Phoenix Documentation

Release 0.8

Birdhouse

Jan 25, 2018

Contents

Phoenix (the bird) *Phoenix is a long-lived bird that is cyclically regenerated or reborn.* (Wikipedia). [..]

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. The user interface gives you the possibility to *register Web Processing Services*. For these registered WPS services you can see which *Processes* they have available. You are provided with a form page to enter the parameters to *execute a process (job)*. You can *monitor the jobs* and see the results.

In the climate science community many analyses are using climate data in the NetCDF format. Phoenix uses the Malleefowl WPS which provides processes to access NetCDF files from the ESGF data archive. Malleefowl provides a workflow process to chain ESGF data retrieval with another WPS process which needs NetCDF data as input. Phoenix has a *Wizard* to collect the parameters to run such a workflow with a process of a registered WPS.

Phoenix should help developers of WPS processes to use their processes more conveniently, especially for feeding their processes with different data sources (like ESGF data archive). Phoenix is also used for demonstration of available WPS processes.

Phoenix has a more generic and technical user interface. To use Phoenix successfully you need to have some knowledge about WPS and the existing data archives. So, Phoenix might not become a good choice for scientists users who just want to run a specific analyses job. There are other climate portals available which address these users. But Phoenix should at least become *developer friendly*.

Phoenix is easy to install using the anaconda python distribution and buildout. So, Phoenix is not only available on production sites where it is close to data archives. You can also install it on your developer machine to make testing of your developed WPS processes easier and to present them to other people.

CHAPTER 1

Installation

This installation works on Linux 64-bit (Ubuntu 14.04, Centos 6, ...). It might still work on MacOSX but packages are updated only from time to time. Most of the dependencies come from Anaconda Python distribution system. Additional conda packages come from the Binstar channel Birdhouse. The installation is done with Buildout.

Phoenix uses WPS processes provided by Malleefowl. As a requiste you should install a local Malleefowl WPS (this will become part of the Phoenix installer). Alternatively you could configure the WPS URL of a running Malleefowl WPS instance in the Phoenix `custom.cfg`.

To install Malleefowl follow the instructions given in the Malleefowl documentation. In short:

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

Now start with downloading Phoenix with sources from github:

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

For install options run `make help` and read the documention for the Makefile.

Before installation you *need* to create a password for the local `phoenix` user which is used to login to the Phoenix web application:

```
$ make passwd
Generate Phoenix password ...
Enter a password with at least 8 characters.
Enter password:
Verify password:

Run 'make install restart' to activate this password.
```

Optionally take a look at `custom.cfg` and make additional changes. When you're finished, run `make clean install` to install Phoenix:

```
$ make clean install
```

You always have to rerun `make update` after making changes in custom.cfg.

After successful installation you need to start the services. All installed files (config etc ...) are below the conda environment `birdhouse` which is by default in your home directory `~/.conda/envs/birdhouse`. Now, start the services:

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

Phoenix web application is available on *http://localhost:8081*.

Check the log file for errors:

```
$ tail -f  ~/birdhouse/var/log/supervisor/phoenix.log
$ tail -f  ~/birdhouse/var/log/supervisor/celery.log
```

1.1 Run Docker

Set the `HOSTNAME` environment variable (not `localhost`) and run `docker-compose`:

```
HOSTNAME=phoenix HTTP_PORT=8081 HTTPS_PORT=8443 SUPERVISOR_PORT=9001 docker-compose up
```

CHAPTER 2

Configuration

You can configure Phoenix by editing `custom.cfg` in the Phoenix source folder:

```
$ cd pyramid-phoenix
$ vim custom.cfg
$ cat custom.cfg
```

```
[settings]
hostname = localhost
http-port = 8081
https-port = 8443
log-level = INFO
# run 'make passwd' and to generate password hash
phoenix-password = sha256:######################
esgf-search-url = http://example.org/esg-search
wps-url = http://localhost:8091/wps
# register at github: https://github.com/settings/applications/new
github-consumer-key = ########################
github-consumer-secret = ############################
```

By default Phoenix runs on localhost. The HTTP port 8081 is redirected to the HTTPS port 8443. If you want to use a different hostname/port then edit the default values in `custom.cfg`:

```
[settings]
hostname = localhost
http-port = 8081
https-port = 8443
```

To be able to login with the `phoenix` admin user you need to create a password. For this run:

```
$ make passwd
```

To activate the GitHub login for external users you need to configure a GitHub application key for your Phoenix web application:

```
[settings]
# register at github:
github-consumer-key = ######################
github-consumer-secret = ##########################
```

See the GitHub Settings on how to generate the application key for Phoenix.

If you want to use a different Malleefowl WPS service then change the `wps-url` value:

```
[settings]
wps-url = http://localhost:8091/wps
```

If you want to use a differnet ESGF index service then change the `esgf-search-url` value:

```
[settings]
esgf-search-url = http://example.org/esg-search
```

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

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

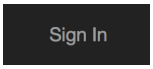
CHAPTER 3

User Guide

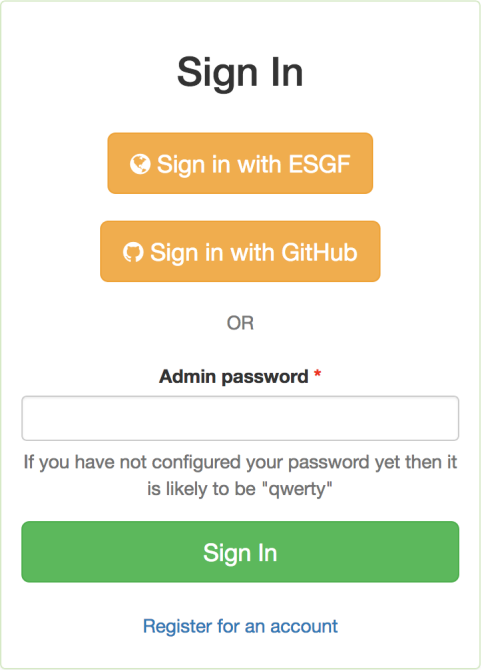
The user guide explains how to use the Phoenix web application to interact with Web Processing Services.

3.1 Login

Press the `Sign in` button in the upper right corner.



The login page offers you several options to login to Phoenix.



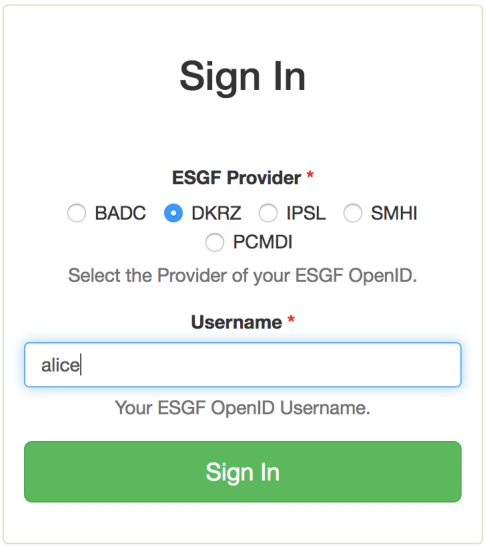
You can login using your ESGF OpenID or your GitHub account. If you login for the first time your account needs to be activated by an administrator.

If you are Phoenix admin you can also enter the admin password here.

ESGF OpenID

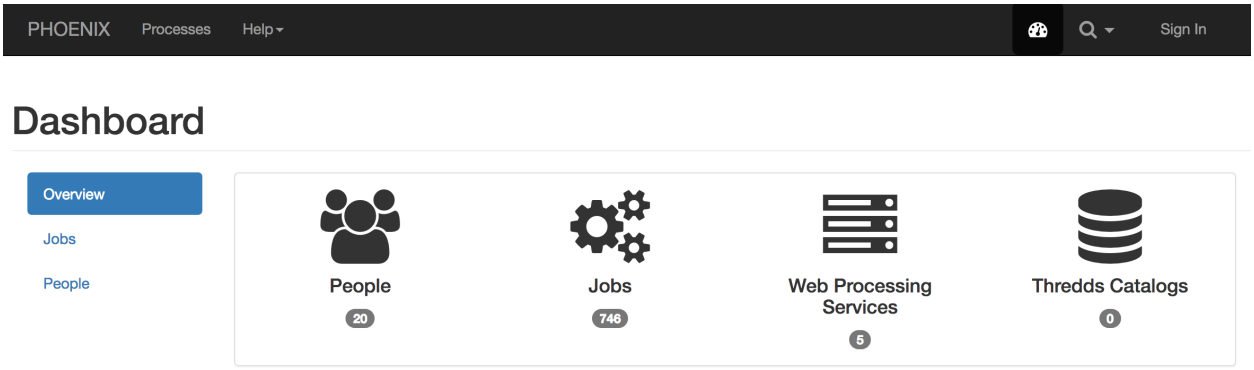
You can use an ESGF OpenID. The ESGF OpenID is used later to access files from ESGF. Make sure, that you have a valid ESGF OpenID of one of the ESGF Providers (for example DKRZ) and that you are able to download a datafile (you need to register for CMIP5 and CORDEX).

Enter the account name of your ESGF OpenID and choose the according ESGF OpenID provider (by default this is DKRZ).



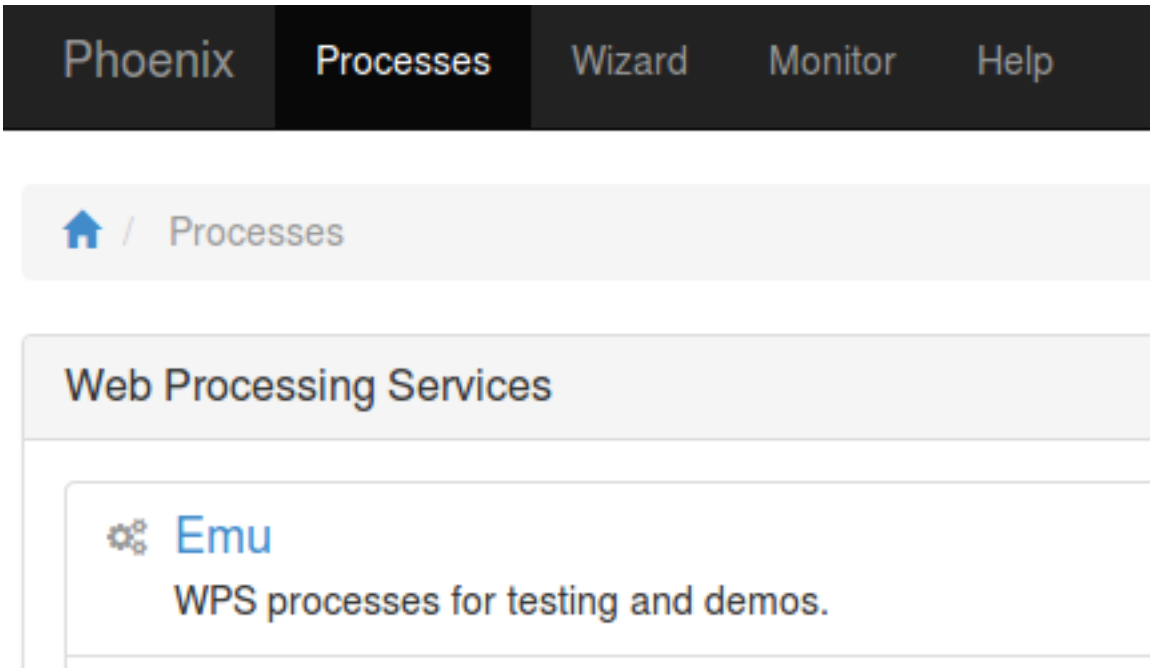
3.2 Dashboard

The dashboard shows some statistics about jobs and users.

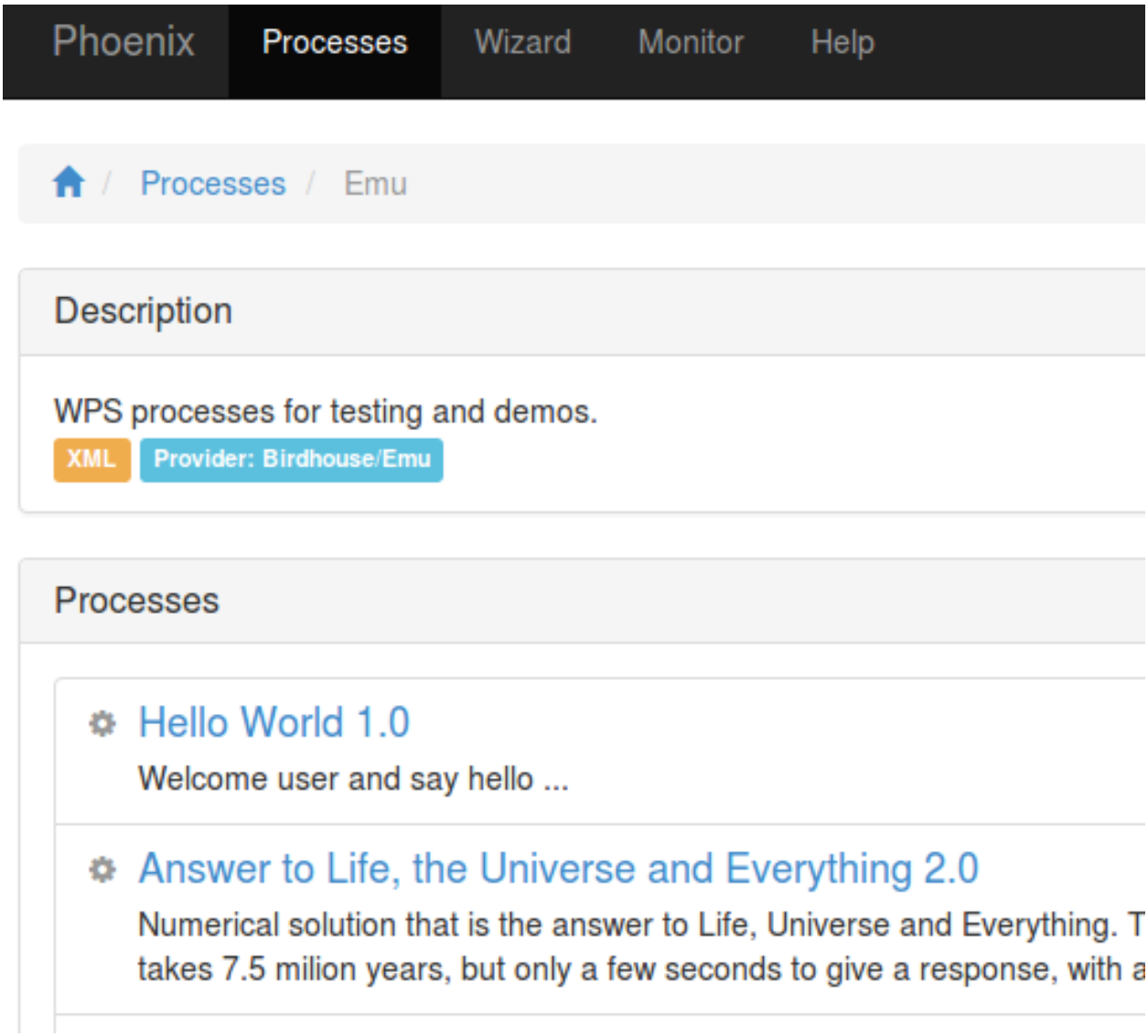


3.3 Processes

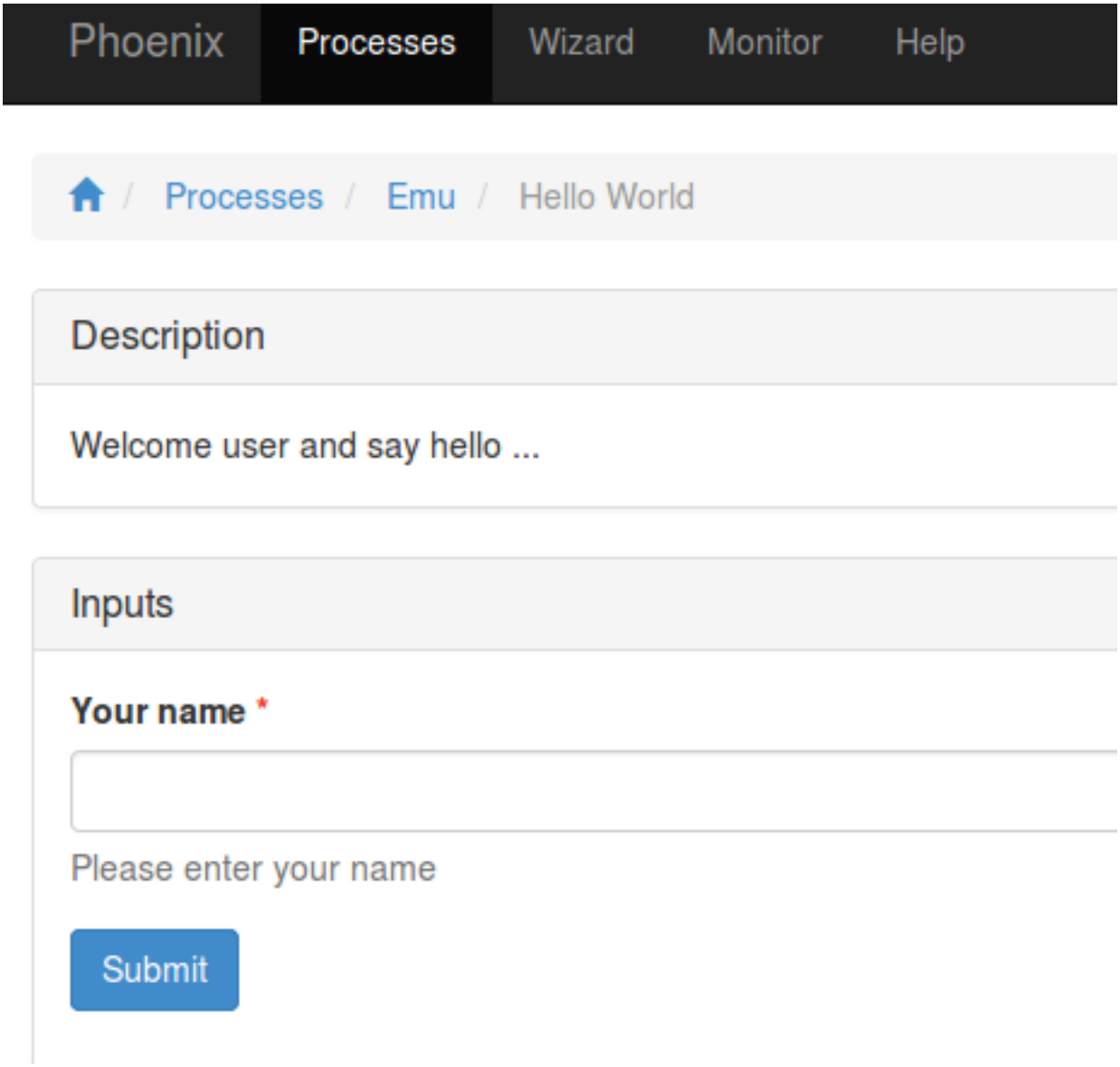
When you have registered WPS services you can run a process. Go to the `Processes` tab.



Choose one of your registered WPS services. You will get a list of available processes (WPS `GetCapabilities` request).



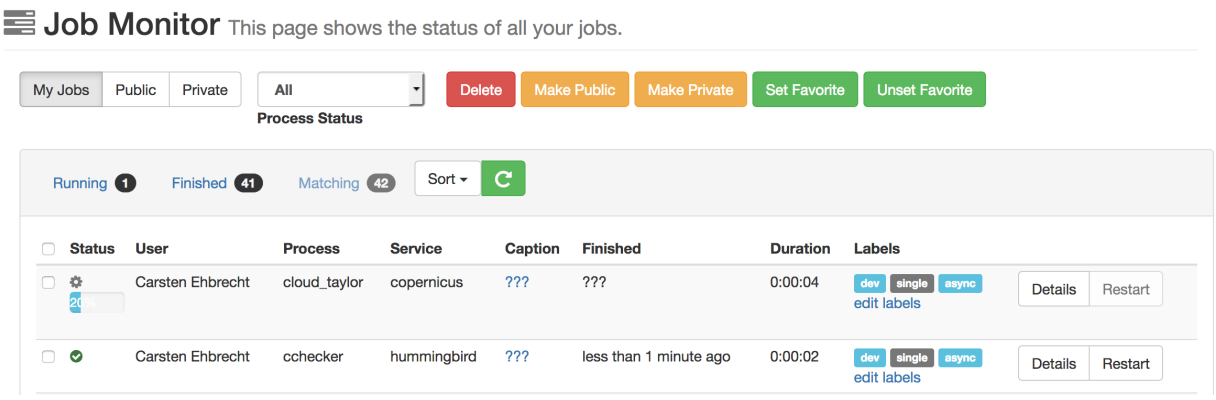
Choose one of these processes by using the `Execute` button. In case of Emu you may try the `Hello World` process. You will then be prompted to enter your username:



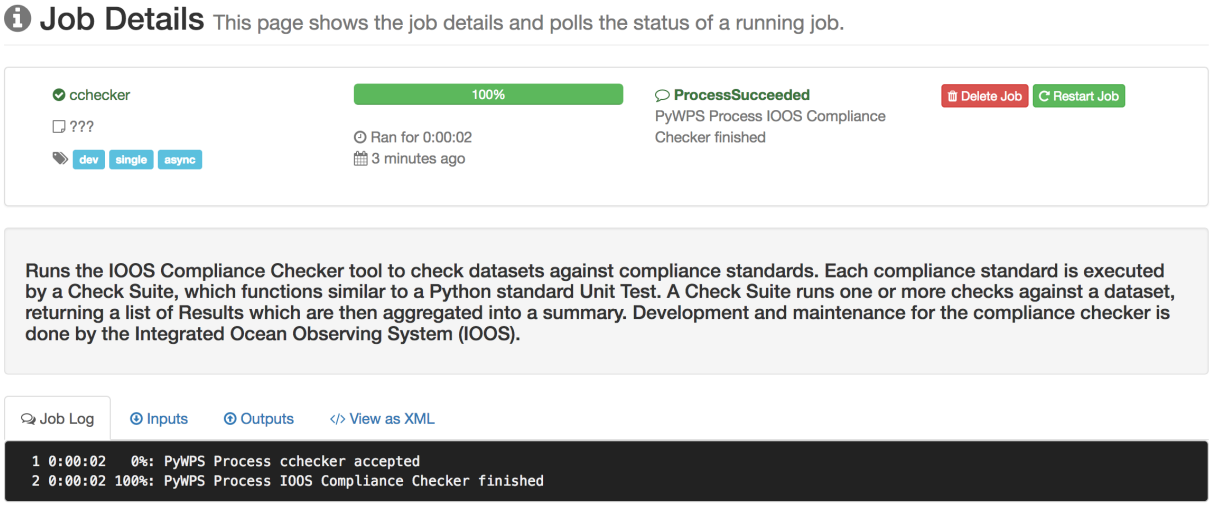
Press the `Submit` button. When the process is submitted you will be shown your job list in `Monitor`.

3.4 Monitor

In `Monitor` all your running or finished jobs are listed. The list shows the status and progress of your jobs.



When a job has finished with success you can see the results by clicking the `Details` button.

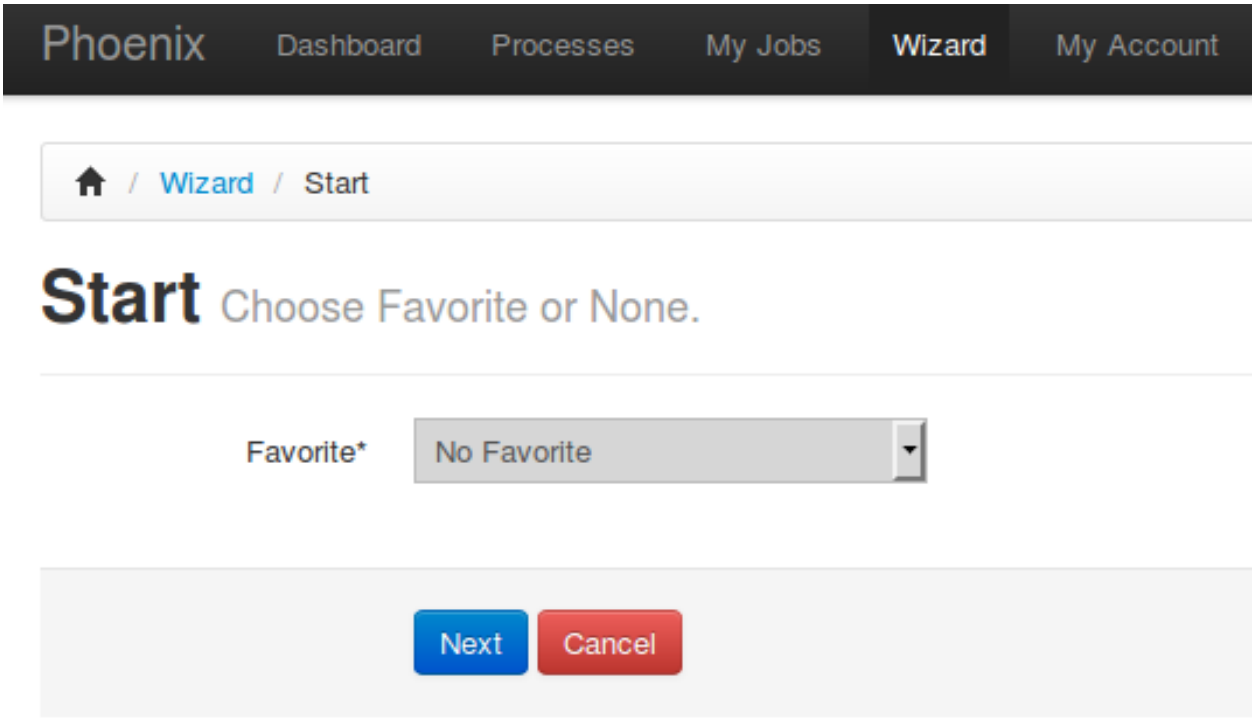


If the result has a document (XML, text, NetCDF, . . .) you can view or download this document with the `Download` button.

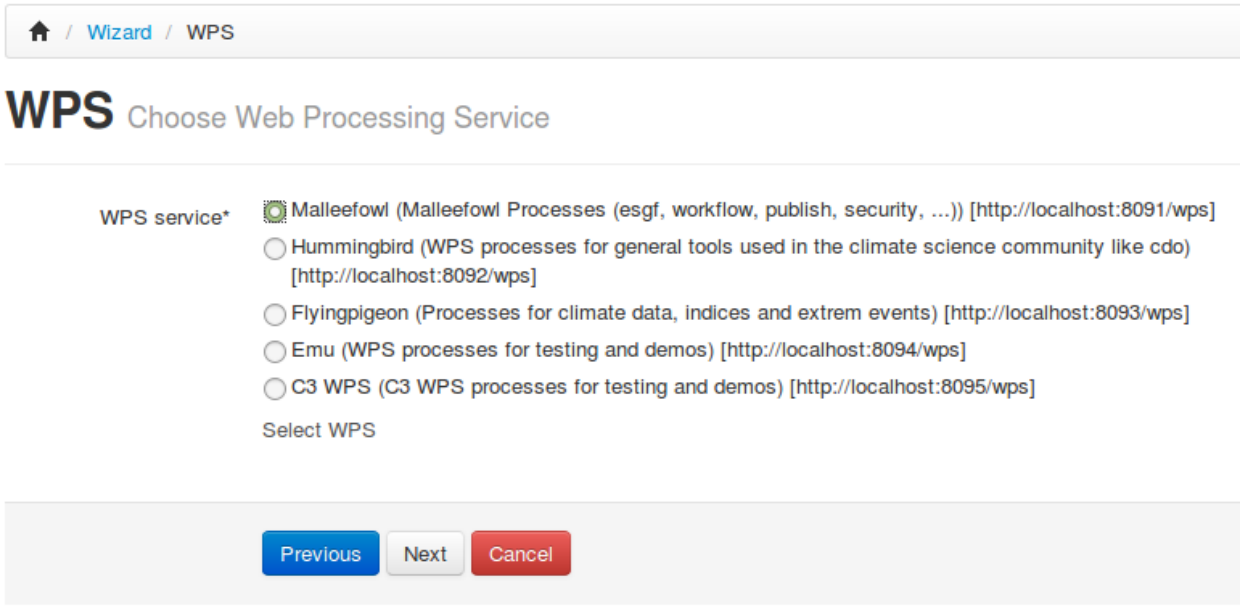
3.5 Wizard

The wizard is used to chain WPS processes and to collect the input parameters for the processes. Currently the wizard chains a user WPS process with a WPS process to retrieve ESGF data. The chained processes are run with a workflow management system which is available as WPS process in Malleefowl.

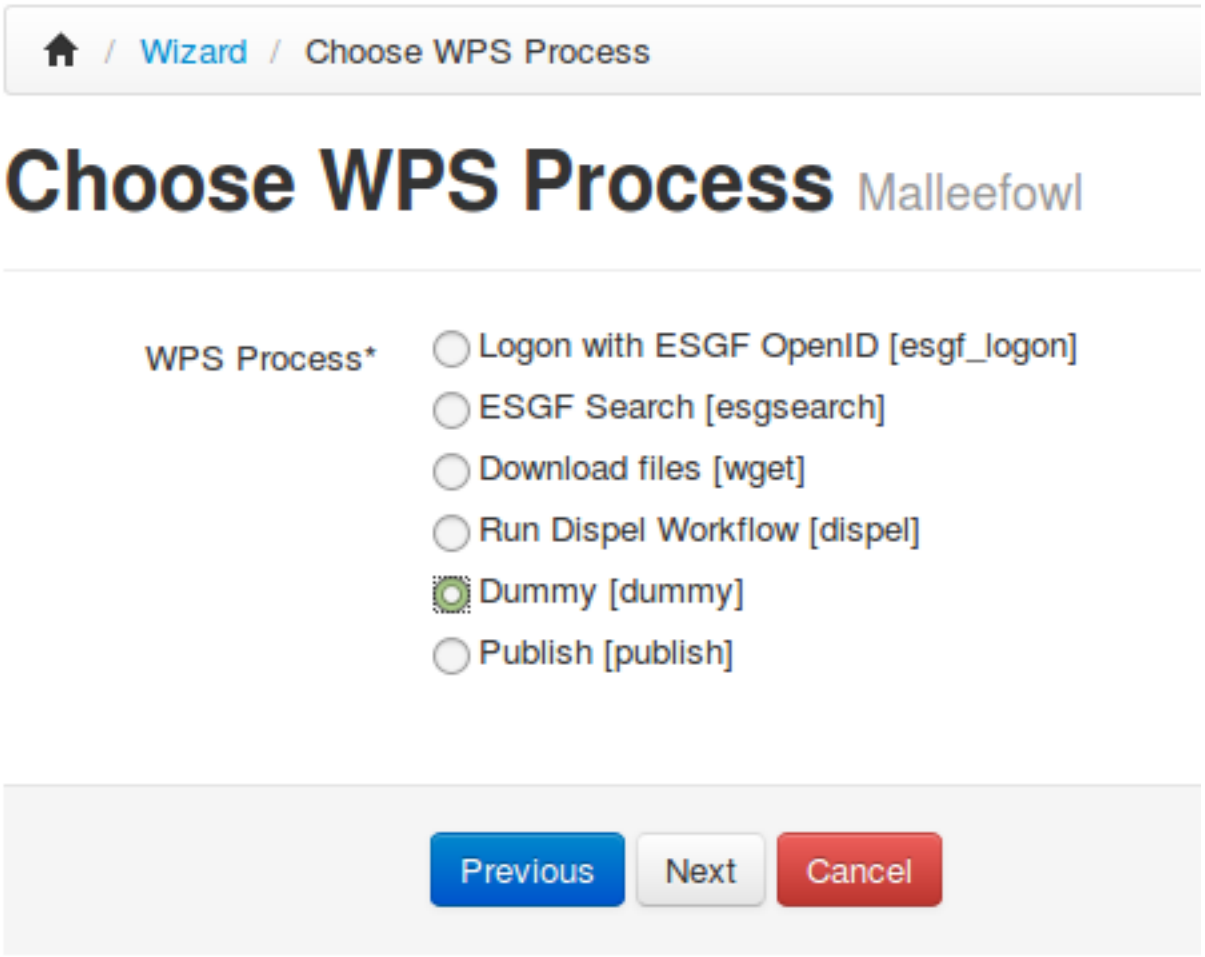
Go to the `Wizard` tab. Enter the appropiate parameters and use `Next` to get to the next wizard page.



