost:8093'
fp = WPSClient(url=fp_url, progress=True)
```

[16]:
```
def plot_ssp_uncertainty(resource, variable, ylim=None, title=None, observation=None,
                         file_extension='png', delta=0, window=None, dir_
↪output=None,
                         figsize=(10, 10)):
    """
    creates a png file containing the appropriate uncertainty plot.

    :param resource: list of files containing the same variable
    :param delta: set a delta for the values e.g. -273.15 to convert Kelvin to Celsius
    :param variable: variable to be visualised. If None (default), variable will be␣
↪detected
    :param ylim: Y-axis limitations: tuple(min,max)
    :param title: string to be used as title
    :param observation:
    :param figsize: figure size defult=(10,10)
    :param window: windowsize of the rolling mean

    :returns str: path/to/file.png
    """
    from flyingpigeon.plt_ncdata import ts_data
    from flyingpigeon.nc_utils import sortssp_by_drsname

    try:
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=figsize, facecolor='w',␣
↪edgecolor='k', sharey=True)
        plt.subplots_adjust( wspace=0, hspace=0.2) #

#         fig = plt.figure(figsize=figsize, dpi=600, facecolor='w', edgecolor='k')
    #         LOGGER.debug('Start visualisation spaghetti plot')
    #         === prepare invironment
```

(continues on next page)

(continued from previous page)

```
        if type(resource) != list:
            resource = [resource]
    #         var = get_variable(nc)
    #         if variable is None:
    #             variable = get_variable(resource[0])
    #         LOGGER.info('plot values preparation done')
    except Exception as ex:
        print("plot values preparation failed {}".format(ex))
    #         LOGGER.exception(msg)
    #         raise Exception(msg)
    try:
        dic = sortssp_by_drsname(resource)  # sort_by_filename(resource, historical_
↪concatination=True)
        df = ts_data(dic, delta=delta)
    except Exception as ex:
        print("failed to sort data".format(e))

    ###################################
    # serach datasets according to scenario
    try:
        ssp119 = [ds for ds in df.columns if 'ssp119' in ds]
        ssp126 = [ds for ds in df.columns if 'ssp126' in ds]
        ssp245 = [ds for ds in df.columns if 'ssp245' in ds]
        ssp370 = [ds for ds in df.columns if 'ssp370' in ds]
        ssp434 = [ds for ds in df.columns if 'ssp434' in ds]
        ssp460 = [ds for ds in df.columns if 'ssp460' in ds]
        ssp585 = [ds for ds in df.columns if 'ssp585' in ds]

        print('all scenarios are seperated')
    except Exception as e:
        print('failed to split scenarios {}'.format(e))


    window = 30 # 30 years

    if len(df.index.values) >= window * 2:
        # TODO: calculate windowsize according to timestapms (day,mon,yr ... with get_
↪frequency)
        df_smooth = df.rolling(window=window, center=True).mean()
        print('rolling mean calculated for all input data')
    else:
        df_smooth = df.copy()
        print('timeseries too short for moving mean')
        fig.text(0.95, 0.05, '!!! timeseries too short for moving mean over 30years !!
↪!',
                 fontsize=20, color='red',
                 ha='right', va='bottom', alpha=0.5)



    try:
        df_ssp119 = df[ssp119].rolling(window=window, center=True, min_periods=2).
↪mean()
        df_ssp126 = df[ssp126].rolling(window=window, center=True, min_periods=2).
↪mean()
        df_ssp245 = df[ssp245].rolling(window=window, center=True, min_periods=2).
↪mean()
```

(continues on next page)

(continued from previous page)

```
        df_ssp370 = df[ssp370].rolling(window=window, center=True, min_periods=2).
↪mean()
        df_ssp434 = df[ssp434].rolling(window=window, center=True, min_periods=2).
↪mean()
        df_ssp460 = df[ssp460].rolling(window=window, center=True, min_periods=2).
↪mean()
        df_ssp585 = df[ssp585].rolling(window=window, center=True, min_periods=2).
↪mean()
    except Exception as e:
        print('failed to group scenarios {}'.format(e))


    ####################
    # calculation of mean and uncertainties

    # ssp119
    try:
        ssp119_rmean = np.squeeze(df_ssp119.quantile([0.5], axis=1,).values)
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp119_q05 = np.squeeze(df_ssp119.quantile([0.10], axis=1,).values)
        ssp119_q33 = np.squeeze(df_ssp119.quantile([0.33], axis=1,).values)
        ssp119_q66 = np.squeeze(df_ssp119.quantile([0.66], axis=1,).values)
        ssp119_q95 = np.squeeze(df_ssp119.quantile([0.90], axis=1,).values)
        print('quantile calculated for all input ssp119 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))


    # ssp126
    try:
        ssp126_rmean = np.squeeze(df_ssp126.quantile([0.5], axis=1,).values)
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp126_q05 = np.squeeze(df_ssp126.quantile([0.10], axis=1,).values)
        ssp126_q33 = np.squeeze(df_ssp126.quantile([0.33], axis=1,).values)
        ssp126_q66 = np.squeeze(df_ssp126.quantile([0.66], axis=1,).values)
        ssp126_q95 = np.squeeze(df_ssp126.quantile([0.90], axis=1,).values)
        print('quantile calculated for all input ssp126 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))

    # ssp245
    try:
        ssp245_rmean = np.squeeze(df_ssp245.quantile([0.5], axis=1,).values)
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp245_q05 = np.squeeze(df_ssp245.quantile([0.10], axis=1,).values)
        ssp245_q33 = np.squeeze(df_ssp245.quantile([0.33], axis=1,).values)
        ssp245_q66 = np.squeeze(df_ssp245.quantile([0.66], axis=1,).values)
        ssp245_q95 = np.squeeze(df_ssp245.quantile([0.90], axis=1,).values)
        print('quantile calculated for all input ssp245 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))

    # ssp370
    try:
        ssp370_rmean = np.squeeze(df_ssp370.quantile([0.5], axis=1,).values)
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp370_q05 = np.squeeze(df_ssp370.quantile([0.10], axis=1,).values)
```

(continues on next page)

(continued from previous page)

```
        ssp370_q33 = np.squeeze(df_ssp370.quantile([0.33], axis=1,).values)
        ssp370_q66 = np.squeeze(df_ssp370.quantile([0.66], axis=1,).values)
        ssp370_q95 = np.squeeze(df_ssp370.quantile([0.90], axis=1,).values)
        print('quantile calculated for all input ssp370 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))

    # ssp460
    try:
        ssp460_rmean = np.squeeze(df_ssp460.quantile([0.5], axis=1,).values)
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp460_q05 = np.squeeze(df_ssp460.quantile([0.10], axis=1,).values)
        ssp460_q33 = np.squeeze(df_ssp460.quantile([0.33], axis=1,).values)
        ssp460_q66 = np.squeeze(df_ssp460.quantile([0.66], axis=1,).values)
        ssp460_q95 = np.squeeze(df_ssp460.quantile([0.90], axis=1,).values)
        print('quantile calculated for all input ssp460 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))

    # ssp585
    try:
        ssp585_rmean = np.squeeze(df_ssp585.quantile([0.5], axis=1,).values)
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp585_q05 = np.squeeze(df_ssp585.quantile([0.10], axis=1,).values)
        ssp585_q33 = np.squeeze(df_ssp585.quantile([0.33], axis=1,).values)
        ssp585_q66 = np.squeeze(df_ssp585.quantile([0.66], axis=1,).values)
        ssp585_q95 = np.squeeze(df_ssp585.quantile([0.90], axis=1,).values)
        print('quantile calculated for all input ssp585 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))


    ###################
    # plot

    try:
        x = pd.to_datetime(df.index.values)
#                df[(df['date'] > '2000-6-1') & (df['date'] <= '2000-6-10')]
        x1 = x[x <= dt.strptime('2015-12-31', "%Y-%m-%d")]
        x2 = x[len(x1)-1:]  # -1 to catch up with the last historical value

        ax1.fill_between(x, ssp126_q05, ssp126_q95, alpha=1, color='lightgrey')
        ax1.fill_between(x, ssp245_q05, ssp245_q95, alpha=1, color='lightgrey')
        ax1.fill_between(x, ssp370_q05, ssp370_q95, alpha=1, color='lightgrey')
        ax1.fill_between(x, ssp460_q05, ssp460_q95, alpha=1, color='lightgrey')
        ax1.fill_between(x, ssp585_q05, ssp585_q95, alpha=1, color='lightgrey')


        ax1.fill_between(x, ssp126_q33, ssp126_q66, alpha=1, color='darkgrey')
        ax1.fill_between(x, ssp245_q33, ssp245_q66, alpha=1, color='darkgrey')
        ax1.fill_between(x, ssp370_q33, ssp370_q66, alpha=1, color='darkgrey')
        ax1.fill_between(x, ssp460_q33, ssp460_q66, alpha=1, color='darkgrey')
        ax1.fill_between(x, ssp585_q33, ssp585_q66, alpha=1, color='darkgrey')

        ax1.plot(x1, ssp126_rmean[:len(x1)], c='dimgrey', lw=2)

        ax1.plot(x2, ssp585_rmean[len(x1)-1:], c=SSP_colors['ssp585'], lw=2)
```

(continues on next page)

(continued from previous page)

```
        ax1.plot(x2, ssp460_rmean[len(x1)-1:], c=SSP_colors['ssp460'], lw=2)
#         plt.plot(x2, ssp370_rmean[len(x1)-1:], c='#63bce4', lw=2)
        ax1.plot(x2, ssp370_rmean[len(x1)-1:], c=SSP_colors['ssp370'], lw=2)
        ax1.plot(x2, ssp245_rmean[len(x1)-1:], c=SSP_colors['ssp245'], lw=2)
        ax1.plot(x2, ssp126_rmean[len(x1)-1:], c=SSP_colors['ssp126'], lw=2)
        ax1.plot(x2, ssp119_rmean[len(x1)-1:], c=SSP_colors['ssp119'], lw=2)

        left = 0.1
        bottom = 0.1
        top = 0.8
        right = 0.8

        ref_start = dt.strptime('1971-01-01', "%Y-%m-%d")
        ref_end = dt.strptime('2000-12-31', "%Y-%m-%d")

        ref_start2 = dt.strptime('1985-01-01', "%Y-%m-%d")
        ref_end2 = dt.strptime('2015-12-31', "%Y-%m-%d")


        ##############################
        ## include les reference lines

        try:
            ref= np.nanmedian(np.mean(df[(df.index >= ref_start )
                                & (df.index <= ref_end)]))
            print('ref = {}'.format(ref))

#             ax1.hlines(ref, ref_start, ref_end,
#                    colors='k', linestyles='solid',
#                    label='reference')

            ax2.hlines(ref, 1.2, 7.5 , colors='grey', linestyles='solid',
                       label='reference')

        except Exception as e:
            print('failed to get the mean {}'.format(e))

        print('timeseries uncertainty plot done for %s' % variable)
    except Exception as e:
        raise Exception('failed to calculate quantiles. {}'.format(e))

    try:

        # ax1.plot(x, y)
        # plot(x, -y)

        bp_ref = ax2.boxplot(np.ravel(df[ssp126][len(ssp126_rmean)-131:-101].values),
                             positions = [1], widths = 0.1,
                             whis=(10, 90), sym='', patch_artist=True)

        for box in bp_ref['boxes']:
            # change outline color
#             box.set(color='red', linewidth=2)
            # change fill color
            box.set(facecolor = 'lightgrey' )
            # change hatch
#             box.set(hatch = '/')
```

(continues on next page)

(continued from previous page)

```
        # 2030
        bplots = ax2.boxplot([np.ravel(df[ssp126][len(ssp126_rmean)-85:-55].values),
                              np.ravel(df[ssp245][len(ssp126_rmean)-85:-55].values),
                              np.ravel(df[ssp460][len(ssp126_rmean)-85:-55].values),
                              np.ravel(df[ssp370][len(ssp126_rmean)-85:-55].values),
                              np.ravel(df[ssp585][len(ssp126_rmean)-85:-55].values)],
                              positions = [3.8,3.9,4,4.1,4.2], widths = 0.1,
                              whis=(10, 90), sym='', patch_artist=True) # , ssp370_
↪rmean
        plt.setp(bplots['boxes'][0], color=SSP_colors['ssp126'])
        plt.setp(bplots['boxes'][1], color=SSP_colors['ssp245'])
        plt.setp(bplots['boxes'][2], color=SSP_colors['ssp460'])
        plt.setp(bplots['boxes'][3], color=SSP_colors['ssp370'])
        plt.setp(bplots['boxes'][4], color=SSP_colors['ssp585'])

        bplots = ax2.boxplot([np.ravel(df[ssp126][len(ssp126_rmean)-66:-35].values),
                              np.ravel(df[ssp245][len(ssp126_rmean)-66:-35].values),
                              np.ravel(df[ssp460][len(ssp126_rmean)-66:-35].values),
                              np.ravel(df[ssp370][len(ssp126_rmean)-66:-35].values),
                              np.ravel(df[ssp585][len(ssp126_rmean)-66:-35].values)],
                              positions = [4.8,4.9,5,5.1,5.2], widths = 0.1,
                              whis=(10, 90), sym='', patch_artist=True)
        plt.setp(bplots['boxes'][0], color=SSP_colors['ssp126'])
        plt.setp(bplots['boxes'][1], color=SSP_colors['ssp245'])
        plt.setp(bplots['boxes'][2], color=SSP_colors['ssp460'])
        plt.setp(bplots['boxes'][3], color=SSP_colors['ssp370'])
        plt.setp(bplots['boxes'][4], color=SSP_colors['ssp585'])

        bplots = ax2.boxplot([np.ravel(df[ssp126][len(ssp126_rmean)-31:-1].values),
                              np.ravel(df[ssp245][len(ssp126_rmean)-31:-1].values),
                              np.ravel(df[ssp460][len(ssp126_rmean)-31:-1].values),
                              np.ravel(df[ssp370][len(ssp126_rmean)-31:-1].values),
                              np.ravel(df[ssp585][len(ssp126_rmean)-31:-1].values)],
                              positions = [6.8,6.9,7,7.1,7.2], widths = 0.1,
                              whis=(10, 90), sym='', patch_artist=True)
        plt.setp(bplots['boxes'][0], color=SSP_colors['ssp126'])
        plt.setp(bplots['boxes'][1], color=SSP_colors['ssp245'])
        plt.setp(bplots['boxes'][2], color=SSP_colors['ssp460'])
        plt.setp(bplots['boxes'][3], color=SSP_colors['ssp370'])
        plt.setp(bplots['boxes'][4], color=SSP_colors['ssp585'])


        from matplotlib.offsetbox import TextArea, VPacker, AnnotationBbox
#         from pylab import *
#         fig = figure(1)
#         ax = gca()
        texts = ['SSP 126','SSP 245','SSP 460','SSP 370','SSP 585']
        colors = [SSP_colors['ssp126'],SSP_colors['ssp245'],SSP_colors['ssp460'],SSP_
↪colors['ssp370'],SSP_colors['ssp585']]
        Texts = []
        for t,c in zip(texts,colors):
            Texts.append(TextArea(t,textprops=dict(color=c)))
        texts_vbox = VPacker(children=Texts,pad=0,sep=0)
        ann = AnnotationBbox(texts_vbox,(.02,.8), xycoords=ax1.transAxes,
                              box_alignment=(0,.5),
                              bboxprops = dict(facecolor='red',boxstyle='round',color=
↪'darkgrey'))
```

(continues on next page)

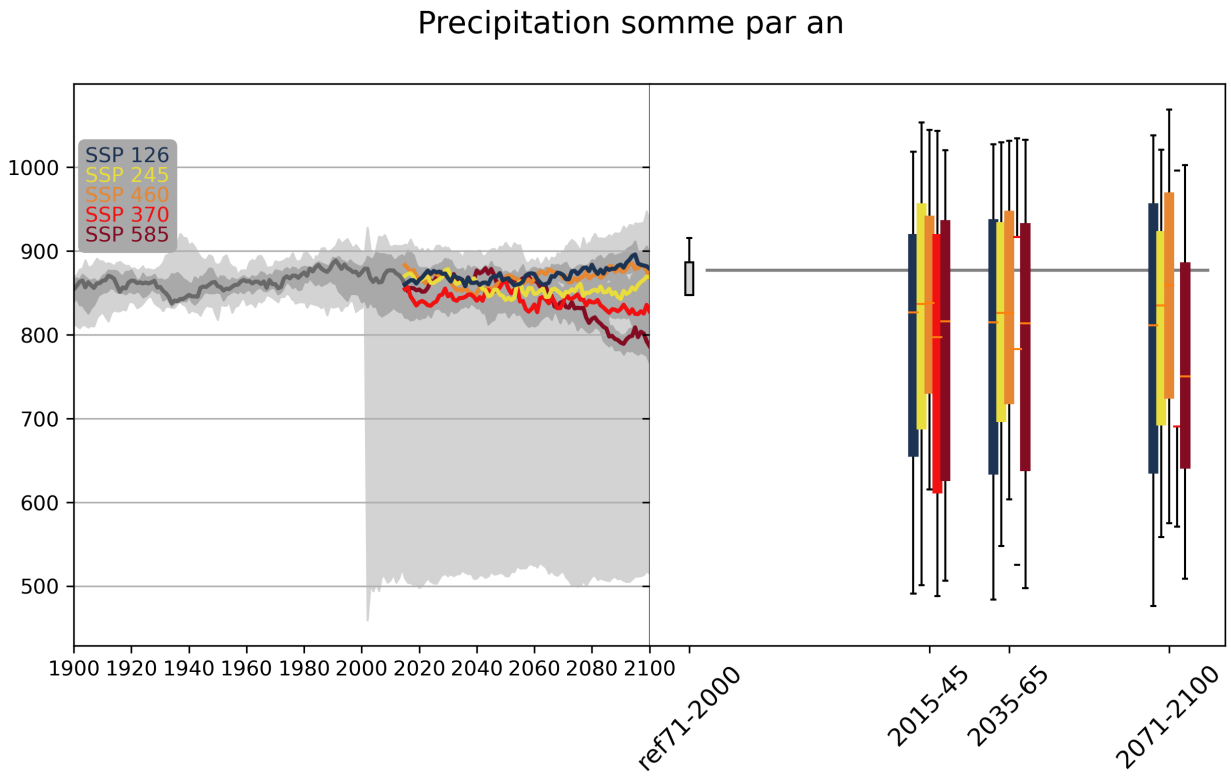
(continued from previous page)

```
        ann.set_figure(fig)
        fig.artists.append(ann)

        fig.suptitle(title, fontsize=15)

        plt.yticks(fontsize=16,)
        plt.ylim(ylim)

        ax1.set_xlim(dt.strptime('1900-01-01', "%Y-%m-%d"), dt.strptime('2100-01-02',
↪"%Y-%m-%d"))

#         ax1.set_xticks(fontsize=16, rotation=45) # ax1.set_xticklabels(rotation=45,␣
↪fontsize=12)
        ax1.grid(axis='y')  # .grid_line_alpha=0.3


        bp_ticks = [1,4,5,7]
        bp_labels = ['ref71-2000','2015-45','2035-65', '2071-2100']

        ax2.set_xticks(bp_ticks)
        ax2.set_xticklabels(bp_labels, rotation=45, fontsize=12)
#         ax2.set_xlables
        ax2.spines['left'].set_visible(False)

        # setBoxColors(bp)
        # bd = plt.boxplot(ssp370_rmean[len(ssp126_rmean)-45:-15], positions = [4],␣
↪widths = 0.1, ) # , ssp370_rmean
        # bd = plt.boxplot(ssp585_rmean[len(ssp126_rmean)-45:-15], positions = [5],␣
↪widths = 0.1) # , ssp370_rmean
        # setBoxColors(bp)

    except Exception as e:
        raise Exception('Failed to make boxplots. {}'.format(e))

    try:
        output_png = fig2plot(fig=fig, file_extension=file_extension, dir_output=dir_
↪output)
        plt.close()

    except Exception as e:
        raise Exception('Failed to make boxplots. {}'.format(e))


    return output_png
```

[64]:
```
i = 3

indices_files = join(path_indices, indices[i])
files = [join(path_indices,indices[i],f) for f in listdir(indices_files)]

plot = plot_ssp_uncertainty(files, indices[i], title=titles[i],
                            figsize=(10, 5), delta=-273.15,
                            dir_output=path_pics)

Image(plot, width=1000)
```

```
all scenarios are seperated
rolling mean calculated for all input data
quantile calculated for all input ssp119 data
quantile calculated for all input ssp126 data
quantile calculated for all input ssp245 data
quantile calculated for all input ssp370 data
quantile calculated for all input ssp460 data
quantile calculated for all input ssp585 data
ref = 877.0137451171875
timeseries uncertainty plot done for prcptot
```

[64]:



[61]:

```
i = 1 # 11 # 10  9   8 7  6  5

indices_files = join(path_indices, indices[i])
files = [join(path_indices,indices[i],f) for f in listdir(indices_files)]

out = fp.plot_spaghetti(files, title=titles[i], figsize=(10, 5))
```

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

```
 owslib.wps.WPSException : {'code': 'NoApplicableCode', 'locator': 'None', 'text':
↪"Process error: method=wps_plot_spaghetti.py._handler, line=162, msg=spaghetti plot␣
↪failed : plot values preparation failed from_bounds() missing 1 required positional␣
↪argument: 'height'"}
```

[19]:

```
from flyingpigeon.plt_ncdata import ts_data
from flyingpigeon.nc_utils import sortssp_by_drsname

# dic = sortssp_by_drsname(files)  # sort_by_filename(resource, historical_
↪concatination=True)
```

(continues on next page)

(continued from previous page)

```
# df = ts_data(dic, delta=delta)
```

[55]:
```
out.get()
```

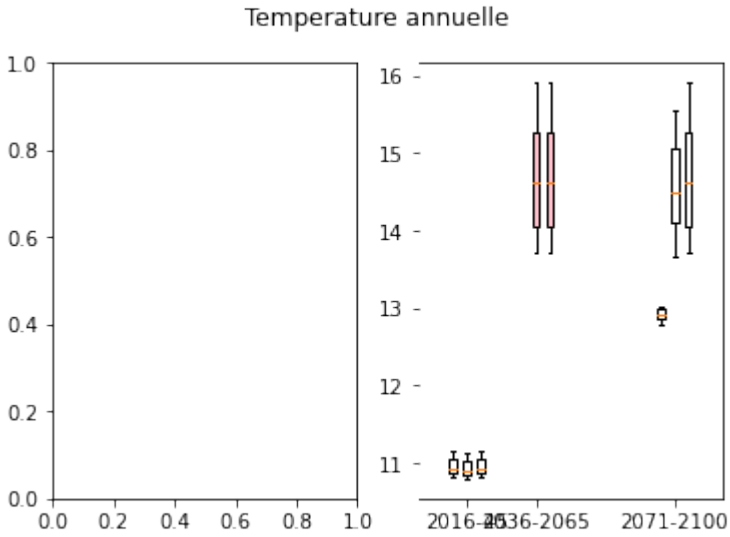
[55]:
```
plot_spaghettiResponse(
    plotout_spaghetti='http://127.0.0.1:8093/outputs/cf23803a-dc9d-11ea-9ff5-
↪9cb6d08a53e7/tmptgslcj61.png'
)
```

[175]:
```
def setBoxColors(bp):
    plt.setp(bp['boxes'][0], color='blue')
    plt.setp(bp['caps'][0], color='blue')
    plt.setp(bp['caps'][1], color='blue')
    plt.setp(bp['whiskers'][0], color='blue')
    plt.setp(bp['whiskers'][1], color='blue')
#     plt.setp(bp['fliers'][0], color='blue')
#     plt.setp(bp['fliers'][1], color='blue')
    plt.setp(bp['medians'][0], color='blue')

    plt.setp(bp['boxes'][1], color='red')
    plt.setp(bp['caps'][2], color='red')
    plt.setp(bp['caps'][3], color='red')
    plt.setp(bp['whiskers'][2], color='red')
    plt.setp(bp['whiskers'][3], color='red')
#     plt.setp(bp['fliers'][2], color='red')
#     plt.setp(bp['fliers'][3], color='red')
    plt.setp(bp['medians'][1], color='red')

ref_start = dt.strptime('1971-01-01', "%Y-%m-%d")
ref_end = dt.strptime('2000-12-31', "%Y-%m-%d")

