PAVICS Documentation

Release 0.1

Ouranos & CRIM

Apr 16, 2024

CONTENTS

PAVICS is a research platform dedicated to climate analysis and visualization. It bundles data search, analytics and visualization services. PAVICS is developped by Ouranos, CRIM and the birdhouse community and been funded by the CANARIE research software program.

To get a sense of what the platform can do, check out our JupyterLab environment.

We plan to build extensive documentation for PAVICS. We're just getting started, but please provide your comments on our issue tracker.

- *Notebooks* and *tutorials* provides step by step instruction on how to use PAVICS. Start here to get a feeling of what can be done.
- *Developer Documentation* explains how to install and configure the various components and run system tests.
- *System Architecture* describes the individual components of the system and how they work together.
- *Processes* documents all available processes on the PAVICS platform.

CHAPTER

ONE

CONTENTS

1.1 Tutorials

1.1.1 Bias correction process

This example shows how to call the bias-correction service based on the Kernel Density Distribution Mapping. We first connect to the WPS server and get some information on the service, identified by id `kddm_bc`.

```
from owslib.wps import WebProcessingService

url = "https://pavics.ouranos.ca/twitcher/ows/proxy/flyingpigeon/wps"
wps = WebProcessingService(url=url)
proc = wps.describeprocess("kddm_bc")
print(proc.abstract)
```

```
---------------------------------------------------------------------------
ServiceException                          Traceback (most recent call last)
Cell In[1], line 4
      1 from owslib.wps import WebProcessingService
      3 url = "https://pavics.ouranos.ca/twitcher/ows/proxy/flyingpigeon/wps"
----> 4 wps = WebProcessingService(url=url)
      5 proc = wps.describeprocess("kddm_bc")
      6 print(proc.abstract)

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/owslib/wps.py:258, in WebProcessingService.__init__(self, url, version,␣
↪username, password, skip_caps, headers, verify, cert, timeout, auth, language)
    255 self.languages = None
    257 if not skip_caps:
--> 258     self.getcapabilities()

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/owslib/wps.py:277, in WebProcessingService.getcapabilities(self, xml)
    275     self._capabilities = reader.readFromString(xml)
    276 else:
--> 277     self._capabilities = reader.readFromUrl(
    278         self.url, headers=self.headers)
    280 log.debug(element_to_string(self._capabilities))
    282 # populate the capabilities metadata obects from the XML tree
```

(continues on next page)

(continued from previous page)

```
File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/owslib/wps.py:546, in WPSCapabilitiesReader.readFromUrl(self, url, username,␣
↪password, headers, verify, cert)
    539 def readFromUrl(self, url, username=None, password=None,
    540                 headers=None, verify=None, cert=None):
    541     """
    542     Method to get and parse a WPS capabilities document, returning an␣
↪elementtree instance.
    543
    544     :param str url: WPS service base url, to which is appended the HTTP␣
↪parameters: service, version, and request.
    545     """
--> 546     return self._readFromUrl(url,
    547                              {'service': 'WPS', 'request':
    548                                  'GetCapabilities', 'version': self.version},
    549                              self.timeout,
    550                              username=username, password=password,
    551                              headers=headers, verify=verify, cert=cert)

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/owslib/wps.py:503, in WPSReader._readFromUrl(self, url, data, timeout, method,
↪ username, password, headers, verify, cert)
    501     # split URL into base url and query string to use utility function
    502     spliturl = request_url.split('?')
--> 503     u = openURL(spliturl[0], spliturl[
    504                 1], method='Get', username=self.auth.username, password=self.
↪auth.password,
    505                 headers=headers, verify=self.auth.verify, cert=self.auth.cert,␣
↪timeout=self.timeout)
    506     return etree.fromstring(u.read())
    508 elif method == 'Post':

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/owslib/util.py:212, in openURL(url_base, data, method, cookies, username,␣
↪password, timeout, headers, verify, cert, auth)
    209 req = requests.request(method.upper(), url_base, headers=headers, **rkwargs)
    211 if req.status_code in [400, 401]:
--> 212     raise ServiceException(req.text)
    214 if req.status_code in [404, 500, 502, 503, 504]:    # add more if needed
    215     req.raise_for_status()

ServiceException: <?xml version="1.0" encoding="utf-8"?>
<ExceptionReport version="1.0.0"
    xmlns="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">
    <Exception exceptionCode="NoApplicableCode" locator="NotAcceptable">
        <ExceptionText>Request failed: HTTPConnectionPool(host=&#x27;flyingpigeon&#x27;,␣
↪port=8093): Max retries exceeded with url: /wps/wps?service=WPS&amp;
↪request=GetCapabilities&amp;version=1.0.0 (Caused by NewConnectionError(&#x27;&lt;
↪urllib3.connection.HTTPConnection object at 0x7fd75097bd10&gt;: Failed to establish a␣
```

(continues on next page)

(continued from previous page)

```
↪new connection: [Errno -3] Try again&#x27;))</ExceptionText>
    </Exception>
</ExceptionReport>
```

For the next step, we'll create small synthetic files and run the process.

```
import numpy as np
import pandas as pd
import xarray as xr

obs_time_index = pd.date_range(start="2000-01-01", end="2000-12-31", freq="D")
obs = xr.DataArray(
    np.arange(len(obs_time_index)), coords={"time": obs_time_index}, dims="time"
)
ref = obs + 1

fut_time_index = pd.date_range(start="2050-01-01", end="2050-12-31", freq="D")
fut = xr.DataArray(
    np.arange(len(fut_time_index)) + 10, coords={"time": fut_time_index}, dims="time"
)

fn = {"obs": "/tmp/obs.nc", "ref": "/tmp/ref.nc", "fut": "/tmp/fut.nc"}

obs.to_netcdf(fn["obs"])
ref.to_netcdf(fn["ref"])
fut.to_netcdf(fn["fut"])

resp = wps.execute(
    "kddm_bc", inputs=[("obs", fn["obs"]), ("ref", fn["ref"]), ("fut", fn["fut"])]
)
```

```
print(resp.status)
```

```
ProcessAccepted
```

Todo: Write tutorial on creating and launching workflows

1.2 Notebooks

These notebooks demonstrate a few features of the PAVICS platform: how to access data through the OPeNDAP protocol and subset and retrieve it, how to render an image using the WMS protocol, how to compute climate indices, and perform bias correction.

If you're unfamiliar with notebooks, note that typing *TAB* after an object will display a drop-down menu of the object's attributes and methods, and that you need to hit *CTRL-Enter* to run a *cell*. You can also type *?* after a function or method to display the corresponding help message.

1.2.1 Spatial and temporal subsetting

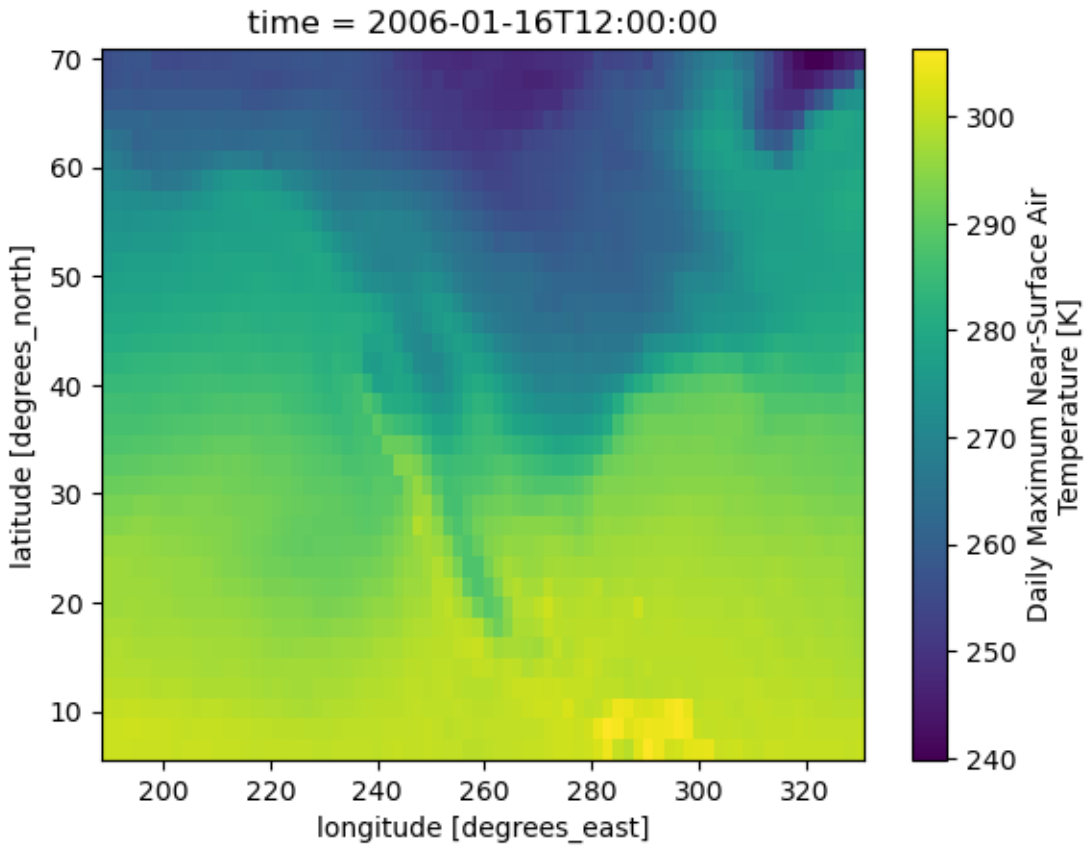
A common task in climate data analysis is subsetting files over a region of interest. Global model simulations and observations cover the entire globe, while impact analyses are often concerned with a region. Instead of downloading the entire file on a local disk, it is often more practical to subset it on the server and only download the relevant part.