You need to choose a WPS service (e.a. Malleefowl).

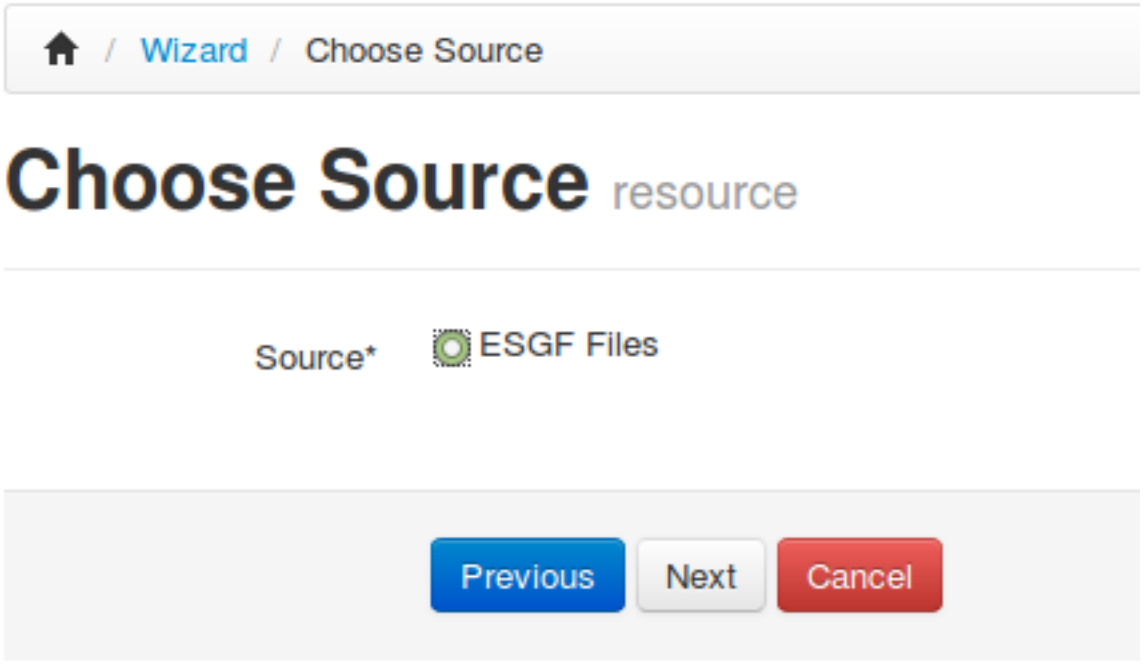


Choose a process (in case of Malleefowl only `Dummy`).

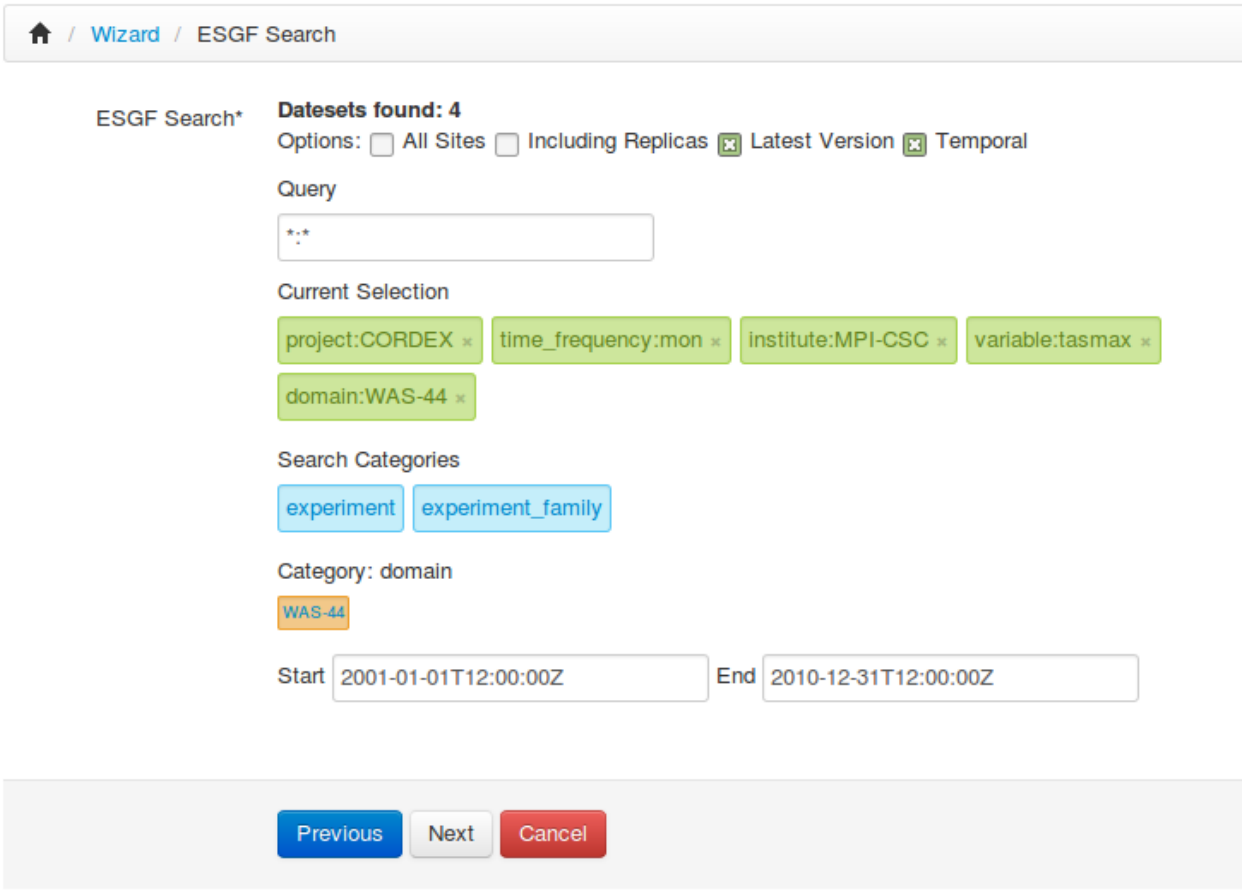


Select the input parameter of the choosen process (mime-type application/netcdf).

Select the input source (ESGF).

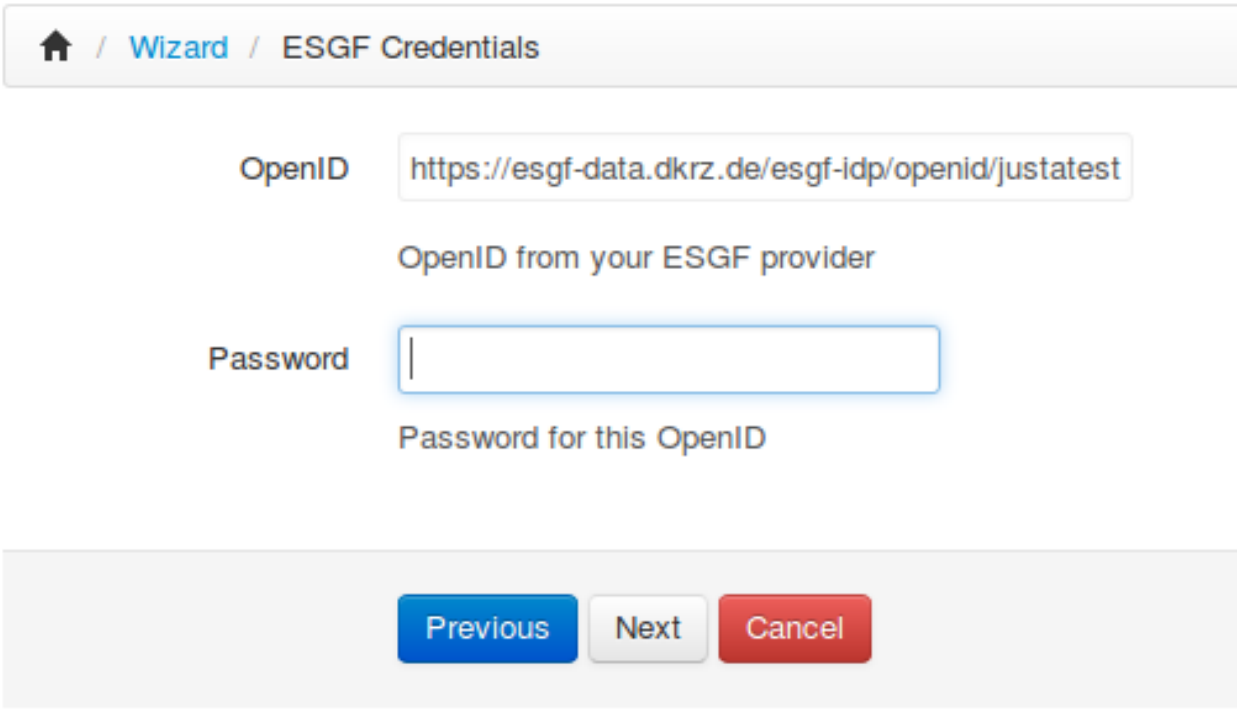


Select an ESGF dataset (select categorie (blue) and values of this category (orange), current selection (green)).

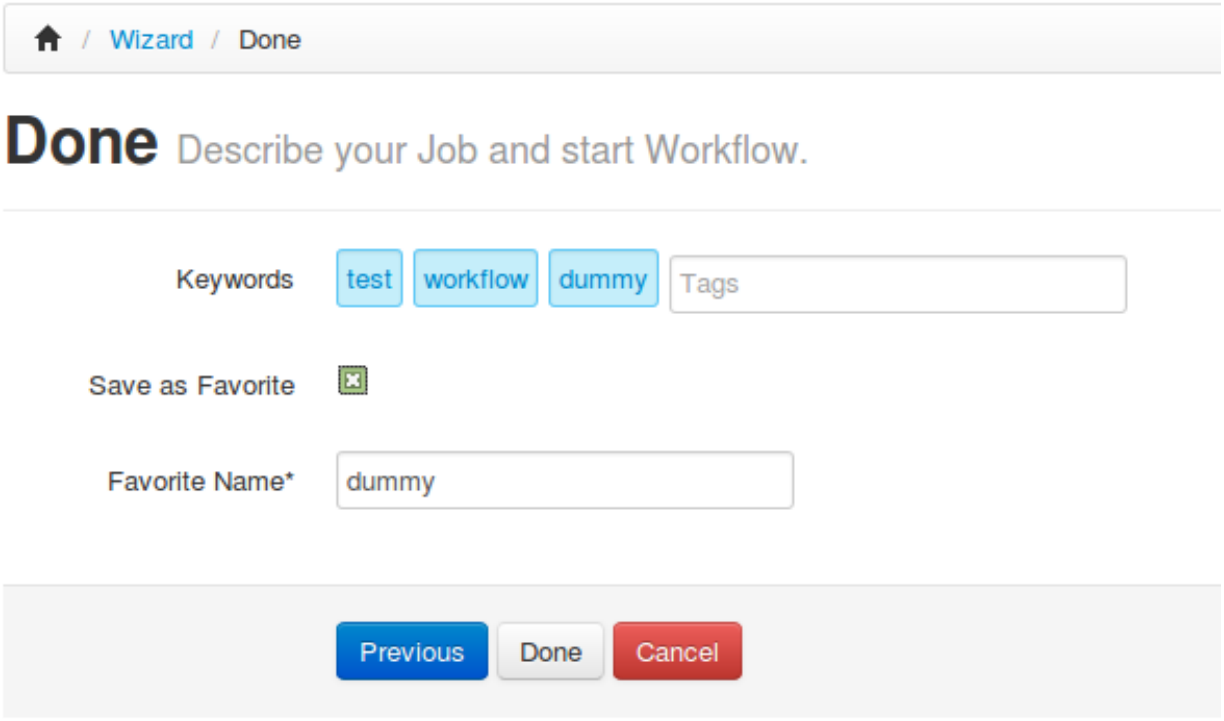


Please select **only one Dataset**!

You will be prompted for your password of your OpenID if your certificate is not valid anymore.

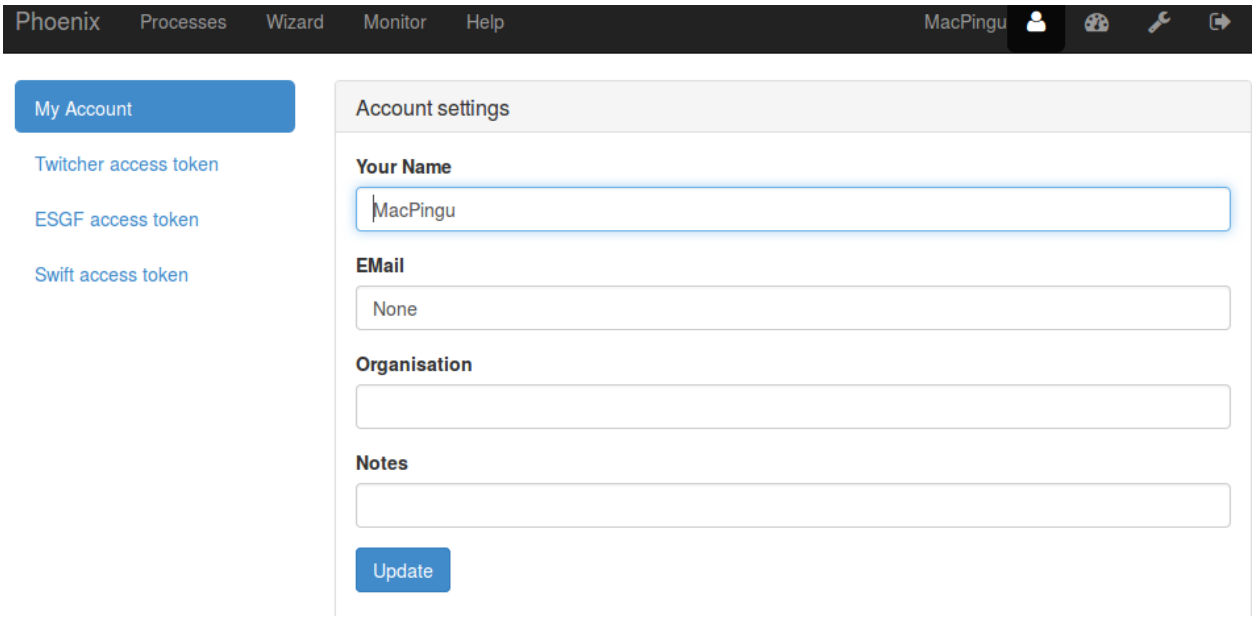


On the final page you can enter some keywords for your process and mark it as favorite (when using a favorite you don't need to enter all parameters again). Press `Done` and the job will be started and shown in your job list `My Jobs`.

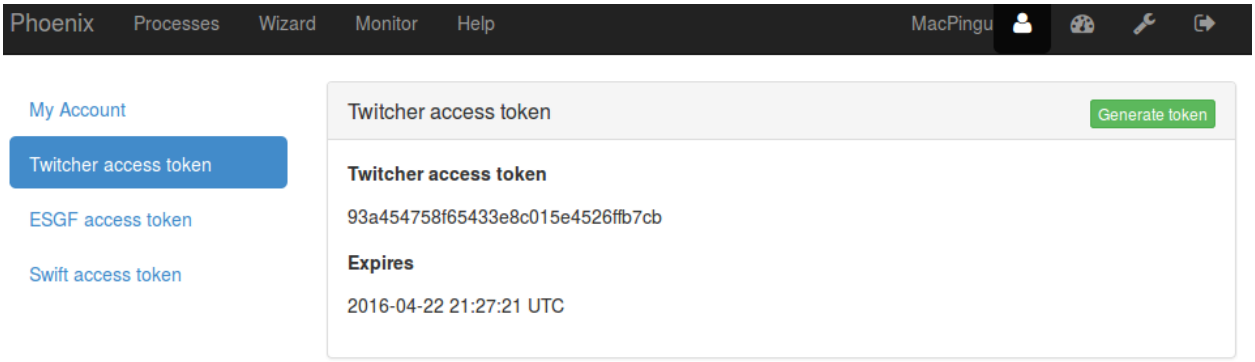


3.6 My Account

In `My Account` you can change your user settings (user name, organisation, openid, . . .).



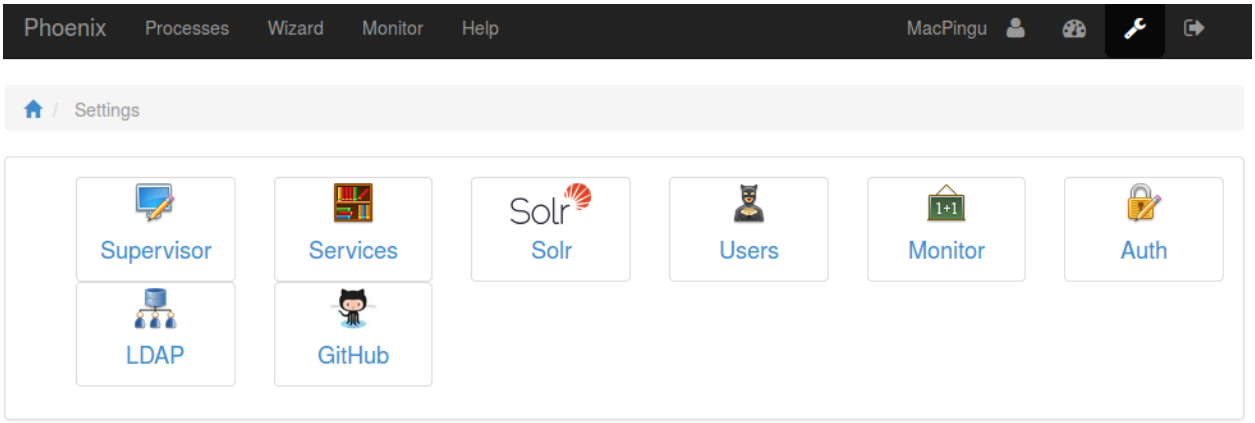
You can also see your current Twitcher access token which you can use to access a registered WPS service directly.



See the Twitcher Tutorial on how to use the token to access a WPS service.

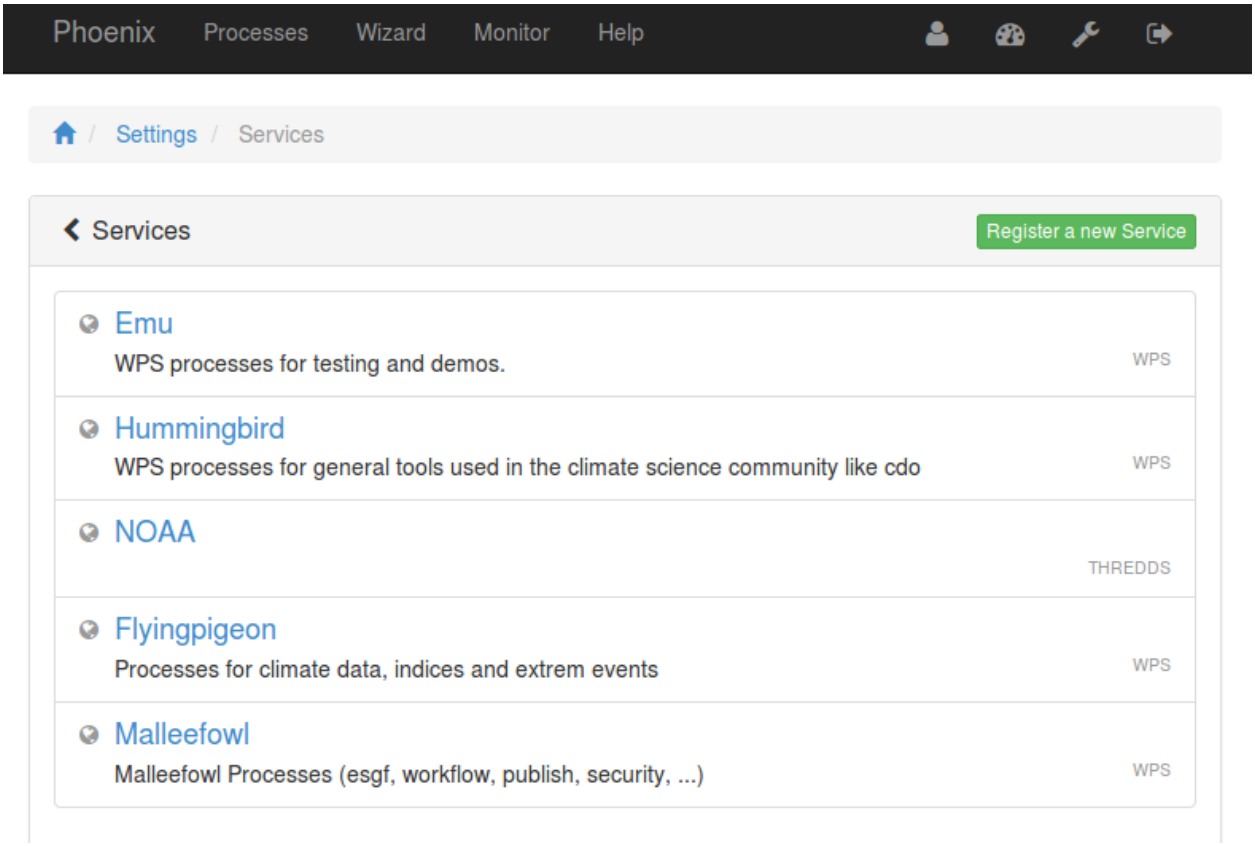
3.7 Settings (admins only)

When you are logged-in as admin user you have the `Settings` page. Here you can make administrative changes and monitor services.



3.7.1 Register a WPS or Thredds service

Open the `Settings/Services` page. Here you can see which services are registered in the catalog service (we are using PyCSW). All theses services are known and useable by Phoenix.

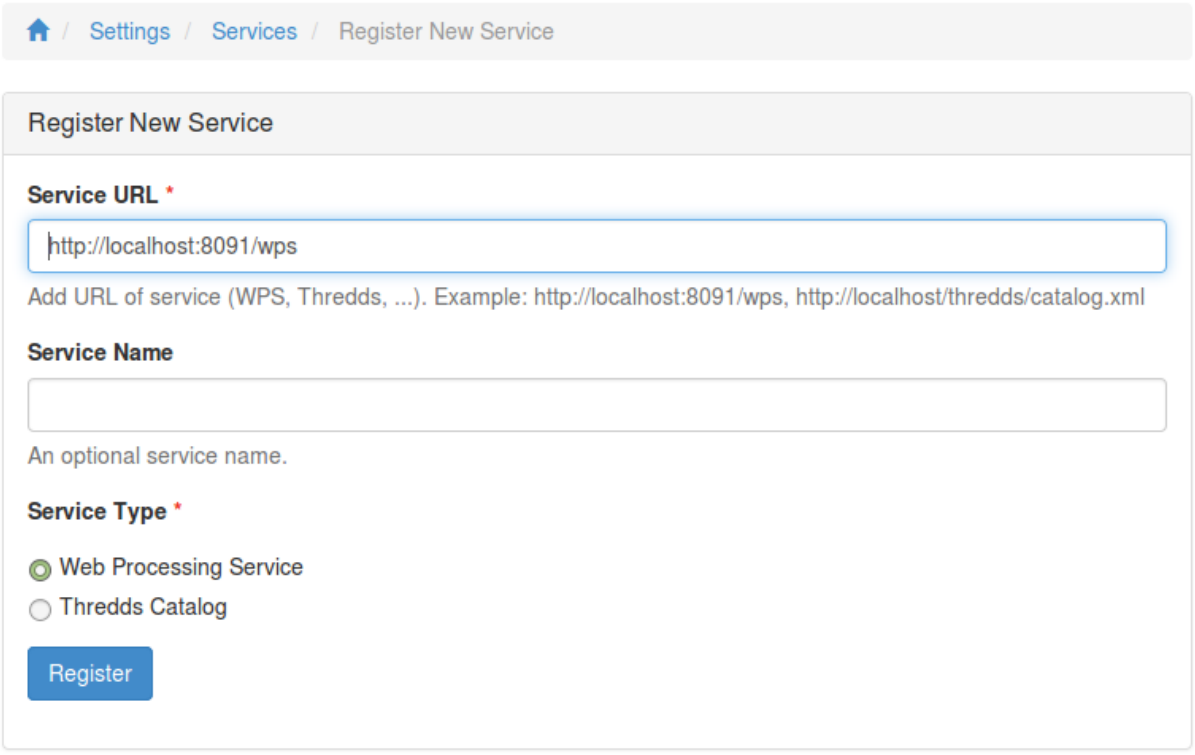


To add a new WPS service, press the `Register a new Service` button and enter the WPS URL in the field `Service URL`:

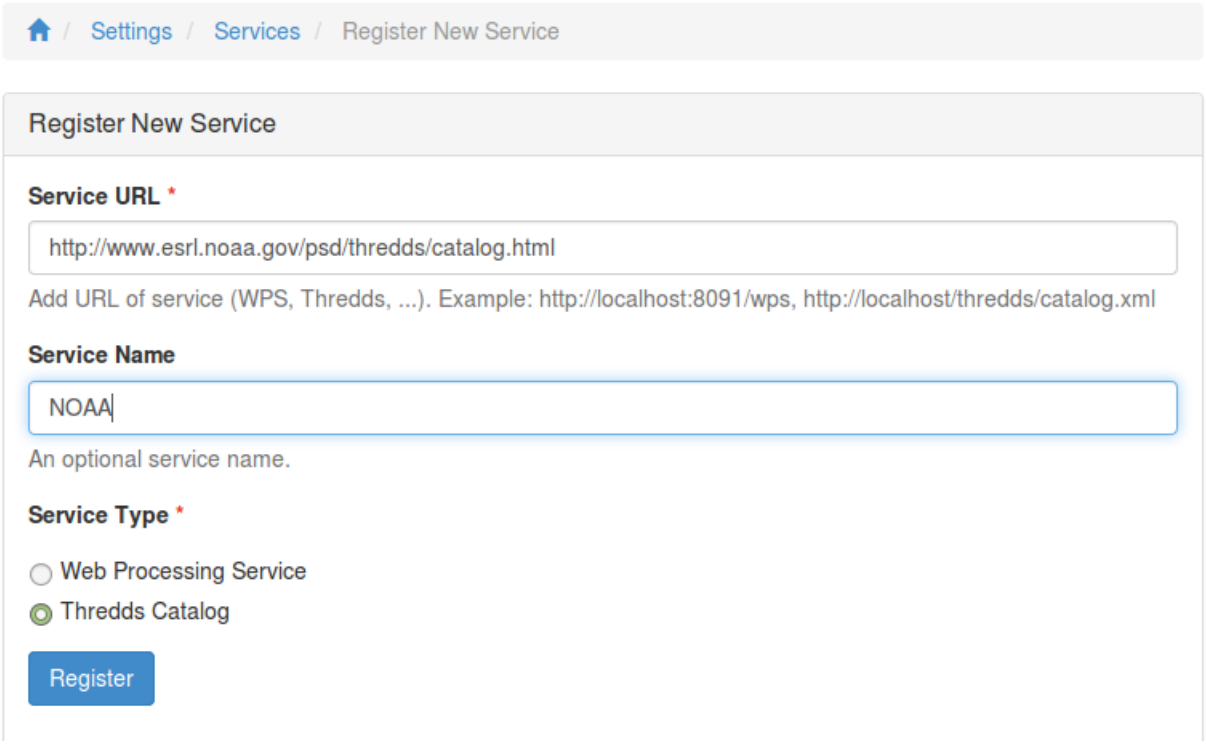
- hummingbird: http://localhost:8092/wps
- flyingpigeon: http://localhost:8093/wps
- emu: http://localhost:8094/wps

For example, to register Malleefowl WPS:

http://localhost:8091/wps



To add a new Thredds service press the `Register a new Service` button again, enter the Thredds URL and choose `Thredds Catalog` as service type.

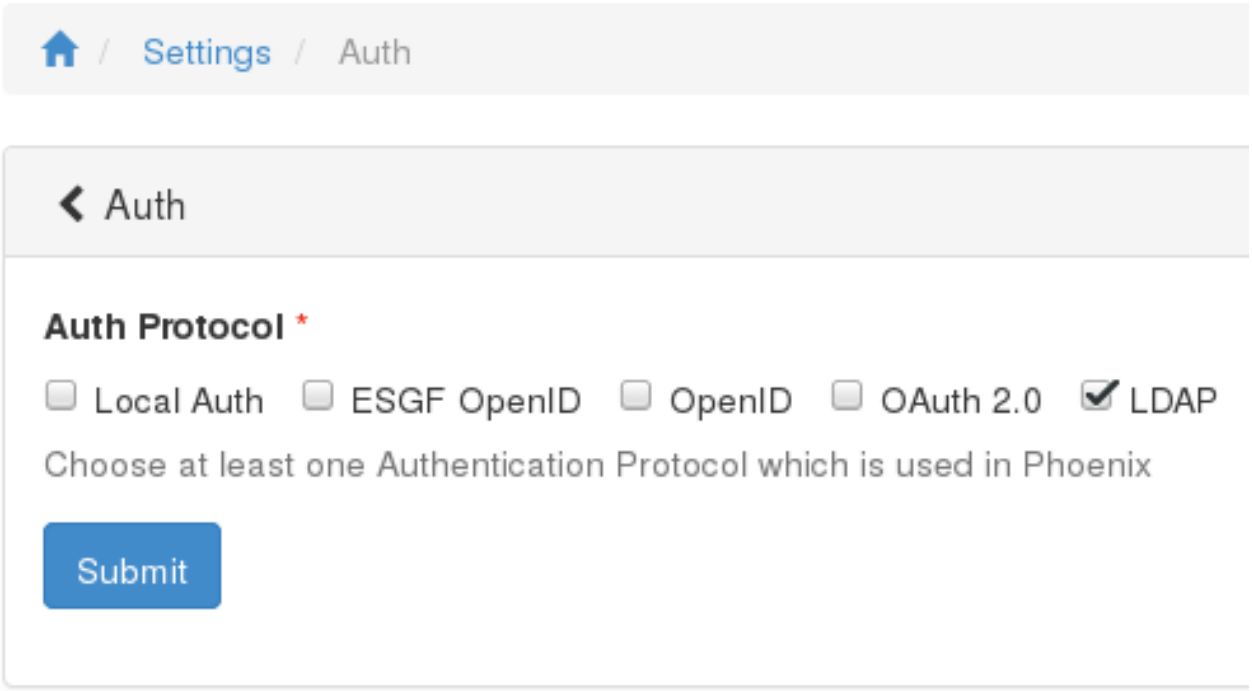


3.7.2 Activate Users

Open the `Settings/Users` page. Here you activate/deactivate users and also remove them. When a user has registerd to the Phoenix web application the user needs to be activated before the user can login.