# fig, axes = plt.subplots(ncols=3, sharey=True)
# fig.subplots_adjust(wspace=0)

# ax.boxplot([data[name][item] ssp126_rmean[len(ssp126_rmean)-45:-15]

#             for item in ['A', 'B', 'C']])
# ax.set(xticklabels=['A', 'B', 'C'], xlabel=name)
# ax.margins(0.05) # Optional


fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Temperature annuelle')

# ax1.plot(x, y)
# plot(x, -y)

bplots = ax2.boxplot([ssp126_rmean[len(ssp126_rmean)-145:-115],
                  ssp370_rmean[len(ssp126_rmean)-145:-115],
                  ssp585_rmean[len(ssp126_rmean)-145:-115]],
                  positions = [0.8,1,1.2], widths = 0.1) # , ssp370_rmean


# bplots['boxes'][0].set_fa
```

(continues on next page)

(continued from previous page)

```
# # fill with colors
# colors = ['pink', 'lightblue', 'lightgreen']

bplot1 = ax2.boxplot(ssp585_rmean[len(ssp585_rmean)-45:-15],
                 positions = [2], widths = 0.1, patch_artist=True)

bplot2 = ax2.boxplot(ssp585_rmean[len(ssp585_rmean)-45:-15],
                 positions = [2.2], widths = 0.1, patch_artist=True)

# # colors =
# for bp in (bd1,bd2) :
#     for patch, color in zip(bp['boxes'], ['blue','green']):
#         patch.set_facecolor(color)

# fill with colors
colors = ['pink', 'lightblue', 'lightgreen']
for bplot in (bplot1, bplot2):
    for patch, color in zip(bplot['boxes'], colors):
        patch.set_facecolor(color)



bd = ax2.boxplot([ssp126_rmean[len(ssp126_rmean)-45:-15],
                  ssp370_rmean[len(ssp126_rmean)-45:-15],
                  ssp585_rmean[len(ssp126_rmean)-45:-15]],
                 positions = [3.8,4,4.2], widths = 0.1) # , ssp370_rmean

# bp['boxes'][0,1,2], color='red'

ticks = [1,2,4]
labels = ['2016-45','2036-2065' , '2071-2100']

# ax2.xlable
plt.xticks(ticks, labels)

ax2.spines['left'].set_visible(False)

# setBoxColors(bp)
# bd = plt.boxplot(ssp370_rmean[len(ssp126_rmean)-45:-15], positions = [4], widths =␣
↪0.1, ) # , ssp370_rmean

# bd = plt.boxplot(ssp585_rmean[len(ssp126_rmean)-45:-15], positions = [5], widths =␣
↪0.1) # , ssp370_rmean
# setBoxColors(bp)
```



[177]:
```
for key , values in bplot1.items():
    for medline in bplot1[key]:
        try:
            linedata = medline.get_xdata()
            print( '{} : {}'.format(key, linedata) )
        except Exception as e:
            print('failed to get data')
```

```
whiskers : [2. 2.]
whiskers : [2. 2.]
caps : [1.975 2.025]
caps : [1.975 2.025]
failed to get data
medians : [1.95 2.05]
fliers : []
```

[176]:
```
for key, value in bplot1.items():
    val = [v.get_data() for v in value]
    print(key, val )

# ['medians']
# res = {key : [v.get_data() for v in value] for key, value in bplot1.items()}
```

```
whiskers [(array([2., 2.]), array([14.05755857, 13.72003886])), (array([2., 2.]),␣
↪array([15.26835963, 15.91097972]))]
caps [(array([1.975, 2.025]), array([13.72003886, 13.72003886])), (array([1.975, 2.
↪025]), array([15.91097972, 15.91097972]))]
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-176-7e24ba9c56c9> in <module>
      1 for key, value in bplot1.items():
----> 2     val = [v.get_data() for v in value]
      3     print(key, val )
      4
      5 # ['medians']

<ipython-input-176-7e24ba9c56c9> in <listcomp>(.0)
```

(continues on next page)

(continued from previous page)

```
      1 for key, value in bplot1.items():
----> 2     val = [v.get_data() for v in value]
      3     print(key, val )
      4
      5 # ['medians']

AttributeError: 'PathPatch' object has no attribute 'get_data'
```

[168]:
```
bplot1.items()
```

[168]:
```
dict_items([('whiskers', [<matplotlib.lines.Line2D object at 0x7f6409aa67d0>,
 ↪<matplotlib.lines.Line2D object at 0x7f6409aa6d50>]), ('caps', [<matplotlib.lines.
 ↪Line2D object at 0x7f6409abc2d0>, <matplotlib.lines.Line2D object at 0x7f6409abc810>
 ↪]), ('boxes', [<matplotlib.patches.PathPatch object at 0x7f640a1e90d0>]), ('medians
 ↪', [<matplotlib.lines.Line2D object at 0x7f6409abcd10>]), ('fliers', [<matplotlib.
 ↪lines.Line2D object at 0x7f6409a8f250>]), ('means', [])])
```

[171]:
```
medline.
```

[171]:
```
<matplotlib.patches.PathPatch at 0x7f640a1e90d0>
```

[20]:
```
def plot_ssp_uncertainty_reduced(resource, variable, ylim=None, title=None,␣
 ↪observation=None, decode_cf=True,
                                 file_extension='png', delta=0, window=None, dir_
 ↪output=None,
                                 figsize=(10, 10)):
    """
    creates a png file containing the appropriate uncertainty plot.

    :param resource: list of files containing the same variable
    :param delta: set a delta for the values e.g. -273.15 to convert Kelvin to Celsius
    :param variable: variable to be visualised. If None (default), variable will be␣
 ↪detected
    :param ylim: Y-axis limitations: tuple(min,max)
    :param title: string to be used as title
    :param observation: optional data of observations
    :param figsize: figure size defult=(10,10)
    :param decode_cf: decode of netCDF values according cf convention
    :param window: windowsize of the rolling mean

    :returns str: path/to/file.png
    """
    from flyingpigeon.plt_ncdata import ts_data
    from flyingpigeon.nc_utils import sortssp_by_drsname

    try:
        fig = plt.figure(figsize=figsize, facecolor='w', edgecolor='k')
        ax = fig.add_subplot(111)
#         plt.subplots_adjust( wspace=0, hspace=0.2) #

#         fig = plt.figure(figsize=figsize, dpi=600, facecolor='w', edgecolor='k')
    #         LOGGER.debug('Start visualisation spaghetti plot')
    #         === prepare invironment
        if type(resource) != list:
            resource = [resource]
    #         var = get_variable(nc)
```

(continues on next page)

(continued from previous page)

```
    #         if variable is None:
    #             variable = get_variable(resource[0])
    #         LOGGER.info('plot values preparation done')
    except Exception as ex:
        print("plot values preparation failed {}".format(ex))
    #         LOGGER.exception(msg)
    #         raise Exception(msg)
    try:
        dic = sortssp_by_drsname(resource)  # sort_by_filename(resource, historical_
↪concatination=True)
        df = ts_data(dic, delta=delta)
    except Exception as ex:
        print("failed to sort data".format(ex))

    ###################################
    # serach datasets according to scenario
    try:
        ssp126 = [ds for ds in df.columns if 'ssp126' in ds]
        ssp245 = [ds for ds in df.columns if 'ssp245' in ds]
        ssp585 = [ds for ds in df.columns if 'ssp585' in ds]

        print('all scenarios are seperated')
    except Exception as e:
        print('failed to split scenarios {}'.format(e))


    window = 30 # 30 years

    if len(df.index.values) >= window * 2:
        # TODO: calculate windowsize according to timestapms (day,mon,yr ... with get_
↪frequency)
        df_smooth = df.rolling(window=window, center=True).mean()
        print('rolling mean calculated for all input data')
    else:
        df_smooth = df.copy()
        fig.text(0.95, 0.05, '!!! timeseries too short for moving mean over 30years !!
↪!',
                 fontsize=20, color='red',
                 ha='right', va='bottom', alpha=0.5)

    try:
        df_ssp126 = df[ssp126].rolling(window=window, center=True, min_periods=2).
↪mean()
        df_ssp245 = df[ssp245].rolling(window=window, center=True, min_periods=2).
↪mean()
        df_ssp585 = df[ssp585].rolling(window=window, center=True, min_periods=2).
↪mean()
    except Exception as e:
        print('failed to group scenarios {}'.format(e))


    ####################
    # calculation of mean and uncertainties

    # ssp126
    try:
        ssp126_rmean = np.squeeze(df_ssp126.quantile([0.5], axis=1,).values)
```

(continues on next page)

(continued from previous page)

```
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp126_q05 = np.squeeze(df_ssp126.quantile([0.05], axis=1,).values)
        ssp126_q33 = np.squeeze(df_ssp126.quantile([0.33], axis=1,).values)
        ssp126_q66 = np.squeeze(df_ssp126.quantile([0.66], axis=1,).values)
        ssp126_q95 = np.squeeze(df_ssp126.quantile([0.95], axis=1,).values)
        print('quantile calculated for all input ssp126 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))

    # ssp245
    try:
        ssp245_rmean = np.squeeze(df_ssp245.quantile([0.5], axis=1,).values)
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp245_q05 = np.squeeze(df_ssp245.quantile([0.05], axis=1,).values)
        ssp245_q33 = np.squeeze(df_ssp245.quantile([0.33], axis=1,).values)
        ssp245_q66 = np.squeeze(df_ssp245.quantile([0.66], axis=1,).values)
        ssp245_q95 = np.squeeze(df_ssp245.quantile([0.95], axis=1,).values)
        print('quantile calculated for all input ssp245 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))


    # ssp585
    try:
        ssp585_rmean = np.squeeze(df_ssp585.quantile([0.5], axis=1,).values)
        # skipna=False  quantile([0.5], axis=1, numeric_only=False )
        ssp585_q05 = np.squeeze(df_ssp585.quantile([0.05], axis=1,).values)
        ssp585_q33 = np.squeeze(df_ssp585.quantile([0.33], axis=1,).values)
        ssp585_q66 = np.squeeze(df_ssp585.quantile([0.66], axis=1,).values)
        ssp585_q95 = np.squeeze(df_ssp585.quantile([0.95], axis=1,).values)
        print('quantile calculated for all input ssp585 data')
    except Exception as e:
        print('failed to calculate quantiles: {}'.format(e))


    ###################
    # plot

    try:
        x = pd.to_datetime(df.index.values)
#               df[(df['date'] > '2000-6-1') & (df['date'] <= '2000-6-10')]
        x1 = x[x <= dt.strptime('2015-12-31', "%Y-%m-%d")]
        x2 = x[len(x1)-1:]  # -1 to catch up with the last historical value

#         plt.fill_between(x, ssp126_q05, ssp126_q95, alpha=1, color='lightgrey')
#         plt.fill_between(x, ssp245_q05, ssp245_q95, alpha=1, color='lightgrey')
#         plt.fill_between(x, ssp585_q05, ssp585_q95, alpha=1, color='lightgrey')

        plt.fill_between(x1, ssp126_q05[:len(x1)], ssp126_q95[:len(x1)], alpha=1,␣
↪color='lightgrey')
        plt.fill_between(x1, ssp245_q05[:len(x1)], ssp245_q95[:len(x1)], alpha=1,␣
↪color='lightgrey')
        plt.fill_between(x1, ssp585_q05[:len(x1)], ssp585_q95[:len(x1)], alpha=1,␣
↪color='lightgrey')


        plt.fill_between(x2, ssp126_q05[len(x1)-1:], ssp126_q95[len(x1)-1:], alpha=0.
↪3, color=SSP_colors['ssp126'])
```

(continues on next page)

(continued from previous page)

```
        plt.fill_between(x2, ssp245_q05[len(x1)-1:], ssp245_q95[len(x1)-1:], alpha=0.
↪3, color=SSP_colors['ssp245'])
        plt.fill_between(x2, ssp585_q05[len(x1)-1:], ssp585_q95[len(x1)-1:], alpha=0.
↪3, color=SSP_colors['ssp585'])

        mean_hist = np.mean([ssp126_rmean[:len(x1)], ssp245_rmean[:len(x1)] , ssp245_
↪rmean[:len(x1)]],axis=0)

        plt.plot(x1, mean_hist , c='darkgrey', lw=2)
#         plt.plot(x1, ssp245_rmean[:len(x1)], c='dimgrey', lw=2)
#         plt.plot(x1, ssp245_rmean[:len(x1)], c='dimgrey', lw=2)

        plt.plot(x2, ssp585_rmean[len(x1)-1:], c=SSP_colors['ssp585'], lw=2)
        plt.plot(x2, ssp245_rmean[len(x1)-1:], c=SSP_colors['ssp245'], lw=2)
        plt.plot(x2, ssp126_rmean[len(x1)-1:], c=SSP_colors['ssp126'], lw=2)


    except Exception as e:
        raise Exception('Failed to make plot. {}'.format(e))


    plt.xlim(dt.strptime('1940-01-01', "%Y-%m-%d"), dt.strptime('2100-01-02', "%Y-%m-
↪%d"))

    #         ax1.set_xticks(fontsize=16, rotation=45) # ax1.set_
↪xticklabels(rotation=45, fontsize=12)
    plt.grid(axis='y')  # .grid_line_alpha=0.3

    plt.title(title)


    from matplotlib.offsetbox import TextArea, VPacker, AnnotationBbox
#         from pylab import *
#         fig = figure(1)
#         ax = gca()
    texts = ['SSP 126','SSP 245','SSP 585','historique', 'observation']
    colors = [SSP_colors['ssp126'],SSP_colors['ssp245'],SSP_colors['ssp585'],
↪'darkgrey','black']
    Texts = []
    for t,c in zip(texts,colors):
        Texts.append(TextArea(t,textprops=dict(color=c)))
    texts_vbox = VPacker(children=Texts,pad=0,sep=0)
    ann = AnnotationBbox(texts_vbox,(.02,.8), xycoords=ax.transAxes,
                         box_alignment=(0,.5),
                         bboxprops = dict(facecolor='red',boxstyle='round', alpha=0.5,
↪ color='lightgrey'))
    ann.set_figure(fig)
    fig.artists.append(ann)

    try:
        plt.axvline(dt.strptime('1985-01-01', "%Y-%m-%d"), color='gray', linestyle='-.
↪', alpha=0.5)
        plt.axvline(dt.strptime('2030-01-01', "%Y-%m-%d"), color='gray', linestyle='--
↪', alpha=0.5)
        plt.axvline(dt.strptime('2050-01-01', "%Y-%m-%d"), color='gray', linestyle='--
↪', alpha=0.5)
        plt.axvline(dt.strptime('2085-01-01', "%Y-%m-%d"), color='gray', linestyle='--
↪', alpha=0.5)
```

(continues on next page)

(continued from previous page)

```
    except:
        raise Exception('Failed to make scatters')

    # include Observation
    if observation is not None:
        try:
            import xarray as xr

            ds = xr.open_dataset(observation, drop_variables='height', decode_
↪cf=decode_cf)
            if delta == 0:
                obs = ds.to_dataframe()
            else:
                obs = ds.to_dataframe() + delta

            obs_rollmean = obs.rolling(window=window, center=True, min_periods=16 ).
↪mean() #   closed='right',

            if decode_cf is True:
                mi = obs.index
                # plt.plot(mi.get_level_values('time'), obs, c='black', lw=1,␣
↪linestyle='--')
                plt.plot(mi.get_level_values('time'), obs_rollmean, c='black', lw=3,␣
↪linestyle='--' )
            else:
                # plt.plot(x, obs, c='black', lw=1, linestyle='--')
                plt.plot(x, obs_rollmean, c='black', lw=3, linestyle='--')

#             plt.scatter(dt.strptime('1985', "%Y"), 12, c='black', s=20)
#             plt.annotate(12, (dt.strptime('1985', "%Y"), 12), c='red')

        except Exception as e:
            raise Exception('Failed to plot observation {}'.format(e))
    try:
        output_png = fig2plot(fig=fig, file_extension=file_extension, dir_output=dir_
↪output)
        plt.close()

    except Exception as e:
        raise Exception('Failed to make boxplots. {}'.format(e))


    try:
        fig = plt.figure()
        ax = fig.add_subplot(111)
        y = [1, 2, 3, 4, 5, 4, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1]
        col_labels = ['1971-2000','2016-2045', '2036-2065', '2071-2100']
        row_labels = ['observation', 'hist(moyen)', 'ssp126', 'ssp245' ,'ssp585']

        ts_ref = x.get_loc(dt.strptime("1985", "%Y"), method='nearest')
        ts_30 = x.get_loc(dt.strptime("2030", "%Y"), method='nearest')
        ts_50 = x.get_loc(dt.strptime("2050", "%Y"), method='nearest')
        ts_85 = x.get_loc(dt.strptime("2085", "%Y"), method='nearest')

        table_vals = [[np.round(obs_rollmean.values[ts_ref],2)[0], '-' , '-', '-'], #␣
↪[obs_rollmean[ts_ref]
```

(continues on next page)

(continued from previous page)

```
                    [round(mean_hist[ts_ref],2), '-', '-', '-'],
                    [round(ssp126_rmean[ts_ref],2) , round(ssp126_rmean[ts_30],2),␣
↪round(ssp126_rmean[ts_50],2),round(ssp126_rmean[ts_85],2)],  # 126
                    [round(ssp245_rmean[ts_ref],2), round(ssp245_rmean[ts_30],2),␣
↪round(ssp245_rmean[ts_50],2),round(ssp245_rmean[ts_85],2)],
                    [round(ssp585_rmean[ts_ref],2), round(ssp585_rmean[ts_30],2),␣
↪round(ssp585_rmean[ts_50],2),round(ssp585_rmean[ts_85],2)]
                   ]

        # Draw table
        the_table = plt.table(cellText=table_vals,
                              colWidths=[0.1] * 30,
                              rowLabels=row_labels,
                              colLabels=col_labels,
                              loc='center')
        the_table.auto_set_font_size(False)
        the_table.set_fontsize(24)
        the_table.scale(4, 4)

        # Removing ticks and spines enables you to get the figure only with table
        plt.tick_params(axis='x', which='both', bottom=False, top=False,␣
↪labelbottom=False)
        plt.tick_params(axis='y', which='both', right=False, left=False,␣
↪labelleft=False)

        for pos in ['right','top','bottom','left']:
            plt.gca().spines[pos].set_visible(False)
        table_png = fig2plot(fig=fig, file_extension=file_extension, dir_output=dir_
↪output)
        plt.close()

    except Exception as e:
        raise Exception('Failed to make table. {}'.format(e))


    return output_png , table_png
```

[15]:
```
import matplotlib.pyplot as plt
import numpy as np
import random

data = {}
data['dataset1'] = {}
data['dataset2'] = {}
data['dataset3'] = {}

n = 500
for k,v in data.keys():
    upper = random.randint(0, 1000)
    v['A'] = np.random.uniform(0, upper, size=n)
    v['B'] = np.random.uniform(0, upper, size=n)
    v['C'] = np.random.uniform(0, upper, size=n)

fig, axes = plt.subplots(ncols=3, sharey=True)
fig.subplots_adjust(wspace=0)
```

(continues on next page)

(continued from previous page)

```
for ax, name in zip(axes, ['dataset1', 'dataset2', 'dataset3']):
    ax.boxplot([data[name][item] for item in ['A', 'B', 'C']])
    ax.set(xticklabels=['A', 'B', 'C'], xlabel=name)
    ax.margins(0.05) # Optional

plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-3fd229b84c63> in <module>
      9
     10 n = 500
---> 11 for k,v in data.keys():
     12     upper = random.randint(0, 1000)
     13     v['A'] = np.random.uniform(0, upper, size=n)

ValueError: too many values to unpack (expected 2)
```

[16]:

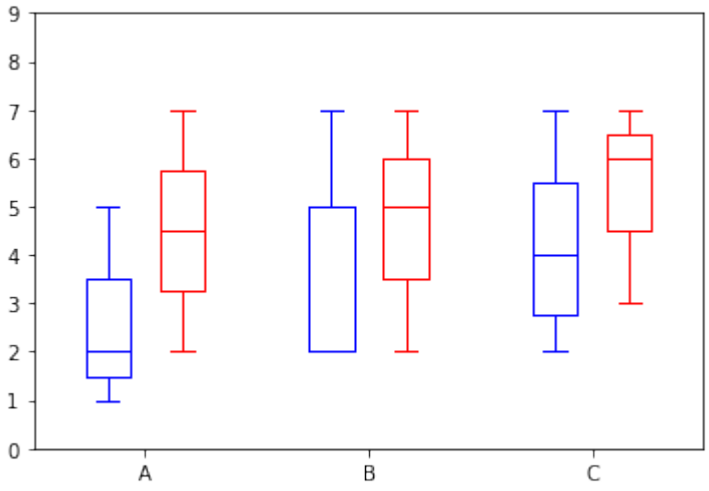
```
    # Some fake data to plot
A= [[1, 2, 5,],  [7, 2]]
B = [[5, 7, 2, 2, 5], [7, 2, 5]]
C = [[3,2,5,7], [6, 7, 3]]

fig = plt.figure()
ax = plt.axes()
# hold(True)

# first boxplot pair
bp = plt.boxplot(A, positions = [1, 2], widths = 0.6)
setBoxColors(bp)

# second boxplot pair
bp = plt.boxplot(B, positions = [4, 5], widths = 0.6)
setBoxColors(bp)

# thrid boxplot pair
bp = plt.boxplot(C, positions = [7, 8], widths = 0.6)
setBoxColors(bp)

# set axes limits and labels
plt.xlim(0,9)
plt.ylim(0,9)
ax.set_xticklabels(['A', 'B', 'C'])
ax.set_xticks([1.5, 4.5, 7.5])