This notebook shows two ways subsetting can be done. The first is to access the file using OPeNDAP, and select the coordinates of interest. The second uses processes defined on the PAVICS server.

Subsetting with OPeNDAP

Let's start with the most direct approach using the OPeNDAP data streaming protocol. For this you'll need the URL to the OPeNDAP link for the file of interst on the PAVICS THREDDS server. Browsing the THREDDS catalog, you should see for each file an OPeNDAP link whose URL contains *dodsC*. We'll use this link and simply pass it to our netCDF library, here `xarray`.

```
%matplotlib inline
import os

import xarray as xr
from matplotlib import pyplot as plt

verify_ssl = True if "DISABLE_VERIFY_SSL" not in os.environ else False

# The OPeNDAP link for the test file
dap = "https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/"
ncfile = "birdhouse/testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r1i1p1_
↪200601-200612.nc"

# Here we open the file and subset it using xarray functionality, which communicates␣
↪directly with
# the OPeNDAP server to retrieve only the data needed.
ds = xr.open_dataset(dap + ncfile)
tas = ds.tasmax
subtas = tas.sel(
    time=slice("2006-01-01", "2006-03-01"), lon=slice(188, 330), lat=slice(6, 70)
)
subtas.isel(time=0).plot()
plt.show()
```



You can then simply write the subset to a local file with `subtas.to_netcdf('subtas.nc')`.

Subset processes with WPS and Finch

For more PAVICS offers a number of subsetting processes through the FlyingPigeon WPS server:

- subset_bbox
- subset_gridpoint
- subset_polygon

The `subset_bbox` process takes the geographical coordinates of the two opposite corner of a rectangle to define the subset region, `subset_gridpoint` returns grid cells closest to the given coordinates, while `subset_polygon` returns the grid cells within a user-defined polygon.

The first step to launch those services is to create a connexion to the WPS server using Birdy's `WPSClient`.

```
from birdy import WPSClient

url = "https://pavics.ouranos.ca/twitcher/ows/proxy/finch/wps"
wps = WPSClient(url, verify=verify_ssl)
```

Now we'll use `wps.subset_gridpoint`, so let's first check what arguments it expects and pass those to the function.

```
help(wps.subset_gridpoint)
```

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

subset_gridpoint(resource, lon, lat=None, start_date=None, end_date=None, variable=None)␣
↪method of birdy.client.base.WPSClient instance
    Return the data for which grid cells includes the point coordinates for each input␣
↪dataset as well as the time range selected.

    Parameters
    ----------
    resource : ComplexData:mimetype:`application/x-netcdf`, :mimetype:`application/x-ogc-
↪dods`
        NetCDF files, can be OPEnDAP urls.
    lon : string
        Longitude coordinate. Accepts a comma separated list of floats for multiple grid␣
↪cells.
    lat : string
        Latitude coordinate. Accepts a comma separated list of floats for multiple grid␣
↪cells.
    start_date : string
        Initial date for temporal subsetting. Can be expressed as year (%Y), year-month (
↪%Y-%m) or year-month-day(%Y-%m-%d). Defaults to first day in file.
    end_date : string
        Final date for temporal subsetting. Can be expressed as year (%Y), year-month (
↪%Y-%m) or year-month-day(%Y-%m-%d). Defaults to last day in file.
    variable : string
        Name of the variable in the NetCDF file.

    Returns
    -------
    output : ComplexData:mimetype:`application/x-netcdf`
        netCDF output
    ref : ComplexData:mimetype:`application/metalink+xml; version=4.0`
        Metalink file storing all references to output files.
```

```
thredds = "https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/fileServer/"
ncfile = "birdhouse/testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r1i1p1_
↪200601-200612.nc"
resp = wps.subset_gridpoint(resource=thredds + ncfile, lon=200, lat=50)
```

The response we're getting can either include the data itself or a reference to the data. Using the `get` method of the response object, we'll get what was included in the response. If the response holds only a reference (link) to the output, we can retrieve it using the `get(as_obj=True)` method. Birdy will then inspect the file format of each output and try to find the appropriate way to open the file and return a Python object. A warning is issued if no converter is found, in which case the original reference is returned.

```
res = resp.get()
print("URL: ", res.output)
res = resp.get(asobj=True)
res.output
```

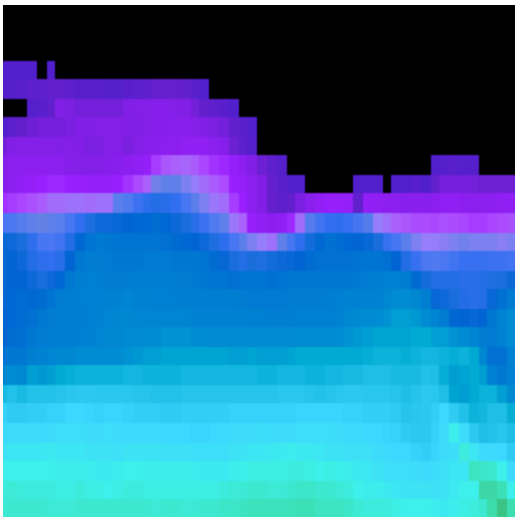
```
URL:  https://pavics.ouranos.ca/wpsoutputs/finch/2c9a8282-fc1f-11ee-ac94-0242ac130003/
↪tasmax_amon_mpi_esm_mr_rcp45_r1i1p1_200601_200612_sub.nc
```

```
<xarray.Dataset> Size: 385B
Dimensions:             (time: 12, region: 1, bnds: 2)
Coordinates:
  * time                (time) datetime64[ns] 96B 2006-01-16T12:00:00 ... 200...
    lat                 float64 8B ...
    lon                 float64 8B ...
Dimensions without coordinates: region, bnds
Data variables:
    time_bnds           (region, time, bnds) datetime64[ns] 192B ...
    lat_bnds            (region, bnds) float64 16B ...
    lon_bnds            (region, bnds) float64 16B ...
    latitude_longitude  (region) |S1 1B ...
    tasmax              (region, time) float32 48B ...
Attributes: (12/27)
    institution:            Max Planck Institute for Meteorology
    institute_id:           MPI-M
    experiment_id:          rcp45
    source:                 MPI-ESM-MR 2011; URL: http://svn.zmaw.de/svn/cosm...
    model_id:               MPI-ESM-MR
    forcing:                GHG,Oz,SD,Sl,Vl,LU
    ...                     ...
    table_id:               Table Amon (27 April 2011) a5a1c518f52ae340313ba0...
    title:                  MPI-ESM-MR model output prepared for CMIP5 RCP4.5
    parent_experiment:      historical
    modeling_realm:         atmos
    realization:            1
    cmor_version:           2.6.0
```

1.2.2 Gridded Data Renderer

PAVICS uses THREDDS WMS service to render netCDF data on a map canvas. The WMS `GetMap` operation passers the layer identification (`<variable name>`), styles, figure size and format, projection and color range and the server returns an image that can be displayed in a figure or a map canvas.

```
from IPython.display import Image
from owslib.wms import WebMapService

server = "https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/"
path = "/birdhouse/testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_
↪200601-200612.nc"
wms = WebMapService(server + "wms" + path, version="1.3.0")
resp = wms.getmap(
    layers=["tasmax"],
    format="image/png",
    colorscalerange=f"{250},{350}",
    size=[256, 256],
    srs="CRS:84",
    bbox=(150, 30, 250, 80),
    time="2006-02-15",
    transparent=True,
)
Image(resp.read())
```



One issue with the figure above is the colorscale range and the colormap, which do not provide a lot of contrast. So lets get a copy of the data using the OPeNDAP protocol to find the actual minimum and maximum values.

```
import xarray as xr

ds = xr.open_dataset(server + "dodsC" + path)
subtas = ds.tasmax.sel(
    time=slice("2006-02-01", "2006-03-01"), lon=slice(188, 330), lat=slice(6, 70)
)
mn, mx = subtas.min().values.tolist(), subtas.max().values.tolist()
mn, mx
```

```
(235.5495147705078, 306.79168701171875)
```

Now we'll simply pass those min/max to `getmap` with the colorscalerange parameter, and change the palette in the same go using the `styles` parameter. The supported styles are stored in the layer's metadata.