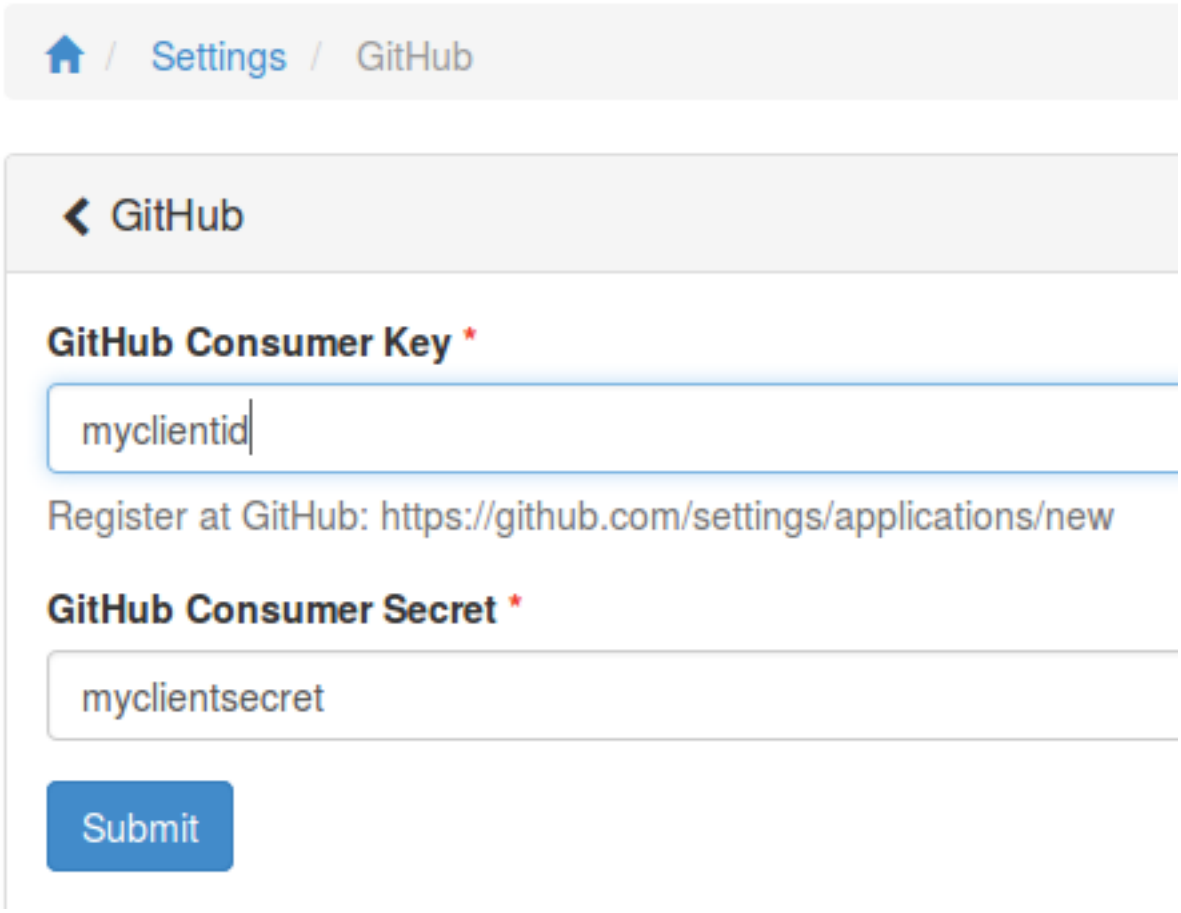
3.7.3 Choose Authentication Protocol

Open the `Settings/Auth` page. Here you can choose the different authentication protocols (OpenID, LDAP, ...) which users can use on the login page. `Local Auth` enables the local admin account whose password is set in `custom.cfg` in your Phoenix installation.



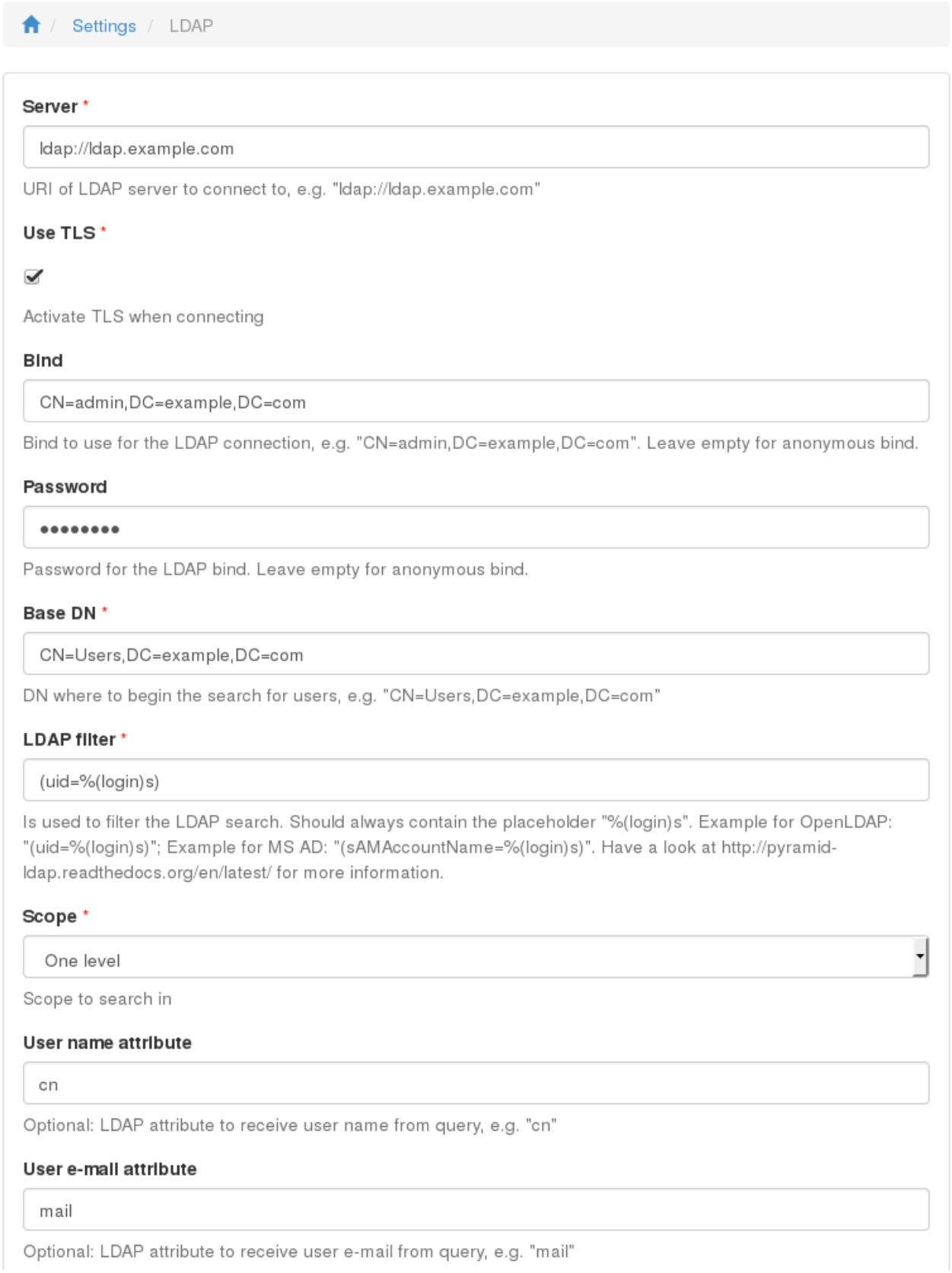
3.7.4 GitHub Support

You can use GitHub accounts to login to Phoenix. GitHub uses OAuth2. First you need to register your Phoenix application at GitHub. Then go to `Settings/GitHub` in your Phoenix application and enter the `GitHub Consumer Key/Client ID` and `GitHub Consumer Secret/Client Secret`:



3.7.5 LDAP Support

Basic support for authentication via LDAP has been added recently. To enable LDAP login for your environment, login with your admin account, navigate to `Settings/LDAP` and configure Phoenix to match your LDAP environment.

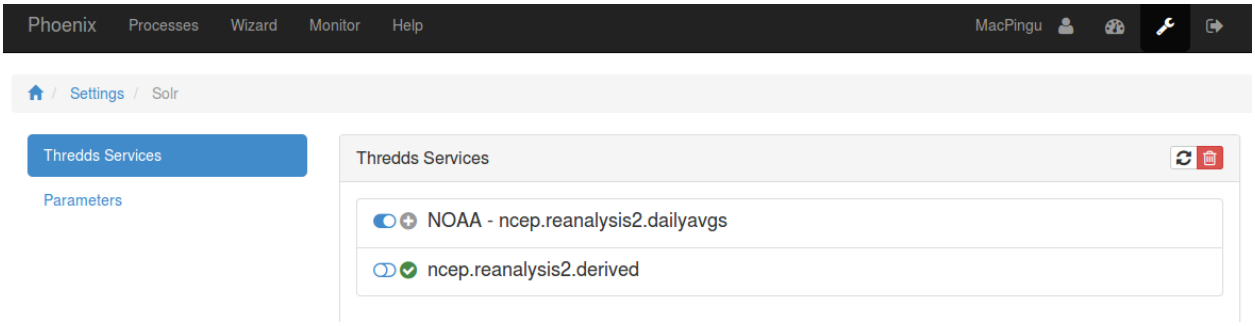


There is no support for LDAP authorization yet. Use the `Settings/Users` backend to manage the access privileges

for your users. There will be an entry for each user that has been logged in once before.

3.7.6 Solr

You can publish the datasets of a registered Thredds service to a Solr index server. The Solr server is setup with the Phoenix installation.



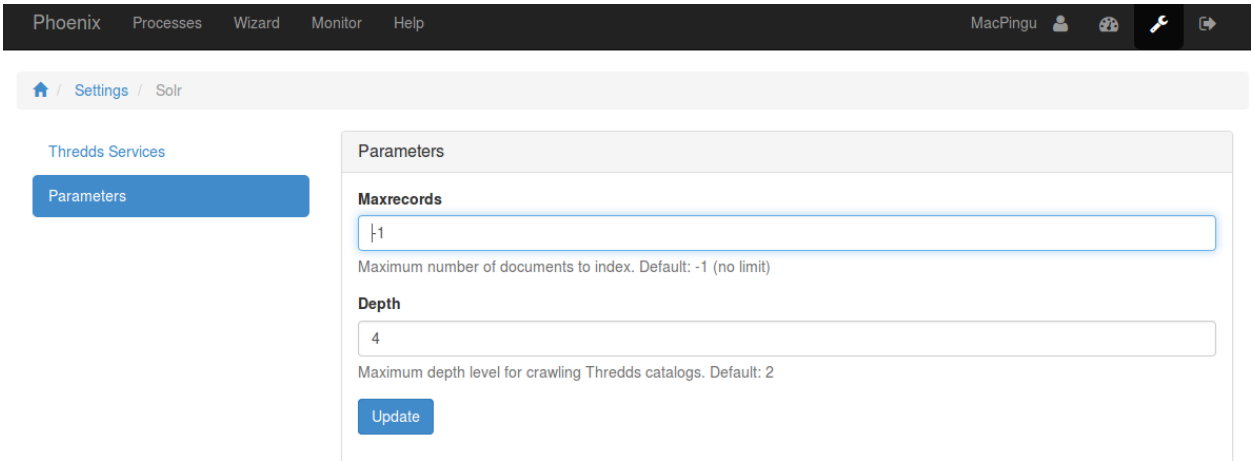
Use the toggle button on the left side of the Thredds service name to activate the publishing. Publishing takes some time. Use the reload button to update the status. The Solr search can then be used in the `Wizard` to select input files.

To clear the whole Solr index use the trash button.

The publisher has two parameters.

maxrecords Maximum number of datasets that will be published. Use -1 for unlimited.

depth The maximum depth level when crawling Thredds catalogs. Default is 2.



CHAPTER 4

Tutorial

In the following tutorial will guide you though the first steps to get famillliar with Phoenix.

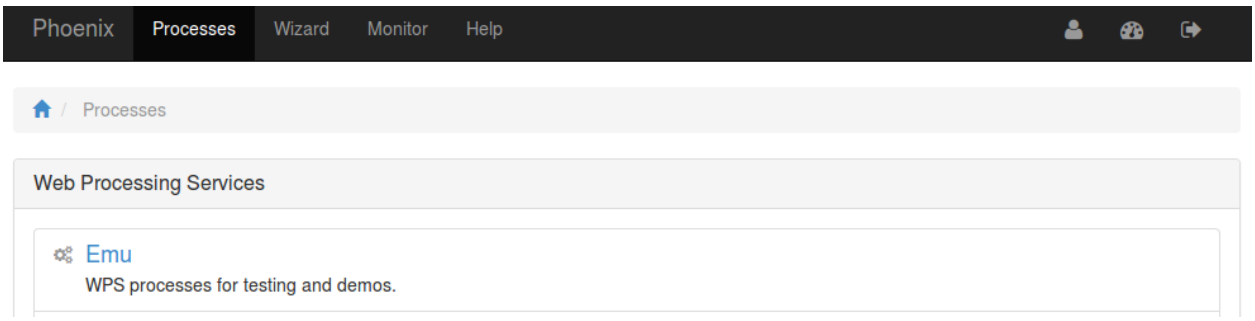
4.1 Hello World

First you need to login. Please follow the login instructions in the *user guide*.

- *Select Emu WPS Service*
- *Choose Hello World Process*
- *Enter Process Parameters*
- *Monitor running Job*
- *Display the outputs*

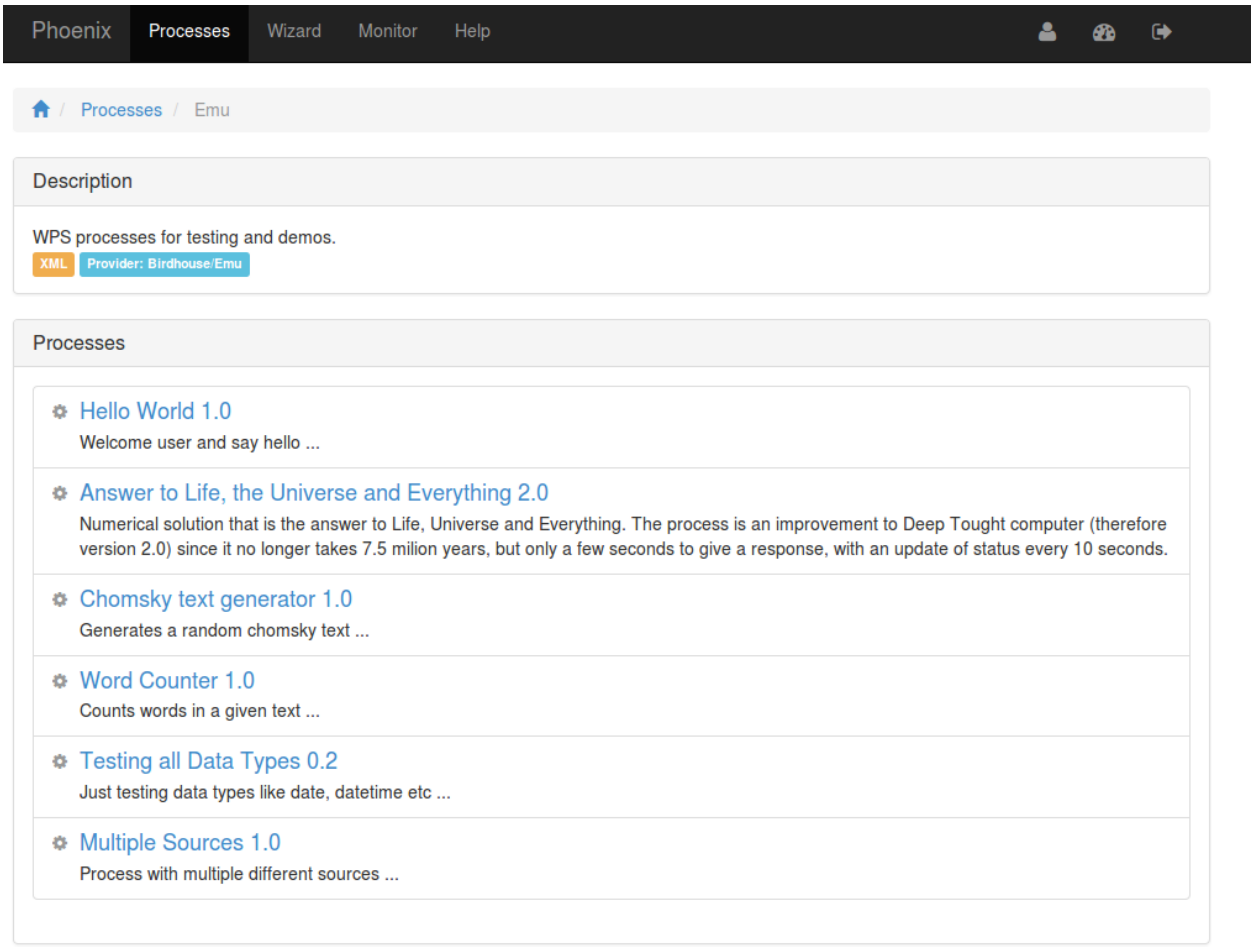
4.1.1 Select Emu WPS Service

For this example choose the Emu WPS service which has test processes. For this go to the `Processes` tab.



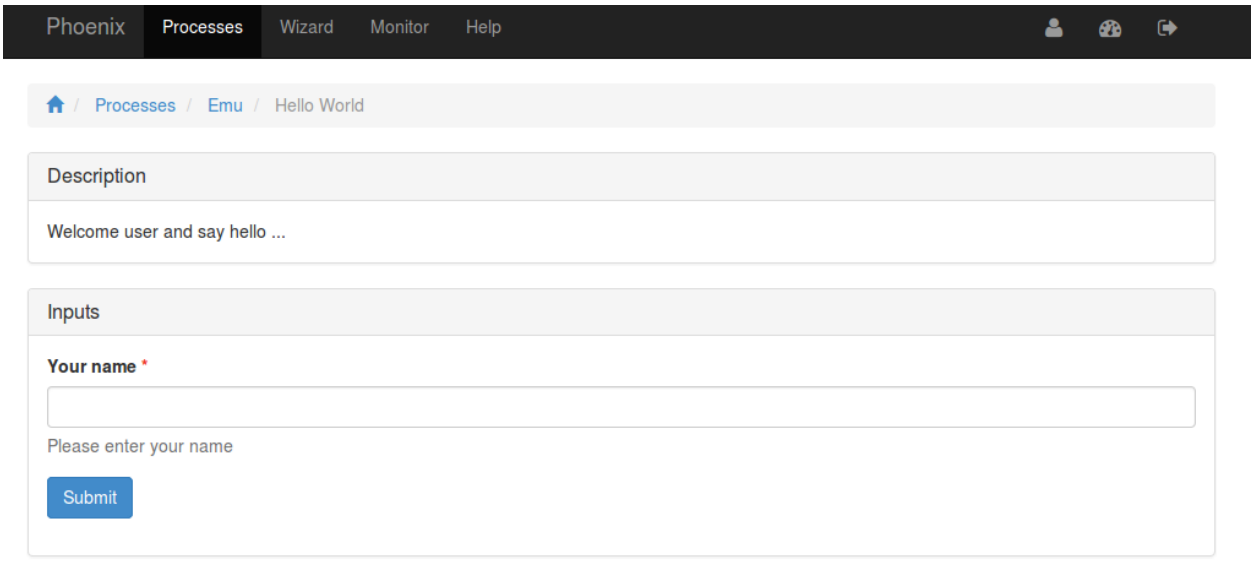
4.1.2 Choose Hello World Process

With clicking on *Emu* you will get the list of available processes in Emu.



4.1.3 Enter Process Parameters

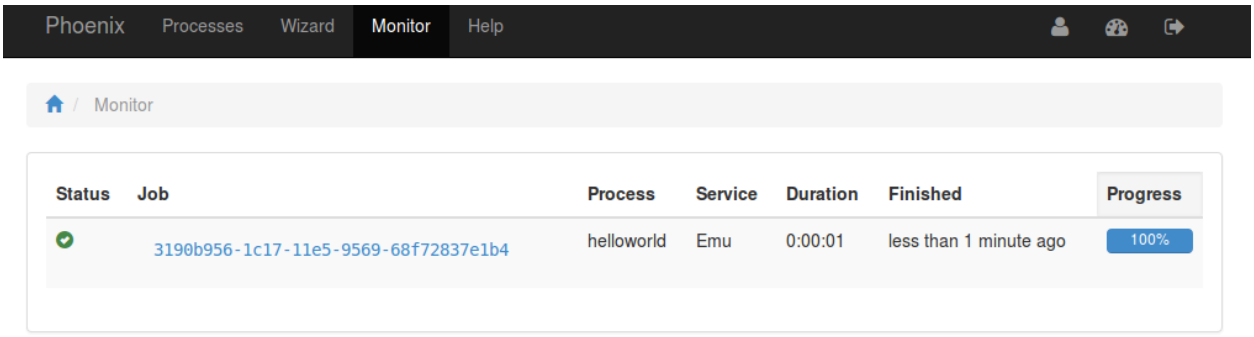
Click on *Hello World* and you will get a form to enter the process parameter:



Enter your name and click `submit`.

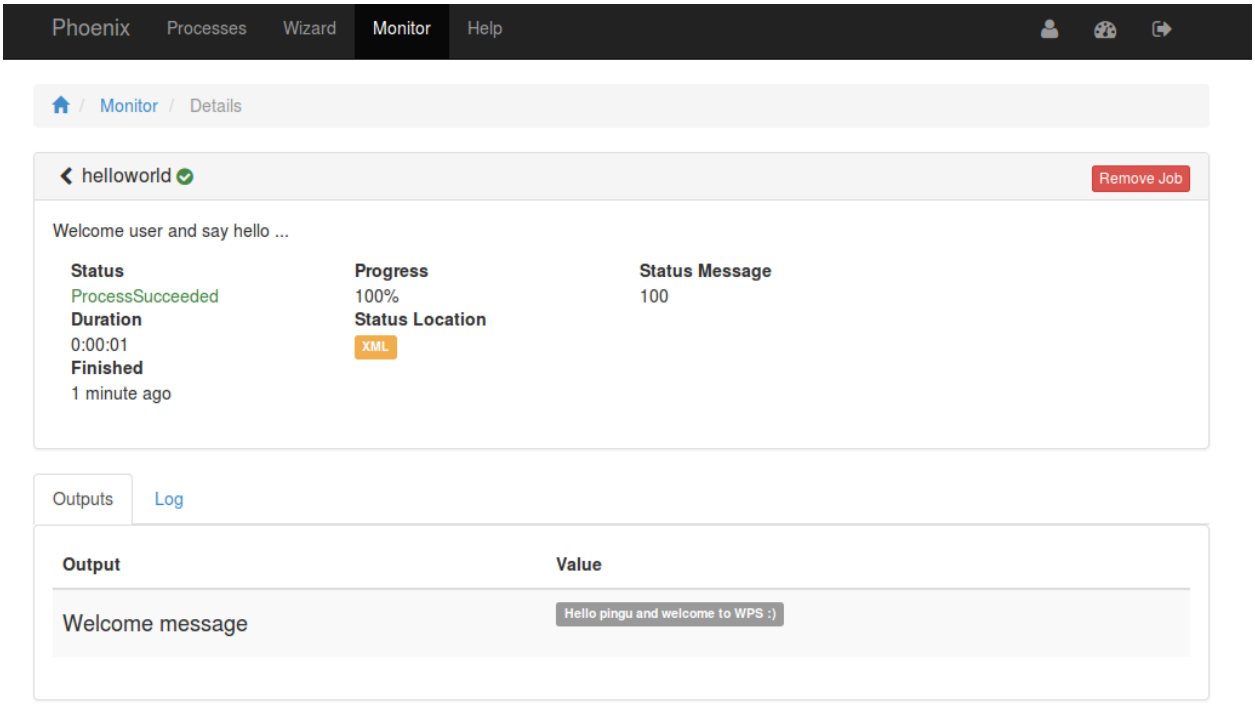
4.1.4 Monitor running Job

The job is now submitted and can be monitored on the *Monitor* page:



4.1.5 Display the outputs

Click on the Job ID link to get to the result of the submitted process.

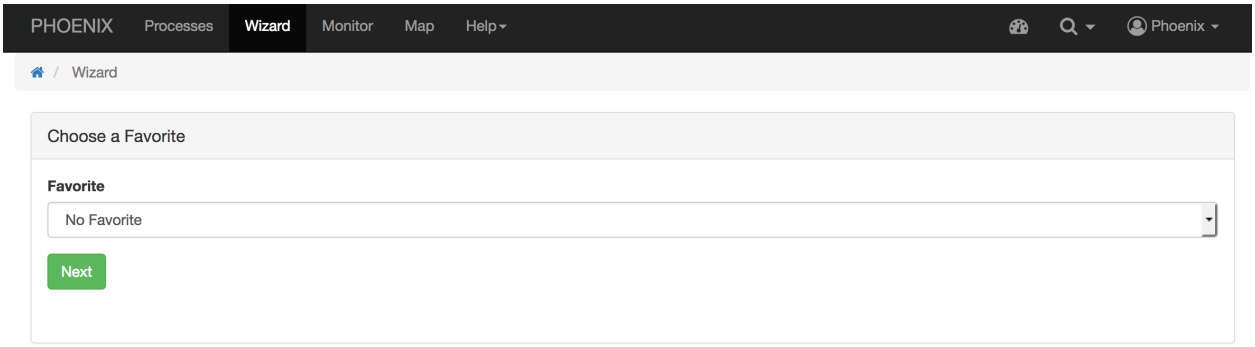


4.2 Run CDO sinfo on data from Thredds Dataservice

First you need to login. Please follow the login instructions in the *user guide*.

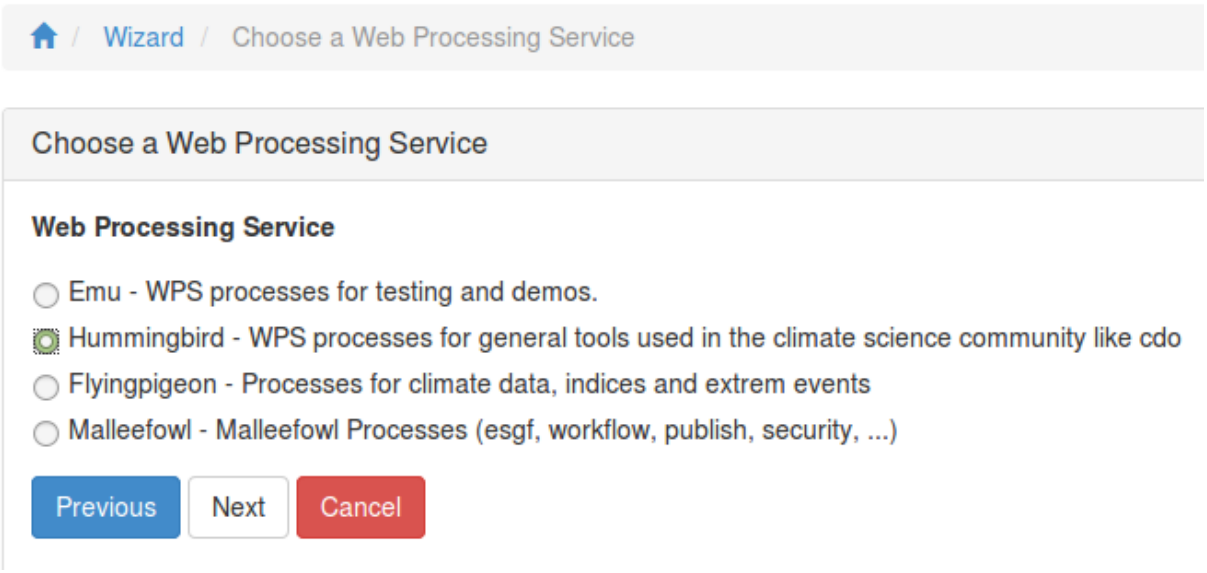
- *Use the Wizard*
- *Select Hummingbird WPS Service*
- *Choose "CDO sinfo" Process*
- *Choose Input Parameter*
- *Choose Thredds as Source*
- *Choose Thredds Service*
- *Choose Data from Thredds Catalog*
- *Start Process*
- *Monitor running Job*
- *Display the outputs*

4.2.1 Use the Wizard

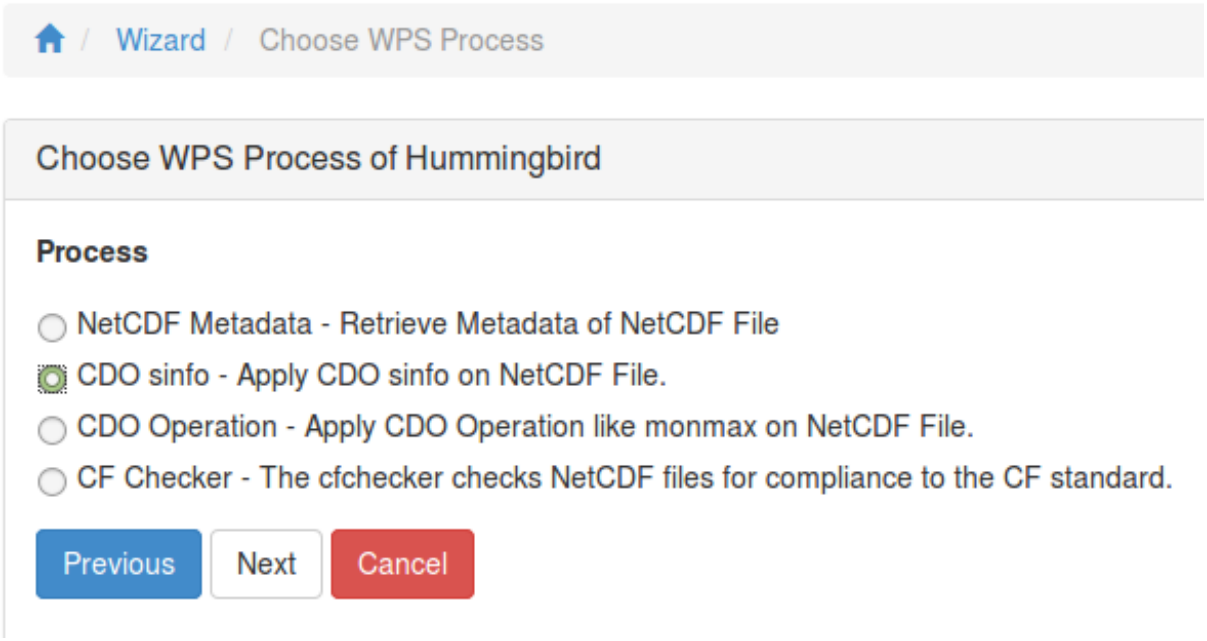


4.2.2 Select Hummingbird WPS Service

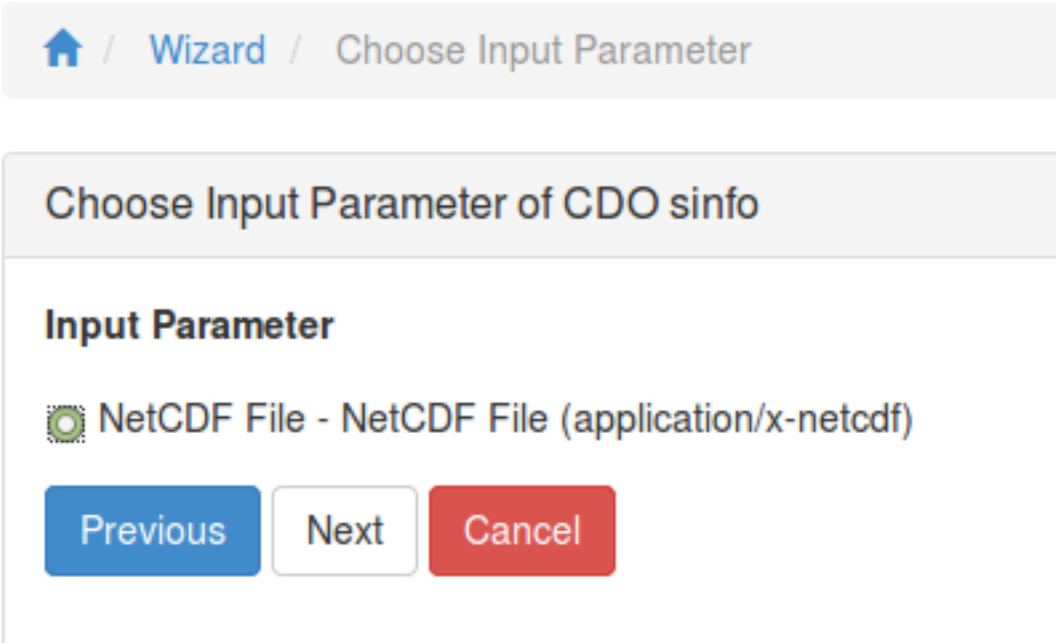
For this example choose the Hummingbird WPS service which has CDO processes.