# draw temporary red and blue lines and use them to create a legend
hB, = plot([1,1],'b-')
hR, = plot([1,1],'r-')
legend((hB, hR),('Apples', 'Oranges'))
hB.set_visible(False)
hR.set_visible(False)

savefig('boxcompare.png')
show()
```

```
---------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
TypeError                                 Traceback (most recent call last)
<ipython-input-16-7f1623cbd218> in <module>
     27
     28 # draw temporary red and blue lines and use them to create a legend
---> 29 hB, = plot([1,1],'b-')
     30 hR, = plot([1,1],'r-')
     31 legend((hB, hR),('Apples', 'Oranges'))

TypeError: 'str' object is not callable
```



[111]:
```
plot = plot_ssp_spaghetti(resource=files, variable='tg_mean', figsize=(15,3), delta=-
↪273.15)
```

```
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_ssp245_r1i1p1f2_gr_2065-2100.nc : x and y can be no greater than 2-D,␣
↪but have shapes (36,) and (36, 2, 3)
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_ssp585_r1i1p1f2_gr_2065-2100.nc : x and y can be no greater than 2-D,␣
↪but have shapes (36,) and (36, 2, 3)
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_ssp370_r1i1p1f2_gr_2065-2100.nc : x and y can be no greater than 2-D,␣
↪but have shapes (36,) and (36, 2, 3)
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_historical_r1i1p1f2_gr_2000-2014.nc : x and y can be no greater than␣
↪2-D, but have shapes (15,) and (15, 2, 3)
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_historical_r1i1p1f2_gr_1900-1949.nc : x and y can be no greater than␣
↪2-D, but have shapes (50,) and (50, 2, 3)
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_historical_r1i1p1f2_gr_1950-1999.nc : x and y can be no greater than␣
↪2-D, but have shapes (50,) and (50, 2, 3)
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_ssp245_r1i1p1f2_gr_2015-2064.nc : x and y can be no greater than 2-D,␣
↪but have shapes (50,) and (50, 2, 3)
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_historical_r1i1p1f2_gr_1850-1899.nc : x and y can be no greater than␣
↪2-D, but have shapes (50,) and (50, 2, 3)
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_ssp126_r1i1p1f2_gr_2065-2100.nc : x and y can be no greater than 2-D,␣
↪but have shapes (36,) and (36, 2, 3)
```

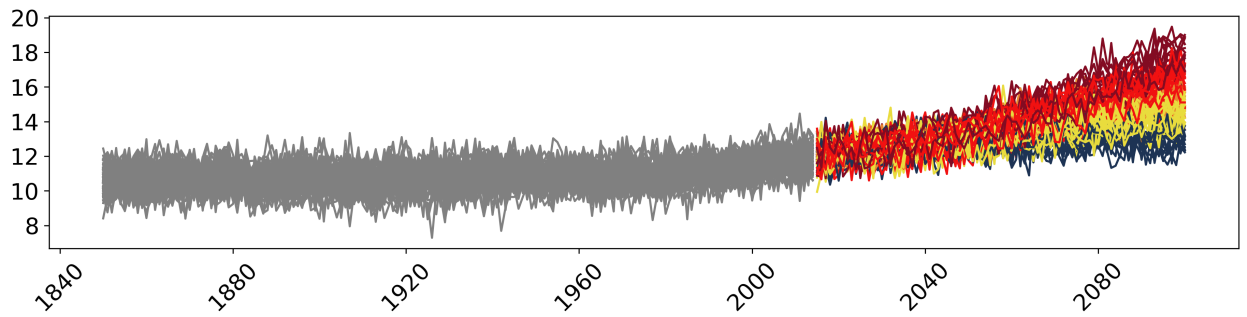
(continues on next page)

(continued from previous page)

```
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_ssp585_r1i1p1f2_gr_2015-2064.nc : x and y can be no greater than 2-D,␣
↪but have shapes (50,) and (50, 2, 3)
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_ssp370_r1i1p1f2_gr_2015-2064.nc : x and y can be no greater than 2-D,␣
↪but have shapes (50,) and (50, 2, 3)
spaghetti plot failed for /home/nils/ramboll/paris/data/indices/tg-mean/tg-mean_yr_
↪CNRM-CM6-1-HR_ssp126_r1i1p1f2_gr_2015-2064.nc : x and y can be no greater than 2-D,␣
↪but have shapes (50,) and (50, 2, 3)
timeseries spaghetti plot done for tg_mean with 166 lines.
```

[112]:
```
Image(plot)
```

[112]:

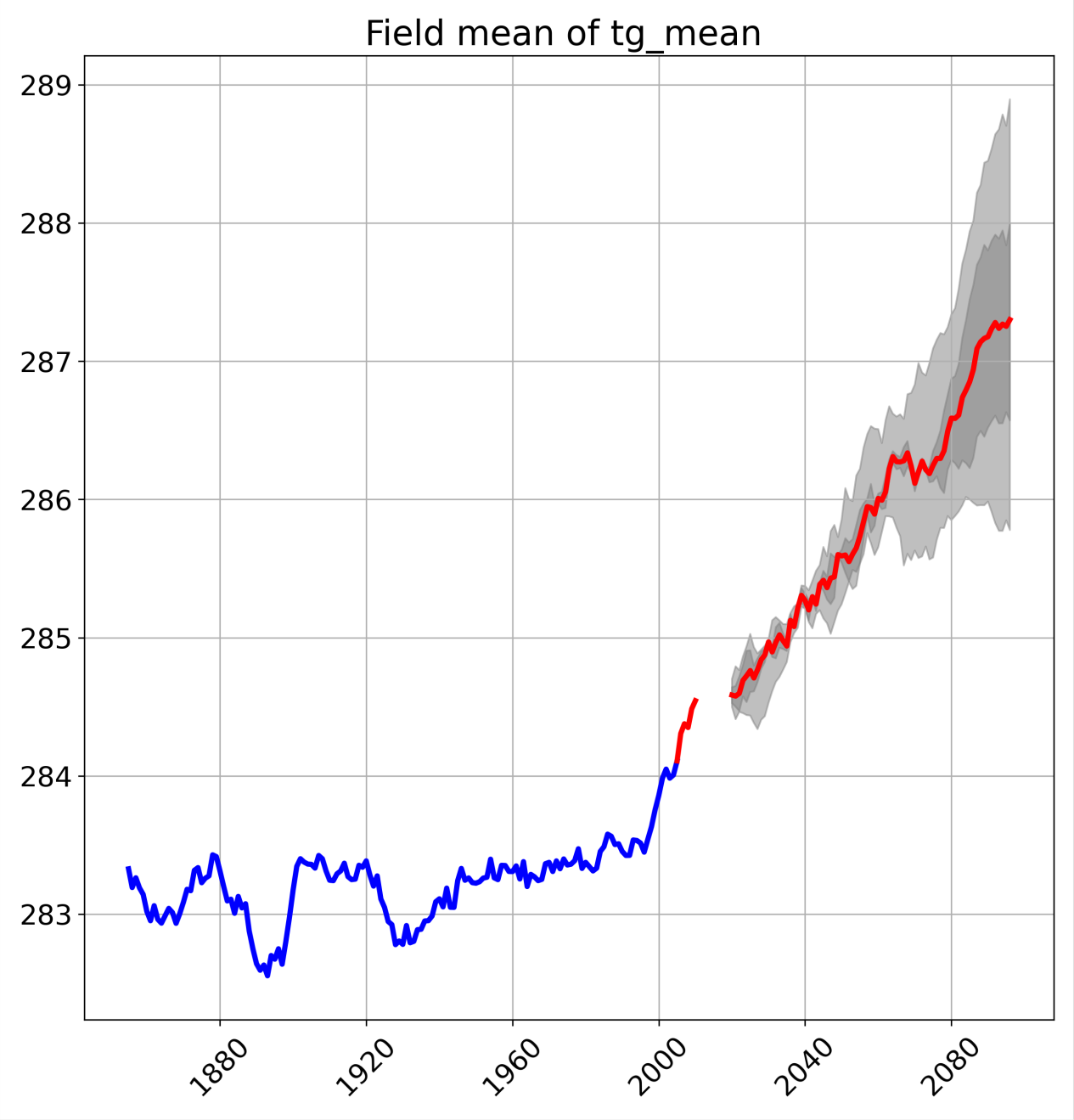


[76]:
```
files = [join('/home/nils/ramboll/paris/data/indices/tg-mean',f) for f in listdir(
↪'data/indices/tg-mean')]
```

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

[78]:
```

```

[78]:



[14]:
```
#     if len(resource) > 1:
#         LOGGER.debug('sort_by_filename module start sorting %s files' %␣
↪len(resource))
#         # LOGGER.debug('resource is list with %s files' % len(resource))
#         try:  # if len(resource) > 1:
#             # collect the different experiment names
#


#             LOGGER.info('found %s datasets', len(nc_datasets.keys()))
#         except Exception:
```

(continues on next page)

(continued from previous page)

```
#             LOGGER.exception('failed to find names of datasets!')
#         LOGGER.info('check for historical/RCP datasets')
#
# try:
#             if historical_concatination is True:
#                 # select only necessary names
#                 rcp_datasets = nc_datasets.copy()
#                 if any("_rcp" in s for s in nc_datasets.keys()):
#                     for key in nc_datasets.keys():
#                         if 'historical' in key:
#                             rcp_datasets.pop(key)
#                     nc_datasets = rcp_datasets.copy()
#                     LOGGER.info('historical data set names removed from dictionary')
#                 else:
#                     LOGGER.info('no RCP dataset names found in dictionary')
#         except Exception:
#             LOGGER.exception('failed to pop historical data set names!')
#         LOGGER.info('start sorting the files')
#

# try:
#             for key in nc_datasets:
#                 try:
#                     if historical_concatination is False:
#                         for n in resource:
#                             if '%s_' % key in n:
#                                 nc_datasets[key].append(path.abspath(n))  # path.
↪join(p, n))

#                     elif historical_concatination is True:
#                         key_hist = key.replace('rcp26', 'historical').\
#                             replace('rcp45', 'historical').\
#                             replace('rcp65', 'historical').\
#                             replace('rcp85', 'historical')
#                         for n in resource:
#                             if '{}_'.format(key_hist) in n:
#                                 nc_datasets[key].append(path.abspath(n))
#                             if '{}_'.format(key) in n:
#                                 nc_datasets[key].append(path.abspath(n))  # path.
↪join(p, n))
#                     else:
#                         LOGGER.error('append file paths to dictionary for key %s␣
↪failed' % key)
#                     nc_datasets[key].sort()
#                 except Exception:
#                     LOGGER.exception('failed for %s ' % key)
#         except Exception:
#             LOGGER.exception('failed to populate the dictionary with appropriate␣
↪files')
#         for key in nc_datasets.keys():
#             try:
#                 nc_datasets[key].sort()
#                 start, _ = get_timerange(nc_datasets[key][0])  # get first timestep␣
↪of first file
#                 _, end = get_timerange(nc_datasets[key][-1])  # get last  timestep␣
↪of last file
#                 newkey = key + '_' + start + '-' + end
```

(continues on next page)

(continued from previous page)

```
#                 tmp_dic[newkey] = nc_datasets[key]
#             except Exception:
#                 msg = 'failed to sort the list of resources and add dates to␣
↪keyname: %s' % key
#                 LOGGER.exception(msg)
#                 tmp_dic[key] = nc_datasets[key]
#                 # raise Exception(msg)
#     elif len(resource) == 1:
#         p, f = path.split(path.abspath(resource[0]))
#         tmp_dic[f.replace('.nc', '')] = path.abspath(resource[0])
#         LOGGER.debug('only one file! Nothing to sort, resource is passed into␣
↪dictionary')
#     else:
#         LOGGER.debug('sort_by_filename module failed: resource is not 1 or >1')
#     LOGGER.info('sort_by_filename module done: %s datasets found' % len(nc_
↪datasets))
# except Exception:
#     msg = 'failed to sort files by filename'
#     LOGGER.exception(msg)
#     raise Exception(msg)
```

4.1.2 Climate data tutorials

Here are examples of basic climate data processing for sustainable development.

To make sure all required dependencies are installed run *conda env create* in the root folder of this repository, than *conda activate climdat*.

An running installation of mini-conda or Anacona is required.

4.2 Getting started with PYWPS

- PyWPS 4.0.0 Slides
- PyWPS Documentation

4.2.1 Birdhouse Workshop

Welcome to the **`Birdhouse`_** Workshop. This workshop is a hands-on session, which will guide you in creating a process for a **`Web Processing Service`_**. During the workshop you will learn how Birdhouse supports this development cycle.

> **Warning:** Under Construction . . .

Let's do it quick

Ready

1. *Conda*.

Steady

2. *Getting Started*

Go

Basics:

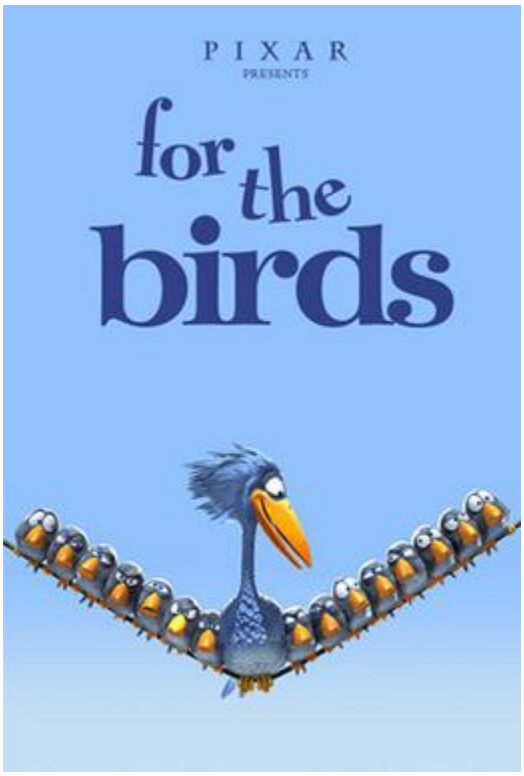
3. *Writing a simple Plot Function*
4. *Testing the Plot Function*
5. *Adding a Command-Line Interface*

PyWPS:

6. *Introduction*
7. *Installation*
8. *Processes*
9. *Clients*

Motivation

Let us see the Birds:



Requirements

Warning: You need a good internet connection! We are using GitHub, Conda and external data.

Warning: Make sure you have enough *disk space* available! Especially when you are using a Virtual Machine. You should have **at least 5 GB of free disk space** available.

The main requirement of this workshop is *Conda*. It works on most common platforms like Linux, macOS and Windows. If you don't have Conda yet ... we will install it at the beginning of this workshop.

Note: You will need a text-editor to edit Python code. Choose your favorite one ... if you don't have one yet, you can try Atom or Kate.

Tip: On Windows you can also use VirtualBox with a Linux Installation, like Linux Mint

Conda

`Conda`_ is an Open Source package and environment manager that can help to manage project dependencies. Conda works on Linux, macOS and Windows. It was created for Python programs, but it can package and distribute software for any language. Therefore it allows us to use it for multi-language projects.

Conda allows you to build your own packages and share them via channels on Anaconda Cloud. There is a community effort to build and maintain packages needed by various projects, called Conda Forge.

You can create conda environments with a specified list of packages, similar to Python virtualenv. These environemnts can be documented by a `environment.yml` configuration file and shared with others.

Warning: In this workshop we will install *all* software packages using `Conda`_.

Installation

Note: You don't need admin rights to install conda and conda packages.

Download and install the appropriate Miniconda installer from https://conda.io/miniconda.html

With Anaconda you can create environments that use any Python version (e.g. Python 2.7 or Python 3.6), so install the latest Python 3.x and if you find out later you need a Python 2.7 environment, you can create one.

Linux/macOS

You can *copy and paste* the following script to install Miniconda with default settings:

```
if [[ $(uname) == "Darwin" ]]; then
  url=https://repo.continuum.io/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
elif [[ $(uname) == "Linux" ]]; then
  url=https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
fi
curl $url -o miniconda.sh
bash miniconda.sh -b
export PATH=$HOME/miniconda3/bin:$PATH
```

We also recommend to add the following line to your `~/.bashrc` file to make Miniconda the Python found first than the system Python:

```
export PATH=$HOME/miniconda3/bin:$PATH
```

Windows

Run the installer, choose **Just Me** (not **All Users**), and choose a **Install Location** owned by you.

See also the Conda documentaion

Check your Python version

We are using Python 3.6:

```
$ which python
~/miniconda3/bin/python
$ python --version
Python 3.6.2 :: Continuum Analytics, Inc.
```

Links

- https://www.anaconda.com/blog/developer-blog/conda-data-science/
- https://docs.anaconda.com/docs_oss/conda/install/quick
- https://docs.anaconda.com/docs_oss/conda/test-drive
- https://conda.io/docs/user-guide/cheatsheet.html
- https://www.anaconda.com/blog/developer-blog/what-to-do-when-things-go-wrong-in-anaconda/

Getting Started

Clone the workshop repo from Github:

```
$ git clone https://github.com/bird-house/birdhouse-workshop.git
```

Note: In this workshop we assume that your workshop sources are in your home folder `~/birdhouse-workshop`. If the sources are located at a different place then you need to adapt the workshop root folder accordingly.

Create the *workshop* conda environment:

```
$ conda create -n workshop python=3
```

Activate the conda *workshop* environment (Linux and macOS):

```
$ source activate workshop
```

Warning: On *Windows* you use the following command:

```
$ activate workshop
```

I don't have git ...

Don't worry ... the quickest way to install git is using conda:

```
$ conda install git
```

If things go wrong ...

Well, this can happen ... you can easily get into troubles with resolving conda package dependencies. The easiest way to solve it is *tabula rasa* ... remove the conda environment and install it from new.

Deactivate the current environment (Linux and MacOS):

```
$ source deactivate
```

Warning: On *Windows* you need to use the following command to deactivate the environment:

```
$ deactivate
```

Remove the *workshop* conda environment:

```
$ conda env remove -n workshop
```

Create a new *workshop* environment with *all* dependencies used in this workshop by using a conda `environment.yml` file in the top level folder:

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

Basics

In the following sections we will write a Python function, which generates a plot from a netCDF file.

Writing a simple Plot Function

Prepare

See *Getting Started*.

Activate the Conda workshop enviroment:

```
$ source activate workshop
```

Aim

We are going to write a simple plot function in Python using *matplotlib* and *cartopy*.

Objectives:

- You will learn how to install packages with Conda.
- You will learn the basic usage of the *netCDF*, *matplotlib* and *cartopy* Python libraries.

Run the plotter

Go to the plotter tutorial source:

```
$ cd ~/birdhouse-workshop/tutorials/01_plotter
```

Try the plotter Python module:

```
$ python plotter.py
Traceback (most recent call last):
File "plotter.py", line 1, in <module>
  import matplotlib.pylab as plt
ModuleNotFoundError: No module named 'matplotlib'
```

Oops, something is missing ... please install the `matplotlib` package via Conda:

```
# Let's see what is available
$ conda search -c conda-forge matplotlib

# ... and install it from the conda-forge channel
$ conda install -c conda-forge matplotlib
```

Conda will show you a list of packages, which are going to be installed. Have a look at this list and answer with `y` or just press `enter`.

```
The following NEW packages will be INSTALLED:

matplotlib:       2.0.2-py36_2  conda-forge

Proceed ([y]/n)?
```

We should check now the `plotter.py` source code. Open the `plotter.py` in your favorite editor, some people like `vim`:

```
$ vim plotter.py
```

Besides `matplotlib` there is another `import` for `netCDF4`:

```
from netCDF4 import Dataset
```

Let us install `netcdf4`:

```
# same procedure as above ...
$ conda search -c conda-forge netcdf4
$ conda install -c conda-forge netcdf4
```

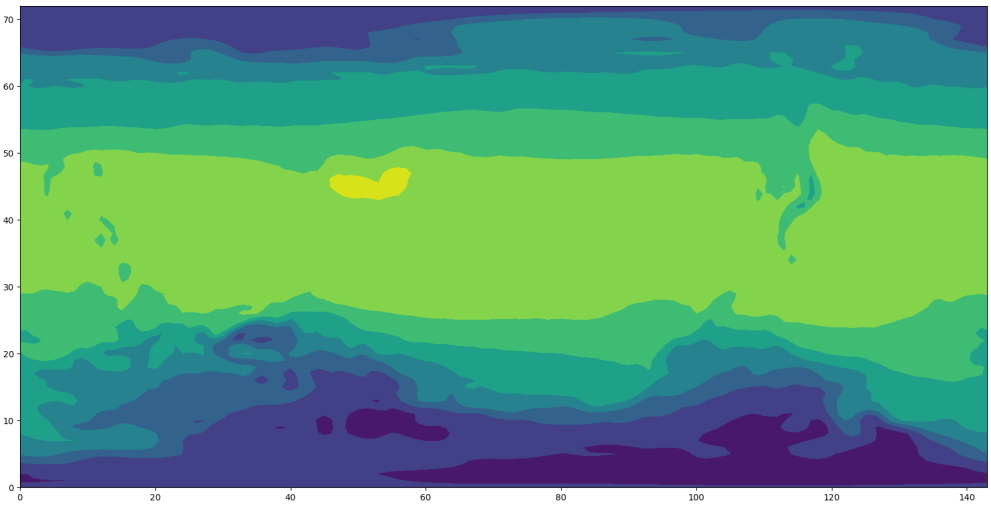
So, we should now be ready to run the plotter:

```
$ python plotter.py
Plotting ../data/air.mon.ltm.nc ...
Plot written to plot.png
```

A plot was generated. Open it in your favorite image viewer. On Ubuntu/LinuxMint you can try *Eye of Gnome*, on macOS just say `open`:

```
$ eog plot.png  # on Ubuntu use Eye of Gnome
or
$ open plot.png # on macOS
```

The image should look like the following:



Exercise

Open the `plotter.py` and implement the following features:

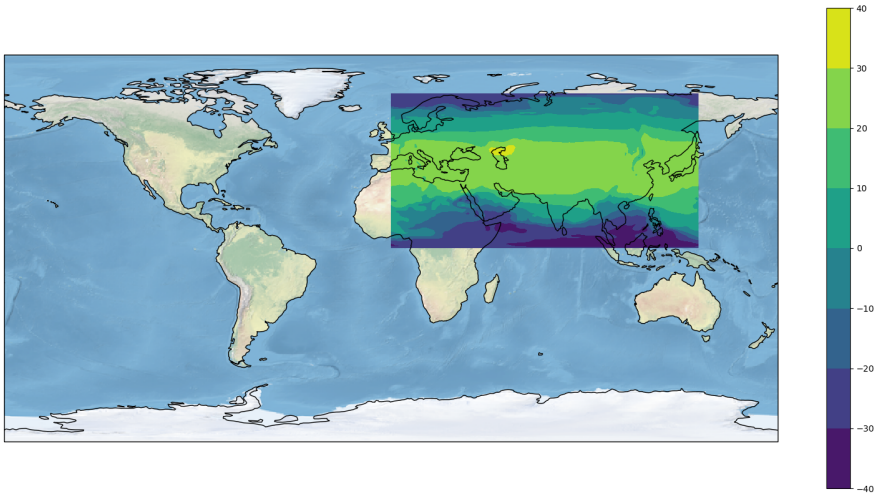
- add a colorbar
- add a background map with coastlines
- use a PlateCarree map projection

You will need an additional Python package, `cartopy`, which you can install with conda. This package is available on the conda-forge channel. You need to provide an option with the conda channel:

```
$ conda install -c conda-forge mypackage
```

Read the code and comments carefully to make this work.

The final result should look like this:



Links

- Matplotlib: http://matplotlib.org/
- Cartopy: http://scitools.org.uk/cartopy/
- Using cartopy with matplotlib
- GeoScience Notebook
- PyEarthScience: matplotlib examples

Testing the Plot Function

Prepare

See *Getting Started*.

Activate the Conda workshop enviroment:

```
$ source activate workshop
```

Aim

We are going to write a unit test for our Python plot function.

Objectives:

- You will learn how to write a unit test with *pytest*.

Run pytest

Go to the plotter tutorial source:

```
$ cd ~/birdhouse-workshop/tutorials/02_testing_plotter
```

Run the `plotter.py` like in the previous tutorial and see if it works:

```
$ python plotter.py
```

Now, we want to implement a unit test for our plot function. We are using pytest as testing framework. Install it via conda:

```
$ conda install -c conda-forge pytest
```

Run now *pytest* on our plotting module:

```
$ pytest plotter.py
E       NotImplementedError: This test is not implemented yet. Help wanted!
```

Oops . . . the test is not working yet.

Exercise

Your task is to implement a meaningful test for our `simple_plot` function.

Start hacking `plotter.py` in your favorite editor and run `pytest` frequently.

> **Warning:** Read the comments carefully to make this work and **do not trust each line of code**.

Links

- pytest: https://docs.pytest.org/en/latest/

Adding a Command-Line Interface

Prepare

See *Getting Started*.

Activate the Conda workshop enviroment:

```
$ source activate workshop
```

Aim

We are going to write a command line interface (CLI) for our Python plot function.

Objectives:

- You will learn how to write a CLI with the Python library argparse.

Run the plotter CLI

Go to the plotter tutorial source:

```
$ cd ~/birdhouse-workshop/tutorials/03_plotter_cli
```

See the command line options of our plotter:

```
$ python plotter.py -h
usage: plotter.py [-h] [-V [VARIABLE]] dataset
```

Plot our well-know image:

```
$ python plotter.py --variable air ../../data/air.mon.ltm.nc
```

Exercise 1

Play a little bit with the command-line options. Try some other options (`-V`), use invalid input (water) and skip some arguments.

Exercise 2

Use external data from a Thredds service, like NOAA:

https://www.esrl.noaa.gov/psd/thredds/catalog/Datasets/ncep.reanalysis.derived/surface/catalog.html

See access methods for a dataset, note *OpenDAP*:

https://www.esrl.noaa.gov/psd/thredds/catalog/Datasets/ncep.reanalysis.derived/surface/catalog.html?dataset=Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc

Access:

1. **OPENDAP:** /psd/thredds/dodsC/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc
2. **HTTPServer:** /psd/thredds/fileServer/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc
3. **WCS:** /psd/thredds/wcs/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc
4. **WMS:** /psd/thredds/wms/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc

Use *OpenDAP* URLs directly as dataset input:

http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc

Data URL: http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets/ncep.reanalysis.derived/surface/air.