```
sorted(wms.contents["tasmax"].styles.keys())
```

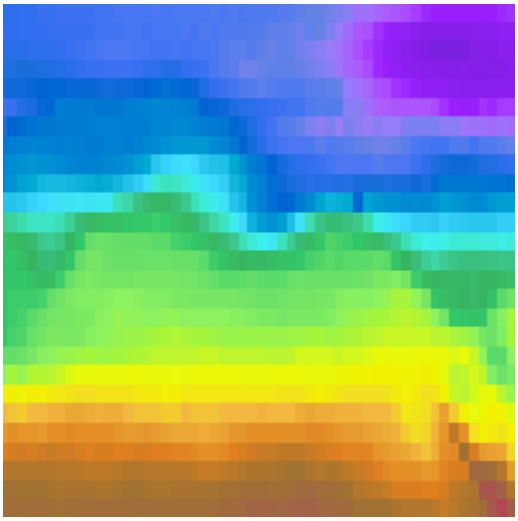
```
['boxfill/alg',
 'boxfill/alg2',
 'boxfill/ferret',
 'boxfill/greyscale',
 'boxfill/ncview',
 'boxfill/occam',
 'boxfill/occam_pastel-30',
 'boxfill/rainbow',
 'boxfill/redblue',
 'boxfill/sst_36']
```

```
resp = wms.getmap(
    layers=["tasmax"],
    styles=["boxfill/occam"],
    format="image/png",
    colorscalerange=f"{mn},{mx}",
    size=[256, 256],
```

(continues on next page)

(continued from previous page)

```
    srs="CRS:84",
    bbox=(150, 30, 250, 80),
    time="2006-02-15",
    transparent=True,
)
Image(resp.read())
```



1.2.3 Data Access Protocol

The Data Access Protocol (DAP) is a standard allowing requests to selected elements of a remote file. The NetCDF library includes support for DAP, which makes accessing remote links as easy as accessing files on disk.

In practice this works fairly well except with two caveats, latency and authorization. Each read operation on the file goes through an http request over the net, which is then processed by a DAP server. Compared to a direct read on the file system, there is often a noticeable lag between a command and the result. This DAP server may also secure some directories, requesting users to provide a authorization in the form of a password, token or certificate.

The main advantage of using DAP is that you don't have to download the entire file. You can open the dataset remotely, access the time slice or the region of interest and make computation on the data.

Some resources:

- https://pypi.org/project/netcdf4_pydap/
- https://portal.enes.org/data/data-metadata-service/search-and-download/opendap

```
%matplotlib inline
import warnings

import netCDF4 as nc

# Disable NumPy np.bool deprecation warnings, see https://numpy.org/devdocs/release/1.20.
↪0-notes.html#deprecations
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Here we access a test server that does not require authentication.

```
url = "http://test.opendap.org:80/opendap/data/nc/20070917-MODIS_A-JPL-L2P-
↪A2007260000000.L2_LAC_GHRSST-v01.nc"
D = nc.Dataset(url)
print(D.ncattrs())
D.variables.keys()
```

```
['title', 'DSD_entry_id', 'platform', 'sensor', 'Conventions', 'references', 'institution
↪', 'contact', 'GDS_version_id', 'netcdf_version_id', 'creation_date', 'history',
↪'product_version', 'spatial_resolution', 'start_date', 'start_time', 'stop_date',
↪'stop_time', 'northernmost_latitude', 'southernmost_latitude', 'easternmost_longitude',
↪ 'westernmost_longitude', 'file_quality_index', 'comment']
```

```
dict_keys(['lat', 'lon', 'time', 'sea_surface_temperature', 'sst_dtime', 'proximity_
↪confidence', 'SSES_bias_error', 'SSES_standard_deviation_error', 'rejection_flag',
↪'confidence_flag', 'sea_surface_temperature4', 'proximity_confidence4', 'SSES_bias_
↪error4', 'SSES_standard_deviation_error4'])
```

```
lat = D.variables["lat"][:]
lon = D.variables["lon"][:]
time = D.variables["time"]
print(time.units, time[0], nc.num2date(time[0], time.units))
```

```
seconds since 1981-01-01 00:00:00 842832339 2007-09-17 00:05:39
```

```
sst = D.variables["sea_surface_temperature"]
print(sst.long_name)
print(sst.units)
print(sst.dimensions, sst.shape)
```

```
sea surface temperature
kelvin
('time', 'nj', 'ni') (1, 2030, 1354)
```

```
a = sst[:]
```

This dataset is problematic because it is entirely masked. We need to extract the data attribute to get to the original, unmasked values.

```
a.data
```

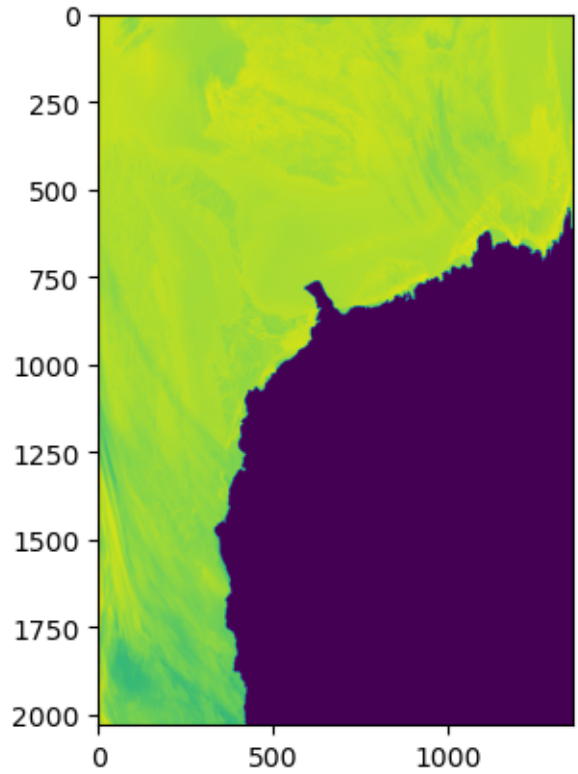
```
array([[[ -3575.,  -3529.,  -3483., ...,  -5279.,  -5268.,  -5239.],
        [ -3636.,  -3657.,  -3527., ...,  -5264.,  -5231.,  -5224.],
        [ -3669.,  -3703.,  -3696., ...,  -5270.,  -5284.,  -5243.],
        ...,
        [ -4451.,  -5568.,  -7066., ..., -32767., -32767., -32767.],
        [ -4392.,  -4950.,  -6888., ..., -32767., -32767., -32767.],
        [ -4524.,  -4661.,  -5933., ..., -32767., -32767., -32767.]]],
      dtype=float32)
```

```
from matplotlib import pyplot as plt
```

(continues on next page)

(continued from previous page)

```
plt.imshow(a.data[0])
plt.show()
```



1.2.4 Earth System Grid Federation Data Access

The Earth System Grid Federation (ESGF) has a search API that can be used by clients to query catalog content matching constraints (see API documentation). It's possible to send requests directly to the API using a simple function (see example), but here we'll use a python client named `pyesgf` to interact with the search API and get data from the ESGF THREDDS servers. The following shows examples of typical queries for data.

If a login username and credentials are required, follow these instructions.

```
# NBVAL_IGNORE_OUTPUT

from pyesgf.search import SearchConnection

# Create a connection for search on ESGF nodes. Note that setting `distrib=True` can lead␣
↪to weird failures.
conn = SearchConnection("https://esgf-node.llnl.gov/esg-search/", distrib=False)

# Launch a search query.
# Here we're looking for any variable related to humidity within the CMIP6 SSP2-4.5␣
↪experiment.
# Results will be stored in a dictionary with keys defined by the `facets` argument.
ctx = conn.new_context(
    project="CMIP6",
```

(continues on next page)

(continued from previous page)

```
    experiment_id="ssp245",
    query="humidity",
    facets="variable_id,source_id",
)

print("Number of results: ", ctx.hit_count)
print("Variables related to humidity: ")
ctx.facet_counts["variable_id"]
```

```
Number of results:  10647
Variables related to humidity:
```

```
{'tnhusscpbl': 163,
 'tnhusscp': 70,
 'tnhuspbl': 70,
 'tnhusmp': 172,
 'tnhusd': 34,
 'tnhusc': 224,
 'tnhusa': 180,
 'tnhus': 76,
 'hussLut': 36,
 'huss': 1993,
 'hus850': 190,
 'hus': 2420,
 'hursmin': 699,
 'hursmax': 682,
 'hurs': 1986,
 'hur': 1652}
```

```
# NBVAL_IGNORE_OUTPUT

# Now let's look for simulations that have the `hurs` variable and pick the first member.
ctx.constrain(variable_id="hurs", ensemble="r1i1p1f1")
ctx.facet_counts["source_id"]
```

```
{'UKESM1-0-LL': 126,
 'TaiESM1': 20,
 'NorESM2-MM': 36,
 'NorESM2-LM': 188,
 'NESM3': 21,
 'MRI-ESM2-0': 663,
 'MPI-ESM1-2-LR': 999,
 'MPI-ESM1-2-HR': 51,
 'MIROC6': 1989,
 'MIROC-ES2L': 1861,
 'MIROC-ES2H': 84,
 'MCM-UA-1-0': 8,
 'KIOST-ESM': 30,
 'KACE-1-0-G': 68,
 'IPSL-CM6A-LR': 236,
 'INM-CM5-0': 26,
```