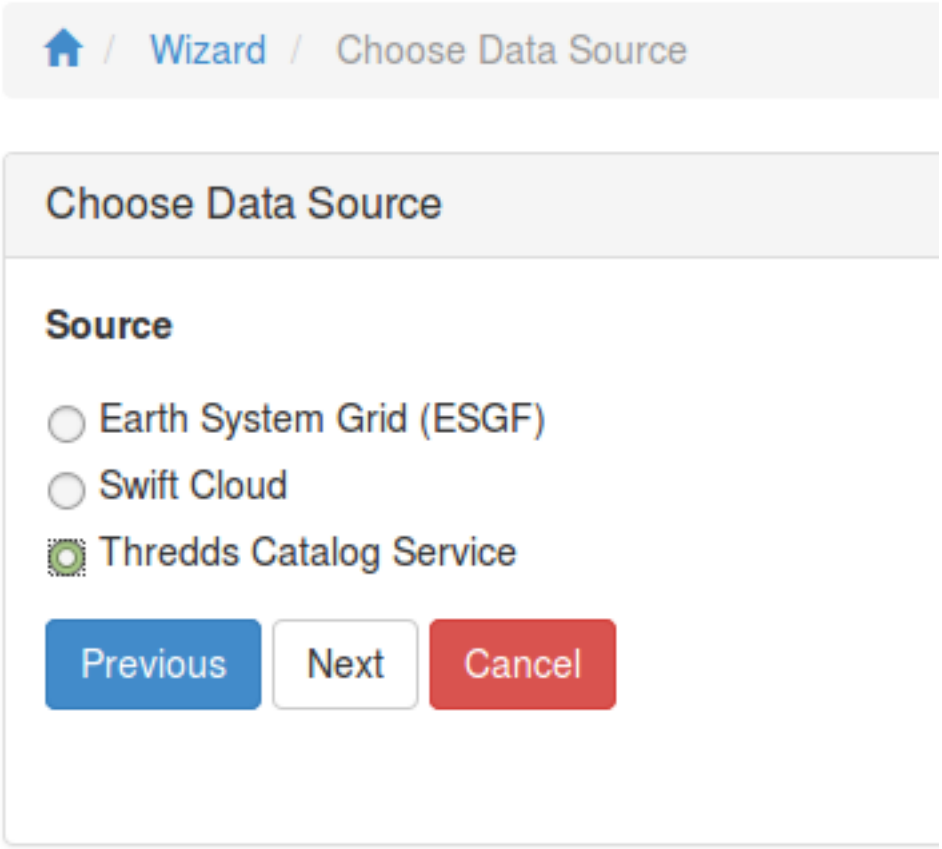
4.2.3 Choose "CDO sinfo" Process



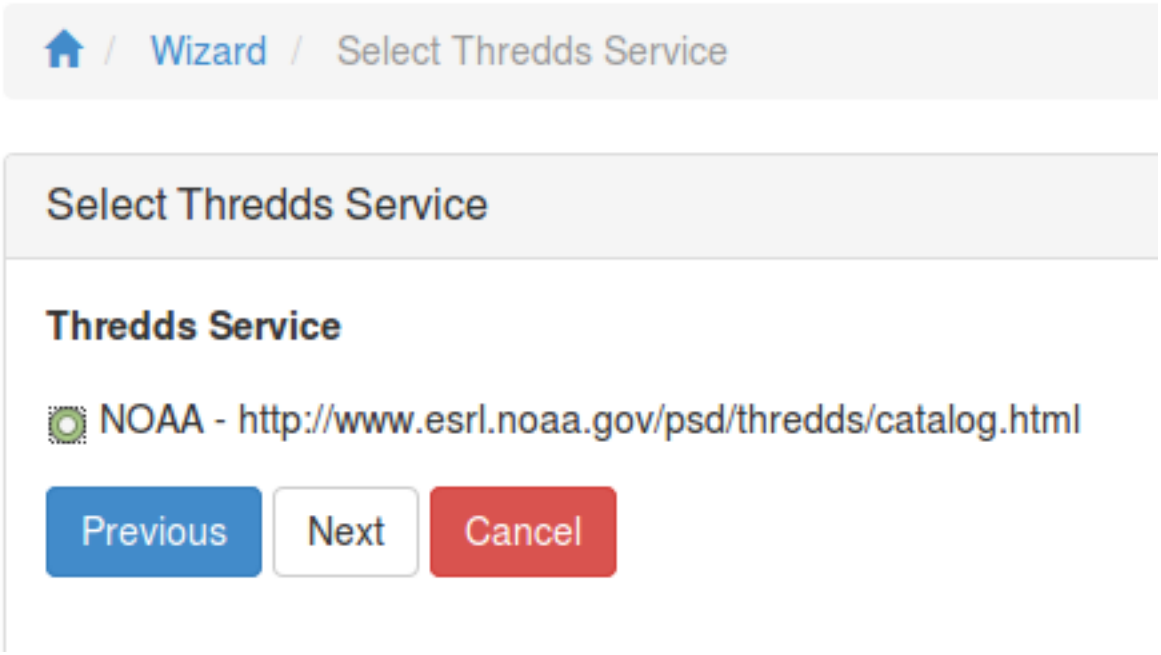
4.2.4 Choose Input Parameter



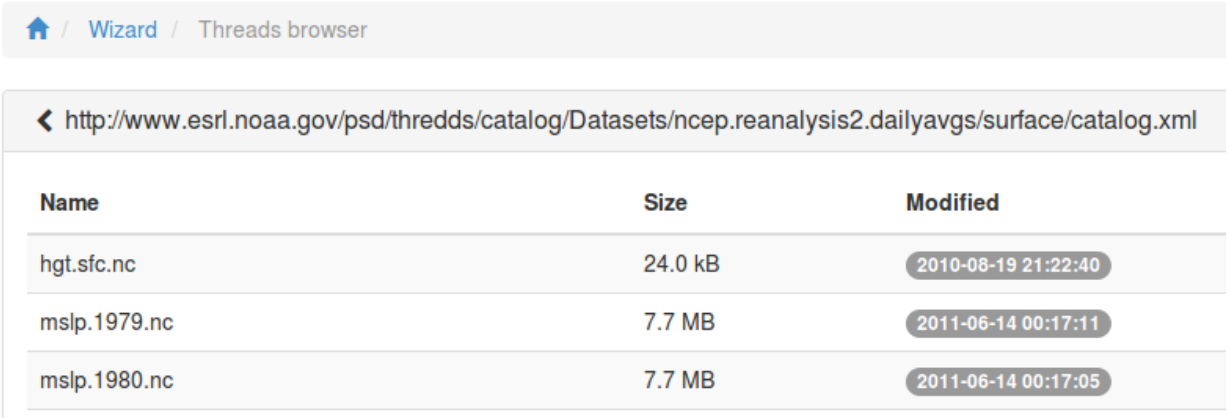
4.2.5 Choose Thredds as Source



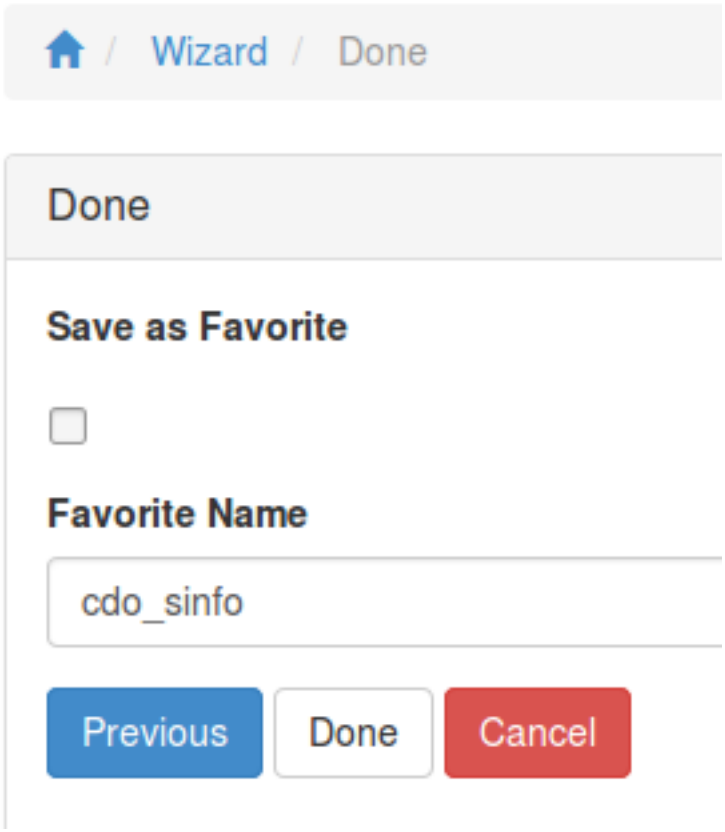
4.2.6 Choose Thredds Service



4.2.7 Choose Data from Thredds Catalog

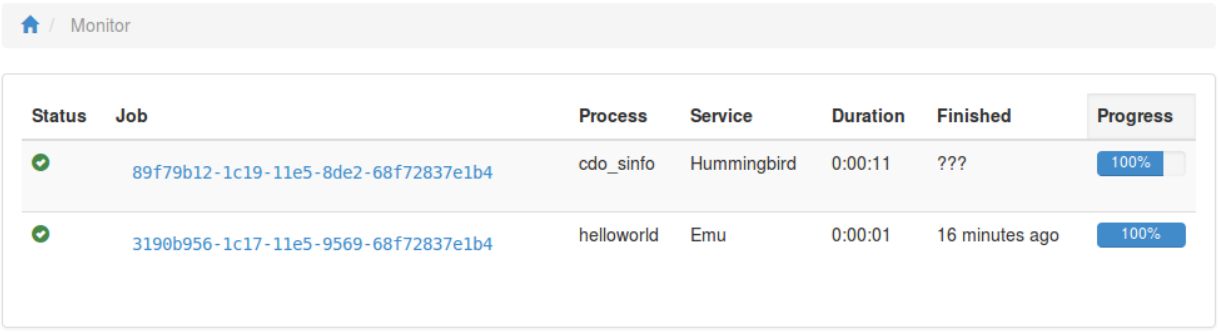


4.2.8 Start Process



4.2.9 Monitor running Job

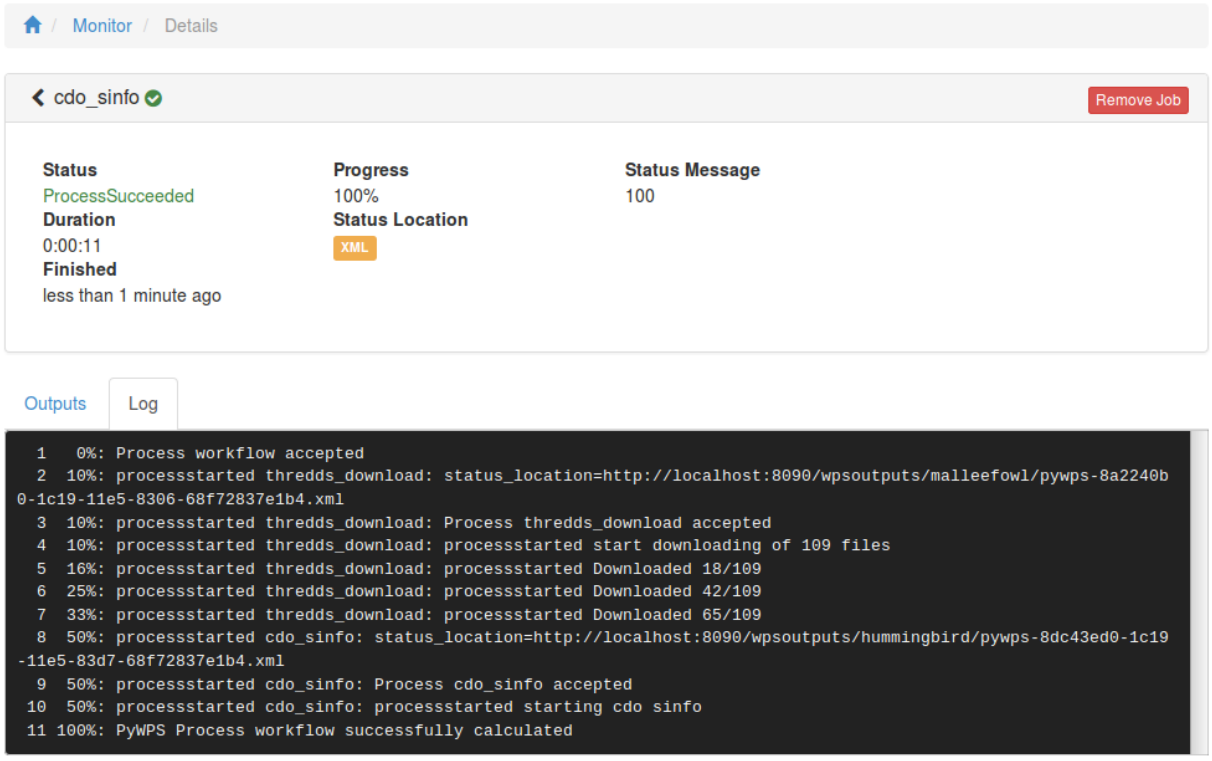
The job is now submitted and can be monitored on the *Monitor* page:



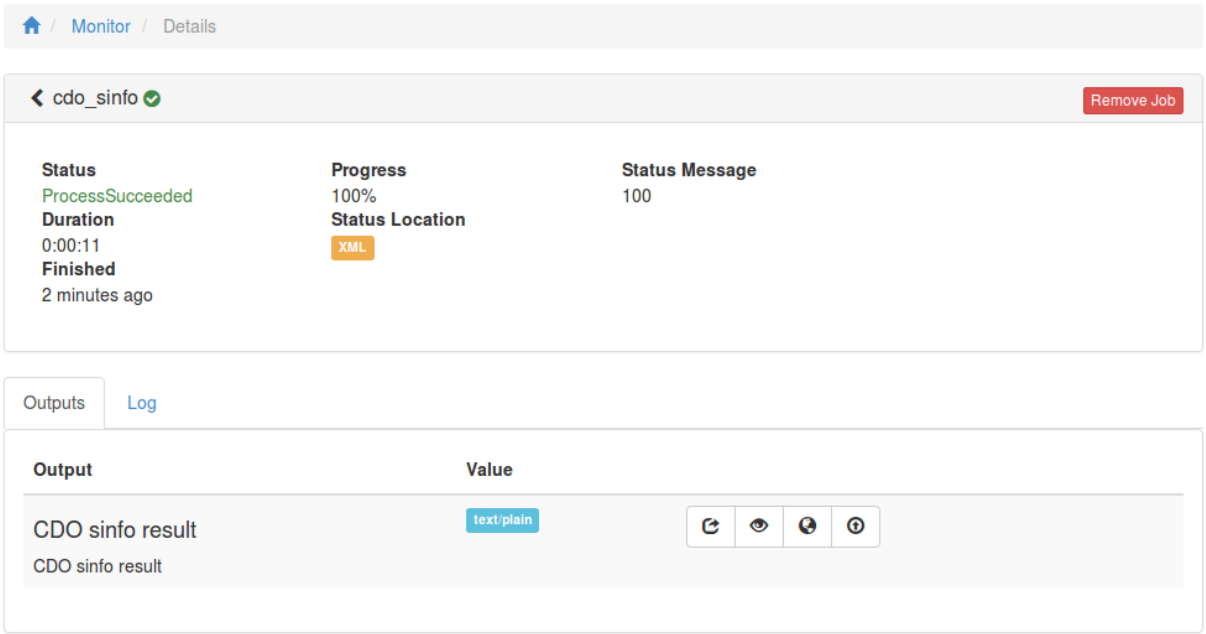
4.2.10 Display the outputs

Click on the Job ID link to get to the result of the submitted process.

Job Log



Job Outputs

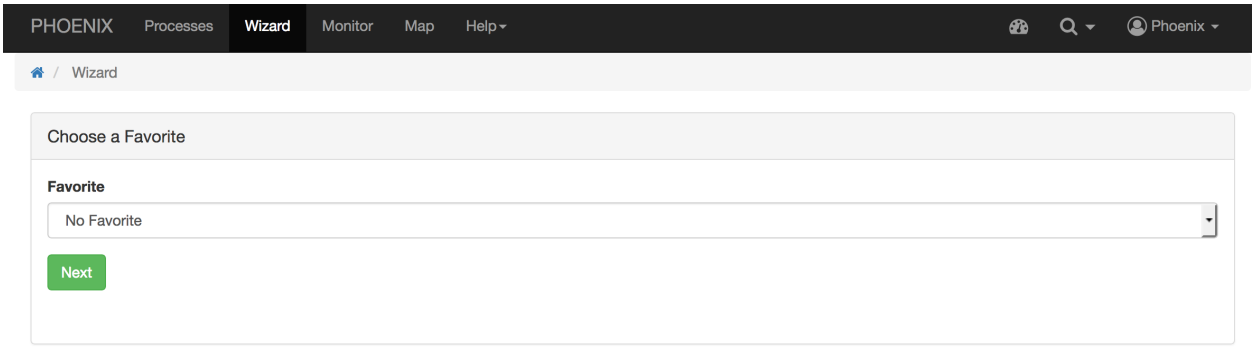


4.3 Run CDO ensemble operation on CMIP5 data from ESGF

First you need to login. Please follow the login instructions in the *user guide*.

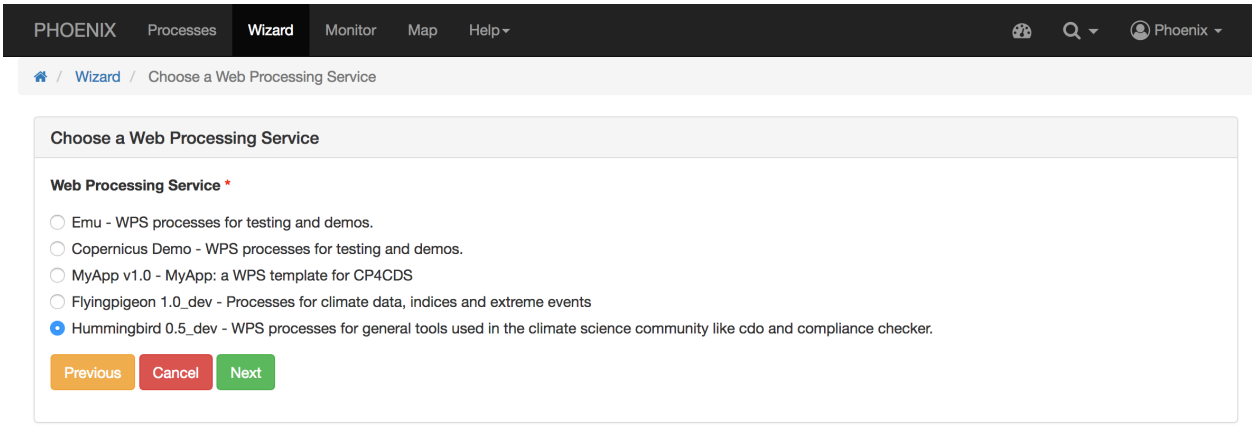
- *Use the Wizard*
- *Select Hummingbird WPS Service*
- *Choose "CDO Ensembles Operation" Process*
- *Choose CDO ensmean Operator*
- *Choose Input Parameter*
- *Choose ESGF as Source*
- *Update your ESGF credentials if asked*
- *Select ensembles of CMIP5 experiment*
- *Start Process*
- *Monitor running Job*
- *Display the outputs*

4.3.1 Use the Wizard

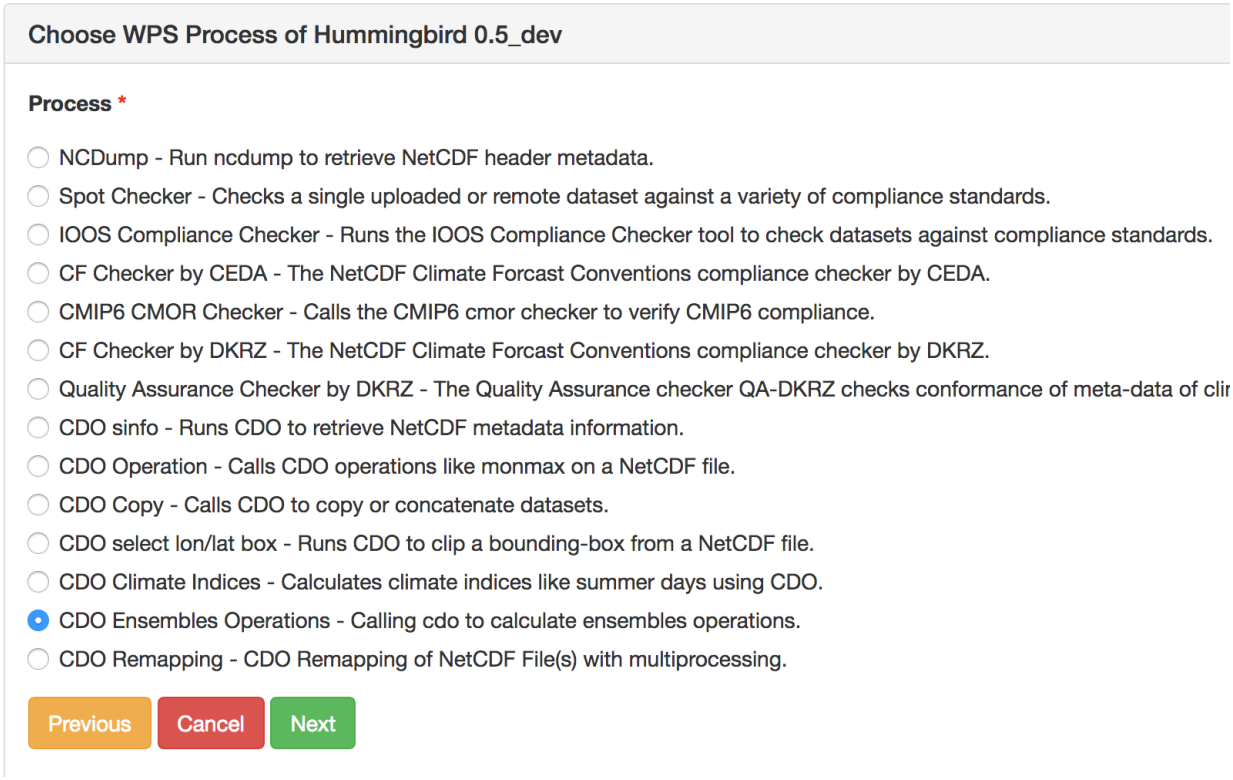


4.3.2 Select Hummingbird WPS Service

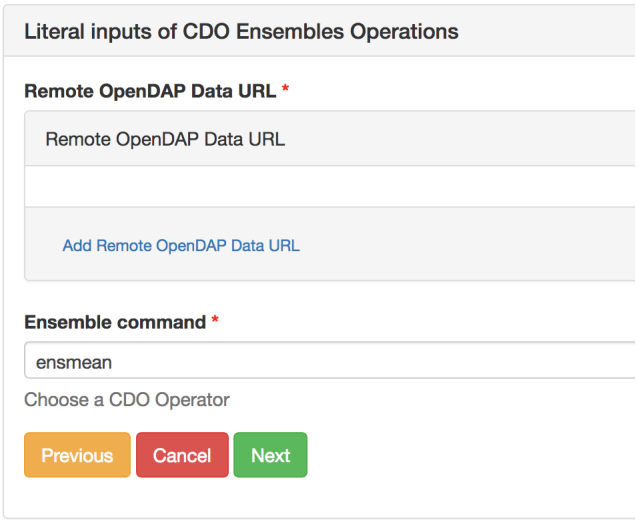
For this example choose the Hummingbird WPS service which has CDO processes.



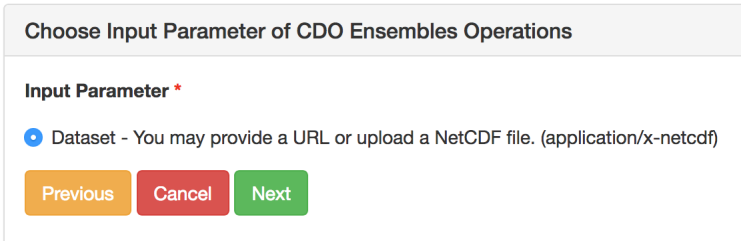
4.3.3 Choose "CDO Ensembles Operation" Process



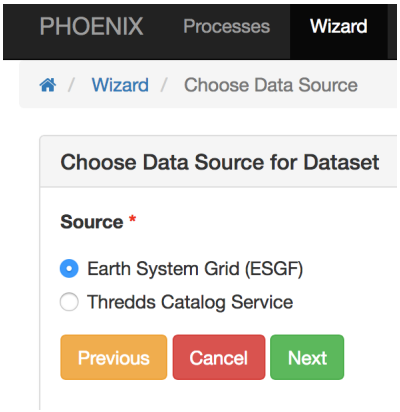
4.3.4 Choose CDO ensmean Operator



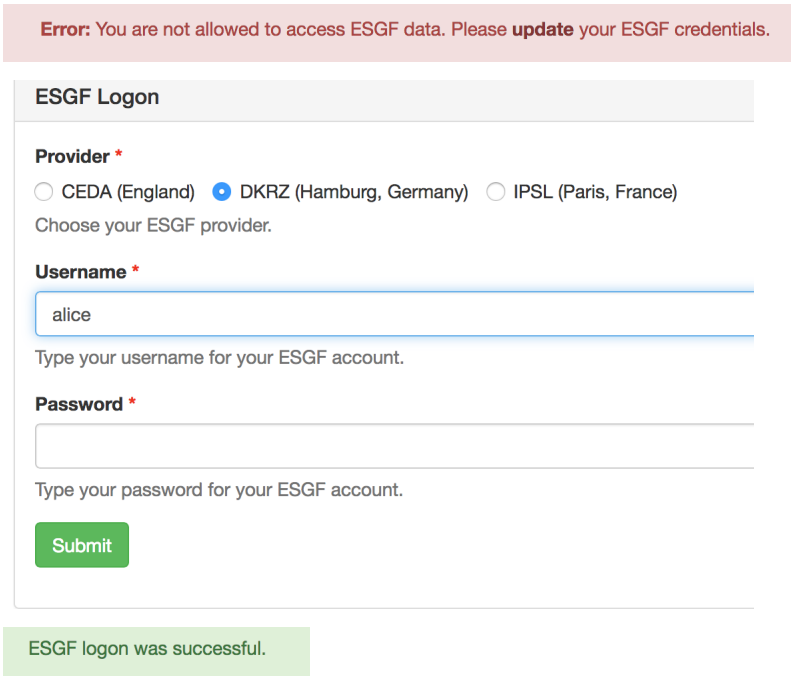
4.3.5 Choose Input Parameter



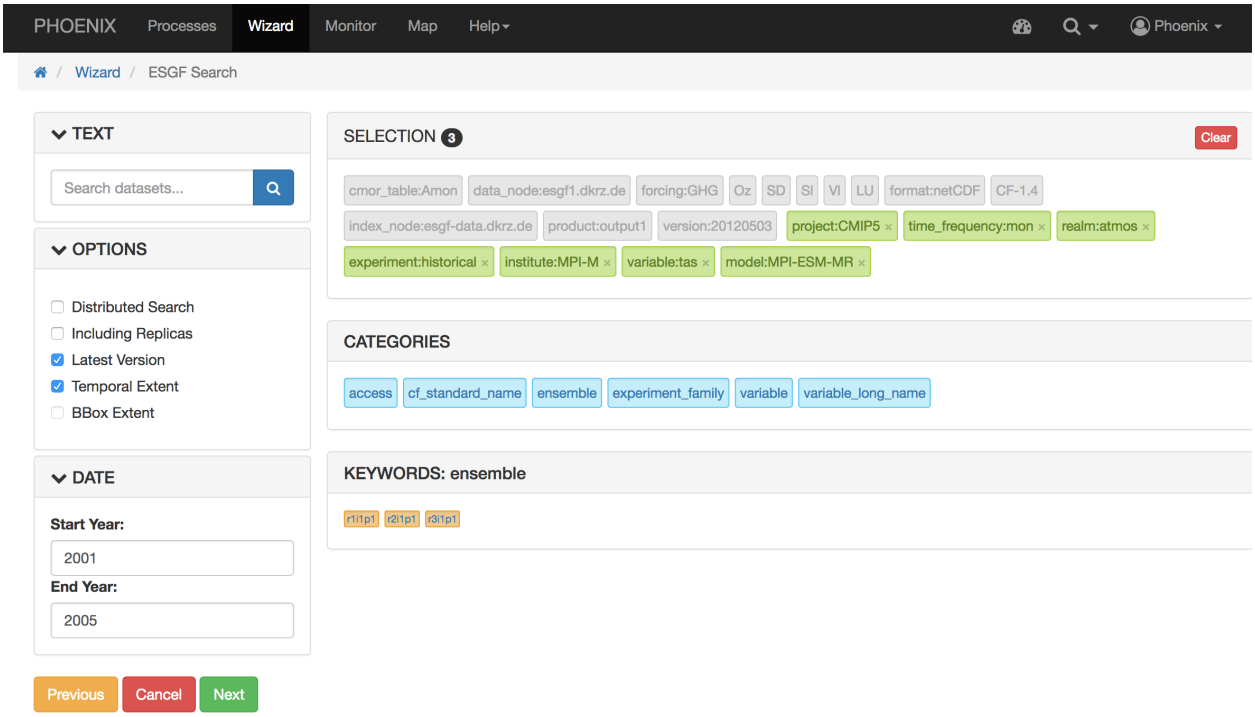
4.3.6 Choose ESGF as Source



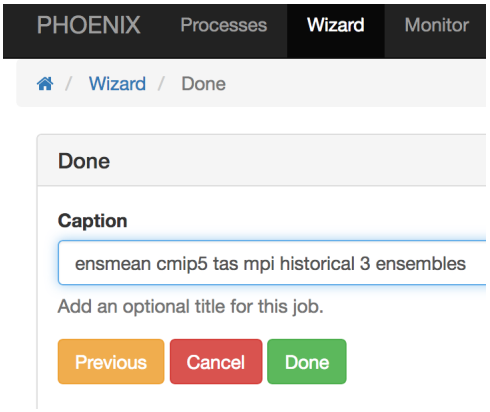
4.3.7 Update your ESGF credentials if asked