```
$ python plotter.py --variable air \
      http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets/ncep.reanalysis.derived/
↪surface/air.mon.ltm.nc
```

Exercise 3

Extend the command line and with an optional parameter for the `timestep` to be plotted.

Open your editor on `plotter.py` ... and happy hacking.

Don't forget to test often:

```
$ pytest plotter.py
```

Exercise 4

The output name of the plot is always `plot.png`. Add an optional `output` parameter to set an output filename.

Links

- Python argparse
- NOAA Thredds Data Service

PyWPS

In the following sections we will introduce PyWPS and write a WPS process for a simple plot function.

Introduction

In the following we describe WPS in general and PyWPS.

What is WPS?

Web Processing Service (WPS) is part of the OWS standards defined by OGC, while WFS, WMS, WCS, SOS are used for transfer of data (upload, download, transformation?), WPS is used for data processing on the server (All processing is done on server side).

WPS provides a standard interface for input, output, process discovery and execution. WPS is normally used for geospatial data to run spatial processes.

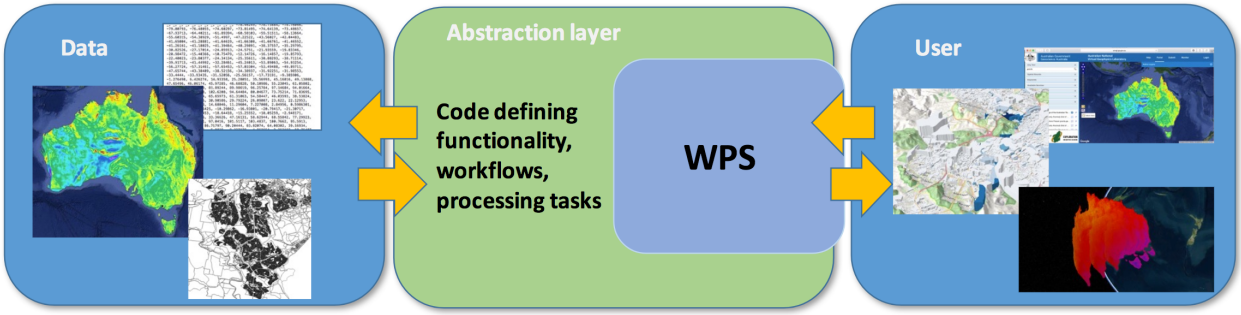


Fig. 1: Taken from: http://pointclouds.nci.org.au/talks/f4g_pointwps_adamsteer.pdf

What is PyWPS?

PyWPS is a WPS implementation written in the Python language. The current version is 4.0.0.

- Server (HTTP-WSGI)
- OGC WPS implementation
- Python 3 support
- Based on a webframework (werkzeug)
- Native WSGI itegration (Better server integration)
- MIT license (can be used in comercial projects)

PyWPS has been used in multiple projects concerning geospatial data processing:

- List of scientifical publications
- PyWPS gallery

A brief introdution to WPS

WPS is part of the OGC service suit (OWS) and some operations are common to other services (e.g. **GetCapabilities**), but others specific to WPS itself.

WPS requests:

- **GetCapabilities**
- **DescribeProcess**
- **Execute**

GetCapabilities this request provides a list of available services.

DescribeProcess describes a process indicating the inputs and outputs required by the process to execute and/or for metadata information.

Execute this request will accept inputs/outputs, processing conditions (async/sync) and will run the process on the server.

WPS async/sync

Some processes are time consuming, so it is better to start the process and later query the server for its status or output. This is refered as a async execute request

If you are confident that the process being executed is fast you can request a sync execution where the client waits for the immeditely reply from server withoutput (no need to pull the output later).

WPS input/output

WPS has 3 sorts of data I/O:

- **Literal**
- **ComplexData**
- **BoundingBox**

The **Literal** is any number (float, int) and string. **ComplexData** is geospatial data in multiple formats (mimetype, e.g: `application/gml+xml`) that can be integrated into the WPS request/response, when using vectorial data this one is transformed into XML and raster binary data coded into base64 (binary coding using ascii symbols).

WPS versions

WPS 1.0.0 was released in 2007, the new WPS 2.0.0 was released in 2015. So far major implementations have only used WPS 1.0.0.

WPS 1.0.0 can start processes, but there is no way to stop them before the process reaches its conclusion... it is like a car without brakes. But not all is bad. New WPS 2.0.0 allows for processes to be cancelled.

Links

- PyWPS
- PyWPS Workshop

Installation

Requirements

See *Getting Started*.

Activate the conda workshop enviroment:

```
$ source activate workshop
```

Aim

We are going to install *PyWPS* and run some example processes.

Objectives:

- You will learn how to install PyWPS, start a WPS service and execute a process.

Install PyWPS

You can install PyWPS via conda. Make sure you install PyWPS from the *birdhouse* conda channel. We also need the *conda-forge* channel, and the channels must be provided in the displayed order (channel priority):

```
$ conda install -c birdhouse -c conda-forge pywps gdal
```

Let's see if this has worked:

```
$ python -c "import pywps"
```

This bash command will load the pywps library and close the console. If the install was properly done *no error messages* will appear.

Start the demo WPS service

This workshop includes a demo service with some example processes. Let's try them.

Start the service by running the following command:

```
# change to workshop root folder
$ cd ~/birdhouse-workshop/
# start demo service
$ python demo/demo.py
```

If everything went well you should have a console output as follows:

```
Configuration file(s) ['demo/default.cfg'] loaded
starting WPS service on http://localhost:5000/wps
  * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Warning: If you need to start the service on a different port then 5000, you must edit the port in the PyWPS configuration `demo/default.cfg`:

```
[server]
url = http://localhost:5001/wps
outputurl = http://localhost:5001/outputs
```

Service check

To test the service, open your internet browser to this address: http://127.0.0.1:5000/wps.

Alternatively, you can also try `curl`:

```
$ curl "http://127.0.0.1:5000/wps"
```

You will get an XML exception report by the PyWPS service:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- PyWPS 4.0.0 -->
<ows:ExceptionReport xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xsi="http://www.
↪w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/ows/1.1␣
↪http://schemas.opengis.net/ows/1.1.0/owsExceptionReport.xsd" version="1.0.0">
  <ows:Exception exceptionCode="MissingParameterValue" locator="service" >
    <ows:ExceptionText>service</ows:ExceptionText>
  </ows:Exception>
</ows:ExceptionReport>
```

The good thing . . . the service is running and talking to you :)

Test PyWPS

Test the WPS service itself using a **GetCapabilities** request; insert this address in your browser:

http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities

```
$ curl "http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities"
```

In the **GetCapabilities** XML document notice the following:

- Abstract describing service
- Service provider
- Process Offerings (Title, Abstract, Metadata)

Say hello

We can run now our first process. The **GetCapabilities** XML document tells us that this WPS serivce has a process with identifier `say_hello`. Please find this description in the document. It should look like this:

```
 <wps:Process wps:processVersion="1.3.2">
   <ows:Identifier>say_hello</ows:Identifier>
   <ows:Title>Process Say Hello</ows:Title>
 </wps:Process>
```

Now, we need some more details about this process. Therefore we do a **DescribeProcess** request; insert this address in your browser:

http://127.0.0.1:5000/wps?service=WPS&request=DescribeProcess&version=1.0.0&identifier=say_hello

```
$ curl "http://127.0.0.1:5000/wps?service=WPS&request=DescribeProcess&version=1.0.0&
↪identifier=say_hello"
```

The resulting XML document tells us something about the *input* and *output* parameters, for example there is an input parameter `name`:

```
   <Input minOccurs="1" maxOccurs="1">
     <ows:Identifier>name</ows:Identifier>
     <ows:Title>Input name</ows:Title>
     <LiteralData>
       <ows:DataType ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:
↪DataType>
       <ows:AnyValue/>
     </LiteralData>
   </Input>
```

Let us now execute the `say_hello` process with an input parameter `name` *Birdy*:

http://127.0.0.1:5000/wps?service=WPS&request=Execute&version=1.0.0&identifier=say_hello&DataInputs=name=Birdy

```
$ curl "http://127.0.0.1:5000/wps?service=WPS&request=Execute&version=1.0.0&
↪identifier=say_hello&DataInputs=name=Birdy"
```

If all went well, you get an output parameter with the value *Hello Birdy*:

```
 <wps:ProcessOutputs>
   <wps:Output>
     <ows:Identifier>response</ows:Identifier>
     <ows:Title>Output response</ows:Title>
     <wps:Data>
       <wps:LiteralData dataType="urn:ogc:def:dataType:OGC:1.1:string" uom="urn:ogc:
↪def:uom:OGC:1.0:unity">Hello Birdy</wps:LiteralData>
     </wps:Data>
   </wps:Output>
 </wps:ProcessOutputs>
```

Exercise 1

Try the `say_hello` again with some other input values.

Exercise 2

Before you fall into *sleep* . . . let's do another exercise. Our service has another process. Which one is it?

Please find it and run an execute request . . . you need to know the input parameters.

Links

- PyWPS Workshop
- PyWPS Flask Demo
- Geoprocessing Info

Processes

Requirements

See *Getting Started*.

Activate the conda workshop enviroment:

```
$ source activate workshop
```

Aim

We are going to write a PyWPS process.

Objectives:

- You will learn how to write a PyWPS process.

What is a WPS Process?

In PyWPS a process is a Python class that has the following structure:

- The parent `Process` class.
- Four input/ouput classes: `ComplexInput`, `LiteralInput`, `ComplexOutput` and `LiteralOutput`
- The `_handler(request, response)` method
- The `request.inputs` and the `response.output` properties.

Go through the PyWPS documentation on Processes.

Create your first process

Let's create a new process that generates a nice and simple plot from a NetCDF file. We have written a `simple_plot` function, which we can use here. We need to do the following:

1. write the PyWPS process definition,
2. call our `simple_plot` method,
3. activate our process in PyWPS.

Check the plotter function

Change into the tutorial `processes` folder:

```
$ cd ~/birdhouse-workshop/tutorials/10_pywps_process/processes
```

You can find here the `plotter.py` module from our previous exercise:

```
$ ls
plotter.py
```

Let's see if it still works:

```
$ python plotter.py -h
```

Generate a plot:

```
$ python plotter.py ../../../data/air.mon.ltm.nc -V air
dataset=['../../../data/air.mon.ltm.nc'], variable=air
Plotting ../../../data/air.mon.ltm.nc ...
Using map projection <cartopy.crs.PlateCarree object at 0x7fae109538e0>
Plot written to plot.png
Output: plot.png
```

Write the process definition

In the `processes/` folder there is another file:

```
$ ls
wps_simple_plot.py
```

This file contains the process definition. Notice the input and output parameters.

Start the service

Change into the tutorials folder:

```
$ cd ~/birdhouse-workshop/tutorials/10_pywps_process
```

Start the WPS service:

```
$ python ../../demo/demo.py
```

Check if the service is running:

http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities

```
$ curl "http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities"
```

Notice that the `simple_plot` service is not activated. Well, time to exercise ...

Exercise 1

Activate the `SimplePlot` process from the `wps_simple_plot` module. See if it shows up in the **GetCapabilites** request.

Tip: You need to edit `processes/__init__.py` and restart the demo service.

Exercise 2

When the `SimplePlot` process is activated then run a **DescribeProcess** request.

Tip: Find the process `identifier` of `SimplePlot` in the **GetCapabilities** document and adapt the **DescribeProcess** URL from our previous exercise.

Exercise 3

Run an **Execute** request with a remote netCDF file from a Thredds data server.

Use the following request URL.

```
http://127.0.0.1:5000/wps?
    Service=WPS&
    Request=Execute&
    Version=1.0.0&
    Identifier=PLOT_IDENTIFIER&
    DataInputs=variable=air;dataset=@xlink:href=NC_URL
```

Or as a one-liner:

http://127.0.0.1:5000/wps?Service=WPS&Request=Execute&Version=1.0.0&Identifier=PLOT_IDENTIFIER&DataInputs=variable=air;dataset=@xlink:href=NC_URL

You need to replace **PLOT_IDENTIFIER** with the correct processes identifier. Replace **NC_URL** with a remote netCDF data file (HTTP, not OpenDAP), for example:

> https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.derived/surface/air.mon.ltm.nc

Notice that the output will be returned as reference, for example:

```
<wps:ProcessOutputs>
  <wps:Output>
    <ows:Identifier>output</ows:Identifier>
    <ows:Title>Simple Plot</ows:Title>
    <wps:Reference xlink:href="http://localhost:5000/outputs/4d075e9a-acf4-11e7-9396-
↪acde48001122/plot_ex33_nbf.png" mimeType="image/png"/>
  </wps:Output>
</wps:ProcessOutputs>
```

Exercise 4

You can also run the process in asynchronous mode by adding the parameters `storeExecuteResponse=true` and `status=true`.

```
http://127.0.0.1:5000/wps?
    Service=WPS&
    Request=Execute&
    Version=1.0.0&
    Identifier=PLOT_IDENTIFIER&
    DataInputs=variable=air;dataset=@xlink:href=NC_URL&
```

(continues on next page)

(continued from previous page)

```
    storeExecuteResponse=true&
    status=true
```

Warning: Asynchronous requests do not work on Windows.

In this case you will a response, which tells you that the process has been accepted, and you need to poll the status document given by the **statusLocation** URL:

```
<wps:ExecuteResponse
  service="WPS" version="1.0.0" xml:lang="en-US"
  serviceInstance="http://localhost:5000/wps?service=WPS&amp;request=GetCapabilities"
  statusLocation="http://localhost:5000/outputs/c894c1b4-acf7-11e7-b989-acde48001122.
↪xml">
  <wps:Process wps:processVersion="1.0">
    <ows:Identifier>simple_plot</ows:Identifier>
    <ows:Title>Simple Plot</ows:Title>
    <ows:Abstract>Returns a nice and simple plot.</ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2017-10-09T15:43:10Z">
    <wps:ProcessAccepted>PyWPS Process simple_plot accepted</wps:ProcessAccepted>
  </wps:Status>
</wps:ExecuteResponse>
```

Exercise 5

You can also return the output directly. For this modify the above request and add the `RawDataOutput` parameter:

```
http://127.0.0.1:5000/wps?
    Service=WPS&
    Request=Execute&
    Version=1.0.0&
    Identifier=PLOT_IDENTIFIER&
    DataInputs=variable=air;dataset=@xlink:href=NC_URL&
    RawDataOutput=output
```

Warning: Due to a bug in PyWPS it works currently only with Python 2.7.

Links

- PyWPS workshop
- Geoprocessing Info
- NOAA Thredds Catalog
- Notebook with WPS requests

Testing

Requirements

See *Getting Started*.

Activate the conda workshop enviroment:

```
$ source activate workshop
```

Aim

As you develop more complex process and use more structured datasets, using simply a web browser to test becomes impractical. In this chapter you get acquainted with alternative tools to interact with a PyWPS instance.

Objectives:

- You will learn how to test a PyWPS process.

wget

Start by trying the *GetCapabilities* request:

```
$ wget -q -O caps.xml \
  "http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities"
```

Important question: Why `-q`, `-O` and `"` in the comnand:

`-q` quit verbose information about requests.

`-O` Output to file. You can use `-`, and the content will be dumped into the prompt.

`"` Otherwise wget would not consider `&` as part of the URL and would cut it.

curl

Similar to *wget* you can also use *curl* to retrieve the *GetCapabilities* XML document:

```
$ curl -s -o caps.xml \
  "http://127.0.0.1:5000/wps?service=WPS&request=GetCapabilities"
```

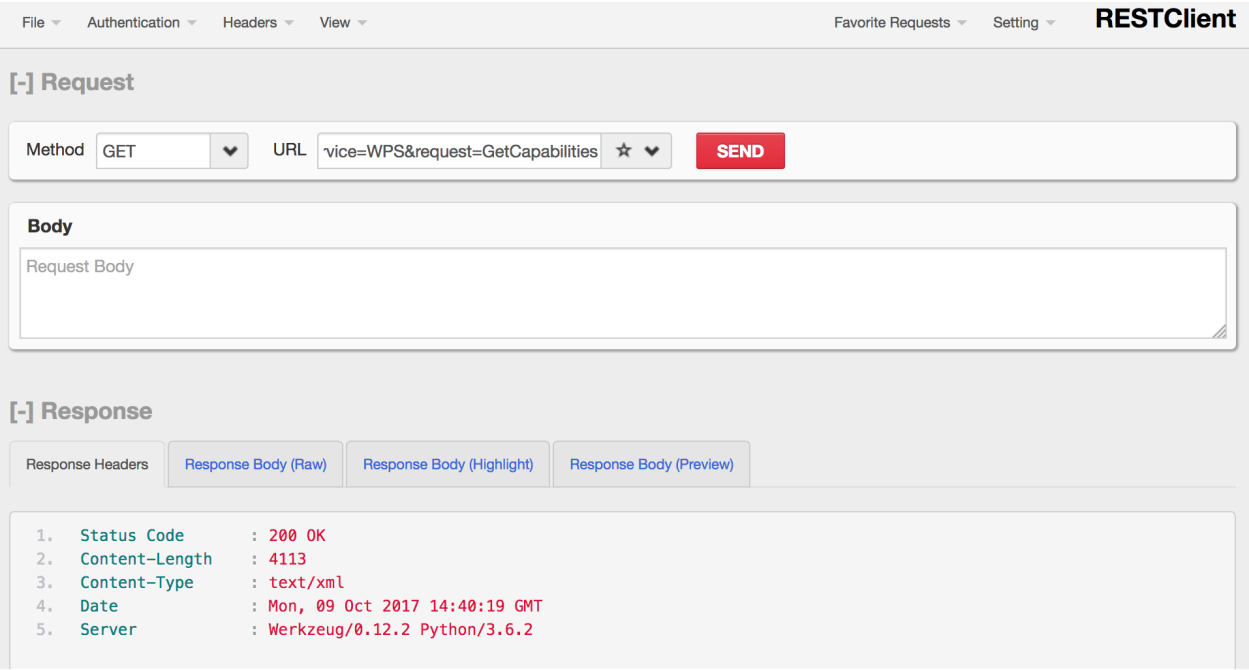
`-s` silent mode . . . no progress bar.

`-o` Output to file. You can use `-`, and the content will be dumped into the prompt.

RESTClient (Firefox only)

You can use the RestClient Firefox plugin to run requests.

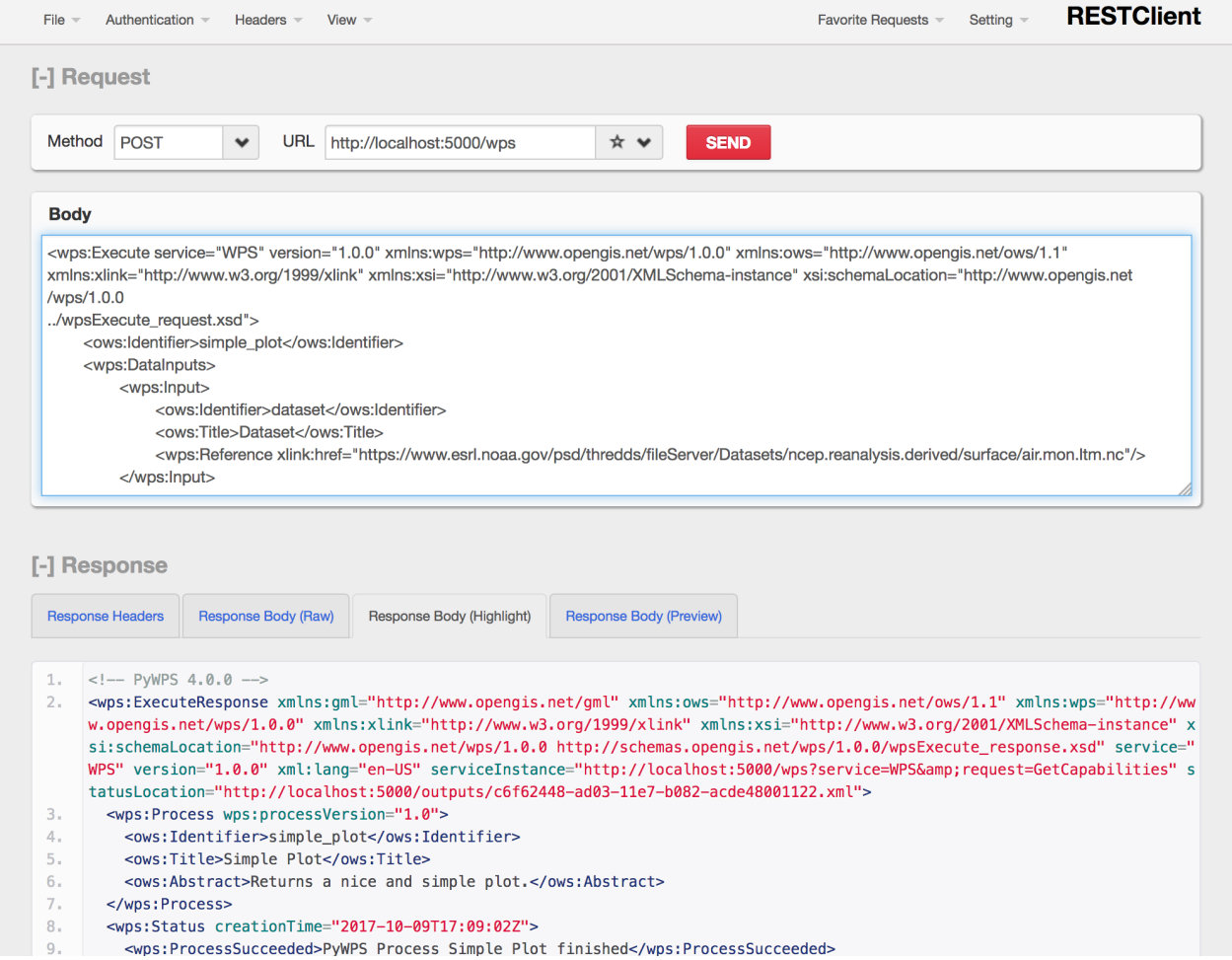
Here is an example with a **GetCapabilities** request using HTTP method **GET**:



XML HTTP Post Request

As requests and data become more structure and lengthy, concatenating all parameters into a URL for a GET type request becomes difficult or impossible. For this reason the WPS standard allows the definition of requests as XML documents sent to the server using the POST method of the HTTP protocol.

Here is an example with an **Execute** request using HTTP method **POST**:



It is using the XML description of the **Execute** request.

It is also possible to use curl (or wget) for POST requests:

```
$ curl -H "Content-Type: text/xml" -X POST \
  -d@execute_req.xml http://localhost:5000/wps
```

-d@ pass data from the given filename (XML payload)

-X HTTP method, GET or POST

-H Header variable, in our case we set the Content-Type.

Exceptions

ExceptionReport is an important feature of WPS. In WPS 1.0.0 we have the following exceptions:

MissingParameterValue The request does not include a parameter value or a default cannot be found.

InvalidParameterValue The request contains an invalid parameter value.

NoApplicableCode Generic exception, no other code could be applied.

NotEnoughStorage The server does not have enough space available.

Try the following request:

http://127.0.0.1:5000/wps?service=WPS&request=DescribeProcess

```
$ curl "http://127.0.0.1:5000/wps?service=WPS&request=DescribeProcess"
```

The exception is *MissingParameterValue*:

```
<?xml version="1.0" encoding="UTF-8"?>
<ows:ExceptionReport xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xsi="http://www.
↪w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/ows/1.1␣
↪http://schemas.opengis.net/ows/1.1.0/owsExceptionReport.xsd" version="1.0.0">
<ows:Exception exceptionCode="MissingParameterValue" locator="version" >
  <ows:ExceptionText>Missing version</ows:ExceptionText>
</ows:Exception>
</ows:ExceptionReport>
```

The *version* parameter is missing.

In case of Python errors in the called process, PyWPS will dump the Python stack into the *ExceptionReport*.

Exercise 1

Try `wget` or `curl` with some of the previous *DescribeProcess* and *Execute* requests.

Exercise 2

Run the **POST** request using the prepared XML payload.

Change into the tutorial `processes` folder:

```
$ cd ~/birdhouse-workshop/tutorials/11_pywps_testing
```

Make sure no WPS service is running ... stop it with `CTRL-C`.

Start the demo service:

```
$ python ../../demo/demo.py
```

Use the above `curl` command with the payload `execute_req.xml`, which you can find in this folder. Modify the input parameters of the payload.

Note: There is another POST request example in the point-clouds talk by NCI.

Links

- RestClient
- Poster on Chrome
- PyWPS workshop
- Geoprocessing Info
- WPS Tutorial

Logging

Requirements

See *Getting Started*.

Activate the conda workshop enviroment:

```
$ source activate workshop
```

Aim

Take a look at the Logging section in the configuration file. PyWPS currently logs events to two different locations:

- A log file where messages are stored. The kind of messages is set in the configuration file.
- A database where each request to the service is registered.

PyWPS uses SQLAlchemy to connect and work with multiple database management systems. SQLite and PostgreSQL tend to be the most used options.

Objectives:

- You will learn how to configure and check the logs.

Check the logs

Our demo WPS service is configured to log to the `pywps.log` file. Using the `tail`, `less` or `cat` commands search for error messages in the `pywps.log` file.

Tip: These messages are preceded by the string "[ERROR]", it is possible to grep the error messages:

```
cat pywps.log | grep "\[ERROR\]"
```

Continuous monitoring

Use the tail command to continuously monitor the activity of the service:

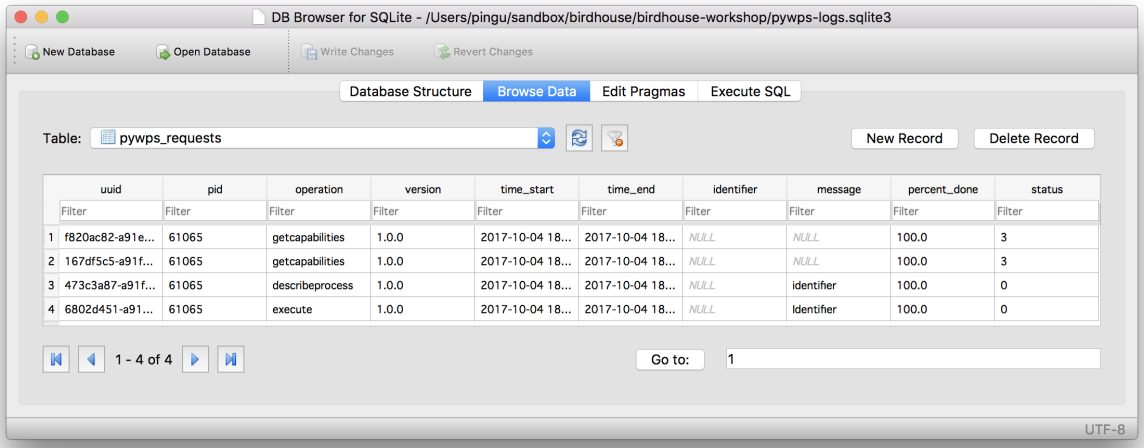
```
$ tail -f pywps.log
```

Database browser

If you have no database browsing programme installed, Install DB Browser for SQLite on your system. On Debian based systems it can be installed from the command line:

```
$ sudo apt install sqlitebrowser
```

See a screenshot with an open SQLite database file `pywps-logs.sqlite3`:



You can also use SQLite on the command-line:

```
$ sqlite3 pywps-logs.sqlite3
sqlite> select * from pywps_requests;
sqlite> .quit
```

Configure logging

Change into the tutorials folder `pywps_logging`:

```
$ cd ~/birdhouse-workshop/tutorials/12_pywps_logging
```

It contains a `pywps.cfg` file with a logging section. You can overwrite the default PyWPS configration by starting the PyWPS service with another config:

```
$ python ../../demo/demo.py -c pywps.cfg
loading configuration
Configuration file(s) ['../../demo/default.cfg', 'pywps.cfg'] loaded
```

Exercise 1

Edit the `pywps.cfg` ... use `DEBUG` logging level. Start the demo WPS service and monitor the log file.

Run a few processes.

Exercse 2

Start the demo WPS service with processes form the previous tutorial. Add some `logging` statements and monitor the service.

Links

- PyWPS workshop

Clients

Requirements

See *Getting Started*.

Activate the conda workshop enviroment:

```
$ source activate workshop
```

Aim

We are going to use a WPS client.

Objectives:

- You will learn how to use the Birdy WPS clients.
- You can try an online demo with the Phoenix Web UI.

Birdy

Birdy is a command-line client for Web Processing Services. It is using the Python library *OWSLib* to interact with WPS services.

Install it via conda:

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

Start the demo WPS service:

```
# go to the workshop root folder
$ cd ~/birdhouse-workshop/
# start wps service
$ python demo/demo.py
```

Warning: The WPS service is running in *foreground*. You need to open a new terminal and activate the conda *workshop* environment for the *birdy* WPS client.

Tip: You can also start the WPS service in *background*:

```
$ python demo/demo.py -d
```

Remember the process id to kill the service:

```
forked process id: 16483
```

Let birdy know the WPS service URL:

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

Note: On Windows you can use:

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

See which processes are available:

```
$ birdy -h
usage: birdy [<options>] <command> [<args>]
```

Show the description of `say_hello`:

```
$ birdy say_hello -h
usage: birdy say_hello [-h] --name [NAME]
                       [--output [{response} [{response} ...]]]
```

Run `say_hello`:

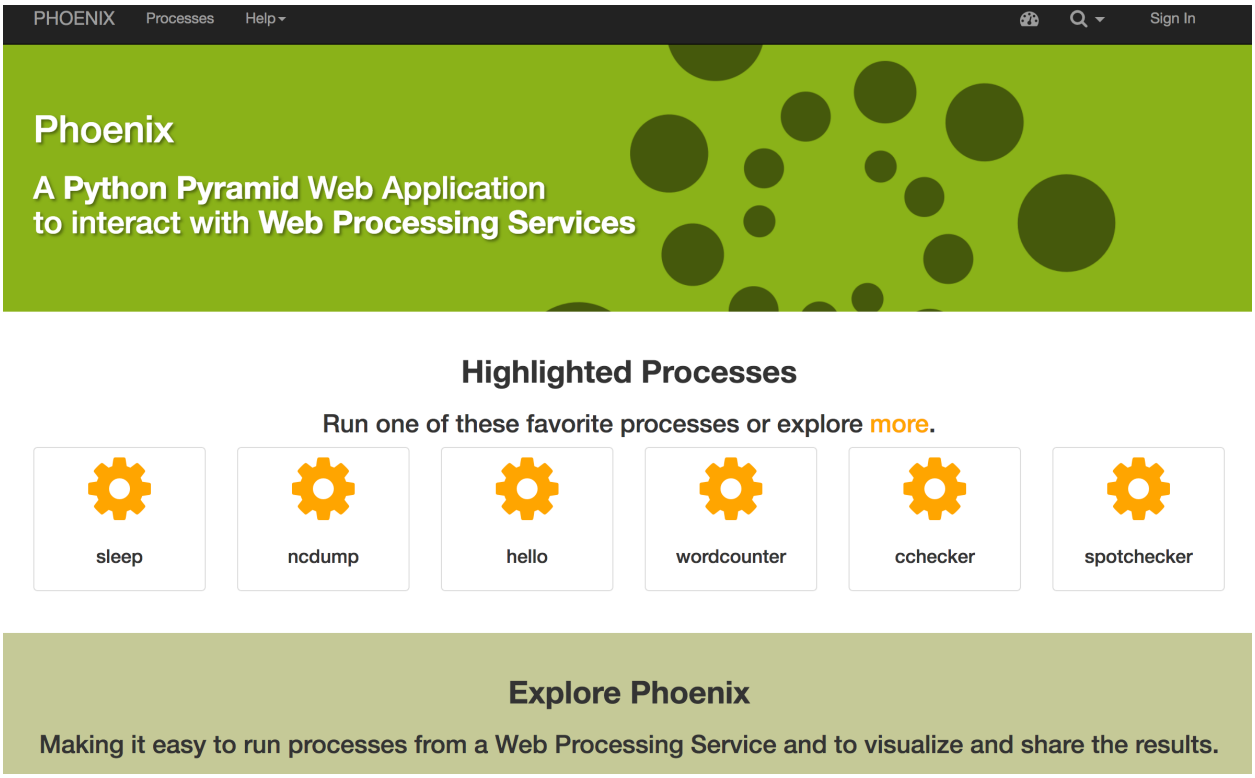
```
$ birdy say_hello --name Birdy
[ProcessAccepted 0/100] PyWPS Process say_hello accepted
[ProcessSucceeded 0/100] PyWPS Process Process Say Hello finished
Output:
response=Hello Birdy
```

Warning: On Windows the asynchronous call does not work, which birdy uses by default. Use the `--sync` option instead:

```
$ birdy --sync say_hello --name Birdy
```

Phoenix

Phoenix is Pyramid web-application to interact with WPS services.



You can try the online demo.

Exercise 1

Play with birdy ... run the `sleep` process.

Exercise 2

Start the demo service with the processes from *Testing* and call the `simple_plot` process with **birdy**.

Exercise 3

Try some of the processes on the Phoenix demo. For example the **wordcounter** and the **spotchecker**.

Links

- Birdy
- Phoenix
- Pyramid
- OWSLib

The Birds

In the following sections we will introduce the Birds.

Introduction

`Birdhouse`_ is a collection of **`Web Processing Service`_** (WPS) related Python components to support climate data analysis. Birdhouse uses OGC/WPS software from the GeoPython project, like PyWPS and OWSLib.

The aim of Birdhouse is to support (climate science) projects to setup a Web Processing Service infrastructure.

Birdhouse is the *Home* of several *Birds*, the components of the Birdhouse ecosystem. There are birds for the Web Processing client side, to make the WPS service access more convenient and also as an example for project own Web UIs. There are fully configured WPS services with example processes, which run *out-of-the-box* and can be forked and used as template. There is also a middleware component to control access to WPS services.

The Birdhouse documentation gives an overview of the architecture.

The Birdhouse components can be installed with a simple `make install`. See the installation documentation for details.

All Birdhouse components are Open Source and released under the Apache License. The source code is available on GitHub.

See the documentation of the Birdhouse components and try the demo.

Live Demo

Phoenix

Showing Phoenix with CDO, Spotchecker and Subsetting process with ESGF, OpenDAP and uploaded data.

Birdy

Showing Birdy with ncdump on ESGF data (using access token).

curl

Using `curl` to run the `wordcounter` on an external service.

Advanced

In the following sections we will go into advanced topics.

OWSLib

OWSLib is a Python library for client programming with Open Geospatial Consortium (OGC) web service (hence OWS), like WMS, CSW and WPS.

We are using an Jupyter notebook to look at some example code.

Todo: Add ssh tunnel or jupyter console example.

We need to install Jupyter via conda:

```
$ conda install -c conda-forge jupyter
```

Go to the tutorials folder `pywps_clients`:

```
$ cd ~/birdhouse-workshop/tutorials/31_owslib
```

You will find there an Jupyter notebook:

```
$ ls
owslib-wps.ipynb
```

Open the Jupyter notebook:

```
$ jupyter notebook
```

And point your browser to the following URL:

http://localhost:8888/notebooks/owslib-wps.ipynb

Or see it on GitHub.

Links

- https://try.jupyter.org/
- https://nbviewer.jupyter.org/

ESGF

Using ESGF pyclient to access ESGF data

Example Notebook: https://github.com/cehbrecht/demo-notebooks

Use birdy command-line with ESGF data

Example: http://birdy.readthedocs.io/en/latest/tutorial.html

Use Phoenix Wizard with ESGF data

Example: http://pyramid-phoenix.readthedocs.io/en/latest/tutorial/visualisation.html

Docker

The Birdhouse WPS services are available as a Docker image on Docker Hub.

What is Docker?

https://www.docker.com/what-docker

Run Emu as Docker container

Example: http://emu.readthedocs.io/en/latest/tutorial/using_docker.html

Links

- Docker Training: http://slides.com/dataduke/docker-001#/

Travis CI

Todo: add travis example

Continuous Integration with Travis . . . triggered by commit on GitHub and via cron job.

See Emu example:

https://travis-ci.org/bird-house/emu

Travis config:

https://github.com/bird-house/emu/blob/master/.travis.yml

WPS tests:

https://github.com/bird-house/emu/blob/master/emu/tests/test_wps_hello.py

Appendix

Why using WPS?

- Web based services could help researchers collaborate
  - The fact that individual researchers are increasingly specialized raises the "cost" of interacting with other disciplines.
  - Due to these costs, multidisciplinary projects are often run in parallel, with no real dependencies and synergies between teams.
  - Open source code has helped tremendously, but there is a large resource gap between installing software and having a working application, especially in earth system modeling.
- Why would individual scientists publish services?
  - Increased visibility and funding opportunities.
  - Improved research quality.
  - Participate to intercomparison projects.
- Access to external resources
  - operations would be calculated on the server, while the system resources could be exposed to clients.
  - large climate data stores
  - compute resources
  - complex software systems
- Cross institutional, cross-community
  - depends *only* on an open standard interface.
  - several implementations for a Processing Service can be used.
  - clients (web portals) can rely on a stable processing service interface.

Who is using WPS?

Copernicus, EU Project

- Copernicus Climate Change Service: http://climate.copernicus.eu/
- WPS Demo: https://github.com/cp4cds/copernicus-wps-demo

CEDA/STFC, UK

- CEDA: http://www.ceda.ac.uk/
- STFC: http://www.stfc.ac.uk/
- COWS WPS: http://wps-web1.ceda.ac.uk/ui/home

IPSL/LSCE, France

- IPSL: https://www.ipsl.fr/en/
- LSCE: http://www.lsce.ipsl.fr/
- Talk at Euro Cordex 2016
- Paper about Flyingpigeon (in Review)

KNMI, Netherlands

- KNMI: http://www.knmi.nl/
- Climate4Impact Portal: https://climate4impact.eu/

ESGF: lead by LLNL, US

- LLNL: https://www.llnl.gov/
- ESGF Project: https://esgf.llnl.gov/
- Compute WPS https://github.com/ESGF/esgf-compute-wps

CRIM/Ouranos, Canada

- CRIM: http://www.crim.ca/en/
- Ouranos: https://www.ouranos.ca/en/
- Talk at AGU 2016

NCI, Australia

- NCI: http://nci.org.au/
- Talk about WPS for Pointclouds

DKRZ, Germany

- DKRZ: https://www.dkrz.de/dkrz-en
- Birdhouse: http://bird-house.github.io/
- Talk at ESGF F2F, 2016

KIT, Germany

- KIT: http://www.kit.edu/english/
- Talk at EGU 2017

APEC Climate Center, South Korea

- APCC: http://www.apcc21.org/
- Talk at FOSS4G, Bonn, 2016

Links

Birdhouse:

- http://bird-house.github.io/
- Birdhouse Workshop: http://birdhouse-workshop.readthedocs.io/en/latest/
- Birdhouse talks: http://birdhouse.readthedocs.io/en/latest/index.html#presentations-blog-posts

WPS:

- http://geoprocessing.info/index.html
- PyWPS: http://pywps.org/
- PyWPS Workshop: https://github.com/PyWPS/pywps-workshop

Conda:

- https://conda.io/docs/
- https://www.anaconda.com/blog/developer-blog/conda-data-science/

Python:

- pytest: https://docs.pytest.org/en/latest/
- Python argparse: https://docs.python.org/3/howto/argparse.html

Jupyter Notebooks:

- Notebook Gallery: https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks
- IOOS Demos: https://ioos.github.io/notebooks_demos/

Cartopy/Matplotlib:

- Matplotlib: http://matplotlib.org/
- Cartopy: http://scitools.org.uk/cartopy/
- Using cartopy with matplotlib
- GeoScience Notebook
- PyEarthScience matplotlib examples: https://github.com/KMFleischer/PyEarthScience/

netCDF:

- http://nbviewer.jupyter.org/github/julienchastang/unidata-python-workshop/blob/master/reading_netCDF.ipynb

- http://schubert.atmos.colostate.edu/~cslocum/netcdf_example.html

OpenDAP:

- https://www.seegrid.csiro.au/wiki/pub/AUKEGGS/FinalWorkshop/seminar.pdf

Docker:

- What is Docker?: https://www.docker.com/what-docker
- Docker Training: http://slides.com/dataduke/docker-001#/
- Play with Docker: http://labs.play-with-docker.com/

Todo List

Todo: This example with Flyingpigeon is outdated.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/example.rst, line 4.)

Todo: Add PEP8 instructions for more editors: PyCharm, Kate, Emacs, Vim, Spyder.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/guide_dev.rst line 94.)

Todo: needs to be updated.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/overview.rst, line 55.)

Todo: Describe the relationship between the frontend and Phoenix.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https line 25.)

Todo: Examine the Birdhouse/Birdhouse-Docs to see if this section can be merged back to it and joined as a submodule here. Birds of interest are listed there.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https line 166.)

Todo: How to add WPS, WMS, WFS servers to PAVICS.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https line 100.)

Todo:

- Add images for the step-by-step processes
- How to modify the meta data associated with layers (how they appear in the interface)
- Add advice on setting styles with SLD4raster and other tools/advice

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https
line 69.)

Todo: Base PAVICS installation is incomplete. The following lines refer to Phoenix instance. Need to specify which birds are needed for a bare installation of PAVICS: Phoenix, FlyingPigeon, Malleefowl, Emu, etc.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https
line 59.)

Todo: Update the installation and config with security changes

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https
line 200.)

Todo: Document how to run integration tests

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https
line 5.)

Todo: How authorizations for services work (the concept) How to grant users access to data and services

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https
line 25.)

Todo: Take a systematic approach and link to other birds and libraries through intersphinx

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https
line 18.)

Todo: Review by CRIM.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https
line 20.)

Todo: Write tutorial on creating and launching workflows

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https
line 10.)

Todo: Describe how to use the UI to add data to the workspace.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https
line 7.)

Todo: Add ssh tunnel or jupyter console example.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https
line 12.)

Todo: add travis example

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/_gitext/https
line 6.)

Todo: explanation of enabling spinx automatic api documentation.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/guide_WPS.r
line 109.)

Todo: Add references to OGC testbed.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/projects.rst,
line 36.)

Todo: How to create a conda package

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/birdhouse/checkouts/latest/docs/source/tutorial_wps.
line 183.)

4.2.2 Birdhouse Workshop

Welcome to the **`Birdhouse`_** Workshop. This workshop is a hands-on session, which will guide you in creating a process for a **`Web Processing Service`_**. During the workshop you will learn how Birdhouse supports this development cycle.

Warning: Under Construction . . .

Useful Links

WPS Documentation

- What is WPS?
- WPS on OSGeo Live
- WPS tutorial
- OGC Web Processing Service Standard

- PyWPS Wiki
- GeoServer tutorial

Talks:

- The WPS 2.0 standard (preliminary information)
- WPS Application Patterns
- Using WPS (PyWPS) with Taverna Orchestration
- Pywps a tutorial for beginners and developers
- Zoo presentation foss4g.jp-2011

WPS Software

WPS Server Software:

- *PyWPS*
- GeoServer - http://docs.geoserver.org/stable/en/user/services/wps/index.html
- Zoo - `http://www.zoo-project.org/`
- *COWS*
- Deegree - http://www.deegree.org/
- 52 North - http://52north.org/communities/geoprocessing/wps/

WPS Client Software:

- *OWSLib Python Client*
- OpenLayers WPS Plugin - http://dev.openlayers.org/docs/files/OpenLayers/WPSClient-js.html
- GeoTools WPS Module - http://docs.geotools.org/latest/userguide/unsupported/wps.html
- 52 North Java Client - http://52north.org/communities/geoprocessing/wps/index.html
- 52 North Javascript Client - http://geoprocessing.demo.52north.org:8080
- WPS Javascript Client by Boundless - https://github.com/boundlessgeo/wps-gui

QGIS Desktop GIS with wps plugins:

- http://www.qgis.org/en/site/
- http://plugins.qgis.org/plugins/wps/
- http://geolabs.fr/plugins.xml

uDig Desktop GIS with wps plugins:

- http://udig.refractions.net/
- https://udig.github.io/docs/user/reference/Using%20the%20WPS%20plugin.html
- https://github.com/52North/uDig-WPS-plugin (outdated)

WMS Software

WMS server:

- ncWMS2 - http://reading-escience-centre.github.io/edal-java/
- adaguc - http://adaguc.knmi.nl/
- sci-wms - http://sci-wms.github.io/sci-wms/

WMS clients:

- OpenLayers - http://openlayers.org/
- **Leaflet - http://leafletjs.com/**
  - time dimension - http://apps.socib.es/Leaflet.TimeDimension/examples/
- GeoExt - http://geoext.github.io/geoext2/

Scientific Workflow Tools

Workflow Engines:

- *Dispel4py*
- *RestFlow*
- *Taverna*
- *VisTrails*
- Kepler - https://kepler-project.org/
- KNIME - http://www.knime.org/

Taverna with WPS:

- `http://rsg.pml.ac.uk/wps/generic.cgi?request=GetCapabilities&service=WPS`
- https://www.youtube.com/watch?v=JNAtoOejVIo
- https://taverna.incubator.apache.org/introduction/services-in-taverna.html
- https://github.com/myGrid/small-area-estimator
- http://comments.gmane.org/gmane.science.biology.informatics.taverna.user/1415
- http://dev.mygrid.org.uk/wiki/display/developer/SCUFL2

VisTrails with WPS:

- https://github.com/ict4eo/eo4vistrails
- http://proj.badc.rl.ac.uk/cows/wiki/CowsWps/CDOWPSWorkingGroup/WPSAndWorkflows
- `http://www.kitware.com/source/home/post/105`

Kepler with WPS:

- https://kepler-project.org/users/sample-workflows

Workflows with PyWPS:

- https://github.com/AnnaHomolka/PyWPS/blob/master/doc/tutorial_process_chaining.pdf

Other Workflow Engines:

- `http://www.yawlfoundation.org/`
- https://en.wikipedia.org/wiki/Scientific_workflow_system
- http://airavata.apache.org/
- http://search.cpan.org/~nuffin/Class-Workflow-0.11/

Scientific Python

- Anaconda - https://www.continuum.io/downloads

Completely free enterprise-ready Python distribution for large-scale data processing, predictive analytics, and scientific computing

- pandas - http://pandas.pydata.org/

Python Data Analysis Library

Python in Climate Science

- OpenClimateGIS - https://earthsystemcog.org/projects/openclimategis/

OpenClimateGIS is a Python package designed for geospatial manipulation, subsetting, computation, and translation of climate datasets stored in local NetCDF files or files served through THREDDS data servers. [..]

- ICCLIM (i see clim . . .) - https://github.com/cerfacs-globc/icclim

Python library for climate indices calculation. Documentation at http://icclim.readthedocs.io/en/latest/

Python Web Frameworks and Utils

- Pyramid - http://www.pylonsproject.org/
- Authomatic - http://peterhudec.github.io/authomatic/
- Bootstrap - http://getbootstrap.com/
- Bootstrap Tutorial - http://www.w3schools.com/bootstrap/default.asp
- Deform - https://github.com/Pylons/deform
- Deform with Bootstrap demo - http://deform2demo.repoze.org/
- Colander - http://docs.pylonsproject.org/projects/colander/en/latest/index.html
- TinyMCE - https://www.tinymce.com/
- Font Awesome - http://fontawesome.io/
- Leaflet - http://leafletjs.com/
- Leaflet TimeDimension - http://apps.socib.es/Leaflet.TimeDimension/examples/

Example WPS Services

List of available Web Processing Services:

- Zoo WPS for PublicaMundi project - http://zoo.dev.publicamundi.eu/cgi-bin/zoo_loader.cgi?service=WPS&version=1.0.0&request=GetCapabilities
- GeoServer Demo WPS - http://demo.opengeo.org/geoserver/wps?request=GetCapabilities&service=WPS
- USGS Geo Data Portal- http://cida.usgs.gov/climate/gdp/process/WebProcessingService
- KNMI climate4impact Portal - http://climate4impact.eu//impactportal/WPS?request=GetCapabilities&service=WPS
- BADC CEDA - http://ceda-wps2.badc.rl.ac.uk/wps?request=GetCapabilities&service=WPS
- delatres - http://dtvirt5.deltares.nl/wps/?Request=GetCapabilities&Service=WPS
- 52 North - http://geoprocessing.demo.52north.org:8080/52n-wps-webapp-3.3.1/WebProcessingService?Request=GetCapabilities&Service=WPS
- 52 North - http://geoprocessing.demo.52north.org:8080/52n-wps-webapp-3.3.1-gt/WebProcessingService?Request=GetCapabilities&Service=WPS
- ZOO Demo WPS - http://zoo-project.org/cgi-bin/zoo_loader3.cgi?Request=GetCapabilities&Service=WPS
- British Antarctic Survey WPS for Meteorological Data - http://sosmet.nerc-bas.ac.uk:8080/wpsmet/WebProcessingService?Request=GetCapabilities&Service=WPS
- PyWPS Demo - http://apps.esdi-humboldt.cz/pywps/?request=GetCapabilities&service=WPS&vers
0.0

Alternatives to WPS

- XML-RPC: Simple cross-platform distributed computing, based on the standards of the Internet. - http://xmlrpc.scripting.com/
- Swagger is a simple yet powerful representation of your RESTful API. - http://swagger.io/

Related Projects

- http://geopython.github.io/
- http://geonode.org/
- http://esgf.llnl.gov/
- http://climate4impact.eu/impactportal/general/index.jsp
- http://adaguc.knmi.nl/
- http://wps-web1.ceda.ac.uk/ui/home
- https://freva.met.fu-berlin.de/
- https://climate.apache.org/

4.3 Calling a Service (birdy)

4.3.1 Examples

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

Basic Usage

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)
```

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')
```

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)
```

Demo

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...
  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.

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

(continues on next page)

(continued from previous page)

```
    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
↪',
    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'...
```

4.3.2 Examples

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

4.4 Basic Usage

4.5 Demo

4.6 WPS general usage

In the following we show an example with a *Word Counter* function which is enabled as a web-service using WPS.

4.6.1 Defining a *Word Counter* function

In the following example we will use the *Word Counter* function:

```
def count_words(file):
    """Calculates number of words in text document.
    Returns JSON document with occurrences of each word.
    """
    return json_doc
```

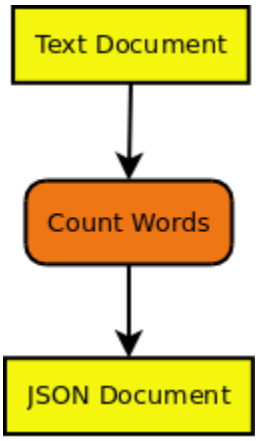
This Python function has the following parts:

- a *name* or *identifier*: count_words
- a *description*: Calculates number of words . . .
- *input parameters*: file (mime type *text/plain*)
- *output parameters*: json_doc (mime type *application/json*)

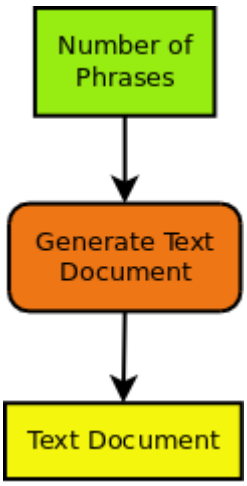
Now, we would like to run this function remotely using a simple web-service. To get this web-service we can use WPS. The function parts (name, parameters) are all we need to know to define a WPS process.

4.6.2 WPS definition of *Word Counter*

To add a new proccess you need to define the input and output parameters. For the *Word Counter* process this looks like the following.



Here is another example for a *Text Generator* process. We will use it later for chaining processes.



There are two types of input/output parameters:

- Literal Parameters (green): these are simple data types like integer, boolean, string, ...
- Complex Parameters (yellow): these are documents with a mime-type (xml, cvs, jpg, netcdf, ...) provided as URL or directly.

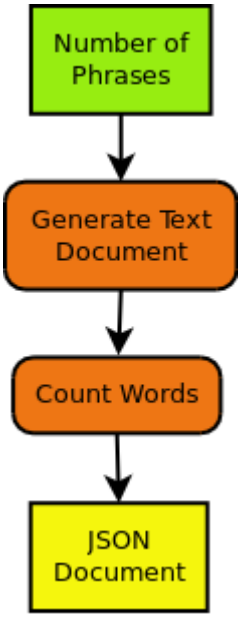
An input/output parameter has:

- a name or *identifier*
- a descriptive *title*
- an *abstract* giving a description of the parameter
- *multiplicity* ... how often can this parameter occur: optional, once, many ...
- in case of literal parameters a list of *allowed values*.

For more details see the following WPS Tutorial.

4.6.3 Chaining WPS processes

If you know the input/output parameters of processes you can chain processes. For example we will chain a *Text Generator* process to our *Word Counter* process.



The *Text document* output of the *Text Generator* process becomes the input of *Word Counter* process.

You can chain process manually by calling them one after the other. The WPS specification allows you to also chain process with a single WPS request. To get even more flexibility (using if-clauses, loops, monitoring ...) you can also use a *workflow engine* (*Taverna*, *VisTrails*, *Dispel4py*, ...).

You will find more details about chaining in the GeoProcessing document and the GeoServer Tutorial.

4.6.4 WPS process implementation with PyWPS

There are several WPS implementations available (*GeoServer*, *COWS*, ...). In birdhouse, we use the Python implementation *PyWPS*. In *PyWPS* the *Word Counter* process could look like the following:

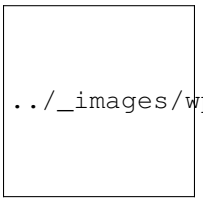
You can see the definition of the input and output parameters and the `execute()` method where the real `count_words()` function is called. You will find more details about implementing a WPS process in the PyWPS Tutorial.

4.6.5 Using WPS

A WPS service has three operations:

- *GetCapabilities*: which processes are available
- *DescribeProcess*: what are the input and output parameters of a specific process
- *Execute*: run a process with parameters.

The following diagram shows these operations:



To call these process one can use simple HTTP request with key/value pairs:

- *GetCapabilites* request:

```
http://localhost:8094/wps?&request=GetCapabilities&service=WPS&version=1.0.0
```

- *DescribeProcess* request for *wordcount* process:

```
http://localhost:8094/wps?&request=DescribeProcess&service=WPS&version=1.0.0&
↪identifier=wordcount
```

- *Exceute* request:

```
http://localhost:8094/wps?request=Execute&service=WPS&version=1.0.0&
↪identifier=wordcount
                        &DataInputs=text=http://birdhouse.readthedocs.org/en/
↪latest/index.html
```

A process can be run *synchronously* or *asynchronously*:

- sync: You make a HTTP request and you need to wait until the request returns with a response (or timeout). This is only useful for short-running processes.
- async: You make a HTTP request and you get immediately a response document. This document gives you a link to a status document which you need to poll until the process has finished.

Processes can be run with simple HTTP get-requests (as shown above) and also with HTTP post-requests. In the later case XML documents are exchanged with the communication details (process, parameters, . . .).

For more details see the following WPS Tutorial.

There are also some IPython notebooks which show the usage of WPS.

4.6.6 Calling *Word Counter* with Birdy

Now, we are using Birdy wps command line client to access the *wordcount* process.

Which proccess are available (*GetCapabilities*):

```
$ birdy -h
usage: birdy [-h] <command> [<args>]

optional arguments:
  -h, --help            show this help message and exit

command:
  List of available commands (wps processes)

  {chomsky,helloworld,inout,ultimatequestionprocess,wordcount}
                        Run "birdy <command> -h" to get additional help.
```

What input and output parameters does *wordcount* have (*DescribeProcess*):

```
$ birdy wordcount -h
usage: birdy wordcount [-h] --text [TEXT] [--output [{output} [{output} ...]]]

optional arguments:
  -h, --help            show this help message and exit
  --text [TEXT]         Text document: URL of text document, mime
                        types=text/plain
  --output [{output} [{output} ...]]
                        Output: output=Word count result, mime
                        types=text/plain (default: all outputs)
```

Run *wordcount* with a text document (*Execute*):

```
$ birdy wordcount --text http://birdhouse.readthedocs.org/en/latest/index.html
Execution status: ProcessAccepted
Execution status: ProcessSucceeded
Output:
output=http://localhost:8090/wpsoutputs/emu/output-37445d08-cf0f-11e4-ab7e-
↪68f72837e1b4.txt
```

Hello World WPS (emu):

- Emu Example with Docker

Create a conda package

Todo: How to create a conda package

Python syntax:

```
"""Python WPS execute"""

from owslib.wps import WebProcessingService, monitorExecution
from os import system
```

4.7 Climate Indices (finch):

WPS finch is providing services to calculate climate indices widely used in climate change adaptation planing processes.

Have a look on the examples of the finch documentation: https://pavics-sdi.readthedocs.io/projects/finch/en/latest/notebooks/index.html

4.8 Hydrological models (raven):

WPS raven is providing hydrological models for e.g. hydro-power controlling and sustainable planing

Have a look on the examples of the raven documentation: https://pavics-raven.readthedocs.io/en/latest/notebooks/index.html

4.9 Server administration

4.9.1 birdhouse installation

Warning: This section is outdated . . .

Birdhouse consists of several components like Malleefowl and Emu. Each of them can be installed individually. The installation is done using the Python-based build system *Buildout*. Most of the dependencies are maintained in the *Anaconda Python distribution*. For convenience, each birdhouse component has a Makefile to ease the installation so you don't need to know how to call the Buildout build tool.

Requirements

Birdhouse uses *Anaconda Python distribution* for most of the dependencies. If Anaconda is not already installed, it will be installed during the installation process. Anaconda has packages for Linux, MacOSX and Windows. But not all packages used by birdhouse are already available in the default package channel of Anaconda. The missing packages are supplied by birdhouse on *Binstar*. But we currently maintain only packages for Linux 64-bit and partly for MacOSX.

So the short answer to the requirements is: **you need a Linux 64-bit installation**.

Birdhouse is currently used on Ubuntu 14.04 and CentOS 6.x. It should also work on Debian, LinuxMint and Fedora.

Birdhouse also installs a few system packages using *apt-get* on Debian based distributions and *yum* on RedHat/CentOS based distributions. For this you need a user account with *sudo* permissions. Installing system packages can be done in a separate step. So your installation user does not need any special permissions. All installed files will go into a birdhouse Anaconda environment in the home folder of the installation user.

Installing from source

The installation of birdhouse components from source is done with some few commands. Here is an example for the Emu WPS service:

```
$ git clone https://github.com/bird-house/emu.git
$ cd emu
$ make clean install
$ make start
$ firefox http://localhost:8094/wps
```

All the birdhouse components follow the same installation pattern. If you want to see all the options of the *Makefile* then type:

```
$ make help
```

You will find more information about these options in the Makefile documentation.

Read the documention of each birdhouse component for the details of the installation and how to configure the components. The birdhouse bootstrap documentation gives some examples of the different ways of making the installation.

On the WPS client side we have:

- Phoenix: a Pyramid web application.
- Birdy: a simple WPS command line tool.

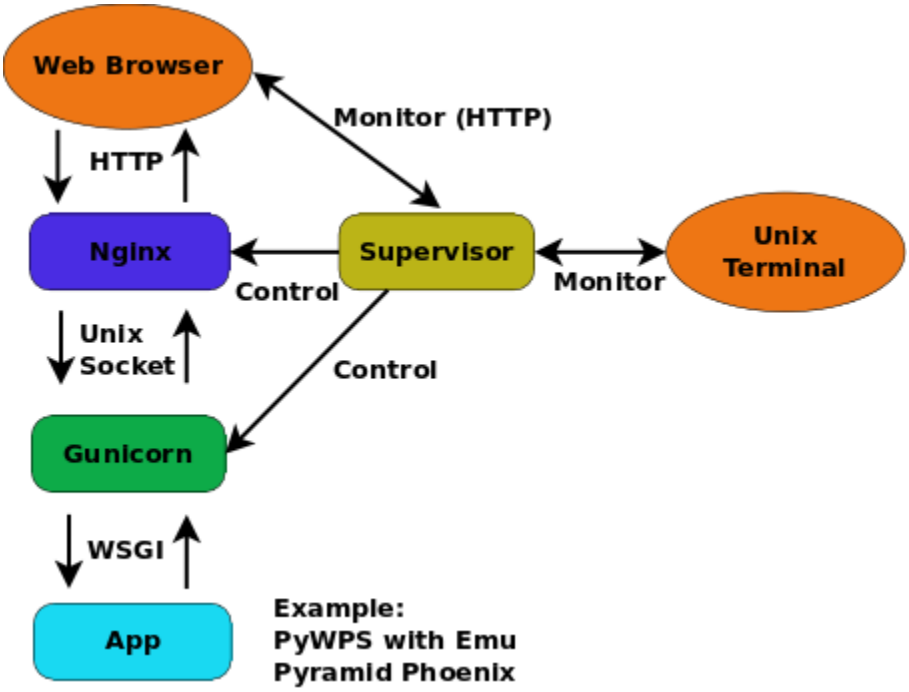
On the WPS server side we have:

- Malleefowl: provides base WPS services to access data.
- Flyingpigeon: provides WPS services for the climate impact community.
- Hummingbird: provides WPS services for CDO and climate metadata checks.
- Emu: just some WPS processes for testing.

Nginx, gunicorn and supervisor

Birdhouse sets up a *PyWPS* server (and also the Phoenix web application) using *Buildout*. We use the *Gunicorn* HTTP application server (similar to Tomcat for Java servlet applications) to run these web applications with the *WSGI* interface. In front of the Gunicorn application server, we use the *Nginx* HTTP server (similar to the Apache web server). All these web services are started/stopped and monitored by a *Supervisor* service.

See the following image for how this looks like:

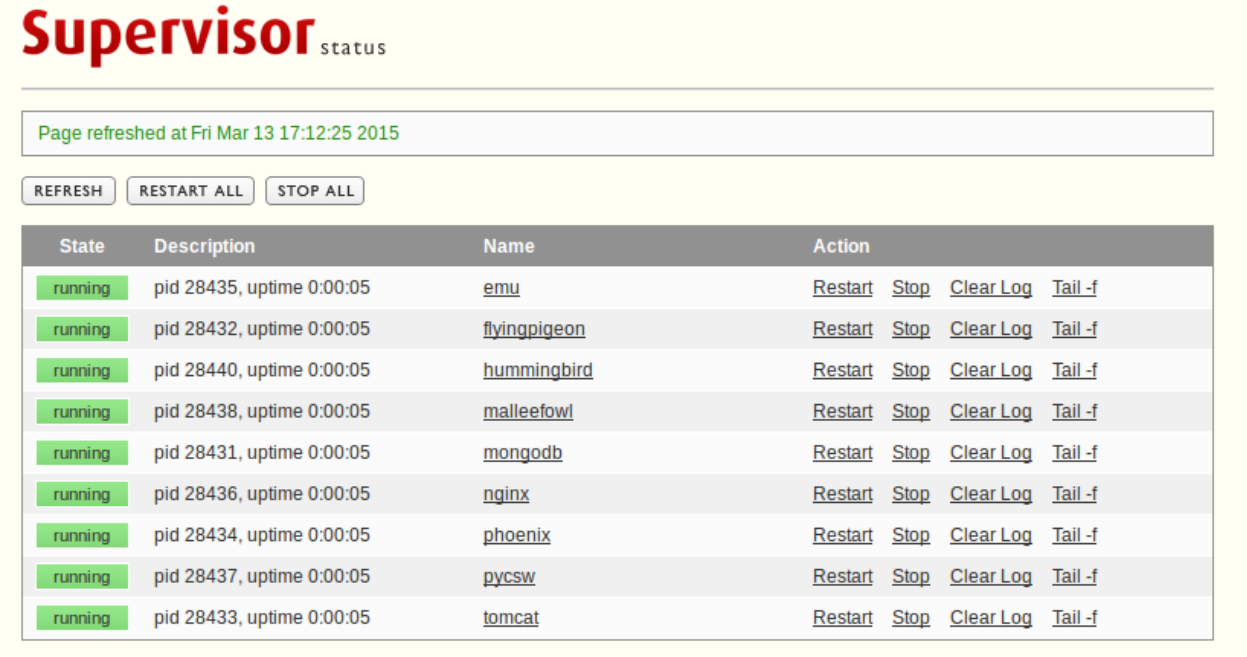


When installing a birdhouse WPS service, you don't need to care about this setup. This is all done by Buildout and using some extensions provided by birdhouse.

The Makefile of a birdhouse application has convenience targets to start/stop a WPS service controlled by the Supervisor and to check the status:

```
$ make start    # start wps service
$ make stop     # stop wps service
$ make status   # show status of wps service
Supervisor status ...
/home/pingu/.conda/envs/birdhouse/bin/supervisorctl status
emu                              RUNNING   pid 25698, uptime 0:00:02
malleefowl                       RUNNING   pid 25702, uptime 0:00:02
mongodb                          RUNNING   pid 25691, uptime 0:00:02
nginx                            RUNNING   pid 25699, uptime 0:00:02
phoenix                          RUNNING   pid 25694, uptime 0:00:02
pycsw                            RUNNING   pid 25700, uptime 0:00:02
tomcat                           RUNNING   pid 25693, uptime 0:00:02
```

You can also use the Supervisor monitor web service which by default is available on port http://localhost:9001/. The Supervisor monitor app looks like in the following screenshot.