(continues on next page)

(continued from previous page)

```
 'INM-CM4-8': 26,
 'IITM-ESM': 17,
 'HadGEM3-GC31-LL': 110,
 'GISS-E2-2-G': 40,
 'GISS-E2-1-H': 84,
 'GISS-E2-1-G-CC': 8,
 'GISS-E2-1-G': 325,
 'GFDL-ESM4': 20,
 'GFDL-CM4': 34,
 'FIO-ESM-2-0': 16,
 'FGOALS-g3': 88,
 'FGOALS-f3-L': 12,
 'EC-Earth3-Veg-LR': 37,
 'EC-Earth3-Veg': 100,
 'EC-Earth3-CC': 64,
 'EC-Earth3-AerChem': 2,
 'EC-Earth3': 817,
 'E3SM-1-1': 22,
 'CanESM5-CanOE': 24,
 'CanESM5-1': 208,
 'CanESM5': 1033,
 'CNRM-ESM2-1': 108,
 'CNRM-CM6-1-HR': 19,
 'CNRM-CM6-1': 78,
 'CMCC-ESM2': 20,
 'CMCC-CM2-SR5': 18,
 'CIESM': 9,
 'CESM2-WACCM': 168,
 'CESM2': 184,
 'CAS-ESM2-0': 20,
 'CAMS-CSM1-0': 6,
 'BCC-CSM2-MR': 20,
 'AWI-CM-1-1-MR': 19,
 'ACCESS-ESM1-5': 408,
 'ACCESS-CM2': 77}
```

```
# We can now refine the search and get datasets corresponding within our search context
results = ctx.constrain(source_id="CanESM5").search()
r = results[0]
r.dataset_id
```

```
'CMIP6.ScenarioMIP.CCCma.CanESM5.ssp245.r14i1p2f1.Amon.hus.gn.v20190429|crd-esgf-drc.ec.
↪gc.ca'
```

```
# To get file download links, there's an extra step
file_ctx = r.file_context()
file_ctx.facets = "*"
files = file_ctx.search()
[f.download_url for f in files]
```

```
['http://crd-esgf-drc.ec.gc.ca/thredds/fileServer/esgD_dataroot/AR6/CMIP6/ScenarioMIP/
↪CCCma/CanESM5/ssp245/r14i1p2f1/Amon/hus/gn/v20190429/hus_Amon_CanESM5_ssp245_r14i1p2f1_
↪gn_201501-210012.nc']
```

```
# Instead of a download URL, we can also get OPeNDAP links.
urls = [f.opendap_url for f in files]
print(urls)

# It's sometimes possible to request aggregations of multiple netCDF into one OPeNDAP␣
↪link,
# but this option is often unavailable.
agg_ctx = r.aggregation_context()
agg_ctx.facets = "*"
agg = agg_ctx.search()[0]
print(agg.opendap_url)
```

```
['http://crd-esgf-drc.ec.gc.ca/thredds/dodsC/esgD_dataroot/AR6/CMIP6/ScenarioMIP/CCCma/
↪CanESM5/ssp245/r14i1p2f1/Amon/hus/gn/v20190429/hus_Amon_CanESM5_ssp245_r14i1p2f1_gn_
↪201501-210012.nc']
```

```
None
```

```
# Open the opendap link with xarray
import xarray as xr

ds = xr.open_mfdataset(urls)
ds
```

```
<xarray.Dataset> Size: 643MB
Dimensions:    (time: 1032, bnds: 2, plev: 19, lat: 64, lon: 128)
Coordinates:
  * time       (time) object 8kB 2015-01-16 12:00:00 ... 2100-12-16 12:00:00
  * plev       (plev) float64 152B 1e+05 9.25e+04 8.5e+04 ... 1e+03 500.0 100.0
  * lat        (lat) float64 512B -87.86 -85.1 -82.31 ... 82.31 85.1 87.86
  * lon        (lon) float64 1kB 0.0 2.812 5.625 8.438 ... 351.6 354.4 357.2
Dimensions without coordinates: bnds
Data variables:
    time_bnds  (time, bnds) object 17kB dask.array<chunksize=(1032, 2), meta=np.ndarray>
    lat_bnds   (lat, bnds) float64 1kB dask.array<chunksize=(64, 2), meta=np.ndarray>
    lon_bnds   (lon, bnds) float64 2kB dask.array<chunksize=(128, 2), meta=np.ndarray>
    hus        (time, plev, lat, lon) float32 643MB dask.array<chunksize=(1032, 19, 64,␣
↪128), meta=np.ndarray>
Attributes: (12/54)
    CCCma_model_hash:                fc4bb7db954c862d023b546e19aec6c588bc0552
    CCCma_parent_runid:              p2-his14
    CCCma_pycmor_hash:               26c970628162d607fffd14254956ebc6dd3b6f49
    CCCma_runid:                     p2-s4514
    Conventions:                     CF-1.7 CMIP-6.2
    YMDH_branch_time_in_child:       2015:01:01:00
    ...                              ...
    variable_id:                     hus
```

(continues on next page)

(continued from previous page)

```
    variant_label:                   r14i1p2f1
    version:                         v20190429
    license:                         CMIP6 model data produced by The Governm...
    cmor_version:                    3.5.0
    DODS_EXTRA.Unlimited_Dimension:  time
```

1.2.5 Accessing PAVICS THREDDS Server

The THREDDS data storing netCDF file on PAVICS has some public and private directories. Data from public directories can be accessed anonymously, while data from private directories require authentication. This notebook shows how to access public and private data on the THREDDS server.

The PAVICS THREDDS server has a `testdata/` folder, in which we store test datasets to validate process requests. Within that directory is a `secure/` folder whose file access requires authentication.

```
# define some useful variables for following steps
import os

PAVICS_HOST = os.getenv("PAVICS_HOST", "pavics.ouranos.ca")
THREDDS_URL = f"https://{PAVICS_HOST}/twitcher/ows/proxy/thredds"

assert PAVICS_HOST != "", "Invalid PAVICS HOST value."
print("THREDDS URL:", THREDDS_URL)
```

```
THREDDS URL: https://pavics.ouranos.ca/twitcher/ows/proxy/thredds
```

First let's just open an unsecured link.

```
# NBVAL_IGNORE_OUTPUT

import xarray as xr

PUBLIC_URL = f"{THREDDS_URL}/dodsC/birdhouse/testdata/ta_Amon_MRI-CGCM3_decadal1980_
↪r1i1p1_199101-200012.nc"
ds = xr.open_dataset(PUBLIC_URL)
ds
```

```
<xarray.Dataset> Size: 565MB
Dimensions:    (time: 120, bnds: 2, plev: 23, lat: 160, lon: 320)
Coordinates:
  * time       (time) datetime64[ns] 960B 1991-01-16T12:00:00 ... 2000-12-16T...
  * plev       (plev) float64 184B 1e+05 9.25e+04 8.5e+04 ... 200.0 100.0 40.0
  * lat        (lat) float64 1kB -89.14 -88.03 -86.91 ... 86.91 88.03 89.14
  * lon        (lon) float64 3kB 0.0 1.125 2.25 3.375 ... 356.6 357.8 358.9
Dimensions without coordinates: bnds
Data variables:
    time_bnds  (time, bnds) datetime64[ns] 2kB ...
    lat_bnds   (lat, bnds) float64 3kB ...
    lon_bnds   (lon, bnds) float64 5kB ...
    ta         (time, plev, lat, lon) float32 565MB ...
Attributes: (12/28)
```

(continues on next page)

(continued from previous page)

```
    institution:                        MRI (Meteorological Research Institute, ...
    institute_id:                       MRI
    experiment_id:                      decadal1980
    source:                             MRI-CGCM3 2011 atmosphere: GSMUV (gsmuv-...
    model_id:                           MRI-CGCM3
    forcing:                            GHG, SA, Oz, LU, Sl, Vl, BC, OC (GHG inc...
    ...                                 ...
    title:                              MRI-CGCM3 model output prepared for CMIP...
    parent_experiment:                  N/A
    modeling_realm:                     atmos
    realization:                        1
    cmor_version:                       2.7.1
    DODS_EXTRA.Unlimited_Dimension:     time
```

Now let's do the same with a secured link.

```
from webob.exc import HTTPError

SECURED_URL = f"{THREDDS_URL}/dodsC/birdhouse/testdata/secure/tasmax_Amon_MPI-ESM-MR_
↪rcp45_r2i1p1_200601-200612.nc"
try:
    ds = xr.open_dataset(SECURED_URL, decode_cf=False)
# depending on 'xarray' version, different errors are raised when failing authentication␣
↪according to how they handle it
except OSError as exc:
    # "NetCDF: Access failure" xarray >= 0.20
    # "Authorization failure" xarray < 0.17
    assert "NetCDF: Access failure" in str(exc) or "Authorization failure" in str(exc)
except HTTPError as exc:  # xarray >= 0.17
    # note: raised error is 500 with 'message' Unauthorized instead of directly raising␣
↪HTTPUnauthorized
    assert "401 Unauthorized" in str(exc)
else:
    raise RuntimeError(
        "Expected unauthorized response, but dataset open operation did not raise!"
    )
print("Unauthorized was raised as expected.")
```

```
Unauthorized was raised as expected.
```

```
syntax error, unexpected WORD_WORD, expecting SCAN_ATTR or SCAN_DATASET or SCAN_ERROR
context: <?xml^ version="1.0" encoding="utf-8"?><ExceptionReport version="1.0.0" xmlns=
↪"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"> <Exception exceptionCode="NoApplicableCode" locator=
↪"AccessForbidden"> <ExceptionText>Access to service is forbidden.</ExceptionText> </
↪Exception></ExceptionReport>
```

To open a secured link, we need to open a session with `Authentication`. Using wrong `Authentication` credentials will not work. They will raise immediately when failing login procedure. Using valid credentials will instead succeed login, but will raise a forbidden response when attempting to retrieve the data. Either way, user must be logged in and have appropriate access to fulfill `Authorization` requirements of the resource.