4.3.8 Select ensembles of CMIP5 experiment

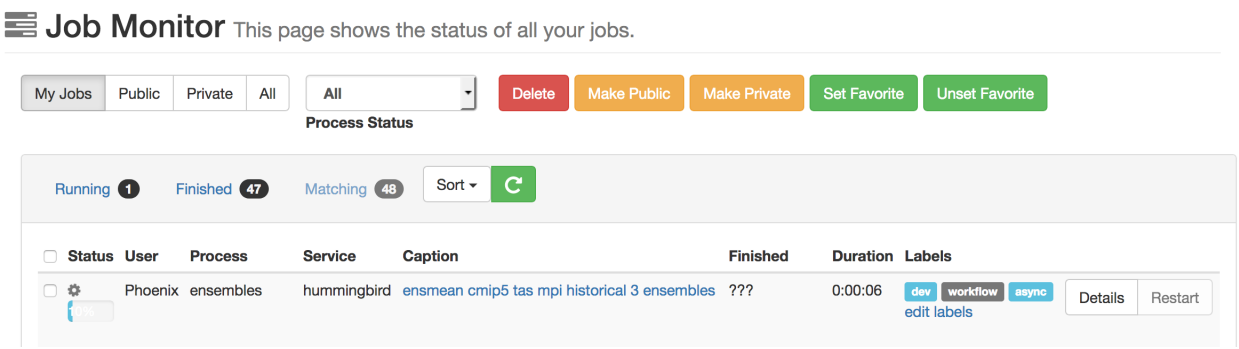


4.3.9 Start Process



4.3.10 Monitor running Job

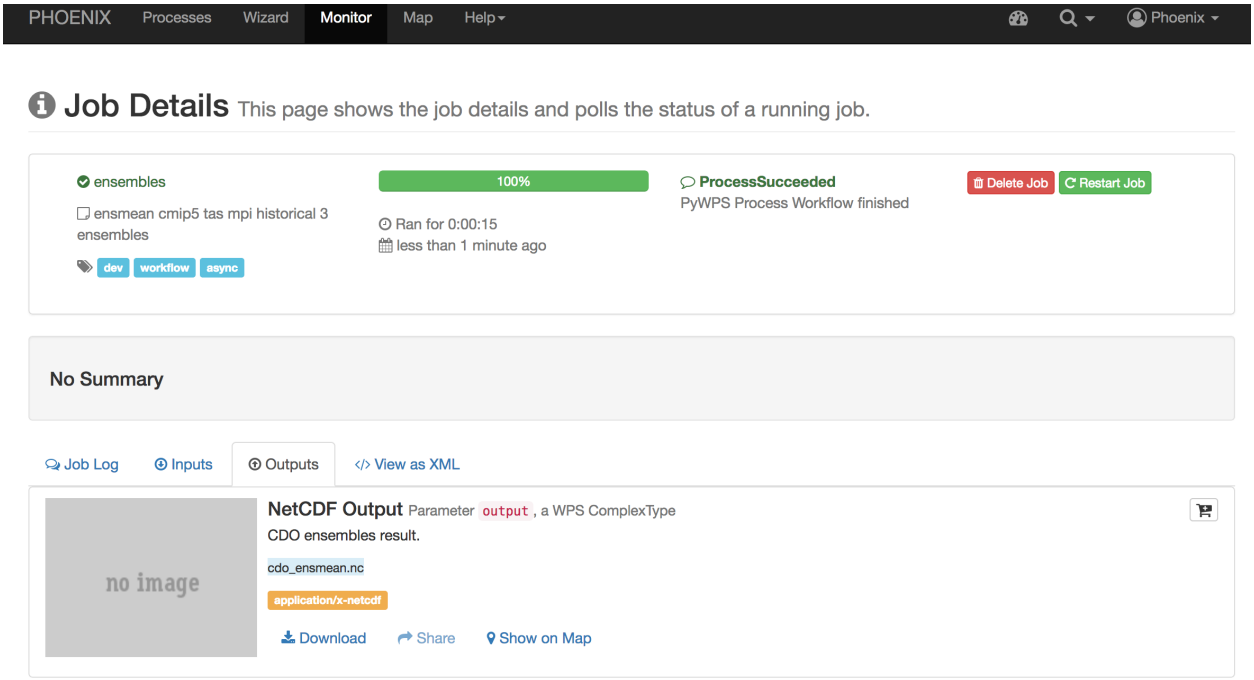
The job is now submitted and can be monitored on the *Monitor* page:



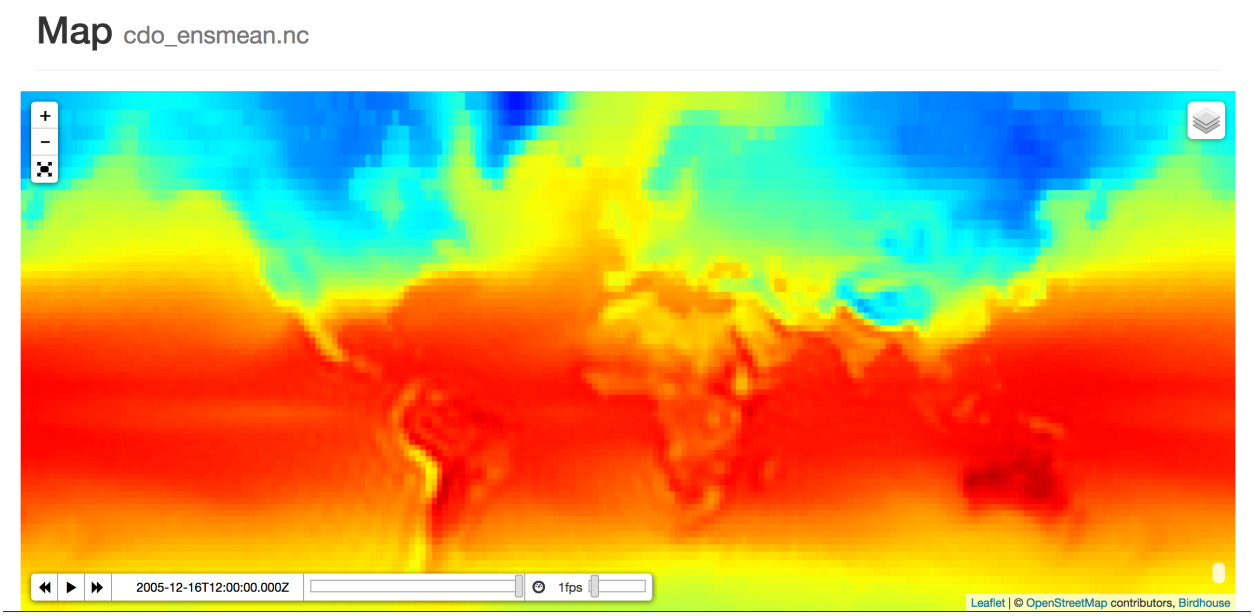
4.3.11 Display the outputs

Click on the `Details` button to get to the result of the submitted process.

Outputs



Map



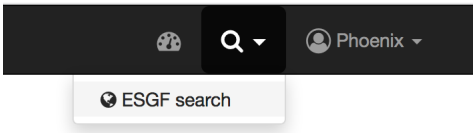
4.4 Run CDO ensemble operation on CORDEX data from ESGF using OpenDAP

First you need to login. Please follow the login instructions in the *user guide*.

- *Search and select CORDEX ensembles*
- *Check the selected files in Cart (optional)*
- *Select Hummingbird WPS Service*
- *Choose "CDO Ensembles Operation" Process*
- *Choose CDO ensmean Operator and OpenDAP datasets*
- *Monitor running Job*
- *Display the outputs*

4.4.1 Search and select CORDEX ensembles

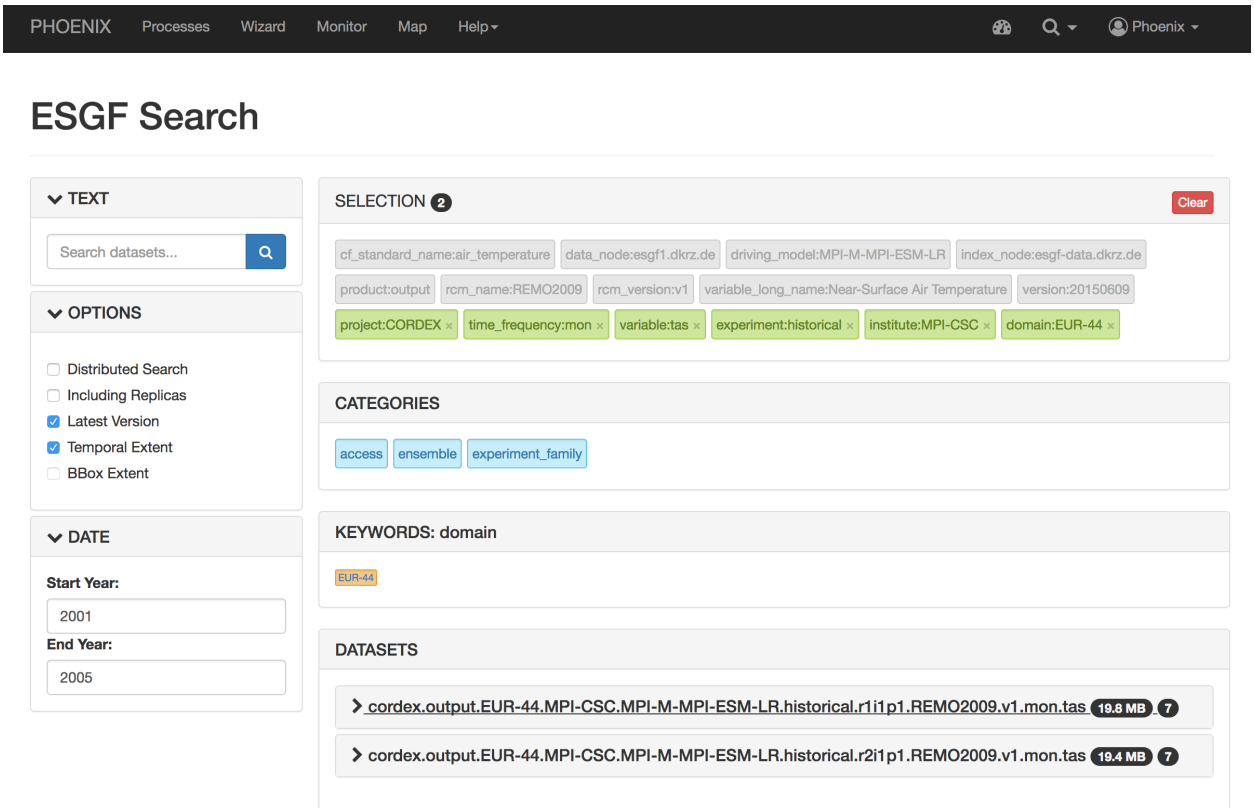
Activate ESGF Search



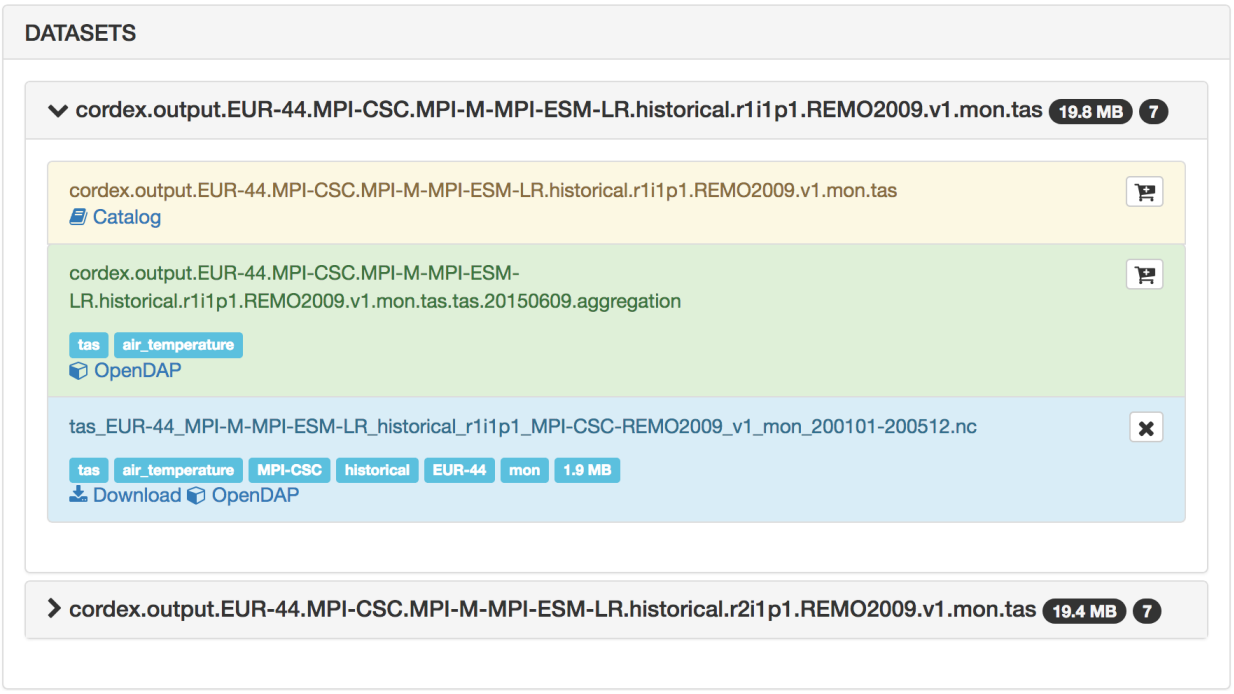
Update ESGF credentials if asked

Warning: You are not allowed to access ESGF data. Please **update** your ESGF credentials.

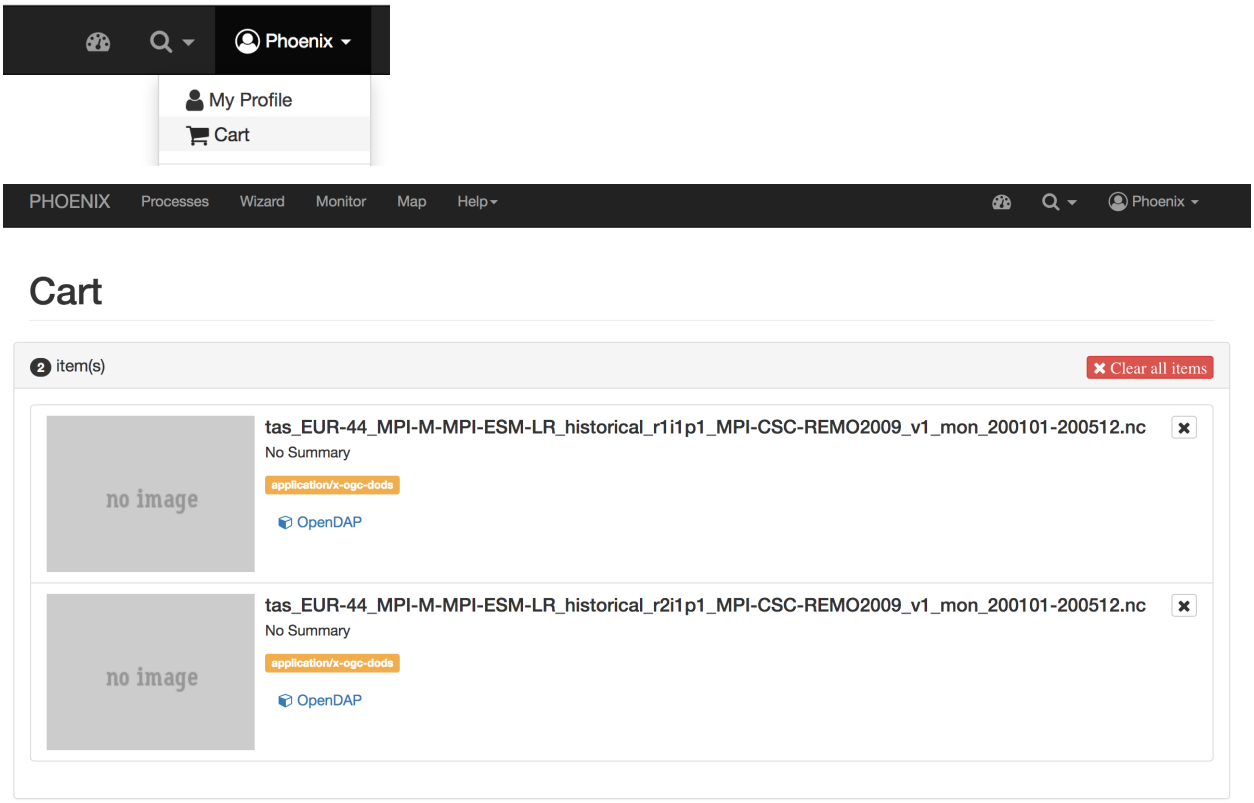
Search CORDEX Ensemble



Select Files (OpenDAP)

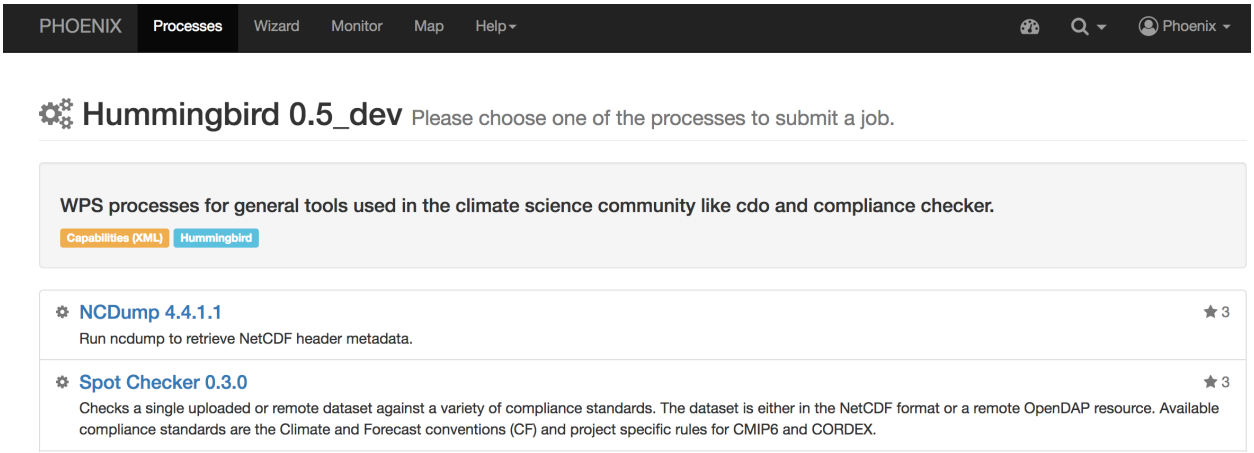


4.4.2 Check the selected files in Cart (optional)

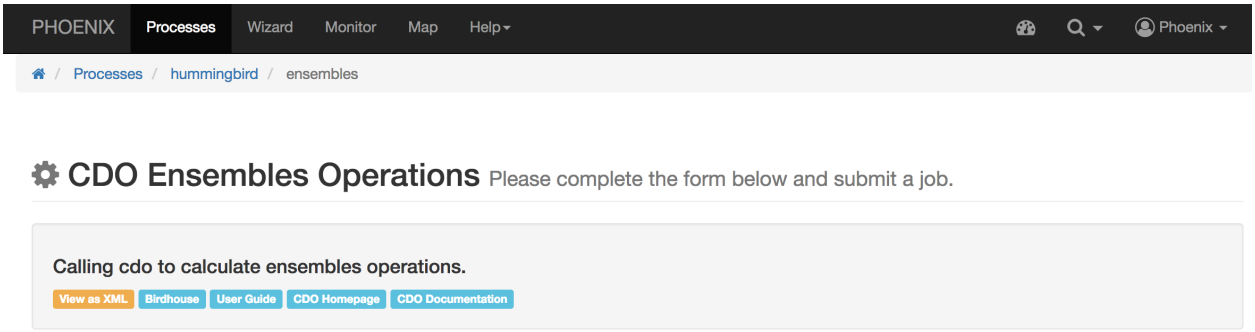


4.4.3 Select Hummingbird WPS Service

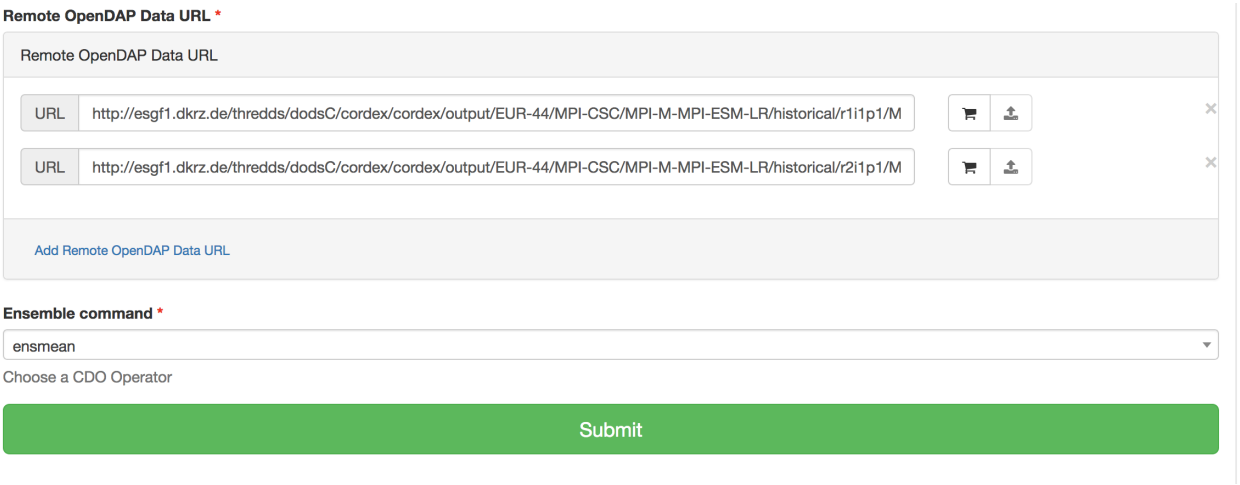
Choose the Hummingbird WPS service which has CDO processes.



4.4.4 Choose "CDO Ensembles Operation" Process

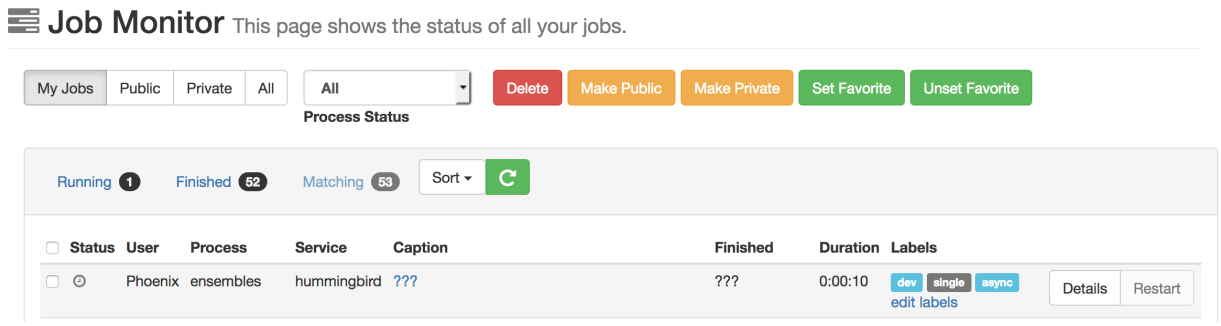


4.4.5 Choose CDO ensmean Operator and OpenDAP datasets



4.4.6 Monitor running Job

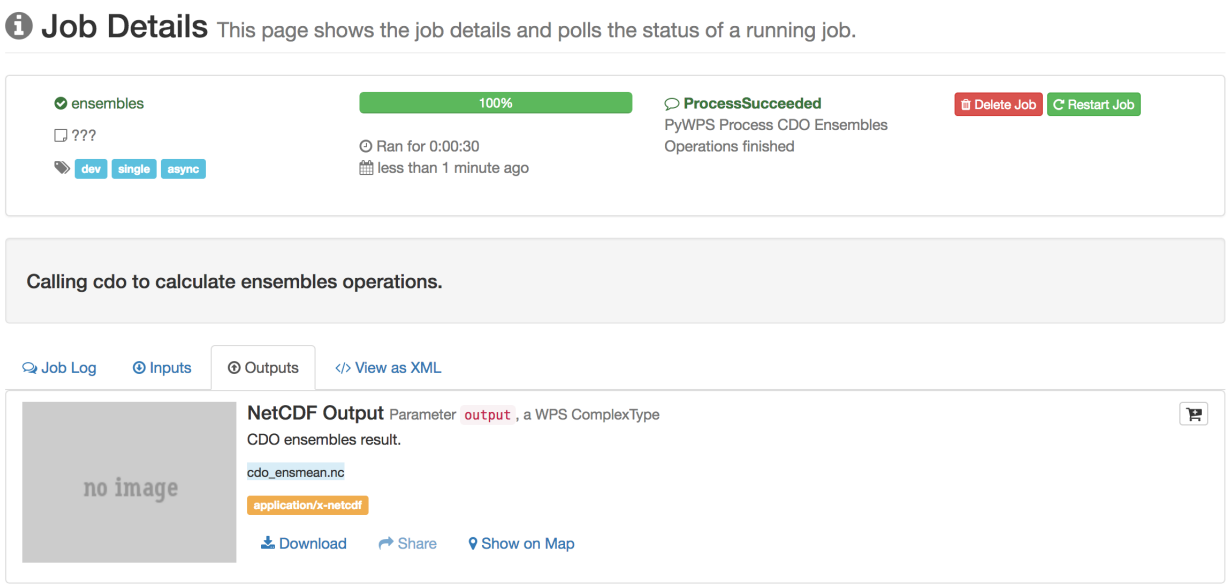
The job is now submitted and can be monitored on the *Monitor* page:



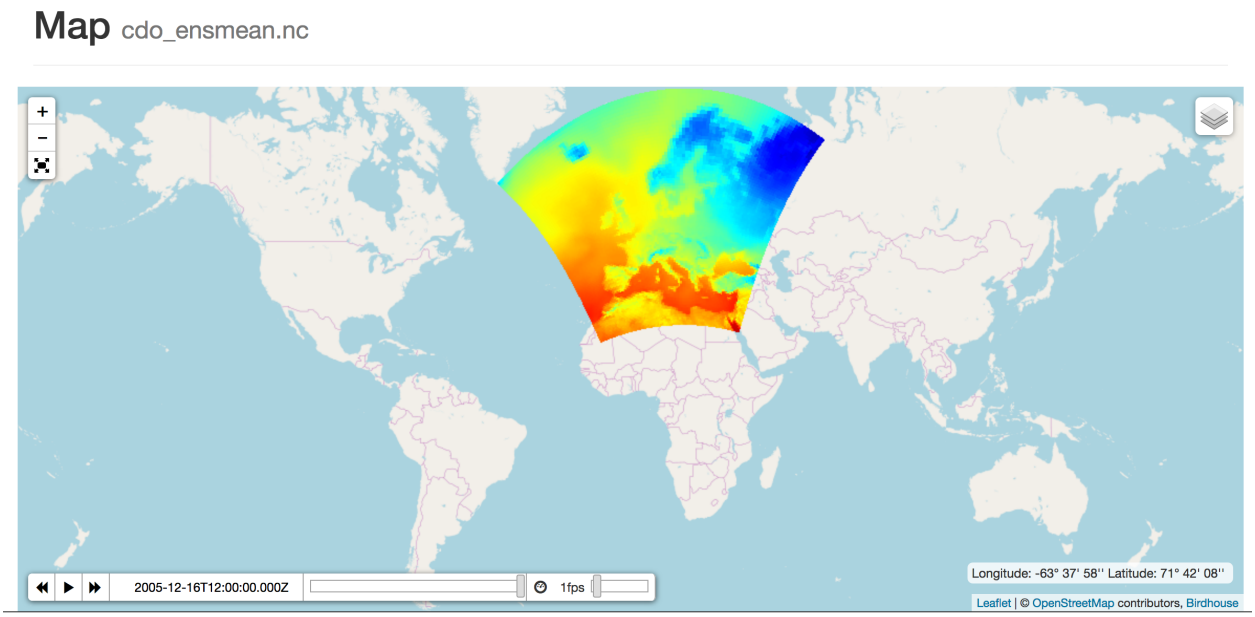
4.4.7 Display the outputs

Click on the `Details` button to get to the result of the submitted process.