Using birdhouse with Docker

An alternative way to install and deploy birdhouse Web Processing Services is by using *Docker*. The birdhouse WPS servers are available as a Docker image on Docker Hub. See an example on how to use them with the Emu WPS Docker image.

4.9.2 birdhouse administration

Warning: This section needs is outdated and needs to be rewritten!

Set up a birdhouse ecosystem server

If you are already familiar with installing single standalone WPS (follow the installation guides in the documentations of e.g. emu), then you are ready to set up a birdhouse containing flyingpigeon (providing scientific analyses methods), malleefowl (to search and fetch data) and the pheonix (a graphic interface for a web browser including a WMS).

General Remarks

Check the *Requirements* of your system!
The installation is done as **normal user**, root rights are causing conflicts.

Prepare Installation

It is recommended to collect the repositories in a separate folder (e.g. birdhouse, but can have a name of your choice):

```
$ mkdir birdhouse
$ cd birdhouse
```

Get the source code from GitHub

```
$ git clone https://github.com/bird-house/flyingpigeon.git
$ git clone https://github.com/bird-house/pyramid-phoenix.git
$ git clone https://github.com/bird-house/malleefowl.git
```

Run Installation

You can run the installation with default settings. It will create a conda environment and deploy all required software dependencies there.

Note: Read the *changing the default configuration* if you want to customize the configuration.

In **all** of the tree folders (malleefowl, flyingpigeon and pyramid-phoenix) run:

```
$ make install
```

This installation will take some minutes to fetch all dependencies and install them into separate conda environments.

Start the Services

in **all** of the birds run:

```
$ make start
```

Launching the Phoenix Web App

If the services are running, you can launch the GUI in a common web browser. By default, phoenix is set to port 8081:

```
firefox http://localhost:8081
```

or:

```
firefox https://localhost:8443/
```

Now you can log in (upper right corner) with your Phoenix password created previously. Phoenix is just a graphical interface with no more function than looking nice ;-).

Register a service in Phoenix Web App

Note: Please read the Phoenix documentation

Your first administration step is to register *flyingpigeon* as a service. For that, log in with your phoenix password. In the upper right corner is a tool symbol to open the *settings*. Click on *Services* and the *Register a Service*.

Flyingpigeon is per default on port 8093.

The appropriate url is:

```
http://localhost:8093/wps
```

Provide service title and name as you like: * Service Title: Flyingpigeon * Service Name: flyingpigeon

check *Service Type*: *Web Processing Service* (default) and register.

Optionally, you can check *Public access?*, to allow unregistered users to launch jobs. (**NOT recommended**)

Launching a Job

Now your birdhouse ecosysem is set up. The also installed malleefowl is already running in the background and will do a lot of work silently. There is **no need to register malleefowl** manually!

Launching a job can be performed as a process (Process menu) or with the wizard. To get familliar with the processes provided by each of the birds, read the approriate documentation for each of the services listed in the overview:

Changing the default configuration

You can customize the configuration of the service. Please read the documentation, for example:

- Phoenix documentation
- Flyingpigeon documentation

Furthermore, you might change the hostname (to make your service accessible from outside), ESGF-node connection, the port or the log-level for more/less information in the administrator logfiles. Here is an example *pyramid-phoenix/custom.cfg*:

```
[settings]
hostname = localhost
http-port = 8081
https-port = 8443
log-level = DEBUG
# run 'make passwd' and to generate password hash
phoenix-password = sha256:513....
# generate secret
# python -c "import os; print(''.join('%02x' % ord(x) for x in os.urandom(16)))"
phoenix-secret = d5e8417....30
esgf-search-url = https://esgf-data.dkrz.de/esg-search
wps-url = http://localhost:8091/wps
```

Update Phoenix Password

To be able to log into the Phoenix GUI once the services are running, it is necessary to generate a password: go into the pyramid-phoenix folder and run:

```
$ make passwd
```

This will automatically write a password hash into pyramid-phoenix/custom.cfg

Backups

See the mongodb documentation on how to backup the database. With the following command you can make a dump of the `users` collection of the Phoenix database:

```
$ mongodump --port 27027 --db phoenix_db --collection users
```

Asking for Support

In case of questions or trouble shooting, feel welcome to join the birdhouse chat and get into contact with the developers directly.

4.10 PyWPS with R

The following shows how you can wrap R software with PyWPS.

4.10.1 Examples of R Birds

- pcic/quail
  - wps_climdex_gsl.py
- pcic/chickadee
  - wps_BCCAQ.py

4.10.2 Rpy2

There's several R-to-python Python libraries but Rpy2 is probably the most well documented and most frequently used. As long as it is installed, a R library can be accessed with `importr([package-name])`. Since R `base` and `utils` are installed with `rpy2` you can import them:

```
from rpy2.robjects.packages import importr
base = importr("base")
utils = importr("utils")
```

Then you can use functions from that package with `package.function_name()`. If the R function name has a period `.` it is replaced with an underscore _ in python.

```
base.all(True)
base.all_equal("hello","world") # all.equal() in R
```

You can execute any regular R code as a string passed to `robjects.r()`

```
from rpy2 import robjects
count = robjects.r("c(1,2,3)")
robjects.r("all(T)")
```

You can also access R functions with the syntax `robjects.r["function.name"]` if you want to avoid the import step.

```
robjects.r["save"](count, file=output_path)
```

Install another package with Rpy2 and use the functions form that package...

```
utils.install_packages("climdex.pcic")
climdex_pcic = importr("climdex.pcic")
climdex_pcic.climdex_gsl(climdexInput, gsl_mode)
```

4.10.3 I/O

Rpy2 handles R-to-python conversions of LITERAL_DATA_TYPES, but objects of other types may need to be stored in a RDS or Rdata file. RDS files and Rdata files are indistinguishable by mime type when read to the server so their handling has to be done elsewhere in the processes. You can see how it's handled in PCIC's quail. Read the file as a `ComplexInput` with `format`:

```
from pywps import ComplexInput, Format

ComplexInput(
   "r_file",
   "R data file",
   supported_formats=[Format("application/x-gzip", encoding="base64")],
)
```

... And if your output is an R object you can save that object to an RDS or Rdata file and return it with `ComplexOutput`:

```
from pywps import ComplexOutput

ComplexOutput(
  "r_output",
  "R output file",
  supported_formats=[Format("application/x-gzip", extension=".rda", encoding="base64
↪")],
)
```

4.10.4 Installing Dependencies

You can write a simple script in `R`, `bash`, or `Python` to automate installation of R package dependencies. `devtools::install_version()` is used to pin versions in PCIC's `quail` and `chickadee`. You can take a look at the R script here.

The script reads from a file similar to `requirements.txt` for Python dependencies:

r_requirements.txt:

```
PCICt==0.5.4.1
climdex.pcic==1.1.11
```

4.10.5 Dockerfile

To install `Rpy2`, R needs to be installed already. A good base image for R is rocker/r-ver and you can install Python on top of it. Check out the pcic/quail Dockerfile as an example.

CHAPTER

FIVE

PUBLICATIONS

5.1 Talks and articles

Articles, book sections and conference proceedings and presentations related to the birdhouse projects:

2019:

- WPS Deployment at CORDEX Copernicus Workshop, Copenhagen
- UN GIS Initiative Workshop at FOSS4G Bucharest

2018:

- Birdhouse in ISPRS photogrammetry and remote-sensing [ELH+18]
- FOSS4G 2018 in Dar-Es-Salaam
- Open Climate GIS and Birdhouse at Pangeo Developer Workshop, 2018
- IGARSS 2018
- D-GEO Days, 2018
- GIZ Fachtagung, 2018
- Copernicus/Birdhouse at EGU 2018, Vienna
- Flyingpigeon in Computes and Geosciences, January 2018 [HEAC+18]

2017:

- Birdhouse in LSDMA book, 2017 [JMS17]
- UNCCC Subgroup 2017 in Kigali

2016:

- AGU 2016 in San Francisco
- ESGF F2F 2016 in Washington
- FOSS4G 2016 in Bonn
- EGI Workshop 2016 in Amsterdam

- EGU 2016 in Vienna
- ICRC-CORDEX 2016
- Model Animation LSCE
- Talk on USGS WebEx 2016/02/18

2015:

- Paris Coding Spring 2015 at IPSL

2014:

- EGI Community Forum 2014 at Helsinki
- Prag
- Optimization of data life cycles [JGG+14]

2013:

- Gerics Hamburg User-Developer Workshop

5.2 References

CHAPTER

SIX

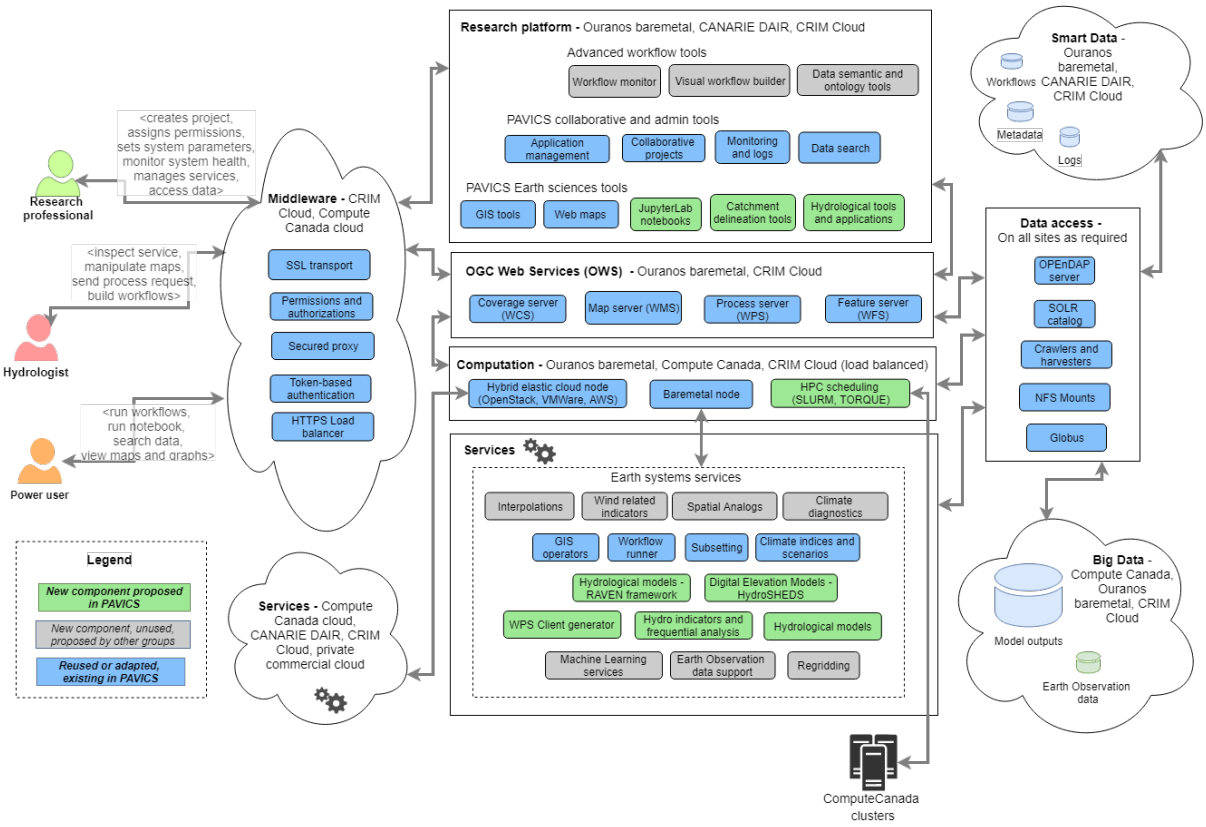
PROJECT EXAMPLES

The birdhouse *framework is modular organized* to enable a flexible architecture design depending on the projects needs. Due to the OCG Standard, software components non-birdhouse components can be combined for interoperability. Here are some examples of real projects to show the flexibility and potential of the birdhouse framework.

6.1 PAVICS

- PAVICS: Platform for climate analysis and visualization by Ouranos and CRIM, Canada.
- PAVICS-Hydro : Additional services for PAVICS allowing users to perform hydrological modeling and analysis.

6.1.1 Backend - PAVICS Node



PAVICS nodes are data, compute and index endpoints accessed through the PAVICS platform or external clients. The Node service is the backend that provides data storage, metadata harvesting, indexation and discovery of local and federated data, user authentication and authorization, server registration and management. The *node service* is therefore composed of several services that are briefly described below, accompanied by links to the full documentation of each individual building block.

The backend of PAVICS-SDI is built entirely with Free and Open Source Software. All of the backend projects (source code and documentation) are open to be inspected, built upon, or contributed to.

Data storage

Data is stored on two different servers: THREDDS for gridded netCDF data, and GeoServer for GIS features (region polygons, river networks).

THREDDS The *Thematic Real-time Environmental Distributed Data Services* (THREDDS) is a server system for providing scientific data and metadata access through various online protocols. The PAVICS platform relies on THREDDS to provide access to all netCDF data archives, as well as output files created by processes. The code is hosted on this GitHub repository. THREDDS support direct file access as well as the OPeNDAP protocol, which allows the netCDF library to access segments of the hosted data without downloading the entire file. Links to files archived on THREDDS are thus used as inputs to WPS processes. File content cannot however be directly displayed by the frontend and require an intermediary (see ncWMS).

GeoServer GeoServer is an OGC compliant server system built for viewing, editing, and presenting geospatial data. PAVICS uses GeoServer as its database for vector geospatial information, such as administrative regions, watersheds and river networks. The frontend sends requests for layers that can be overlayed on the map canvas. See the GeoServer documentation for more information on its capabilities.

Indexation

Although information about file content is stored in the netCDF metadata fields, accessing and reading those fields one by one takes a considerable amount of time. The strategies used here mimic those used by ESGF, and comprises running a crawler over all netCDF files hosted on THREDDS, extracting relevant metadata and storing them in a SOLR database. Search queries are thus directed at SOLR, which returns a list of links matching the search terms. The crawler is part of the PAVICS-DataCatalog library.

SOLR SOLR is a search platform part of the Apache Lucene project. It is used in this project for its faceted search capability. Search queries are relayed from the UI or WPS processes to the SOLR database, which returns a json file with the links to matching files.

PAVICS-DataCatalog PAVICS-DataCatalog is a database system for storing and serving information about available climate data.

Climate Analytic Processes with Birdhouse

The climate computing aspect of PAVICS is largely built upon the many components developed as part of the Birdhouse Project. The goal of Birdhouse is to develop a collection of easy-to-use Web Processing Service (WPS) servers providing climate analytic algorithms. Birdhouse servers are called 'birds', each one offering a set of individual processes:

Birdhouse/Finch Provides access to a large suite of climate indicators, largely inspired by **`ICCLIM`_**. Finch Official Documentation

Raven Provides hydrological modeling capability using the Raven framework, along with model calibration utilities, regionalization tools, hydrological indicators and frequency analysis.

Birdhouse/Malleefowl Provides processes to access ESGF data nodes and THREDDS catalogs, as well as a workflow engine to string different processes together. Malleefowl Official Documentation

Birdhouse/Flyingpigeon Provides a wide array of climate services including indices computation, spatial analogs, weather analogs, species distribution model, subsetting and averaging, climate fact sheets, etc. FlyingPigeon is the sand box for emerging services, which eventually will make their way to more stable and specialized birds. Flyingpigeon Official Documentation

Birdhouse/Hummingbird Provides access to climate Data Operators (CDO) functions and compliance-checker for netCDF files. Hummingbird Official Documentation

Virtually all individual processes ingest and return netCDF files (or OPeNDAP links), such that one process' output can be used as the input of another process. This lets scientist create complex workflows. By insisting that process inputs and outputs comply with the CF-Convention, we make sure that data is accompanied by clear and unambiguous metadata.

Authentication and authorization

Access to files and services is controlled by a security proxy called **`Twitcher`_**, also part of Birdhouse. Upon login, the proxy issues access tokens that allow users to access services behind the proxy. CRIM developed a Twitcher extension called Magpie that provides a higher level of granularity for service access.

Twitcher Proxy service issuing access tokens necessary to run WPS processes or any other OWS service.

Magpie Manages user/group/resource permissions for services behind Twitcher.

Gridded data visualization

The UI can display 2D netCDF fields by making a request to a ncWMS server. The UI will specify which time step of which file to map, and ncWMS will fetch the data from the THREDDS server, then convert the array into an image embedded into a WMS response. This conversion requires a mapping of numerical value to a color scale: a colormap and min/max values. The colormap is defined by the user through the UI, while default min/max values are stored in our SOLR database by the metadata crawler. Users may also specify min/max values directly within the UI.

ncWMS ncWMS is an implementation of the OGC's Web Mapping Service (WMS) specifically built for multidimensional gridded data such as the netCDF format. The PAVICS platform uses it to convert gridded netCDF data layers from a file or an OPeNDAP link to an image that can be accessed through WMS `GetMap` requests. See this reference paper for more information.

6.1.2 Backend - PAVICS Node

PAVICS nodes are data, compute and index endpoints accessed through the PAVICS platform or external clients. The Node service is the backend that provides data storage, metadata harvesting, indexation and discovery of local and federated data, user authentication and authorization, server registration and management. The *node service* is therefore composed of several services that are briefly described below, accompanied by links to the full documentation of each individual building block.

The backend of PAVICS-SDI is built entirely with Free and Open Source Software. All of the backend projects (source code and documentation) are open to be inspected, built upon, or contributed to.

Data storage

Data is stored on two different servers: THREDDS for gridded netCDF data, and GeoServer for GIS features (region polygons, river networks).

THREDDS The *Thematic Real-time Environmental Distributed Data Services* (THREDDS) is a server system for providing scientific data and metadata access through various online protocols. The PAVICS platform relies on THREDDS to provide access to all netCDF data archives, as well as output files created by processes. The code is hosted on this GitHub repository. THREDDS support direct file access as well as the OPeNDAP protocol, which allows the netCDF library to access segments of the hosted data without downloading the entire file. Links to files archived on THREDDS are thus used as inputs to WPS processes. File content cannot however be directly displayed by the frontend and require an intermediary (see ncWMS).

GeoServer GeoServer is an OGC compliant server system built for viewing, editing, and presenting geospatial data. PAVICS uses GeoServer as its database for vector geospatial information, such as administrative regions, watersheds and river networks. The frontend sends requests for layers that can be overlayed on the map canvas. See the GeoServer documentation for more information on its capabilities.

Indexation

Although information about file content is stored in the netCDF metadata fields, accessing and reading those fields one by one takes a considerable amount of time. The strategies used here mimic those used by ESGF, and comprises running a crawler over all netCDF files hosted on THREDDS, extracting relevant metadata and storing them in a SOLR database. Search queries are thus directed at SOLR, which returns a list of links matching the search terms. The crawler is part of the PAVICS-DataCatalog library.

SOLR SOLR is a search platform part of the Apache Lucene project. It is used in this project for its faceted search capability. Search queries are relayed from the UI or WPS processes to the SOLR database, which returns a json file with the links to matching files.

PAVICS-DataCatalog PAVICS-DataCatalog is a database system for storing and serving information about available climate data.

Climate Analytic Processes with Birdhouse

The climate computing aspect of PAVICS is largely built upon the many components developed as part of the Birdhouse Project. The goal of Birdhouse is to develop a collection of easy-to-use Web Processing Service (WPS) servers providing climate analytic algorithms. Birdhouse servers are called 'birds', each one offering a set of individual processes:

Birdhouse/Finch Provides access to a large suite of climate indicators, largely inspired by **`ICCLIM`**_. Finch Official Documentation

Raven Provides hydrological modeling capability using the Raven framework, along with model calibration utilities, regionalization tools, hydrological indicators and frequency analysis.

Birdhouse/Malleefowl Provides processes to access ESGF data nodes and THREDDS catalogs, as well as a workflow engine to string different processes together. Malleefowl Official Documentation

Birdhouse/Flyingpigeon Provides a wide array of climate services including indices computation, spatial analogs, weather analogs, species distribution model, subsetting and averaging, climate fact sheets, etc. FlyingPigeon is the sand box for emerging services, which eventually will make their way to more stable and specialized birds. Flyingpigeon Official Documentation

Birdhouse/Hummingbird Provides access to climate Data Operators (CDO) functions and compliance-checker for netCDF files. Hummingbird Official Documentation

Virtually all individual processes ingest and return netCDF files (or OPeNDAP links), such that one process' output can be used as the input of another process. This lets scientist create complex workflows. By insisting that process inputs and outputs comply with the CF-Convention, we make sure that data is accompanied by clear and unambiguous metadata.

Authentication and authorization

Access to files and services is controlled by a security proxy called **`Twitcher`**_, also part of Birdhouse. Upon login, the proxy issues access tokens that allow users to access services behind the proxy. CRIM developed a Twitcher extension called Magpie that provides a higher level of granularity for service access.

Twitcher Proxy service issuing access tokens necessary to run WPS processes or any other OWS service.

Magpie Manages user/group/resource permissions for services behind Twitcher.

Gridded data visualization

The UI can display 2D netCDF fields by making a request to a ncWMS server. The UI will specify which time step of which file to map, and ncWMS will fetch the data from the THREDDS server, then convert the array into an image embedded into a WMS response. This conversion requires a mapping of numerical value to a color scale: a colormap and min/max values. The colormap is defined by the user through the UI, while default min/max values are stored in our SOLR database by the metadata crawler. Users may also specify min/max values directly within the UI.

ncWMS ncWMS is an implementation of the OGC's Web Mapping Service (WMS) specifically built for multidimensional gridded data such as the netCDF format. The PAVICS platform uses it to convert gridded netCDF data layers from a file or an OPeNDAP link to an image that can be accessed through WMS `GetMap` requests. See this reference paper for more information.

6.2 COPERNICUS

- CP4CDS: Climate Projects for the Climate Data Store (part of the European Union's Copernicus Climate Change Service).

6.3 OGC-Testbeds

Todo: Add references to OGC testbed.

- OGC Testbed 13: Enhancement of scheduling services
- OGC Testbed 14: Enhancement of security

CHAPTER

SEVEN

IDEAS

In this section we are collection ideas how we could improve our coding and design in the Birdhouse/WPS context.

7.1 PyWPS Profiles

Warning: Work in progress.

7.1.1 Motivation

It happens quite often that we have a set of processes with common input (and output) parameters. In WPS the process signature (inputs+outputs) is called a WPS profile. In the following we show examples how to avoid *copy+paste* of these process parameters.

7.1.2 Python Mixins

One could use Python *mixin classes* to define a commonly used profile which can be adapted by each individual process.

See how a mixin class looks like:

https://www.ianlewis.org/en/mixins-and-python

See notebook examples how it could be used with PyWPS:

https://nbviewer.jupyter.org/github/bird-house/notebooks/blob/master/pywps-profiles/notebooks/process_mixin.ipynb

7.1.3 Python Decorators

We can also use function decorator to define a WPS profile for PyWPS.