Let's see the result when credentials are invalid.

```
import requests
from requests_magpie import MagpieAuth, MagpieAuthenticationError

BAD_USR = "an-invalid-user"
BAD_PWD = "or-bad-password"

try:
    with requests.session() as session:
        session.auth = MagpieAuth(f"https://{PAVICS_HOST}/magpie", BAD_USR, BAD_PWD)
        xr.open_dataset(
            SECURED_URL, decode_cf=False
        )  # Attributes are problematic with this file.
# specific error depends on what raises (unauthorized, forbidden, login failure) and
↪'xarray' version
except (OSError, HTTPError, MagpieAuthenticationError) as exc:
    print("Access with invalid credentials was not permitted as expected.")
else:
    raise RuntimeError(
        "Expected authentication failure response, but login operation did not raise!"
    )
```

```
Access with invalid credentials was not permitted as expected.
```

```
syntax error, unexpected WORD_WORD, expecting SCAN_ATTR or SCAN_DATASET or SCAN_ERROR
context: <?xml^ version="1.0" encoding="utf-8"?><ExceptionReport version="1.0.0" xmlns=
↪"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"> <Exception exceptionCode="NoApplicableCode" locator=
↪"AccessForbidden"> <ExceptionText>Access to service is forbidden.</ExceptionText> </
↪Exception></ExceptionReport>
```

As we can see, the server identified that credentials were provided, but they were incorrect and could not log in. Similar result would happen if login succeeded, but user was forbidden access due to insufficient permissions.

We've created an `authtest` user in advance that has access to the `secure` contents to facilitate testing.

Let's use it now to obtain the secured resource.

```
# NBVAL_IGNORE_OUTPUT

AUTH_USR = os.getenv("TEST_MAGPIE_AUTHTEST_USERNAME", "authtest")
AUTH_PWD = os.getenv("TEST_MAGPIE_AUTHTEST_PASSWORD", "authtest1234")

# Open session
with requests.Session() as session:
    session.auth = MagpieAuth(f"https://{PAVICS_HOST}/magpie", AUTH_USR, AUTH_PWD)
    # Open a PyDAP data store and pass it to xarray
    store = xr.backends.PydapDataStore.open(SECURED_URL, session=session)
    ds = xr.open_dataset(
        store, decode_cf=False
    )  # Attributes are problematic with this file.
ds
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[5], line 10
      8     session.auth = MagpieAuth(f"https://{PAVICS_HOST}/magpie", AUTH_USR, AUTH_
↪PWD)
      9     # Open a PyDAP data store and pass it to xarray
---> 10     store = xr.backends.PydapDataStore.open(SECURED_URL, session=session)
     11     ds = xr.open_dataset(
     12         store, decode_cf=False
     13     )  # Attributes are problematic with this file.
     14 ds

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/xarray/backends/pydap_.py:110, in PydapDataStore.open(cls, url, application,␣
↪session, output_grid, timeout, verify, user_charset)
     99 @classmethod
    100 def open(
    101     cls,
   (...)
    108     user_charset=None,
    109 ):
--> 110     import pydap.client
    111     import pydap.lib
    113     if timeout is None:

ModuleNotFoundError: No module named 'pydap'
```

Successful listing of the above data means the user was granted access for this reference.

1.2.6 Analyzing the ClimEx Large Ensemble

ClimEx is a project investigating the influence of climate change on hydro-meteorological extremes. To understand the effect of natural variability on the occurrence of rare extreme events, ClimEx ran 50 independent simulations, increasing the sample size available.

Analyzing simulation outputs from large ensembles can get tedious due to the large number of individual netCDF files to handle. To simplify data access, we've created an aggregated view of daily precipitation and temperature for all 50 ClimEx realizations from 1955 to 2100.

The first step is to open the dataset, whose path can be found in the `climex` catalog. Although there are currently 36 250 individual netCDF files, there is only one link in the catalog.

```
import warnings

import numba

warnings.simplefilter("ignore", category=numba.core.errors.NumbaDeprecationWarning)

import logging

import intake
import xarray as xr
import xclim
```

(continues on next page)

(continued from previous page)

```
from clisops.core.subset import subset_gridpoint
from dask.diagnostics import ProgressBar
from dask.distributed import Client, LocalCluster
from IPython.display import HTML, Markdown, clear_output
from xclim import ensembles as xens

# fmt: off
climex = "https://pavics.ouranos.ca/catalog/climex.json"  # TEST_USE_PROD_DATA
cat = intake.open_esm_datastore(climex)
# fmt: on

cat.df.head()
```

```
                license_type                                title  \
0  permissive non-commercial  The ClimEx CRCM5 Large Ensemble

                                          institution driving_model_id  \
0  Ouranos Consortium on Regional Climatology and...   CCCma-CanESM2

  driving_experiment type processing project_id frequency modeling_realm  \
0  historical, rcp85  RCM        raw     CLIMEX       day          atmos

                                      variable_name  \
0  ['tasmin', 'tasmax', 'tas', 'pr', 'prsn', 'rot...

                                   variable_long_name  \
0  ['Daily Minimum Near-Surface Temperature minim...

                                               path
0  https://pavics.ouranos.ca/twitcher/ows/proxy/t...
```

```
# Opening the link takes a while, because the server creates an aggregated view of 435,
↪000 individual files.
url = cat.df.path[0]
ds = xr.open_dataset(url, chunks=dict(realization=2, time=30 * 3))
ds
```

```
<xarray.Dataset> Size: 4TB
Dimensions:       (rlat: 280, rlon: 280, time: 52924, realization: 50)
Coordinates:
  * rlat          (rlat) float64 2kB -12.61 -12.51 -12.39 ... 17.85 17.96 18.07
  * rlon          (rlon) float64 2kB 2.695 2.805 2.915 ... 33.17 33.28 33.39
  * time          (time) object 423kB 1955-01-01 00:00:00 ... 2099-12-30 00:0...
  * realization   (realization) |S64 3kB b'historical-r1-r10i1p1' ... b'histo...
    lat           (rlat, rlon) float32 314kB dask.array<chunksize=(280, 280), meta=np.
↪ndarray>
    lon           (rlat, rlon) float32 314kB dask.array<chunksize=(280, 280), meta=np.
↪ndarray>
Data variables:
    rotated_pole  |S64 64B ...
    tasmin        (realization, time, rlat, rlon) float32 830GB dask.array<chunksize=(2,␣
```

(continues on next page)

(continued from previous page)

```
↪90, 280, 280), meta=np.ndarray>
    tasmax        (realization, time, rlat, rlon) float32 830GB dask.array<chunksize=(2,␣
↪90, 280, 280), meta=np.ndarray>
    tas           (realization, time, rlat, rlon) float32 830GB dask.array<chunksize=(2,␣
↪90, 280, 280), meta=np.ndarray>
    pr            (realization, time, rlat, rlon) float32 830GB dask.array<chunksize=(2,␣
↪90, 280, 280), meta=np.ndarray>
    prsn          (realization, time, rlat, rlon) float32 830GB dask.array<chunksize=(2,␣
↪90, 280, 280), meta=np.ndarray>
Attributes: (12/30)
    Conventions:              CF-1.6
    DODS.dimName:             string1
    DODS.strlen:              0
    EXTRA_DIMENSION.bnds:     2
    NCO:                      "4.5.2"
    abstract:                 The ClimEx CRCM5 Large Ensemble of high-resolut...
    ...                       ...
    project_id:               CLIMEX
    rcm_version_id:           v3331
    terms_of_use:             http://www.climex-project.org/sites/default/fil...
    title:                    The ClimEx CRCM5 Large Ensemble
    type:                     RCM
    EXTRA_DIMENSION.string1:  1
```

The CLIMEX dataset currently stores 5 daily variables, simulated by CRCM5 driven by CanESM2 using the representative concentration pathway RCP8.5:

- `pr`: mean daily precipitation flux
- `prsn`: mean daily snowfall flux
- `tas`: mean daily temperature
- `tasmin`: minimum daily temperature
- `tasmax`: maximum daily temperature