Outputs



Map



4.5 Creating a timeseries plot

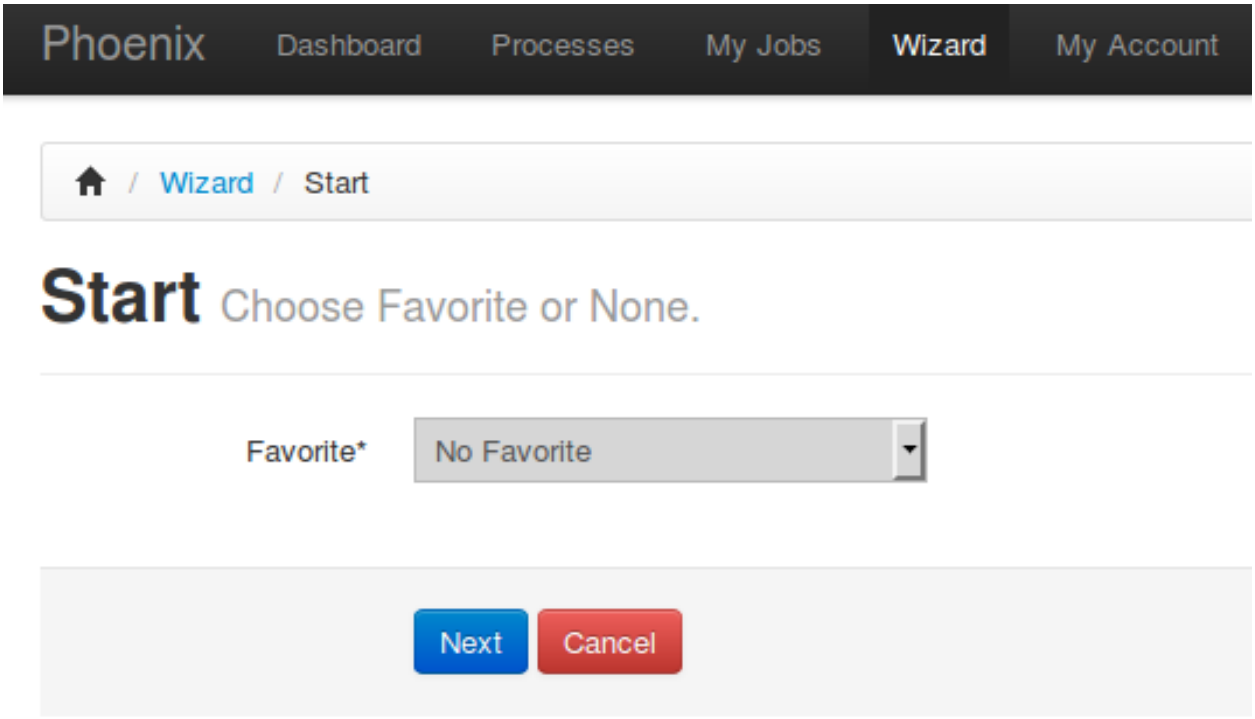
First you need to login. Please follow the login instructions in the *user guide*.

Once the login procedure is done, processes are operable and data search and download within the ESGF data archive is possible.

In this timeseries plot example we will use the Flyingpigeon WPS. Make sure Flyingpigeon is installed and running and check that it is *registered in Phoenix*.

There are two ways to submit a job: either with *Processes* or *Wizard*.

While with *Processes* you can select single operational processes the *Wizard* is guiding you through the necessary steps to submit a job. For getting an idea of the operation procedure choose the *Wizard* tab:



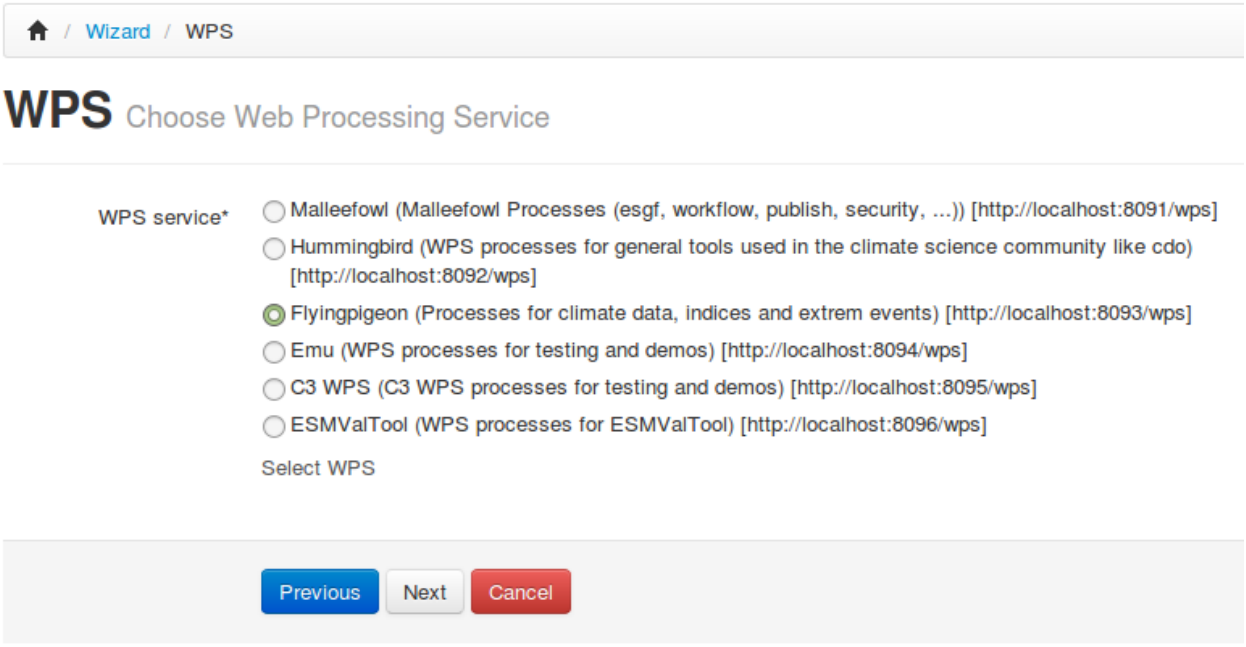
You could choose a favorite here of a previous run job but in this case please choose *No Favorite* and click *Next*.

The following steps are necessary to run a visualisation job:

- *Select WPS Service*
- *Choose Process*
- *Enter Process Parameters*
- *Select Data Source*
- *Search Input Files*
- *Check your credentials*
- *Start the process*
- *Monitor running Job*
- *Display the outputs*

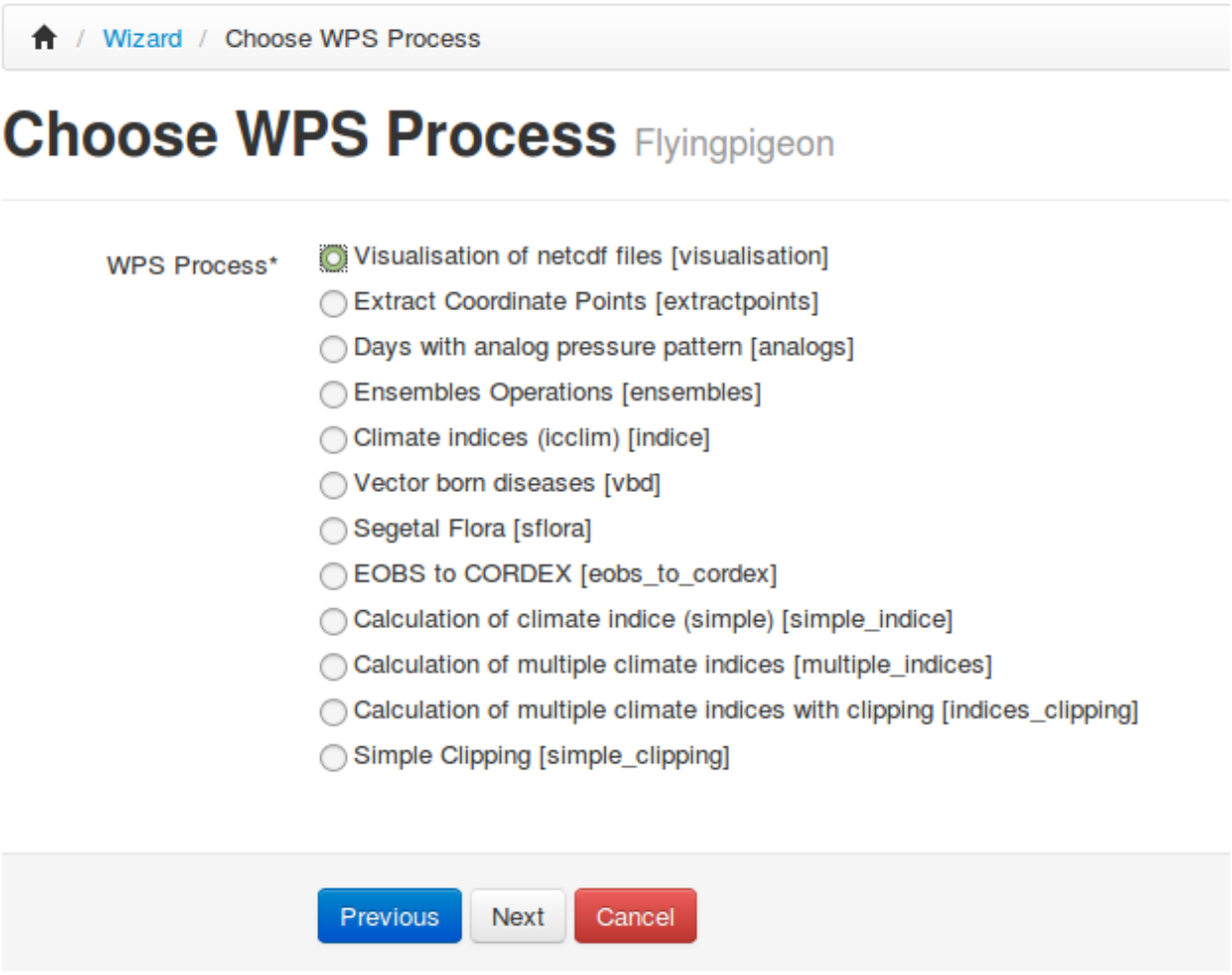
4.5.1 Select WPS Service

For this example choose the Flyingpigeon WPS service which has processes for the climate impact community.



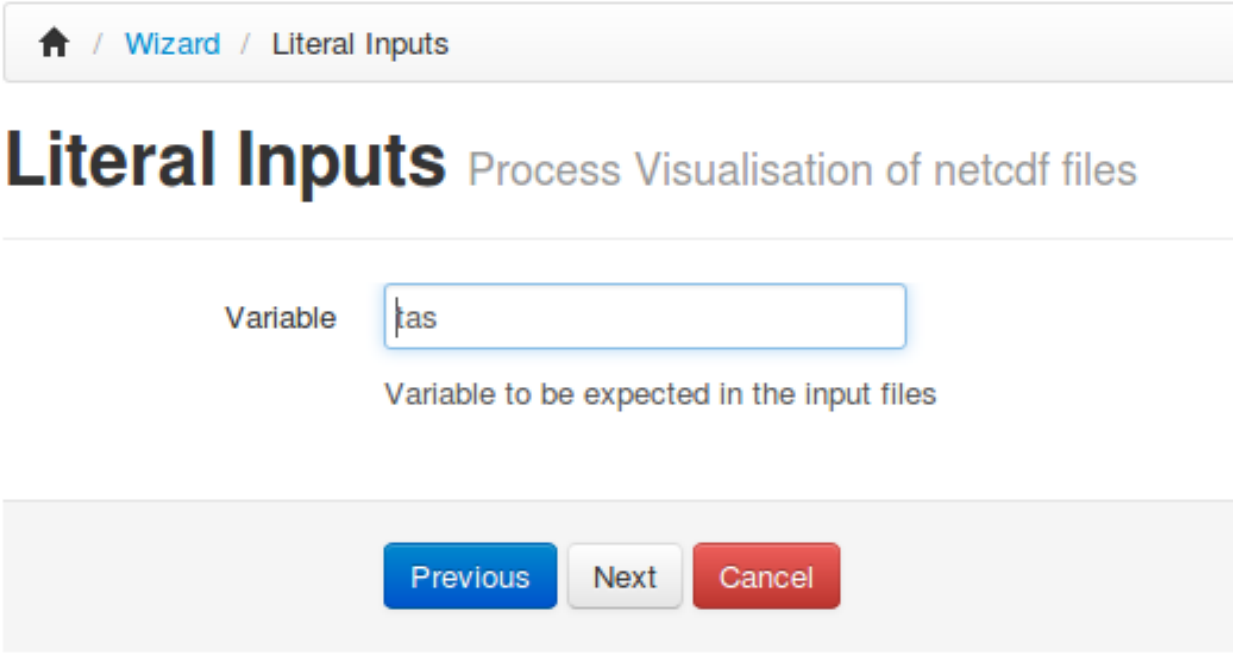
4.5.2 Choose Process

With clicking on *Next* you'll find the list of available processes. Check the *Visualisation of NetCDF files*.



4.5.3 Enter Process Parameters

Click on *Next* which guides you to the process parameter:



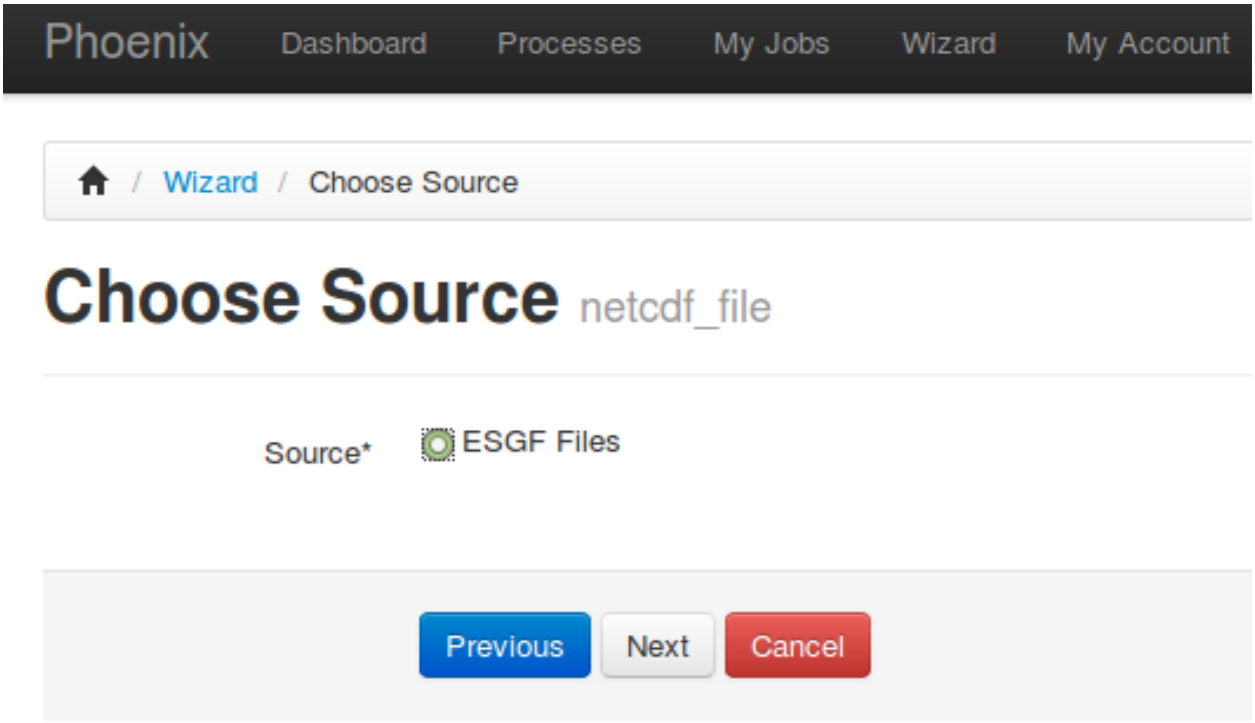
The values in the data files are stored with defined variable names. Here are the most common ones:

- tas – mean air temperaure at 2m (in Kelvin)
- tasmin – minimum air temperaure at 2m (in Kelvin)
- tasmax – maximum air temperaure at 2m (in Kelvin)
- pr – pricipitation fulx at surface (in kg/second)
- ps – air pressure at surface
- huss – specific humidiy (in Kg/Kg)

A list of available variable names used for CMIP5 and CORDEX experiment can be found in the Appendix B of the CORDEX archive design.

4.5.4 Select Data Source

In the next step you will choose the data source. Currently there is only the ESGF data archive:



4.5.5 Search Input Files

This is a search GUI to find appropriate files stored in ESGF data archive. By selecting a *Search Categorie* (blue buttons), you can choose the appropriate options (in orange).

In this example select the following parameter:

Categorie	Option
project	CORDEX
domain	WAS-44
insitute	MPI-CSC
variable	tas
time_frequency	day

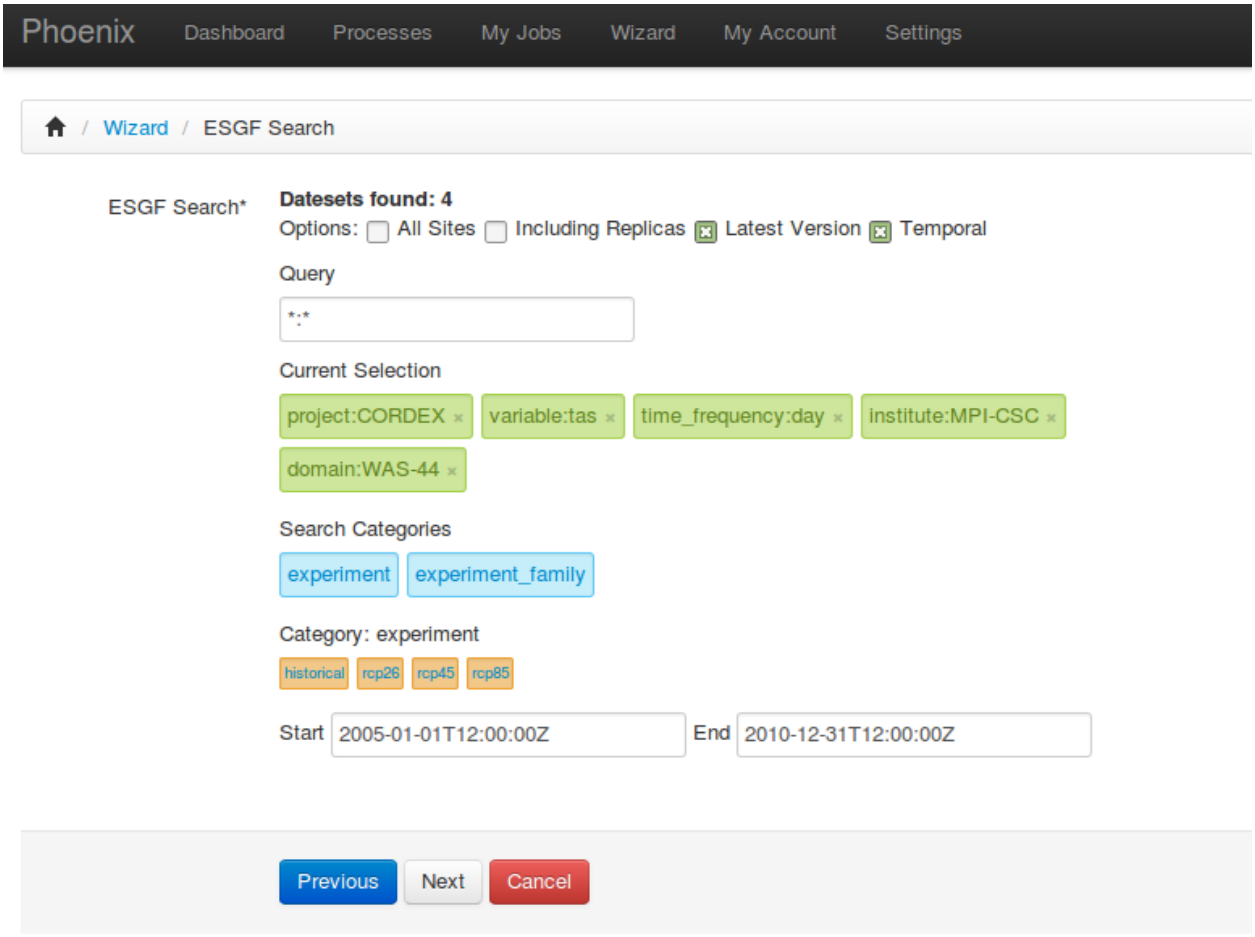
Double selection (like two domains) can be realized with pressing *Ctrl* - tab.

For the visualisation process it is necessary that the selected variable (`tas`) is the same as the variable argument in the *Process Parameters*

And optionally you can set the time bounds:

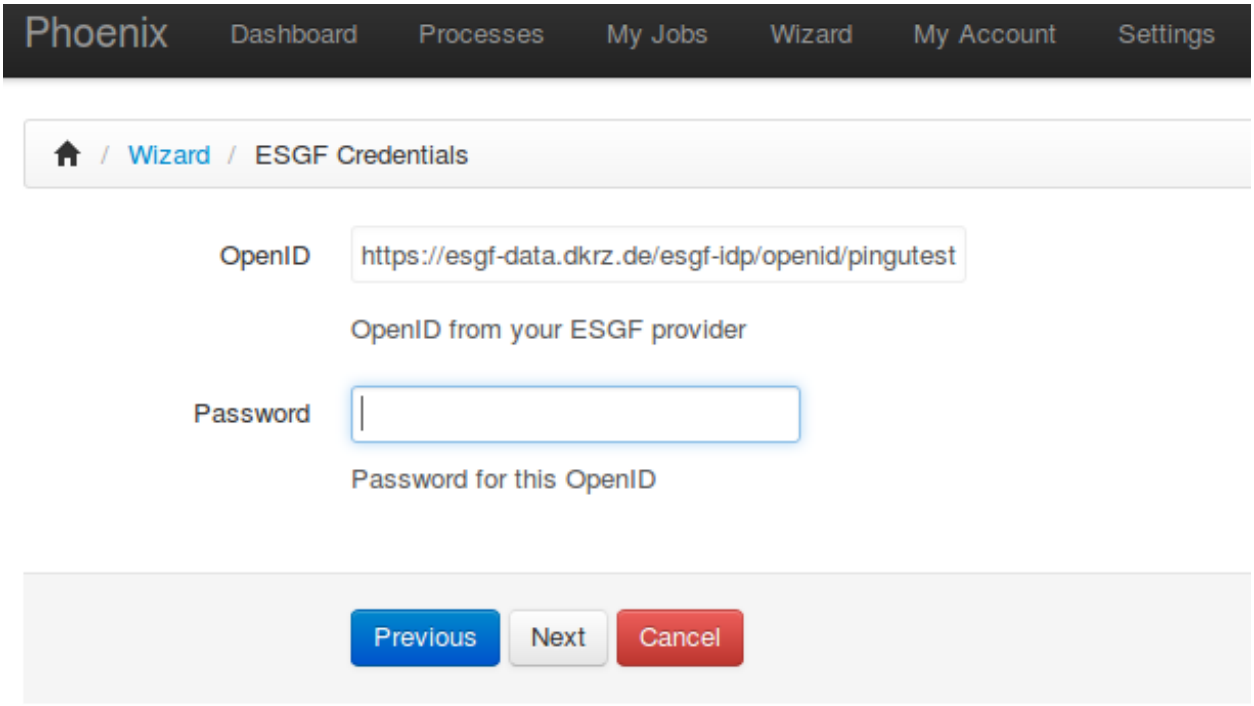
```
Start: 2005-01-01T12:00:00Z
End:   2010-12-31T12:00:00Z
```

The Selection should look similar to the following screenshot:



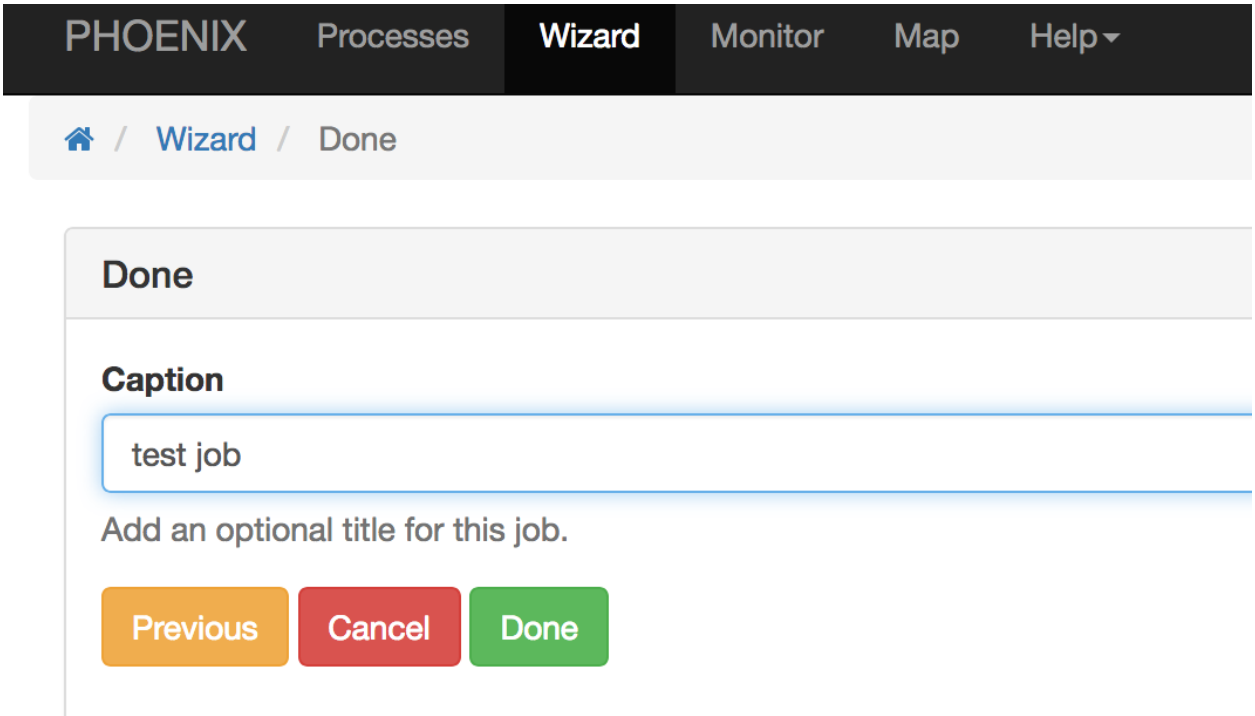
4.5.6 Check your credentials

To access ESGF data you need an x509 proxy certificate from ESGF. You can update your certificate in *My Account*. The x509 proxy certificate is valid only for a few hours. The wizard checks if your certificate is still valid and if not you will be asked to update it on the following wizard page.



4.5.7 Start the process

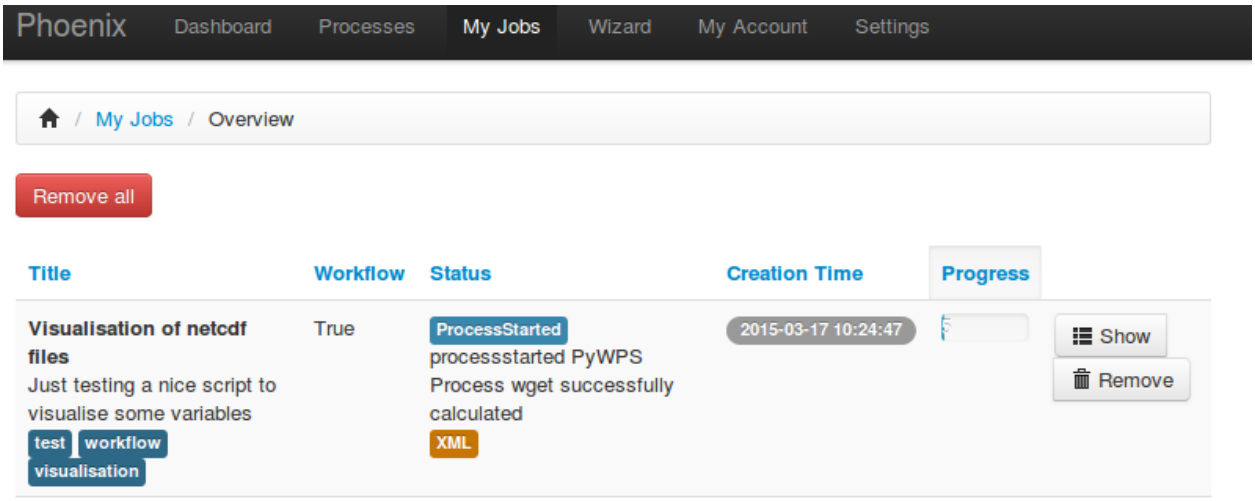
On the final page *Done* of the wizard you can give some descriptive keywords for your process. You can also save it as a favorite so that later you can run the same job again.



Press *Done* and the job will start.

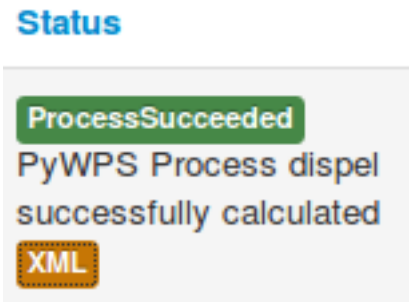
4.5.8 Monitor running Job

The job is now submitted and can be monitored on the *My Jobs* page:



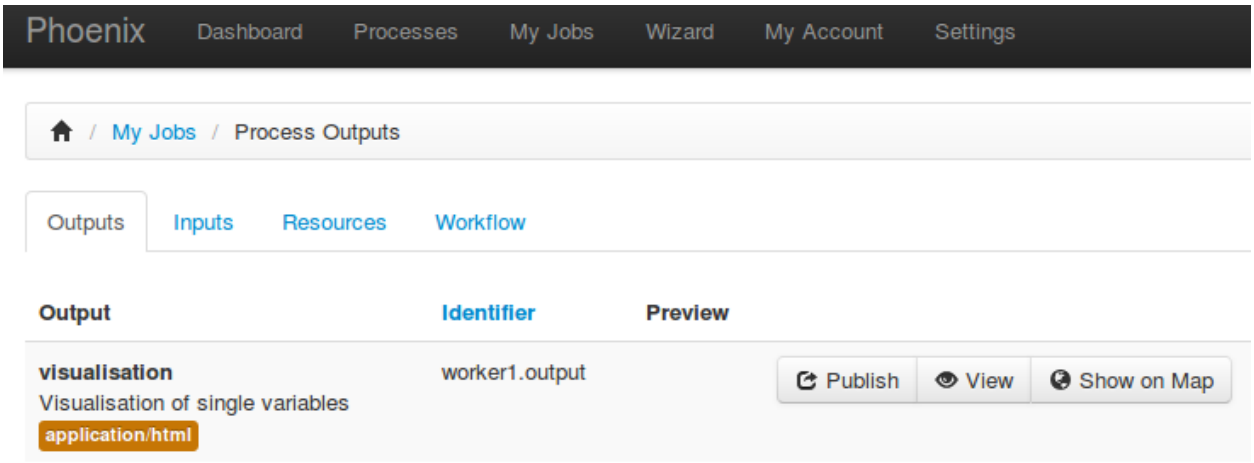
The job is running . . . data will be downloaded and the analyzing of the data starts. In this case, a field mean over the several experiments will be performed and an appropriate timeline drawn.