See how a function decorator looks like:

https://krzysztofzuraw.com/blog/2016/python-class-decorators.html

Here are some notebook examples how it could be used with PyWPS:

- notebooks: https://nbviewer.jupyter.org/github/bird-house/notebooks/blob/master/pywps-profiles/notebooks/process_decorator.ipynb
- Emu subset with ESGF-API: https://github.com/bird-house/emu/blob/esgfwps/emu/processes/wps_esgf_subset.py

7.1.4 Simple Alternative: Shared Profile Module/Class

Relatively few developers will be familiar with the concepts of *mixins* and *decorators*. In other words, it might look a bit too much like magic. We could also simply create a module with all the common inputs and outputs used throughout the different WPS processes (*wpsio.py*). For a given Process definition, one then just import *wpsio* and refer to the objects in the inputs and outputs fields of the *Process.init* method.

See for example:

https://github.com/Ouranosinc/raven/blob/master/raven/processes/wps_regionalisation.py

Here is a notebook showing this approach which includes also an optional decorator:

https://nbviewer.jupyter.org/github/bird-house/notebooks/blob/master/pywps-profiles/notebooks/process_simple_profile_and_decorator.ipynb

CHAPTER

EIGHT

RELEASE NOTES

8.1 Niamey (October 2020, v0.10.0)

Highlighted Changes:

- Updated FlyingPigeon WPS with improved plot and subset processes.
- Improved cookiecutter template for PyWPS with cruft update.
- Ansible PyWPS playbook with support for Slurm cluster.

Released Tools:

- Twitcher WPS Proxy: 0.6.0
- Ansible Playbook for PyWPS 0.4.0
- Ansible Playbook for Twitcher 0.1.0
- Cookiecutter Template for PyWPS 0.5.0
- Birdy WPS Client: 0.6.9

Released WPS services:

- Emu WPS: 0.12.0
- FlyingPigeon WPS: 1.6.0
- Finch WPS: 0.5.3
- Hummingbird WPS: 0.9.0

Maintained Apps with Buildout:

- Phoenix Web App: 0.11.0

8.2 Oxford (April 2020, v0.9.0)

Highlighted Changes:

- Keycloak support in Twitcher and Phoenix.

Released Tools:

- Twitcher WPS Proxy: 0.6.0
- Ansible Playbook for PyWPS 0.3.0
- Ansible Playbook for Twitcher 0.1.0
- Cookiecutter Template for PyWPS 0.4.2
- Birdy WPS Client: 0.6.9

Released WPS services:

- Emu WPS: 0.11.1
- FlyingPigeon WPS: 1.5.1
- Finch WPS: 0.5.1
- Hummingbird WPS: 0.9.0

Maintained Apps with Buildout:

- Phoenix Web App: 0.11.0

8.3 Bucharest (October 2019, v0.8.0)

PyWPS was present at FOSS4G 2019 in Bucharest.

Highlighted Changes:

- Skipped buildout in Twitcher.
- Skipped conda handling in Makefile.
- Working on OAuth support in Twitcher and birdy.
- Released OWSLib extension for ESGF compute API.

Released Birds:

- Twitcher WPS Proxy: 0.5.2
- Ansible Playbook for PyWPS 0.2.2

- Cookiecutter Template for PyWPS 0.4.1
- Birdy WPS Client: 0.6.5
- Emu WPS: 0.11.0
- FlyingPigeon WPS: 1.5
- Finch WPS: 0.2.5
- Hummingbird WPS: 0.8.0
- Malleefowl WPS: 0.9.0
- OWSLib extension for ESGF: 0.2.0

Maintained Birds with Buildout:

- Phoenix Web App: 0.10.0

New Birds in the making:

- Kingfisher: https://github.com/bird-house/kingfisher
- Black Swan: https://github.com/bird-house/blackswan
- Eggshell: https://github.com/bird-house/eggshell
- Pelican: https://github.com/bird-house/pelican

8.4 San Francisco (May 2019, v0.7.0)

Highlighted Changes:

- All released birds support only Python >3.6.
- Support for the ESGF WPS profile with a Pelican WPS demo and an OWSLib extension.
- Support for MetaLink in Birdy and PyWPS to return multiple files as WPS output.
- Release of Finch, a WPS for climate indicators.

Released Birds:

- Ansible Playbook for PyWPS 0.2.1
- Cookiecutter Template for PyWPS 0.4.0
- Birdy WPS Client: 0.6.0
- Emu WPS: 0.10.0
- FlyingPigeon WPS: 1.4.1
- Finch WPS: 0.2.0
- Hummingbird WPS: 0.7.0
- Malleefowl WPS: 0.8.0

Maintained Birds with Buildout:

- Phoenix Web App: 0.9.0
- Twitcher WPS Proxy: 0.4.0

New Birds in the making:

- Kingfisher: https://github.com/bird-house/kingfisher
- Black Swan: https://github.com/bird-house/blackswan
- Eggshell: https://github.com/bird-house/eggshell
- Pelican: https://github.com/bird-house/pelican
- OWSLib extension for ESGF: https://github.com/bird-house/OWSLib-esgfwps

8.5 Washington (December 2018, v0.6.1)

Birdhouse was present at the AGU 2018 and ESGF Face to Face 2018 both in Washington D.C.

Highlighted Changes:

- Improved *Birdy WPSClient* as a pythonic library for WPS client with support for Jupyter Notebooks.
- Converted *Malleefowl* and *FlyingPigeon* to new deployment layout without buildout.
- New birds: *Finch* WPS for Climate Indicators and *Kingfisher* for Earth Observation Data Analysis.
- *FlyingPigeon* has been reborn as the *Curious Climate Explorer*. Most of its original functionallity has moved to other birds: *BlackSwan*, *Kingfisher* and *Finch*.

Released Birds:

- Ansible Playbook for PyWPS 0.2.0
- Cookiecutter Template for PyWPS 0.3.1
- Birdy WPS Client: 0.5.0
- Emu WPS: 0.9.1
- Hummingbird WPS: 0.6.1
- Malleefowl WPS: 0.7.0

Maintained Birds with Buildout:

- Phoenix Web App: 0.8.3
- Twitcher WPS Proxy: 0.3.8

New Birds in the making:

- FlyingPigeon (reborn): https://github.com/bird-house/flyingpigeon
- Kingfisher: https://github.com/bird-house/kingfisher
- Finch: https://github.com/bird-house/finch
- Black Swan: https://github.com/bird-house/blackswan
- Eggshell: https://github.com/bird-house/eggshell

8.6 Dar es Salaam (September 2018, v0.6.0)

Birdhouse was present at the FOSS4G 2018 in Dar es Salaam.

Highlighted Changes:

- Ansible playbook to install PyWPS applications.
- Skipped Buildout deployment . . . not all birds are converted yet.
- Updated Cookiecutter template for new deployment.
- Using PyWPS OpenDAP support.
- Initial version of Birdy native client.

Released Birds:

- Ansible Playbook for PyWPS 0.1.0
- Cookiecutter Template for PyWPS 0.3.0
- Birdy WPS Client: 0.4.0
- Emu WPS: 0.9.0
- Hummingbird WPS: 0.6.0

Maintained Birds with Buildout:

- Phoenix Web App: 0.8.2
- Twitcher WPS Proxy: 0.3.8
- Flyingpigeon WPS: 1.2.1
- Malleefowl WPS: 0.6.8

New Birds in the making:

- Black Swan: https://github.com/bird-house/blackswan
- Eggshell: https://github.com/bird-house/eggshell

8.7 Montréal (March 2018, v0.5.0)

We had a workshop in Montréal with CRIM and Ouranos.

Highlighted Changes:

- Birdhouse has a Logo :)
- A Cookiecutter template for Birdhouse WPS birds is available.
- A new WPS Bird Black Swan for extreme weather event assessments is started by LSCE, Paris. This bird is spawned off Flyingpigeon.
- A new Python library, Eggshell, is started to provide common base functionallity to WPS birds like Flyingpigeon and Black Swan.
- The Twitcher security proxy supports now X509 certificates for authentication to WPS services.

Released Birds:

- Phoenix 0.8.1

- Birdy 0.2.1
- Twitcher 0.3.7
- Flyingpigeon 1.2.0
- Hummingbird 0.5.7
- Malleefowl 0.6.7
- Emu 0.6.3

New Birds in the making:

- Black Swan: https://github.com/bird-house/blackswan
- Eggshell: https://github.com/bird-house/eggshell
- Cookiecutter: https://github.com/bird-house/cookiecutter-birdhouse

8.8 Bonn (August 2016, v0.4.0)

Birdhouse was present at the FOSS4G 2016 in Bonn.

Highlighted Changes:

- Leaflet map with time-dimension plugin.
- using twitcher security proxy.
- using conda environments for each birdhouse compartment.
- using ansible to deploy birdhouse compartments.
- added weather-regimes and analogs detection processes.
- allow upload of files to processes.
- updated Phoenix user interface.

8.9 Paris (October 2015, v0.3.0)

- updated documents on readthedocs
- OAuth2 used for login with GitHub, Ceda, ...
- LDAP support for login
- using ncWMS and adagucwms
- register and use Thredds catalogs as data source
- publish local netcdf files and Thredds catalogs to birdhouse Solr
- qualtiy check processes added (cfchecker, qa-dkrz)
- generation of docker images for each birdhouse component
- using dispel4py as workflow engine in Malleefowl
- using Celery task scheduler/queue to run and monitor WPS processes
- improved Phoenix web client
- using birdy wps command line client

8.10 Paris (September 2014, v0.2.0)

- Phoenix UI as WPS client with ESGF faceted search component and a wizard to chain WPS processes
- PyWPS based processing backend with supporting processes of Malleefowl
- WMS service (inculded in Thredds) for visualization of NetCDF files
- OGC CSW catalog service for published results and OGC WPS services
- ESGF data access with wget and OpenID
- Caching of accessed files from ESGF Nodes and Catalog Service
- WPS processes: cdo, climate-indices, ensemble data visualization, demo processes
- IPython environment for WPS processes
- initial unit tests for WPS processes
- Workflow engine Restflow for running processing chains. Currently there is only a simple workflow used: get data with wget - process data.
- Installation based on anaconda and buildout
- buildout recipes (birdhousebuilder) available on PyPI to simplify installation and configuration of multiple WPS server
- Monitoring of all used services (WPS, WMS, CSW, Phoenix) with supervisor
- moved source code and documentation to birdhouse on GitHub

8.11 Helsinki (May 2014, v0.1.2)

- presentation of birdhouse at EGI, Helsinki
- stabilized birdhouse and CSC processes
- updated documenation and tutorials

8.12 Vienna (April 2014, v0.1.1)

- presentation of birdhouse at EGU, Vienna.
- “quality check” workflow for CORDEX data.

8.13 Hamburg (December 2013, v0.1.0)

- First presentation of Birdhouse at GERICS (German Climate Service Center), Hamburg.

CHAPTER

NINE

COMMUNICATION

There are numerous ways to interact with the Birdhouse community, for example join the chat or follow our blog. Also we are present on several conferences where you can enjoy one of our good presentations.

9.1 Chat-room

The most easiest way to drop a line to the developers is our Gitter chat room. If you want to have a quick technical question to one of the developers, or just wants to follow the discussions, feel welcome to join.

9.2 Meetings

More complex and real discussions are done regularly in video conferences. Check out the information for upcoming birdhouse meetings. Here you also find the minutes of previews video conferences and feel welcome to join an upcoming one.

9.3 Blog-post

In the blog you can find interesting articles and information related to birdhouse in general. We also inform regularly abut the main steps forward in the software development that you can keep track on whats going on in the birdhouse. If you want to receive a notification of new articles follow birdhouse news on our blog:

- The IT Landscape for Climate Services
- Cyberinfrastructures for Sustainable Development

9.4 Newsletter

To be informed about the main progress in the birdhouse development as well as related information you can subscribe to our newsletter.

9.5 Wiki

The birdhouse wiki provides an area for supporting information that frequently changes and / or is outside the scope of the formal documentation.

CHAPTER

TEN

LICENSE

Birdhouse is Open Source and released under the Apache License, Version 2.0.

CHAPTER

ELEVEN

GLOSSARY

Anaconda

Anaconda Python distribution Python distribution for large-scale data processing, predictive analytics, and scientific computing. https://www.continuum.io/

Binstar

Anaconda Server

Anaconda cloud Binstar is a service that allows you to create and manage public and private *Anaconda* package repositories. https://anaconda.org/ https://docs.continuum.io/

Bokeh Bokeh is a Python interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of novel graphics in the style of D3.js, but also deliver this capability with high-performance interactivity over very large or streaming datasets. http://bokeh.pydata.org/en/latest/

Buildout *Buildout* is a Python-based build system for creating, assembling and deploying applications from multiple parts, some of which may be non-Python-based. It lets you create a buildout configuration and reproduce the same software later. http://www.buildout.org/en/latest/

CDO

Climate Data Operators *CDO* is a collection of command line Operators to manipulate and analyse Climate and NWP model Data. https://code.zmaw.de/projects/cdo

cfchecker The NetCDF Climate Forcast Conventions compliance checker. https://pypi.python.org/pypi/cfchecker

climate indice A climate index is a calculated value that can be used to describe the state and the changes in the climate system. http://icclim.readthedocs.io/en/latest/intro.html#climate-indices-label

CMIP5 In climatology, the Coupled Model Intercomparison Project (CMIP) is a framework and the analog of the Atmospheric Model Intercomparison Project (AMIP) for global coupled ocean-atmosphere general circulation models. https://en.wikipedia.org/wiki/Coupled_model_intercomparison_project

Conda The *conda* command is the primary interface for managing Anaconda installations. http://conda.pydata.org/docs/index.html

CORDEX The CORDEX vision is to advance and coordinate the science and application of regional climate downscaling through global partnerships. http://www.cordex.org/

COWS The COWS Web Processing Service (WPS) is a *generic* web service and offline processing tool developed within the Centre for Environmental Data Archival (CEDA). http://cows.ceda.ac.uk/cows_wps.html

CSW

Catalog Service *Catalog Service for the Web (CSW)*, sometimes seen as Catalog Service - Web, is a standard for exposing a catalogue of geospatial records in XML on the Internet (over HTTP). The catalogue is made up of records that describe geospatial data (e.g. KML), geospatial services (e.g. WMS), and related resources. https://en.wikipedia.org/wiki/Catalog_Service_for_the_Web

Dispel4py *Dispel4Py* is a Python library for describing abstract workflows for distributed data-intensive applications. http://www2.epcc.ed.ac.uk/~amrey/VERCE/Dispel4Py/index.html

Docker *Docker* - An open platform for distributed applications for developers and sysadmins. https://www.docker.com/

Docker Hub Docker Hub manages the lifecycle of distributed apps with cloud services for building and sharing containers and automating workflows. https://hub.docker.com/

Emu *Emu* is a Python package with some test proccess for *Web Processing Services*. http://emu.readthedocs.io/en/latest/

ESGF

Earth System Grid Federation An open source effort providing a robust, distributed data and computation platform, enabling world wide access to Peta/Exa-scale scientific data. http://esgf.llnl.gov/

GeoPython GitHub organisation of Python projects related to geospatial. https://geopython.github.io/

GeoServer GeoServer is an open source software server written in Java that allows users to share and edit geospatial data. http://docs.geoserver.org/stable/en/user/index.html

GitHub GitHub is a web-based Git repository hosting service. https://github.com/ https://en.wikipedia.org/wiki/GitHub

Gunicorn Gunicorn *Green Unicorn* is a Python WSGI HTTP Server for UNIX. http://gunicorn.org/

Homebrew The missing package manager for OS X. http://brew.sh/

ICCLIM

Indice Calculation CLIMate *ICCLIM* (Indice Calculation CLIMate) is a Python library for computing a number of *climate indices*. http://icclim.readthedocs.io/en/latest/

Linuxbrew Linuxbrew is a fork of Homebrew, the Mac OS package manager, for Linux. http://brew.sh/linuxbrew/

Malleefowl *Malleefowl* is a Python package to simplify the usage of *Web Processing Services*. http://malleefowl.readthedocs.io/en/latest/

NetCDF NetCDF (Network Common Data Form) is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data. https://en.wikipedia.org/wiki/NetCDF

Nginx nginx [engine x] is an HTTP and reverse proxy server. http://nginx.org/

ocgis

OpenClimateGIS *OpenClimateGIS* (OCGIS) is a Python package designed for geospatial manipulation, subsetting, computation, and translation of climate datasets stored in local *NetCDF* files or files served through *THREDDS* data servers. https://www.earthsystemcog.org/projects/openclimategis/ https://github.com/NCPP/ocgis

OGC

Open Geospatial Consortium The *Open Geospatial Consortium (OGC)* is an international voluntary consensus standards organization, originated in 1994. https://en.wikipedia.org/wiki/Open_Geospatial_Consortium, http://www.opengeospatial.org/standards/wps

OpenID OpenID (OID) is an open standard and decentralized protocol by the non-profit OpenID Foundation that allows users to be authenticated by certain co-operating sites (known as Relying Parties or RP) using a third party service. https://en.wikipedia.org/wiki/OpenID, http://openid.net/

OWSLib OWSLib is a Python package for client programming with *Open Geospatial Consortium* web service interface standards, and their related content models. OWSLib has *WPS* client library which is used in Birdhouse to access WPS services. http://geopython.github.io/OWSLib/, http://geopython.github.io/OWSLib/#wps

Phoenix Pyramid *Phoenix* is a web-application build with the Python web-framework pyramid. Phoenix has a user interface to make it easier to interact with *Web Processing Services*. http://pyramid-phoenix.readthedocs.io/en/latest

PyCSW pycsw is an *OGC* CSW server implementation written in Python. Started in 2010 (more formally announced in 2011), pycsw allows for the publishing and discovery of geospatial metadata, providing a standards-based metadata and catalogue component of spatial data infrastructures. http://pycsw.org/, https://github.com/geopython/pycsw

PyPi

Python Package Index The Python Package Index is a repository of software for the Python programming language. https://pypi.python.org/pypi

Pyramid Pyramid is a Python web framework. http://www.pylonsproject.org/

PyWPS *Python Web Processing Service* is an implementation of the *Web processing Service* standard from *Open Geospatial Consortium*. http://pywps.org/

RestFlow *RestFlow* is a dataflow programming language and runtime engine designed to make it easy for scientists to build and execute computational pipelines. https://github.com/restflow-org/restflow/wiki

Supervisor Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems. http://supervisord.org/

Taverna *Taverna* is an open source and domain-independent Workflow Management System – a suite of tools used to design and execute scientific workflows. http://www.taverna.org.uk/

TDS

THREDDS The THREDDS Data Server (TDS) is a web server that provides metadata and data access for scientific datasets, using a variety of remote data access protocols. http://www.unidata.ucar.edu/software/thredds/current/tds/

VisTrails *VisTrails* is an open-source scientific workflow and provenance management system that supports data exploration and visualization. http://www.vistrails.org/index.php/Main_Page

WMS

Web Mapping Service A Web Map Service (WMS) is a standard protocol for serving georeferenced map images over the Internet that are generated by a map server using data from a GIS database. https://en.wikipedia.org/wiki/Web_Map_Service

Workflow

Workflow Management System A workflow management system (WfMS) is a software system for the set-up, performance and monitoring of a defined sequence of tasks, arranged as a workflow. https://en.wikipedia.org/wiki/Workflow_management_system

WPS

Web Processing Service WPS is an open standard to search and run processes with a simple web-based interface. See: *WPS general usage*.

WSGI WSGI is an interface specification by which server and application communicate. http://wsgi.tutorial.codepoint.net/

x509 In cryptography, X.509 is an ITU-T standard for a public key infrastructure (PKI) and Privilege Management Infrastructure (PMI). https://en.wikipedia.org/wiki/X.509

XML-RPC It's a spec and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet. http://xmlrpc.scripting.com/default.html

BIBLIOGRAPHY

[MNV+17] Barend Mons, Cameron Neylon, Jan Velterop, Michel Dumontier, Luiz Olavo Bonino da Silva Santos, and Mark D Wilkinson. Cloudy, increasingly FAIR; revisiting the FAIR Data guiding principles for the European Open Science Cloud. *Information Services & Use*, 37(1):49–56, 2017. URL: https://content.iospress.com/articles/information-services-and-use/isu824, doi:10.3233/ISU-170824.

[SL17] Martina Stockhause and Michael Lautenschlager. CMIP6 data citation of evolving data. *Data Science Journal*, 16:30, 2017. doi:10.5334/dsj-2017-030.

[Tho10] Bejoy K Thomas. Participation in the Knowledge Society: the Free and Open Source Software (FOSS) movement compared with participatory development. *Development in Practice*, 20(2):270–276, 2010. URL: https://doi.org/10.1080/09614520903566509, doi:10.1080/09614520903566509.

[WLTK13] Tobias Weigel, Michael Lautenschlager, Frank Toussaint, and Stephan Kindermann. A framework for extended persistent identification of scientific assets. *Data Science Journal*, 12(March):10–22, 2013. doi:10.2481/dsj.12-036.

[WDA+16] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, Jildau Bouwman, Anthony J Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J G Gray, Paul Groth, Carole Goble, Jeffrey S Grethe, Jaap Heringa, Peter A C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J Lusher, Maryann E Martone, Albert Mons, Abel L Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1):160018, 2016. URL: https://doi.org/10.1038/sdata.2016.18, doi:10.1038/sdata.2016.18.

[ELH+18] C. Ehbrecht, T. Landry, N. Hempelmann, D. Huard, and S. Kindermann. Projects based on the web processing service framework birdhouse. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-4/W8:43–47, 2018. URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-4-W8/43/2018/, doi:10.5194/isprs-archives-XLII-4-W8-43-2018.

[HEAC+18] N. Hempelmann, C. Ehbrecht, C. Alvarez-Castro, P. Brockmann, W. Falk, J. Hoffmann, S. Kindermann, B. Koziol, C. Nangini, S. Radanovics, R. Vautard, and P. Yiou. Web processing service for climate impact and extreme weather event analyses. flyingpigeon (version 1.0). *Computers & Geosciences*, 110(Supplement C):65 – 72, 2018. URL: http://www.sciencedirect.com/science/article/pii/S0098300416302801, doi:https://doi.org/10.1016/j.cageo.2017.10.004.

[JGG+14] C. Jung, M. Gasthuber, A. Giesler, M. Hardt, J. Meyer, F. Rigoll, K. Schwarz, R. Stotzka, and A. Streit. Optimization of data life cycles. *Journal of Physics: Conference Series*, 513(3):032047, 2014. URL: http://stacks.iop.org/1742-6596/513/i=3/a=032047.

[JMS17] Christopher Jung, Jörg Meyer, and Achim Streit, editors. *Helmholtz Portfolio Theme Large-Scale Data Management and Analysis (LSDMA)*. KIT Scientific Publishing, Karlsruhe, 2017. ISBN 978-3-7315-0695-9. 46.12.02; LK 01. doi:10.5445/KSP/1000071931.

INDEX

V

W

X