There variables are stored along spatial dimensions (rotated latitude and longitude), time (1955-2100) and realizations (50 members). These members are created by first running five members from CanESM2 from 1850 to 1950, then small perturbations are applied in 1950 to spawn 10 members from each original member, to yield a total of 50 global realizations. Each realization is then dynamically downscaled to 12 km resolution over Québec.

```
ds.realization[:5].data.tolist()
```

```
[b'historical-r1-r10i1p1',
 b'historical-r1-r1i1p1',
 b'historical-r1-r2i1p1',
 b'historical-r1-r3i1p1',
 b'historical-r1-r4i1p1']
```

1.2.7 Creating maps of ClimEx fields

The data is on a rotated pole grid, and the actual geographic latitudes and longitudes of grid centroids are stored in the variables with the same name. We can plot the data directly using the native `rlat` and `rlon` coordinates easily.

```
# NBVAL_IGNORE_OUTPUT
from matplotlib import pyplot as plt

with ProgressBar():
    field = ds.tas.isel(time=0, realization=0).load()
    field.plot()
```

```
[                                        ] | 0% Completed | 181.94 us
```

```
[                                        ] | 0% Completed | 101.69 ms
```

```
[                                        ] | 0% Completed | 202.52 ms
```

```
[                                        ] | 0% Completed | 303.31 ms
```

```
[                                        ] | 0% Completed | 404.06 ms
```

```
[                                        ] | 0% Completed | 504.78 ms
```

```
[                                        ] | 0% Completed | 605.72 ms
```

```
[                                        ] | 0% Completed | 706.44 ms
```

```
[                                        ] | 0% Completed | 807.17 ms
```

```
[                                        ] | 0% Completed | 907.87 ms
```

```
[                                        ] | 0% Completed | 1.01 s
```

```
[                                        ] | 0% Completed | 1.11 s
```

```
[                                        ] | 0% Completed | 1.21 s
```

```
[                                        ] | 0% Completed | 1.31 s
```

```
[####################                    ] | 50% Completed | 1.41 s
```

```
[####################                    ] | 50% Completed | 1.51 s
```

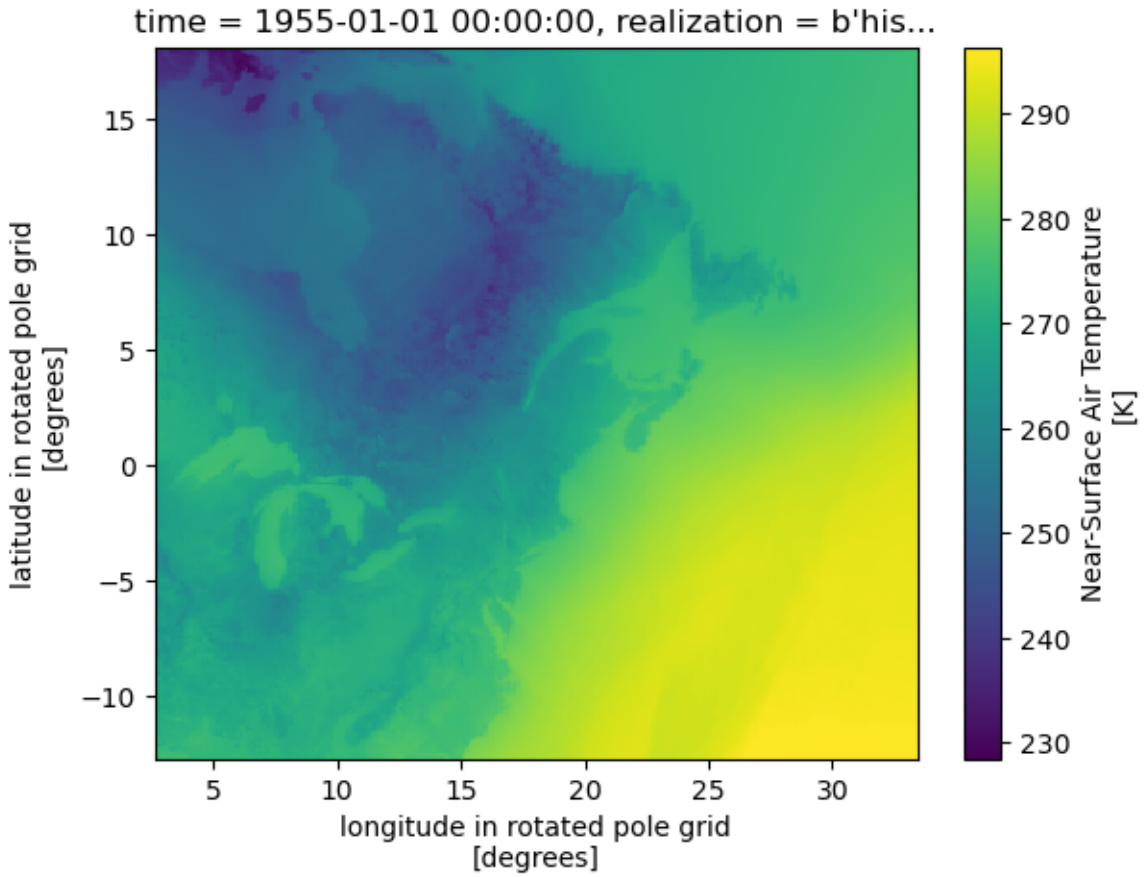
```
[####################                    ] | 50% Completed | 1.61 s
```

```
[##############################          ] | 75% Completed | 1.71 s
```

```
[############################          ] | 75% Completed | 1.81 s
```

```
[########################################] | 100% Completed | 1.92 s
```

```

```



However, the axes are defined with respect to the rotated coordinates. To create a map with correct geographic coordinates, we could use the real longitudes and latitudes (`plt.pcolormesh(ds.lon, ds.lat, field)`), but a better option is to create axes with the same projection used by the model. That is, we set the map projection to `RotatedPole`, using the coordinate reference system defined in the `rotated_pole` variable. We can use a similar approach to project the model output on another projection.

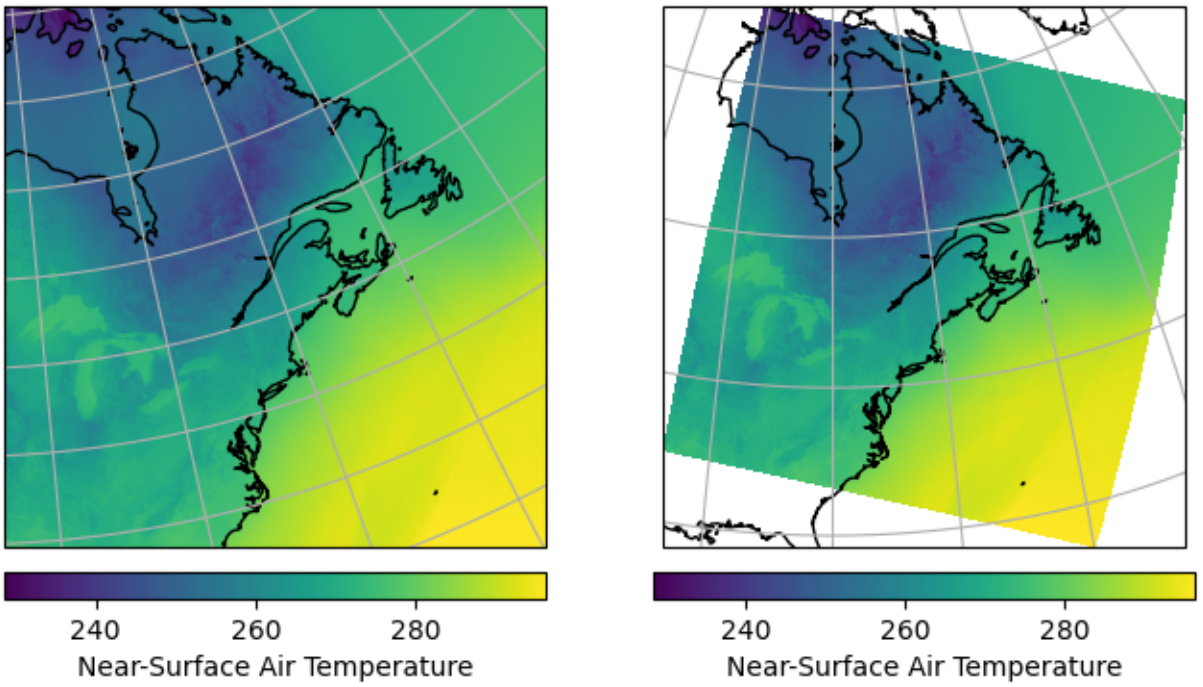
```
# NBVAL_IGNORE_OUTPUT
import cartopy.crs as ccrs

with ProgressBar():
    # The CRS for the rotated pole (data)
    rotp = ccrs.RotatedPole(
        pole_longitude=ds.rotated_pole.grid_north_pole_longitude,
        pole_latitude=ds.rotated_pole.grid_north_pole_latitude,
    )

    # The CRS for your map projection (can be anything)
```

(continues on next page)