When the job has finished, the status bar is turning into green:

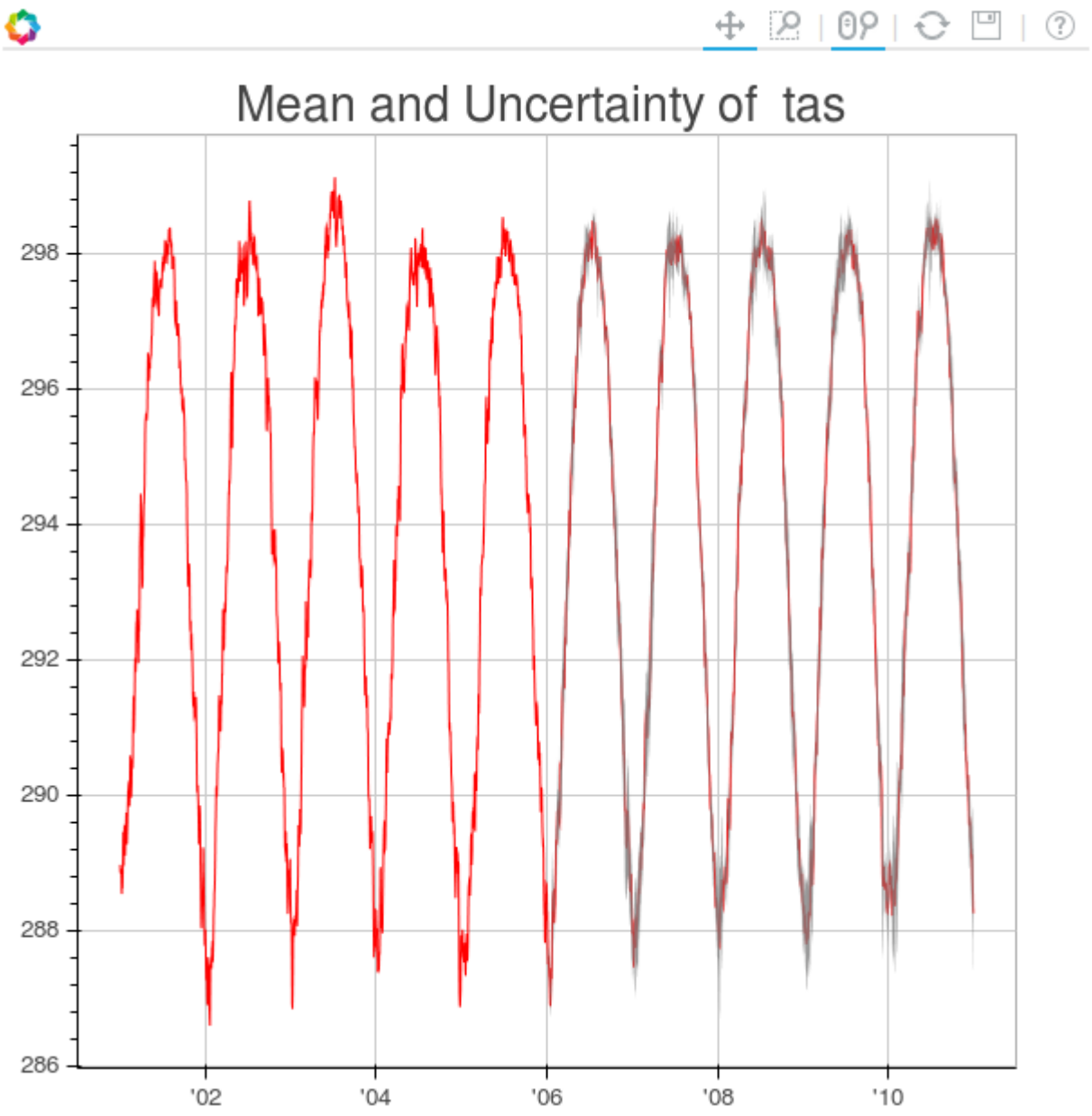


4.5.9 Display the outputs

Click on the *Show* button to get to the result of the submitted process.



In this case, it is an URL pointing to a HTML page with an embedded interactive plot using bokeh. Opening it in a new browser tab gives the following result:



4.6 Use the Birdhouse Solr Search in the Wizard

First you need to login. Please follow the login instructions in the *user guide*.

- *Prepare Solr Search (Admins only)*
- *Use the Wizard*
- *Select Hummingbird WPS Service*

- *Choose "CDO sinfo" Process*
- *Choose Input Parameter*
- *Choose Birdhouse Solr as Source*
- *Choose Data from Solr Search*
- *Start Process*
- *Monitor running Job*
- *Display the outputs*

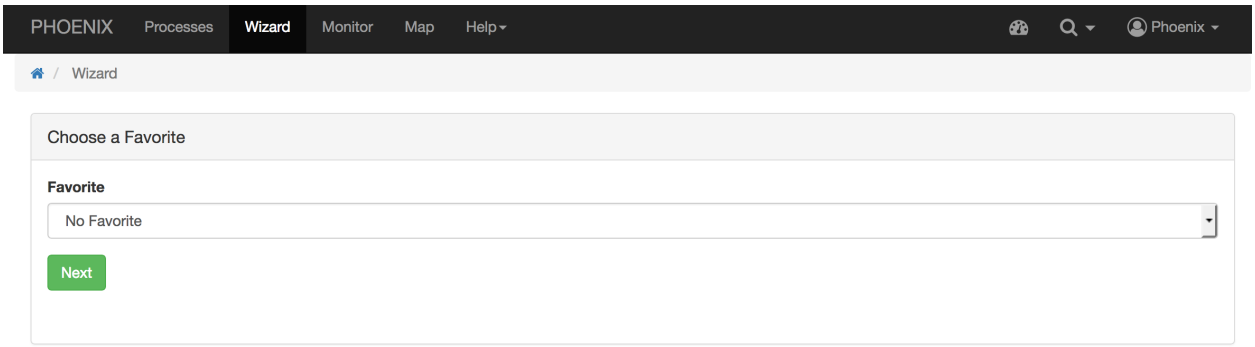
4.6.1 Prepare Solr Search (Admins only)

Register a thredds catalog in `Settings/Services`. For example use:

```
http://www.esrl.noaa.gov/psd/thredds/catalog/Datasets/ncep.reanalysis2.dailyavgs/
↪catalog.html
```

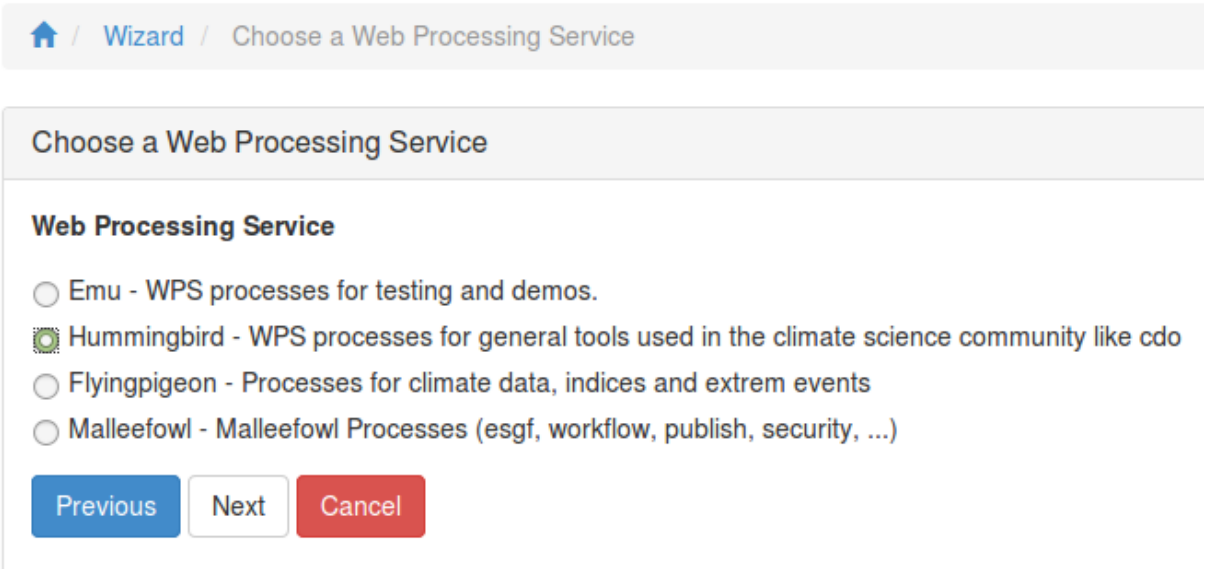
Index this Thredds Catalog in `Settings/Solr`.

4.6.2 Use the Wizard

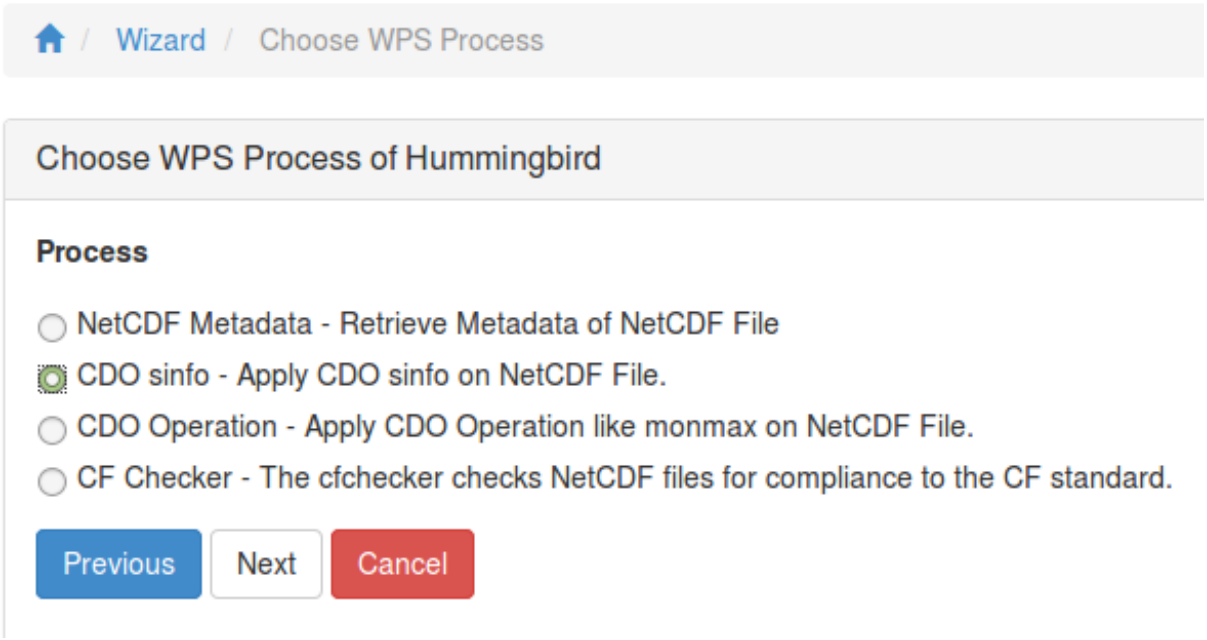


4.6.3 Select Hummingbird WPS Service

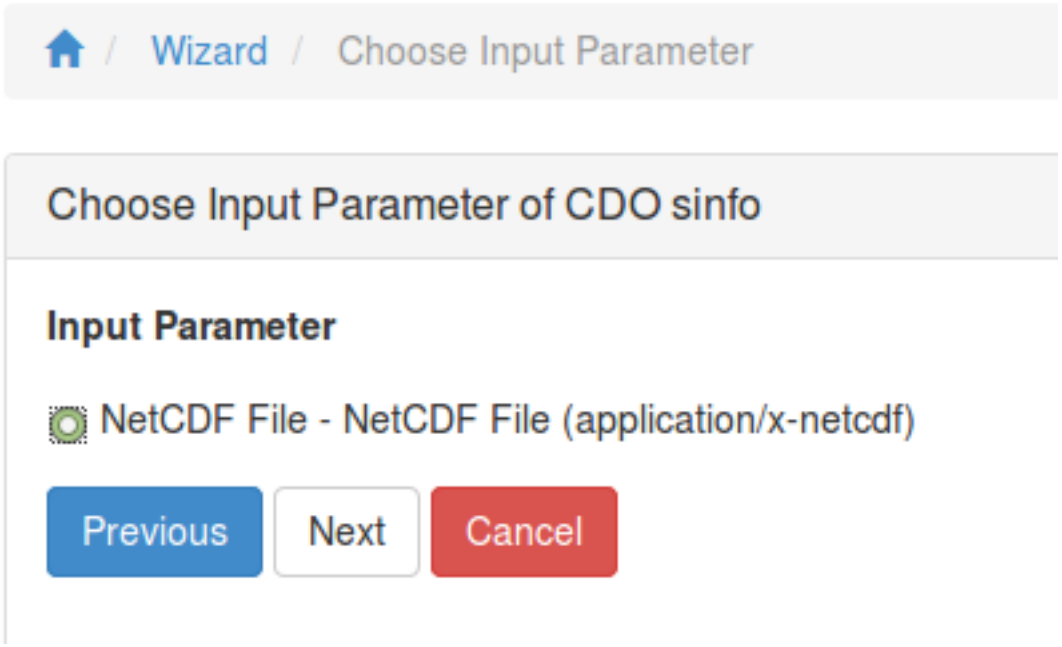
For this example choose the Hummingbird WPS service which has CDO processes.



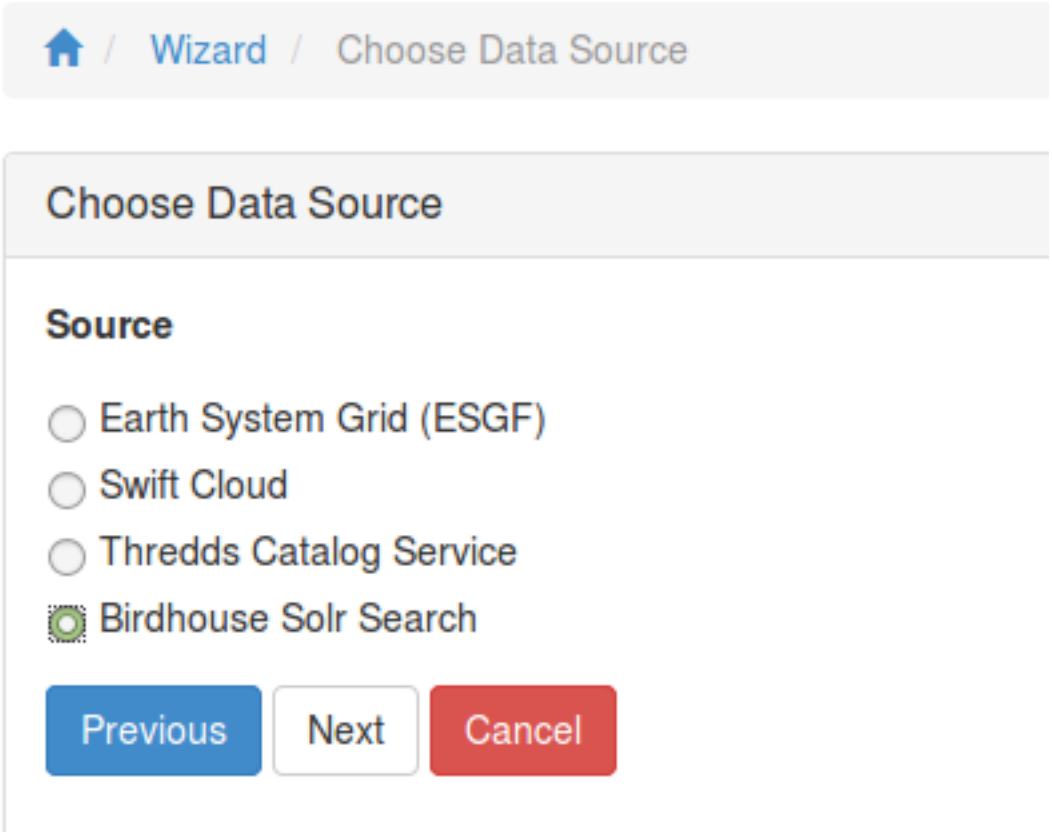
4.6.4 Choose "CDO sinfo" Process



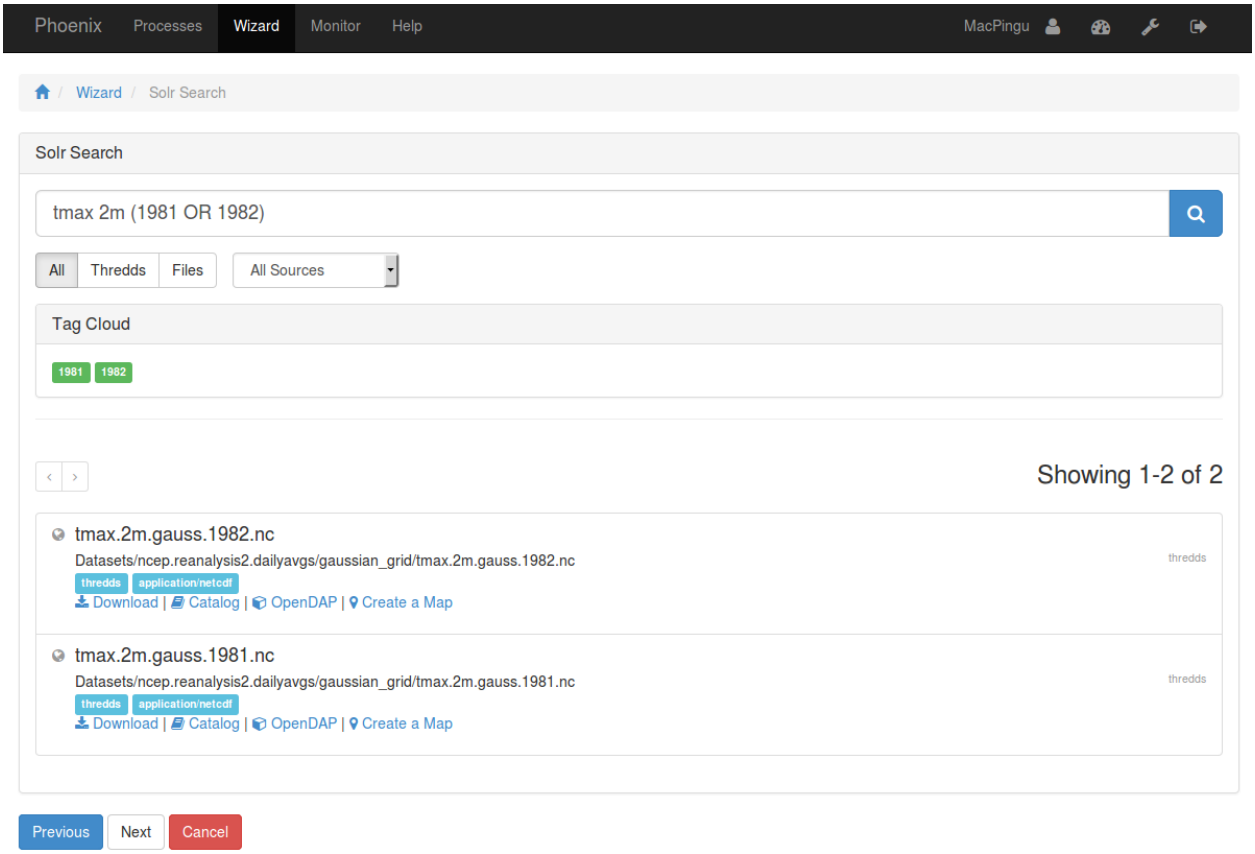
4.6.5 Choose Input Parameter



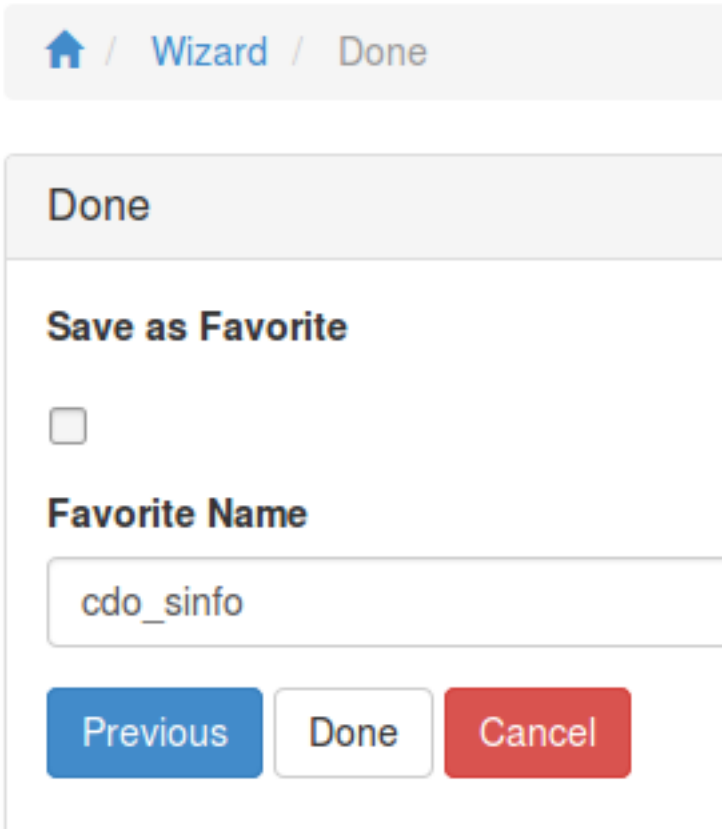
4.6.6 Choose Birdhouse Solr as Source



4.6.7 Choose Data from Solr Search

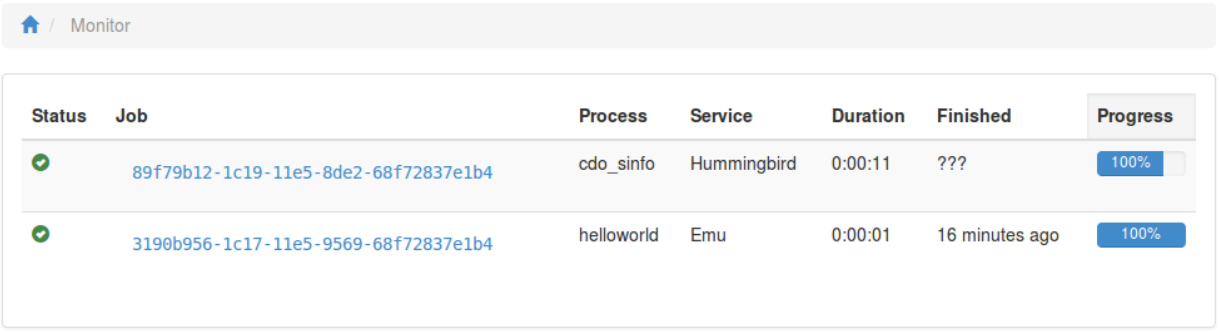


4.6.8 Start Process



4.6.9 Monitor running Job

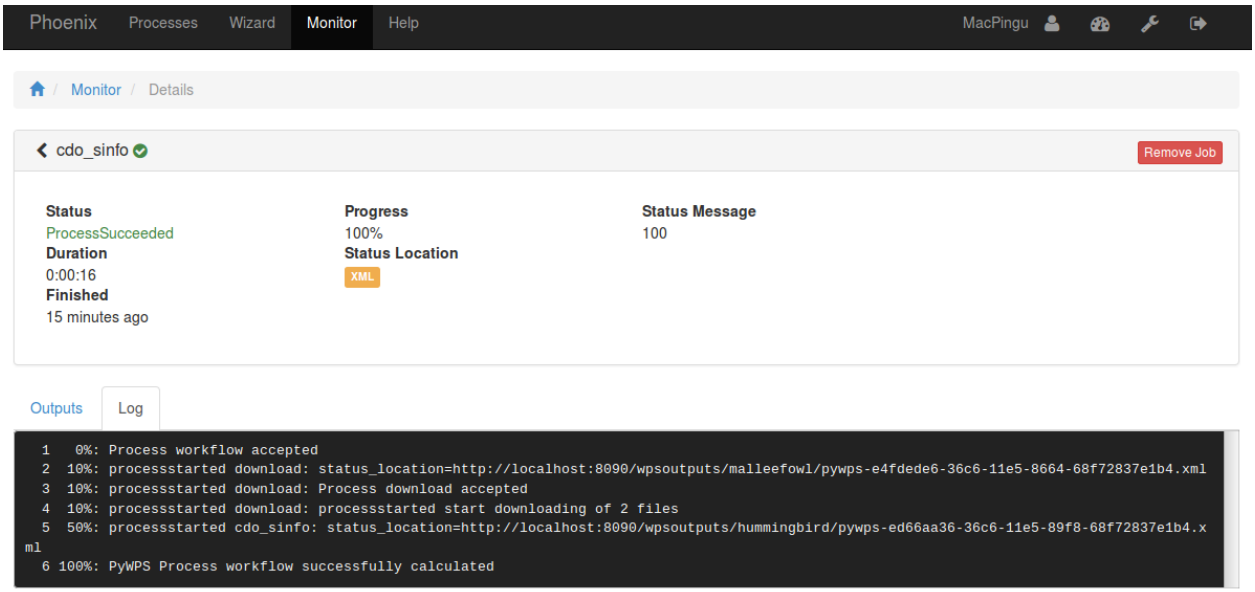
The job is now submitted and can be monitored on the *Monitor* page:



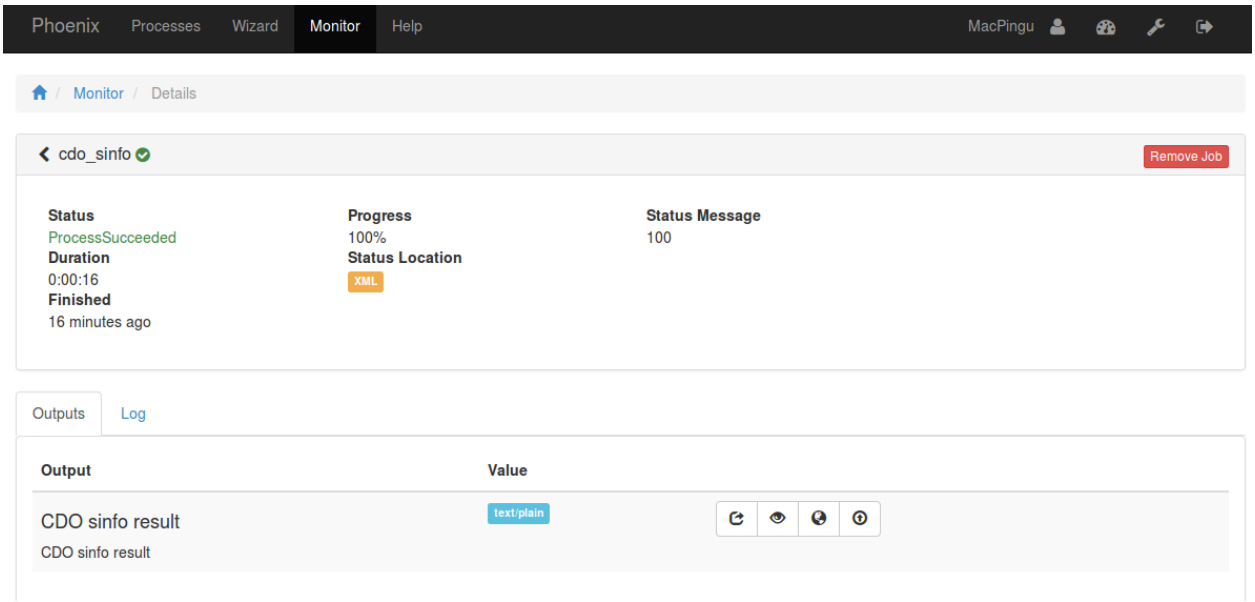
4.6.10 Display the outputs

Click on the Job ID link to get to the result of the submitted process.

Job Log


Phoenix
Processes
Wizard
Monitor
Help
MacPingu
Monitor / Details
cdo_sinfo
Remove Job
Status
ProcessSucceeded
Duration
0:00:16
Finished
15 minutes ago
Progress
100%
Status Location
XML
Status Message
100
Outputs
Log
1 0%: Process workflow accepted
2 10%: processstarted download: status_location=http://localhost:8090/wpsoutputs/malleefowl/pywps-e4fdede6-36c6-11e5-8664-68f72837e1b4.xml
3 10%: processstarted download: Process download accepted
4 10%: processstarted download: processstarted start downloading of 2 files
5 50%: processstarted cdo_sinfo: status_location=http://localhost:8090/wpsoutputs/hummingbird/pywps-ed66aa36-36c6-11e5-89f8-68f72837e1b4.xml
6 100%: PyWPS Process workflow successfully calculated

Job Outputs


Phoenix
Processes
Wizard
Monitor
Help
MacPingu
Monitor / Details
cdo_sinfo
Remove Job
Status
ProcessSucceeded
Duration
0:00:16
Finished
16 minutes ago
Progress
100%
Status Location
XML
Status Message
100
Outputs
Log
Output
Value
CDO sinfo result
CDO sinfo result
text/plain

CHAPTER 5

Troubleshooting

5.1 Phoenix does not start

Phoenix needs a running mongodb and pycsw service. Sometimes Phoenix is started when these service are not ready yet. In that case start theses services manually in the order mongodb, pycsw and Phoenix with:

```
$ source activate pyramid-phoenix     # activate conda environment used by phoenix
$ supervisorctl restart mongodb
$ supervisorctl restart pycsw
$ supervisorctl restart phoenix
```

You can also try to restart all services with:

```
$ supervisorctl restart all
```

or:

```
$ make restart
```

Check the log files to see the error messages:

```
$ tail -f  ~/birdhouse/var/log/supervisor/phoenix.log
$ tail -f  ~/birdhouse/var/log/supervisor/celery.log
```

5.2 Nginx does not start

From a former installation there might be nginx files with false permissions. Remove those files:

```
$ ~/birdhouse/etc/init.d/supervisord stop
$ sudo rm -rf ~/birdhouse/var/run
$ sudo rm -rf ~/birdhouse/var/log
$ ~/birdhouse/etc/init.d/supervisord start
```

CHAPTER 6

Sphinx AutoAPI Index

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

6.1 ldap

6.1.1 Module Contents

ldap.**includeme**(*config*)

6.2 _compat

This python 2.x/3.x compatibility modules is based on the pywps 4.x code.

6.3 db

6.3.1 Module Contents

db.**mongodb**(*registry*)

db.**includeme**(*config*)

6.4 events

6.4.1 Module Contents

class events.**JobStarted**(*request*, *task_id*)

class events.**JobFinished**(*job*)

succeeded()

class events.**SettingsChanged**(*request*, *new_settings*)

converted_settings()

6.5 layouts

6.5.1 Module Contents

class layouts.**PageLayout**(*context*, *request*)

project_title()

add_breadcrumb(*route_path*, *title*)

add_heading(*name*, **args*, ***kw*)

6.6 catalog

6.6.1 Module Contents

catalog.**includeme**(*config*)

catalog.**catalog_factory**(*registry*)

catalog.**_fetch_thredds_metadata**(*url*, *title=None*)
Fetch capabilities metadata from thredds catalog service and return record dict.

catalog.**_fetch_wps_metadata**(*url*, *title=None*)
Fetch capabilities metadata from wps service and return record dict.

class catalog.**Catalog**

get_record_by_id(*identifier*)

delete_record(*identifier*)

insert_record(*record*)

harvest(*url*, *service_type*, *service_name=None*, *service_title=None*, *public=False*, *c4i=False*)

get_service_name(*record*)
Get service name from twitcher registry for given service (url).

get_service_by_name(*name*)
Get service from twitcher registry by given service name.

get_service_by_url(*url*)
Get service from twitcher registry by given url.

get_services(*service_type=None*, *maxrecords=100*)

clear_services()

class catalog.**CatalogService**(*csw*, *service_registry*)

get_record_by_id(*identifier*)

delete_record(*identifier*)

insert_record(*record*)

harvest(*url*, *service_type*, *service_name=None*, *service_title=None*, *public=False*, *c4i=False*)

get_services(*service_type=None*, *maxrecords=100*)

catalog.**doc2record**(*document*)
Converts `document` from mongodb to a `Record` object.

class catalog.**MongodbCatalog**(*collection*, *service_registry*)
Implementation of a Catalog with MongoDB.

get_record_by_id(*identifier*)

delete_record(*identifier*)

insert_record(*record*)

harvest(*url*, *service_type*, *service_name=None*, *service_title=None*, *public=False*, *c4i=False*)

get_services(*service_type=None*, *maxrecords=100*)

clear_services()

6.7 security

see pyramid security:

- http://docs.pylonsproject.org/projects/pyramid/en/latest/tutorials/wiki2/authentication.html

6.7.1 Module Contents

security.**check_csrf_token**(*request*)

security.**has_execute_permission**(*request*, *service_name*)

security.**passwd_check**(*request*, *passphrase*)
code taken from IPython.lib.security TODO: maybe import ipython

```
>>> passwd_check('sha1:0e112c3ddfce:a68df677475c2b47b6e86d0467eec97ac5f4b85a',
...              'anotherpassword')
False
```

security.**groupfinder**(*userid*, *request*)

class security.**Root**(*request*)

security.**root_factory**(*request*)

security.**authomatic**(*request*)

security.**authomatic_config**(*request*)

class security.**MyAuthenticationPolicy**

authenticated_userid(*request*)

security.**get_user**(*request*)

security.**includeme**(*config*)

6.8 patch

6.8.1 Module Contents

patch.**patch_myproxy_client**()
Patch myproxyclient to use MESSAGE_DIGEST_TYPE sha256.