(continued from previous page)

```
    ortho = ccrs.Orthographic(central_longitude=-80, central_latitude=45)

    fig = plt.figure(figsize=(8, 4))

    # Plot data on the rotated pole projection directly
    ax = plt.subplot(1, 2, 1, projection=rotp)
    ax.coastlines()
    ax.gridlines()
    m = ax.pcolormesh(ds.rlon, ds.rlat, field)
    plt.colorbar(
        m, orientation="horizontal", label=field.long_name, fraction=0.046, pad=0.04
    )

    # Plot data on another projection, transforming it using the rotp crs.
    ax = plt.subplot(1, 2, 2, projection=ortho)
    ax.coastlines()
    ax.gridlines()
    m = ax.pcolormesh(
        ds.rlon, ds.rlat, field, transform=rotp
    )  # Note the transform parameter
    plt.colorbar(
        m, orientation="horizontal", label=field.long_name, fraction=0.046, pad=0.04
    )

    plt.show()
```

```
ERROR 1: PROJ: proj_create_from_database: Open of /home/docs/checkouts/readthedocs.org/
↪user_builds/pavics-sdi/conda/latest/share/proj failed
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/cartopy/io/__init__.py:241: DownloadWarning: Downloading: https://
↪naturalearth.s3.amazonaws.com/50m_physical/ne_50m_coastline.zip
  warnings.warn(f'Downloading: {url}', DownloadWarning)
```



1.2.8 Computing climate indicators

Next we'll compute a climate indicator as an example. We'll compute the mean precipitation intensity for 50 members for the year 2000, then compute ensemble statistics between realizations. The amount of data that we'll be requesting is however too big to go over the network in one request. To work around this, we need to first chunk the data in smaller `dask.array` portions by setting the `chunks` parameter when creating our dataset.

- When selecting `xarray` chunk sizes it is important to consider both the "on-disk" chunk sizes of the netcdf data as well as the type of calculation you are performing. For example the Climex data variables `_ChunkSizes` attributes indicate on-disk chunk size for dims `time, rlat, rlon` of `[365 50 50]`**1**. Using multiples of these values is a reasonable way to ensure that in-memory `dask` chunks line up with the way that the data is stored on disk.
- The next step is to think about the calculation we wish to perform. In this case we wish to analyse a single year of data for all realizations over the entire spatial domain. Using something like `chunks = dict(realization=1, time=365, rlat=50*3, rlon=50*3)` seems reasonable as a start. The general goal is to find a balance between chunk-size and a number of chunks ... too many small chunks to process will create a lot of overhead and slow the calculation down. Very large chunks will use a lot of system memory causing other issues or data timeouts from the PAVICS thredds server. In this specific case having a large `time` chunk could result in reading more data than necessary seeing as we are only interested in a single year.
- Enabling chunking also has the effect of making further computations on these chunks *lazy*. The next cell is thus going to execute almost instantly, because no data is transferred, and no computation executed. Calculations are launched when the data is actually needed, as when plotting graphics, saving to file, or when requested explicitly by calling the compute, persist or load methods. Below we make use of a dask.distributed Client of worker processes to parallelize computations and improve performance.

```
ds = xr.open_dataset(
    url, chunks=dict(realization=1, time=365, rlat=50 * 3, rlon=50 * 3)
)
```

(continues on next page)

(continued from previous page)

```
xclim.set_options(check_missing="pct", missing_options={"pct": {"tolerance": 0.05}})
sdii = xclim.atmos.daily_pr_intensity(pr=ds.pr.sel(time="2000"))
sdii
```

```
<xarray.DataArray 'sdii' (realization: 50, time: 1, rlat: 280, rlon: 280)> Size: 31MB
dask.array<where, shape=(50, 1, 280, 280), dtype=float64, chunksize=(1, 1, 150, 150),␣
↪chunktype=numpy.ndarray>
Coordinates:
  * rlat         (rlat) float64 2kB -12.61 -12.51 -12.39 ... 17.85 17.96 18.07
  * rlon         (rlon) float64 2kB 2.695 2.805 2.915 ... 33.17 33.28 33.39
  * realization  (realization) |S64 3kB b'historical-r1-r10i1p1' ... b'histor...
    lat          (rlat, rlon) float32 314kB dask.array<chunksize=(150, 150), meta=np.
↪ndarray>
    lon          (rlat, rlon) float32 314kB dask.array<chunksize=(150, 150), meta=np.
↪ndarray>
  * time         (time) object 8B 2000-01-01 00:00:00
Attributes:
    units:          mm d-1
    cell_methods:   time: mean
    history:        [2024-04-16 18:26:26] sdii: SDII(pr=pr, thresh='1 mm/day'...
    standard_name:  lwe_thickness_of_precipitation_amount
    long_name:      Average precipitation during days with daily precipitatio...
    description:    Annual simple daily intensity index (sdii) or annual aver...
```

```
# NBVAL_IGNORE_OUTPUT

# Create a local client with 6 workers processes.
dask_kwargs = dict(
    n_workers=6,
    threads_per_worker=6,
    memory_limit="4GB",
    local_directory="/notebook_dir/writable-workspace/tmp",
    silence_logs=logging.ERROR,
)

with Client(**dask_kwargs) as client:
    clear_output()
    display(
        HTML(
            f'<div class="alert alert-info"> Consult the client <a href="{client.
↪dashboard_link}" target="_blank"><strong><em><u>Dashboard</u></em></strong></a> or the
↪<strong><em>Dask jupyter sidebar</em></strong> to follow progress ... </div>'
        )
    )
    out = xens.ensemble_mean_std_max_min(sdii.to_dataset())
    out = out.load()

clear_output()
```

```
INFO:distributed.http.proxy:To route to workers diagnostics web server please install␣
↪jupyter-server-proxy: python -m pip install jupyter-server-proxy
```

```
INFO:distributed.scheduler:State start
```

```
INFO:distributed.scheduler:  Scheduler at:     tcp://127.0.0.1:37225
```

```
INFO:distributed.scheduler:  dashboard at:  http://127.0.0.1:8787/status
```

```
INFO:distributed.scheduler:Registering Worker plugin shuffle
```

```
---------------------------------------------------------------------------
PermissionError                           Traceback (most recent call last)
Cell In[7], line 12
      1 # NBVAL_IGNORE_OUTPUT
      2
      3 # Create a local client with 6 workers processes.
      4 dask_kwargs = dict(
      5     n_workers=6,
      6     threads_per_worker=6,
   (...)
      9     silence_logs=logging.ERROR,
     10 )
---> 12 with Client(**dask_kwargs) as client:
     13     clear_output()
     14     display(
     15         HTML(
     16             f'<div class="alert alert-info"> Consult the client <a href="{client.
↪dashboard_link}" target="_blank"><strong><em><u>Dashboard</u></em></strong></a> or the
↪<strong><em>Dask jupyter sidebar</em></strong> to follow progress ... </div>'
     17         )
     18     )

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/distributed/client.py:1017, in Client.__init__(self, address, loop, timeout,␣
↪set_as_default, scheduler_file, security, asynchronous, name, heartbeat_interval,␣
↪serializers, deserializers, extensions, direct_to_workers, connection_limit, **kwargs)
   1014 preload_argv = dask.config.get("distributed.client.preload-argv")
   1015 self.preloads = preloading.process_preloads(self, preload, preload_argv)
-> 1017 self.start(timeout=timeout)
   1018 Client._instances.add(self)
   1020 from distributed.recreate_tasks import ReplayTaskClient

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/distributed/client.py:1219, in Client.start(self, **kwargs)
   1217     self._started = asyncio.ensure_future(self._start(**kwargs))
   1218 else:
-> 1219     sync(self.loop, self._start, **kwargs)

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/distributed/utils.py:434, in sync(loop, func, callback_timeout, *args,␣
↪**kwargs)
    431         wait(10)
    433 if error is not None:
--> 434     raise error
```

(continues on next page)

(continued from previous page)

```
    435 else:
    436     return result

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/distributed/utils.py:408, in sync.<locals>.f()
    406         awaitable = wait_for(awaitable, timeout)
    407     future = asyncio.ensure_future(awaitable)
--> 408     result = yield future
    409 except Exception as exception:
    410     error = exception

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/tornado/gen.py:767, in Runner.run(self)
    765 try:
    766     try:
--> 767         value = future.result()
    768     except Exception as e:
    769         # Save the exception for later. It's important that
    770         # gen.throw() not be called inside this try/except block
    771         # because that makes sys.exc_info behave unexpectedly.
    772         exc: Optional[Exception] = e

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/distributed/client.py:1284, in Client._start(self, timeout, **kwargs)
   1281 elif self._start_arg is None:
   1282     from distributed.deploy import LocalCluster
-> 1284     self.cluster = await LocalCluster(
   1285         loop=self.loop,
   1286         asynchronous=self._asynchronous,
   1287         **self._startup_kwargs,
   1288     )
   1289     address = self.cluster.scheduler_address
   1291 self._gather_semaphore = asyncio.Semaphore(5)

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/distributed/deploy/spec.py:420, in SpecCluster.__await__.<locals>._()
    418     await self._start()
    419 await self.scheduler
--> 420 await self._correct_state()
    421 if self.workers:
    422     await asyncio.wait(
    423         [
    424             asyncio.create_task(_wrap_awaitable(w))
    425             for w in self.workers.values()
    426         ]
    427     )  # maybe there are more

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/distributed/deploy/spec.py:374, in SpecCluster._correct_state_internal(self)
    372 if isinstance(cls, str):
    373     cls = import_term(cls)
--> 374 worker = cls(
```