6.9 __init__

6.9.1 Package Contents

__init__.**main**(*global_config*, ***settings*)
This function returns a Pyramid WSGI application.

6.10 panels

6.10.1 Module Contents

panels.**navbar**(*context*, *request*)

panels.**messages**(*context*, *request*)

panels.**breadcrumbs**(*context*, *request*)

panels.**footer**(*context*, *request*)

panels.**headings**(*context*, *request*)

6.11 exceptions

all Exceptions defined by Phoenix ...

6.12 wps

6.12.1 Module Contents

wps.**is_opendap**(*data_input*)

wps.**check_status**(*url=None*, *response=None*, *sleep_secs=2*, *verify=False*)
Run owslib.wps check_status with additional exception handling.

Parameters **verify** – Flag to enable SSL verification. Default: False

Returns OWSLib.wps.WPSExecution object.

wps.**appstruct_to_inputs**(*request*, *appstruct*)
Transforms appstruct to wps inputs.

class wps.**WPSSchema**(*request*, *hide_complex=False*, *process=None*, *use_async=False*, *user=None*, ***kw*)
Build a Colander Schema based on the WPS data inputs.

This Schema generator is based on: http://colanderalchemy.readthedocs.io/en/latest/

TODO: fix dataType in wps client

add_async_check()

add_nodes(*process*)

literal_data(*data_input*)

colander_literal_type(*data_input*)

colander_literal_widget(*node*, *data_input*)

bbox_data(*data_input*)

complex_data(*data_input*)

_url_node_default(*data_input*)

bind(***kw*)

clone()

6.13 utils

6.13.1 Module Contents

class utils.**ActionButton**(*name*, *title=None*, *no_children=False*, *href=None*, *new_window=False*, *disabled=False*, *css_class="btn btn-default"*, *icon=None*)

url(*context*, *request*)

permitted(*context*, *request*)

utils.**pinned_processes**(*request*)

utils.**skip_csrf_token**(*appstruct*)

utils.**headline**(*text*, *max_length=120*)

utils.**make_tags**(*tags_str*)

utils.**format_tags**(*tags*)

utils.**localize_datetime**(*dt*, *tz_name="UTC"*)
Provide a timzeone-aware object for a given datetime and timezone name

utils.**is_url**(*url*)
Check wheather given text is url or not

utils.**build_url**(*url*, *query*)

utils.**wps_caps_url**(*url*)

utils.**wps_describe_url**(*url*, *identifier*)

utils.**time_ago_in_words**(*from_time*)

utils.**root_path**(*path*)

6.14 grid

6.14.1 Module Contents

grid.**get_value**(*record*, *attribute*, *default=None*)

class grid.**CustomGrid**(*request*, **args*, ***kwargs*)

checkbox_column_format(*column_number*, *i*, *record*)

render_td(*renderer*, ***data*)

label_td(*attribute*, *default=None*)

time_ago_td(*attribute*)

timestamp_td(*attribute*)

size_td(*attribute*)

userid_td(*attribute*)

user_td(*attribute*)

render_title_td(*title*, *abstract=None*, *keywords=list*, *data=list*, *format=None*, *source="#"*)

render_flag_td(*flag=False*, *tooltip=""*)

render_format_td(*format*, *source*)

render_preview_td(*format*, *source*)

render_buttongroup_td(*buttons=list*)

generate_header_link(*column_number*, *column*, *label_text*)
Override of the ObjectGrid to customize the headers. This is mostly taken from the example code in ObjectGrid itself.

default_header_column_format(*column_number*, *column_name*, *header_label*)
Override of the ObjectGrid to use <th> for header columns

default_header_ordered_column_format(*column_number*, *column_name*, *header_label*)
Override of the ObjectGrid to use <th> and to add an icon that represents the sort order for the column.

6.15 twitcherclient

6.15.1 Module Contents

twitcherclient.**includeme**(*config*)

twitcherclient.**twitcher_service_factory**(*registry*)

twitcherclient.**generate_access_token**(*registry*, *userid*, *valid_in_hours=1*)

twitcherclient.**is_public**(*registry*, *name*)

6.16 views

6.16.1 Module Contents

class views.**MyView**(*request*, *name*, *title*, *description=None*)

breadcrumbs()

views.**notfound**(*request*)
This special view just renders a custom 404 page. We do this so that the 404 page fits nicely into our global layout.

views.**add_global**(*event*)

views.**unknown_failure**(*request*, *exc*)

views.**favicon_view**(*request*)

views.**robotstxt_view**(*request*)

class views.**Home**(*request*)

view()

6.17 tasks

6.17.1 Submodules

tasks.esgflogon

Module Contents

tasks.esgflogon.**esgf_logon**(*self*, *userid*, *hostname*, *username*, *password*)

tasks.execute

Module Contents

tasks.execute.**execute_process**(*self*, *url*, *service_name*, *identifier*, *inputs*, *outputs*, *async=True*, *userid=None*, *caption=None*)

tasks.solr

Module Contents

tasks.solr.**index_thredds**(*self*, *url*, *maxrecords=-1*, *depth=2*)

tasks.solr.**clear_index**(*self*)

tasks.utils

Module Contents

tasks.utils.**task_result**(*task_id*)

tasks.utils.**wait_secs**(*run_step=-1*)

tasks.utils.**dump_json**(*obj*)

tasks.utils.**save_log**(*job*, *error=None*)

tasks.utils.**add_job**(*db*, *task_id*, *process_id*, *title=None*, *abstract=None*, *service_name=None*, *service=None*, *status_location=None*, *is_workflow=False*, *caption=None*, *userid=None*, *async=True*)

tasks.utils.**get_access_token**(*userid*)

tasks.utils.**wps_headers**(*userid*)

tasks.workflow

Module Contents

tasks.workflow.**execute_workflow**(*self*, *userid*, *url*, *service_name*, *workflow*, *caption=None*)

6.18 processes

6.18.1 Subpackages

processes.views

Submodules

processes.views.actions

Module Contents

processes.views.actions.**includeme**(*config*)

class processes.views.actions.**ProcessesActions**(*context*, *request*)

Actions related to processes.

list_processes()

processes.views.execute

Module Contents

class processes.views.execute.**ExecuteProcess**(*request*)

breadcrumbs()

appstruct()

generate_form(*formid="deform"*)

process_form(*form*)

execute(*appstruct*)

view()

processes.views.execute_json

Module Contents

class processes.views.execute_json.**ExecuteProcessJson**(*request*)

jsonify(*value*)

view()

processes.views.list

Module Contents

class processes.views.list.**ProcessList**(*request*)

view()

processes.views.list_json

Module Contents

class processes.views.list_json.**ProcessListJson**(*request*)

view()

processes.views.overview

Module Contents

class processes.views.overview.**Overview**(*request*)

wps_services()

pinned_processes()

view()

processes.views.overview_json

Module Contents

class processes.views.overview_json.**OverviewJson**(*request*)

view()

6.18.2 Package Contents

processes.**includeme**(*config*)

6.19 providers

6.19.1 Submodules

providers.esgfopenid

Providers which implement the |openid|_ protocol based on the ‘python-openid‘_ library. .. warning:

```
This providers are dependent on the |pyopenid|_ package.
```

Module Contents

class providers.esgfopenid.**MyFetcher**

_urlopen()

class providers.esgfopenid.**ESGFOpenID**(**args*, ***kwargs*)
ESGF OpenID provider with a common provider url template "https://{hostname}/esgf-idp/openid/{username}".

6.20 storage

6.20.1 Submodules

storage.views

Module Contents

storage.views.**download**(*request*)

storage.views.**delete**(*request*)
A DELETE request. If found, deletes a file with the corresponding UUID from the servers filesystem.

storage.views.**upload**(*request*)

storage.views.**handle_delete**(*request*, *uuid*)
Handles a filesystem delete based on UUID.

storage.views.**handle_upload**(*request*, *attrs*)
Handle a chunked or non-chunked upload.

See example code: https://github.com/FineUploader/server-examples/blob/master/python/flask-fine-uploader/app.py

storage.views.**save_chunk**(*fs*, *path*)
Save an uploaded chunk.

Chunks are stored in chunks/

storage.views.**combine_chunks**(*total_parts*, *source_folder*, *dest*)
Combine a chunked file into a whole file again. Goes through each part, in order, and appends that part's bytes to another destination file.

Chunks are stored in chunks/

6.20.2 Package Contents

storage.**includeme**(*config*)

6.21 services

6.21.1 Subpackages

services.views

Submodules

services.views.actions

Module Contents

class services.views.actions.**ServiceActions**(*context*, *request*)
Actions related to service registration.

remove_service()

clear_services()

services.views.actions.**includeme**(*config*)
Pyramid includeme hook.

Parameters **config** (pyramid.config.Configurator) – app config

services.views.registerservice

Module Contents

class services.views.registerservice.**Schema**

class services.views.registerservice.**RegisterService**(*request*)

breadcrumbs()

generate_form()

process_form(*form*)

view()

services.views.services

Module Contents

class services.views.services.**Services**(*request*)

breadcrumbs()

details_view()

list_view()

6.21.2 Package Contents

services.**includeme**(*config*)

6.22 wizard

6.22.1 Subpackages

wizard.views

Submodules

wizard.views.complexinputs

Module Contents

wizard.views.complexinputs.**includeme**(*config*)

wizard.views.complexinputs.**deferred_widget**(*node*, *kw*)

class wizard.views.complexinputs.**Schema**

class wizard.views.complexinputs.**ComplexInputs**(*request*)

breadcrumbs()

schema()

success(*appstruct*)

next_success(*appstruct*)

view()

custom_view()

wizard.views.done

Module Contents

wizard.views.done.**includeme**(*config*)

class wizard.views.done.**DoneSchema**

class wizard.views.done.**Done**(*request*)

breadcrumbs()

schema()

workflow_description()

success(*appstruct*)

next_success(*appstruct*)

view()

wizard.views.esgfsearch

Module Contents

wizard.views.esgfsearch.**includeme**(*config*)

class wizard.views.esgfsearch.**ESGFSearchView**(*request*)

breadcrumbs()

schema()

appstruct()

next_ok()

next_success(*appstruct*)

custom_view()

wizard.views.literalinputs

Module Contents

wizard.views.literalinputs.**includeme**(*config*)

class wizard.views.literalinputs.**LiteralInputs**(*request*)

breadcrumbs()

schema()

next_success(*appstruct*)

view()

custom_view()

wizard.views.solrsearch

Module Contents

wizard.views.solrsearch.**includeme**(*config*)

class wizard.views.solrsearch.**SolrSearch**(*request*)

breadcrumbs()

schema()

appstruct()

next_success(*appstruct*)

view()

wizard.views.source

Module Contents

wizard.views.source.**includeme**(*config*)

class wizard.views.source.**SourceSchemaNode**

after_bind(*node*, *kw*)

class wizard.views.source.**Schema**

class wizard.views.source.**ChooseSource**(*request*)

breadcrumbs()

schema()

next_success(*appstruct*)

view()

wizard.views.start

Module Contents

wizard.views.start.**includeme**(*config*)

wizard.views.start.**job_to_state**(*request*, *job_id*)

class wizard.views.start.**FavoriteSchema**

deferred_favorite_widget(*kw*)

class wizard.views.start.**Start**(*request*)

schema()

appstruct()

success(*appstruct*)

next_success(*appstruct*)

view()

wizard.views.threddsbrowser

Module Contents

class wizard.views.threddsbrowser.**Schema**

class wizard.views.threddsbrowser.**ThreddsBrowser**(*request*)

breadcrumbs()

schema()

appstruct()

next_success(*appstruct*)

custom_view()

view()

class wizard.views.threddsbrowser.**Grid**(*request*, **args*, ***kwargs*)

name_td(*col_num*, *i*, *item*)

wizard.views.threddsservice

Module Contents

wizard.views.threddsservice.**includeme**(*config*)

wizard.views.threddsservice.**deferred_widget**(*node*, *kw*)

class wizard.views.threddsservice.**Schema**

class wizard.views.threddsservice.**ThreddsService**(*request*)

breadcrumbs()

schema()

success(*appstruct*)

next_success(*appstruct*)

view()

wizard.views.wps

Module Contents

wizard.views.wps.**includeme**(*config*)

class wizard.views.wps.**ChooseWPSSchema**

deferred_validator(*kw*)

deferred_widget(*kw*)

class wizard.views.wps.**ChooseWPS**(*request*)

breadcrumbs()

schema()

next_success(*appstruct*)

view()

wizard.views.wpsprocess

Module Contents

wizard.views.wpsprocess.**includeme**(*config*)

wizard.views.wpsprocess.**count_literal_inputs**(*wps*, *identifier*)

class wizard.views.wpsprocess.**Schema**

deferred_validator(*kw*)

deferred_widget(*kw*)

class wizard.views.wpsprocess.**ChooseWPSProcess**(*request*)

breadcrumbs()

schema()

next_success(*appstruct*)

view()

custom_view()

Package Contents

class wizard.views.**WizardFavorite**(*request*, *session*)
Stores wizard state in session with a name (favorite). TODO: implement as a dict?

names()

get(*name*)

set(*name*, *state*)

clear()

class wizard.views.**WizardState**(*session*, *initial_step="wizard"*, *final_step="wizard_done"*)

load(*state*)

dump()

current_step()

is_first()

is_last()

next(*step*)

previous()

get(*key*, *default=None*)

set(*key*, *value*)

clear()

class wizard.views.**Wizard**(*request*, *name*, *title*, *description=None*)

buttons()

prev_ok()

next_ok()

use_ajax()

ajax_options()

success(*appstruct*)

appstruct()

schema()

previous_success(*appstruct*)

previous_failure(*validation_failure*)

next_success(*appstruct*)

next_failure(*validation_failure*)

generate_form(*formid="deform"*)

process_form(*form*, *action*)

previous()

next(*step*, *query=None*)

cancel()

custom_view()

breadcrumbs()

resources()

view()

6.22.2 Package Contents

wizard.**includeme**(*config*)

6.23 cart

6.23.1 Submodules

cart.actions

Module Contents

class cart.actions.**CartActions**(*context*, *request*)

Actions related to cart.

list_cart()

add_to_cart()

remove_from_cart()

clear_cart()

remove_item()

cart.cart

Module Contents

class cart.cart.**CartItem**(*url*, *title=None*, *abstract=None*, *mime_type=None*, *dataset=None*)

title()

abstract()

filename()

is_service()

is_opendap()

is_thredds_catalog()

to_json()

class cart.cart.**Cart**(*request*)

add_item(*url*, *title=None*, *abstract=None*, *mime_type=None*)
Add cart item.

remove_item(*url*)
Remove cart item with given url.

count()
Returns: number of cart items.

has_items()
Returns: True if cart items available, otherwise False.

clear()
Removes all items of cart and updates session.

save()
Store cart items in session.

load()
Load cart items from session.

to_json()
Returns: json representation of all cart items.

cart.views

Module Contents

class cart.views.**Cart**(*request*)

    view()

6.23.2 Package Contents

cart.**includeme**(*config*)

6.24 tests

6.24.1 Submodules

tests.test_cart

Module Contents

tests.test_cart.**test_cart**()

tests.test_cart.**test_clear_cart**()

class tests.test_cart.**CartTests**

    setUp()

    tearDown()

    test_cart_from_request()

tests.test_catalog

Module Contents

tests.test_catalog.**test_doc2record**()

tests.test_esgf_metadata

Module Contents

tests.test_esgf_metadata.**test_convert_constraints**()

tests.test_esgf_search

Module Contents

tests.test_esgf_search.**test_build_constraints_dict**()

tests.test_esgf_search.**test_date_from_filename**()

tests.test_esgf_search.**test_temporal_filter**()

tests.test_esgf_search.**test_variable_filter**()

class tests.test_esgf_search.**ESGFSearchTests**

setUp()

tearDown()

test_query_params()

test_params()

test_search_datasets()

test_search_items()

tests.test_form

Module Contents

tests.test_form.**invalid_exc**(*func*, **arg*, ***kw*)

class tests.test_form.**TestBBoxValidator**

test_default()

test_minx()

test_miny()

test_maxx()

test_maxy()

class tests.test_form.**TestURLValidator**

test_default()

test_file_scheme()

test_invalid_path()

test_invalid_relative_path()

class tests.test_form.**TestTextValidator**

test_default()

test_empty()

test_restricted_chars()

tests.test_settings

Module Contents

class tests.test_settings.**UserSettingsTests**

    setUp()

    tearDown()

    test_user_view()

class tests.test_settings.**UserSettingsFunctionalTests**

    setUp()

    test_user_view()

tests.test_utils

Module Contents

tests.test_utils.**test_headline**()

tests.test_utils.**test_time_ago_in_words**()

tests.test_utils.**test_make_tags**()

tests.test_utils.**test_format_tags**()

tests.test_wizard

Module Contents

tests.test_wizard.**test_convert_states_to_nodes**()

tests.test_wps

Module Contents

tests.test_wps.**test_check_status**()

6.25 supervisor

6.25.1 Subpackages

supervisor.views

Submodules

supervisor.views.supervisor

Module Contents

class supervisor.views.supervisor.**Supervisor**(*request*)

supervisor_process()

view()

class supervisor.views.supervisor.**Grid**(*request*, **args*, ***kwargs*)

state_td(*col_num*, *i*, *item*)

buttongroup_td(*col_num*, *i*, *item*)

supervisor.views.supervisor_log

Module Contents

class supervisor.views.supervisor_log.**SupervisorLog**(*request*)

view()

6.25.2 Package Contents

supervisor.**includeme**(*config*)

6.26 dashboard

6.26.1 Submodules

dashboard.panels

Module Contents

dashboard.panels.**dashboard_overview**(*context*, *request*)

dashboard.panels.**dashboard_people**(*context*, *request*)

dashboard.panels.**dashboard_jobs**(*context*, *request*)

dashboard.views

Module Contents

class dashboard.views.**Dashboard**(*request*)

    view()

6.26.2 Package Contents

dashboard.**includeme**(*config*)

6.27 settings

6.27.1 Subpackages

settings.views

Submodules

settings.views.ldap_config

Module Contents

class settings.views.ldap_config.**Ldap**(*request*)

    breadcrumbs()
    view()

settings.views.overview

Module Contents

class settings.views.overview.**Overview**(*request*)

    view()

settings.views.processes

Module Contents

class settings.views.processes.**Processes**(*request*)

breadcrumbs()

generate_form()

process_form(*form*)

appstruct()

view()

settings.views.solr

Module Contents

class settings.views.solr.**SolrSettings**(*request*)

breadcrumbs()

index_service()

clear_index()

view()

6.27.2 Submodules

settings.schema

Module Contents

settings.schema.**deferred_processes_widget**(*node*, *kw*)

class settings.schema.**ProcessesSchema**

class settings.schema.**AuthProtocolSchema**

class settings.schema.**LdapSchema**

6.27.3 Package Contents

settings.**includeme**(*config*)

6.28 account

6.28.1 Submodules

account.base

Module Contents

account.base.**forbidden**(*request*)

class account.base.**Account**(*request*)

schema()

generate_form()

process_form(*form*)

_handle_appstruct(*appstruct*)

send_notification(*email*, *subject*, *message*)
Sends email notification to admins.

Sends email with the pyramid_mailer module. For configuration look at documentation http://pythonhosted.org//pyramid_mailer/

add_user(*login_id*, *email=None*)

login()

login_success(*login_id*, *email=None*, *name=None*, *openid=None*, *local=False*)

login_failure(*message=None*)

logout()

register()

authomatic_login()

account.esgf

Module Contents

class account.esgf.**ESGFSchema**

class account.esgf.**ESGFAccount**

schema()

_handle_appstruct(*appstruct*)

esgf_login()

account.ldap

Module Contents

class account.ldap.**LDAPSchema**

class account.ldap.**LDAPAccount**

schema()

_handle_appstruct(*appstruct*)
Handle LDAP login.

ldap_login()

init_ldap()
Lazy LDAP connector construction

account.local

Module Contents

class account.local.**LocalSchema**

class account.local.**LocalAccount**

schema()

_handle_appstruct(*appstruct*)

sign_in()

6.28.2 Package Contents

account.**includeme**(*config*)

6.29 people

6.29.1 Subpackages

people.views

Submodules

people.views.actions

Module Contents

class people.views.actions.**Actions**(*request*)

update_esgf_certs()

forget_esgf_certs()

generate_twitcher_token()

generate_esgf_slcs_token()
Update ESGF slcs token.

forget_esgf_slcs_token()
Forget ESGF slcs token.

esgf_oauth_callback()
Convert an authorisation grant into an access token.

delete_user()

people.views.actions.**includeme**(*config*)
Pyramid includeme hook. :param config: app config :type config: pyramid.config.Configurator

people.views.list

Module Contents

class people.views.list.**People**(*request*)

view()

class people.views.list.**PeopleGrid**(*request*, **args*, ***kwargs*)

group_td(*col_num*, *i*, *item*)

buttongroup_td(*col_num*, *i*, *item*)

people.views.profile

Module Contents

class people.views.profile.**Profile**(*request*)

panel_title()

appstruct()

readonly()

schema()

generate_form()

generate_buttons()

process_form(*form*)

view()

6.29.2 Submodules

people.schema

Module Contents

class people.schema.**ProfileSchema**

class people.schema.**GroupSchema**

class people.schema.**TwitcherSchema**

class people.schema.**ESGFSLCSTokenSchema**

class people.schema.**ESGFCredentialsSchema**

6.29.3 Package Contents

people.**includeme**(*config*)

6.30 map

6.30.1 Package Contents

class map.**Map**(*request*)

view()

map.**includeme**(*config*)

6.31 solrsearch

6.31.1 Subpackages

solrsearch.views

Submodules

solrsearch.views.actions

Module Contents

class solrsearch.views.actions.**Actions**(*request*)

solrsearch.views.solrsearch

Module Contents

class solrsearch.views.solrsearch.**SolrSearch**(*request*)

view()

6.31.2 Submodules

solrsearch.panels

Module Contents

solrsearch.panels.**query_path**(*request*)

solrsearch.panels.**solrsearch_script**(*context*, *request*)

solrsearch.panels.**solrsearch**(*context*, *request*)

solrsearch.schema

Module Contents

class solrsearch.schema.**SolrSearchSchema**

solrsearch.search

Module Contents

solrsearch.search.**solr_search**(*url*, *query*, *page*, *category*, *source*, *tag*)

6.31.3 Package Contents

solrsearch.**includeme**(*config*)

6.32 geoform

6.32.1 Submodules

geoform.form

Module Contents

class geoform.form.**BBoxValidator**
Bounding-Box validator which succeeds if the bbox value has the format `minx,miny,maxx,maxy` and values are in range (`-180 <= x <=180`, `-90 <= y <=90`).

class geoform.form.**URLValidator**(*allowed_schemes=None*)
URL validator which can configured with allowed URL schemes.

class geoform.form.**TextValidator**(*restricted_chars=None*)

class geoform.form.**FileUploadValidator**(*storage*, *max_size*)
Runs all validators for file upload checks.

class geoform.form.**FileFormatAllowedValidator**(*storage*)
File format extension is allowed.

https://pythonhosted.org/pyramid_storage/

class geoform.form.**FileSizeLimitValidator**(*storage*, *max_size=2*)
File size limit validator.

You can configure the maximum size by setting the max_size option to the maximum number of megabytes that you want to allow.

geoform.widget

Module Contents

class geoform.widget.**ResourceWidget**
Renders an WPS ComplexType input widget with a cart and upload button.

It is based on deform.widget.TextInputWidget.

serialize(*field*, *cstruct*, ***kw*)

deserialize(*field*, *pstruct*)

class geoform.widget.**BBoxWidget**
Renders a BoundingBox Widget.

Attributes/Arguments template

The template name used to render the input widget. Default: `bbox`.

readonly_template The template name used to render the widget in read-only mode. Default: `readonly/bbox`.

serialize(*field*, *cstruct*, ***kw*)

deserialize(*field*, *pstruct*)

class geoform.widget.**TagsWidget**

serialize(*field*, *cstruct*, ***kw*)

deserialize(*field*, *pstruct*)

6.33 solr

6.33.1 Submodules

solr.panels

Module Contents

class solr.panels.**Schema**

class solr.panels.**SolrPanel**(*context*, *request*)

class solr.panels.**SolrIndexPanel**

panel()

class solr.panels.**SolrParamsPanel**

appstruct()

panel()

6.33.2 Package Contents

solr.**includeme**(*config*)

6.34 esgf

6.34.1 Subpackages

esgf.views

Submodules

esgf.views.esgflogon

Module Contents

class esgf.views.esgflogon.**ESGFLogon**(*request*)

appstruct()

generate_form()

process_form(*form*)

check_logon()

loading()

callback()

view()

esgf.views.esgfsearch

Module Contents

class esgf.views.esgfsearch.**ESGFSearchActions**(*request*)

search_datasets()

search_items()

6.34.2 Submodules

esgf.logon

Module Contents

esgf.logon.**save_credentials**(*registry*, *userid*, *file=None*, *filename=None*)

esgf.logon.**logon**(*username=None*, *password=None*, *hostname=None*, *interactive=False*, *outdir=None*)
Logon to MyProxy and fetch proxy certificate.

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

esgf.metadata

Module Contents

esgf.metadata.**process_constraints**(*process*)

esgf.metadata.**convert_constraints**(*url*)
converts esgf search query to constraints parameter. TODO: constraints parameter should have the same structure as the esgf query.

esgf.schema

Module Contents

class esgf.schema.**ESGFLogonSchema**

esgf.schema.**esgfsearch_validator**(*node*, *value*)

class esgf.schema.**ESGFSearchSchema**

esgf.search

Module Contents

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

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

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

esgf.search.**query_params_from_appstruct**(*appstruct*, *defaults*)

esgf.search.**build_constraint_dict**(*constraints*)

class esgf.search.**ESGFSearch**(*request*, *url=None*)

_parse_params()
parse search params.

query_params()
search params as string used for query.

params()
search params as object.

search_items()
search files and aggregations with download url and opendap url.

_run_search_items(*dataset_id*, *search_type*)

search_datasets()
search datasets according to search parameters.

esgf.slcsclient

Module Contents

esgf.slcsclient.**refresh_token**(*registry*, *token*, *userid*)

esgf.slcsclient.**save_token**(*registry*, *token*, *userid*)
Store the token in the database.

class esgf.slcsclient.**ESGFSLCSClient**(*request*)

authorize()
Redirect the user to the ESGF SLCS Server for authorisation.

callback()
Convert an authorisation grant into an access token.

refresh_token()

get_token()

save_token(*token*)
Store the token in the database.

delete_token()
Remove token from database.

get_certificate()
Generates a new private key and certificate request, submits the request to be signed by the SLCS CA and prints the resulting key/certificate pair.

Uses automatic refreshing of tokens if they have expired.

esgf.validator

Module Contents

esgf.validator.**cert_ok**(*request*, *valid_hours=3*)

6.34.3 Package Contents

esgf.**includeme**(*config*)

6.35 monitor

6.35.1 Subpackages

monitor.panels

Submodules

monitor.panels.inputs

Module Contents

monitor.panels.inputs.**collect_inputs**(*status_location=None*, *response=None*)

monitor.panels.inputs.**process_inputs**(*request*, *job_id*)

class monitor.panels.inputs.**Inputs**(*context*, *request*)

    panel()

monitor.panels.outputs

Module Contents

monitor.panels.outputs.**collect_outputs**(*status_location=None*, *response=None*)

monitor.panels.outputs.**process_outputs**(*request*, *job_id*)

class monitor.panels.outputs.**Outputs**(*context*, *request*)

    panel()

Package Contents

monitor.panels.**job_details**(*request*, *job_id*)

monitor.panels.**details**(*context*, *request*)

monitor.panels.**log**(*context*, *request*)

monitor.panels.**xml**(*context*, *request*)

monitor.views

Submodules

monitor.views.actions

Module Contents

class monitor.views.actions.**NodeActions**(*context*, *request*)

Actions related to job monitor.

_selected_children()

Get the selected children of the given context.

Result List with select children.

Return type *list*

restart_job()

delete_job()

delete_jobs()

Delete selected jobs.

delete_all_jobs()

make_public()

Make selected jobs public.

make_private()

Make selected jobs private.

set_favorite()

Set selected jobs as favorite.

unset_favorite()

Unset selected jobs as favorite.

edit_job()

active_jobs()

monitor.views.actions.**monitor_buttons**(*context*, *request*)

Build the action buttons for the monitor view based on the current state and the permissions of the user.

Result List of ActionButtons.

Return type *list*

monitor.views.actions.**download_wpsoutputs**(*request*)

monitor.views.actions.**includeme**(*config*)

Pyramid includeme hook.

Parameters **config** (pyramid.config.Configurator) – app config

monitor.views.details

Module Contents

class monitor.views.details.**Details**(*request*)

view()

monitor.views.list

Module Contents

class monitor.views.list.**CaptionSchema**
This is the form schema to add and edit form for job captions.

class monitor.views.list.**LabelsSchema**
This is the form schema to add and edit form for job tags/labels.

class monitor.views.list.**JobList**(*request*)

filter_jobs(*page=0*, *limit=10*, *tag=None*, *access=None*, *status=None*, *sort="created"*)

generate_caption_form(*formid="deform_caption"*)
This helper code generates the form that will be used to add and edit job captions based on the schema of the form.

process_caption_form(*form*)

generate_labels_form(*formid="deform_tags"*)
This helper code generates the form that will be used to add and edit job tags/labels based on the schema of the form.

process_labels_form(*form*)

view()

class monitor.views.list.**JobsGrid**(*request*, **args*, ***kwargs*)

status_td(*col_num*, *i*, *item*)

duration_td(*col_num*, *i*, *item*)

caption_td(*col_num*, *i*, *item*)

labels_td(*col_num*, *i*, *item*)

buttongroup_td(*col_num*, *i*, *item*)

monitor.views.list_json

Module Contents

class monitor.views.list_json.**JobListJson**(*request*)

view()

monitor.views.status

Module Contents

class monitor.views.status.**JobStatus**(*request*)

    view()

Package Contents

monitor.views.**notify_job_started**(*event*)

monitor.views.**notify_job_finished**(*event*)

6.35.2 Submodules

monitor.utils

Module Contents

monitor.utils.**escape_output**(*output*)

monitor.utils.**output_details**(*request*, *output*)

6.35.3 Package Contents

monitor.**includeme**(*config*)

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

t

U

V

W

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X