(continues on next page)

(continued from previous page)

```
    375     getattr(self.scheduler, "contact_address", None)
    376     or self.scheduler.address,
    377     **opts,
    378 )
    379 self._created.add(worker)
    380 workers.append(worker)

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/distributed/nanny.py:200, in Nanny.__init__(self, scheduler_ip, scheduler_
↪port, scheduler_file, worker_port, nthreads, loop, local_directory, services, name,␣
↪memory_limit, reconnect, validate, quiet, resources, silence_logs, death_timeout,␣
↪preload, preload_argv, preload_nanny, preload_nanny_argv, security, contact_address,␣
↪listen_address, worker_class, env, interface, host, port, protocol, config, **worker_
↪kwargs)
    186     preload_nanny_argv = dask.config.get("distributed.nanny.preload-argv")
    188 handlers = {
    189     "instantiate": self.instantiate,
    190     "kill": self.kill,
   (...)
    198     "plugin_remove": self.plugin_remove,
    199 }
--> 200 super().__init__(
    201     handlers=handlers,
    202     connection_args=self.connection_args,
    203     local_directory=local_directory,
    204     needs_workdir=False,
    205 )
    207 self.preloads = preloading.process_preloads(
    208     self, preload_nanny, preload_nanny_argv, file_dir=self.local_directory
    209 )
    211 self.death_timeout = parse_timedelta(death_timeout)

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/distributed/core.py:403, in Server.__init__(self, handlers, blocked_handlers,␣
↪stream_handlers, connection_limit, deserialize, serializers, deserializers, connection_
↪args, timeout, io_loop, local_directory, needs_workdir)
    394 self._original_local_dir = local_directory
    396 with warn_on_duration(
    397     "1s",
    398     "Creating scratch directories is taking a surprisingly long time. (
↪{duration:.2f}s) "
   (...)
    401     "scratch data to a local disk.",
    402 ):
--> 403     self._workspace = WorkSpace(local_directory)
    405     if not needs_workdir:  # eg. Nanny will not need a WorkDir
    406         self._workdir = None

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/distributed/diskutils.py:128, in WorkSpace.__init__(self, base_dir)
    127 def __init__(self, base_dir: str):
--> 128     self.base_dir = self._init_workspace(base_dir)
```

(continues on next page)

(continued from previous page)

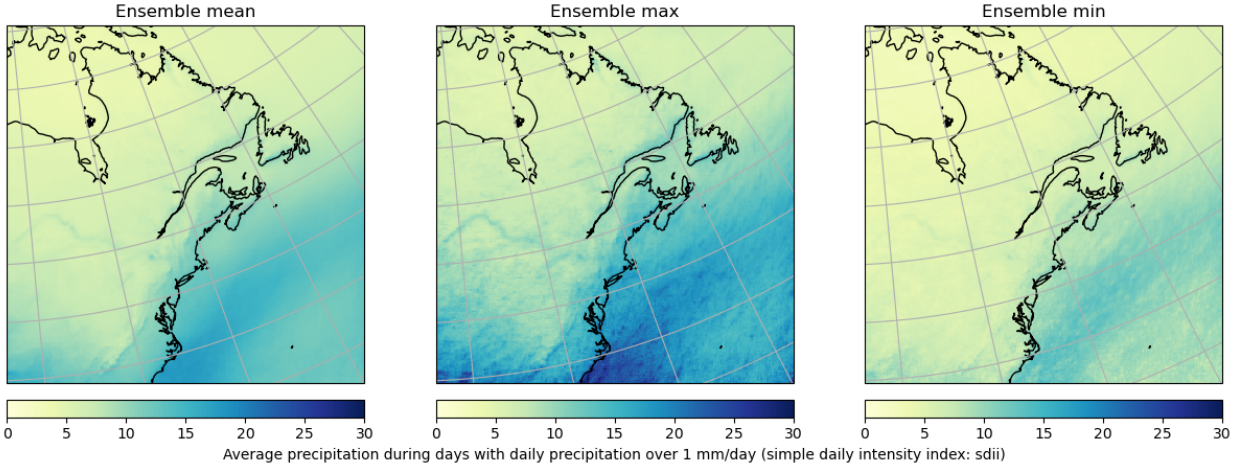
```
    129     self._global_lock_path = os.path.join(self.base_dir, "global.lock")
    130     self._purge_lock_path = os.path.join(self.base_dir, "purge.lock")

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/distributed/diskutils.py:146, in WorkSpace._init_workspace(self, base_dir)
    144 for try_dir in try_dirs:
    145     try:
--> 146         os.makedirs(try_dir)
    147     except FileExistsError:
    148         try:

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/os.
↪py:215, in makedirs(name, mode, exist_ok)
    213 if head and tail and not path.exists(head):
    214     try:
--> 215         makedirs(head, exist_ok=exist_ok)
    216     except FileExistsError:
    217         # Defeats race condition when another thread created the path
    218         pass

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/os.
↪py:215, in makedirs(name, mode, exist_ok)
    213 if head and tail and not path.exists(head):
    214     try:
--> 215         makedirs(head, exist_ok=exist_ok)
    216     except FileExistsError:
    217         # Defeats race condition when another thread created the path
    218         pass

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/os.
↪py:215, in makedirs(name, mode, exist_ok)
    213 if head and tail and not path.exists(head):
    214     try:
--> 215         makedirs(head, exist_ok=exist_ok)
    216     except FileExistsError:
    217         # Defeats race condition when another thread created the path
    218         pass

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/os.
↪py:225, in makedirs(name, mode, exist_ok)
    223         return
    224 try:
--> 225     mkdir(name, mode)
    226 except OSError:
    227     # Cannot rely on checking for EEXIST, since the operating system
    228     # could give priority to other errors like EACCES or EROFS
    229     if not exist_ok or not path.isdir(name):

PermissionError: [Errno 13] Permission denied: '/notebook_dir'
```

```
fig = plt.figure(figsize=(15, 5))
for ii, vv in enumerate([v for v in out.data_vars if "stdev" not in v]):
```

(continues on next page)

(continued from previous page)

```
    ax = plt.subplot(1, 3, ii + 1, projection=rotp)

    ax.coastlines()
    ax.gridlines()
    m = ax.pcolormesh(
        out.rlon, out.rlat, out[vv].isel(time=0), vmin=0, vmax=30, cmap="YlGnBu"
    )
    c = plt.colorbar(
        m, orientation="horizontal", label=out[vv].long_name, fraction=0.046, pad=0.04
    )
    ax.set_title(f"Ensemble {vv.split('_')[-1]}")
    if ii != 1:
        c.set_label("")
```



In the next example, we extract one grid point over Montréal in order to calculate the annual maximum 1-day precipitation for all years and ensemble members. In this case we would likely benefit by rethinking out chunk sizes.

- Since we are only interested in a single grid-cell there is no real need to have relatively large spatial chunks in the `rlon` and `rlat` dimensions read into memory to simply be immediately discarded. In this case we reduce the spatial chunk size.
- Similarly, since we are only looking at a single point we can likely get away with loading a large number of time steps into a single chunk.

```
# Subset over the Montreal gridpoint
ds = xr.open_dataset(url, chunks=dict(realization=1, time=365 * 50, rlon=25, rlat=25))
pt = subset_gridpoint(ds, lon=-73.69, lat=45.50)
print("Input dataset for Montreal :")
display(pt)
out = xclim.atmos.max_1day_precipitation_amount(pr=pt.pr, freq="YS")
print("Maximim 1-day precipitation `lazy` output ..")
out
```

```
Input dataset for Montreal :
```

```
<xarray.Dataset>
Dimensions:       (time: 52924, realization: 50)
```

(continues on next page)

(continued from previous page)

```
Coordinates:
    rlat          float64 0.365
    rlon          float64 16.12
  * time          (time) object 1955-01-01 00:00:00 ... 2099-12-30 00:00:00
  * realization   (realization) |S64 b'historical-r1-r10i1p1' ... b'historica...
    lat           float32 45.45
    lon           float32 -73.7
Data variables:
    rotated_pole  |S64 b''
    tasmin        (realization, time) float32 dask.array<chunksize=(1, 18250), meta=np.
↪ndarray>
    tasmax        (realization, time) float32 dask.array<chunksize=(1, 18250), meta=np.
↪ndarray>
    tas           (realization, time) float32 dask.array<chunksize=(1, 18250), meta=np.
↪ndarray>
    pr            (realization, time) float32 dask.array<chunksize=(1, 18250), meta=np.
↪ndarray>
    prsn          (realization, time) float32 dask.array<chunksize=(1, 18250), meta=np.
↪ndarray>
Attributes: (12/30)
    Conventions:              CF-1.6
    DODS.dimName:             string1
    DODS.strlen:              0
    EXTRA_DIMENSION.bnds:     2
    NCO:                      "4.5.2"
    abstract:                 The ClimEx CRCM5 Large Ensemble of high-resolut...
    ...                       ...
    project_id:               CLIMEX
    rcm_version_id:           v3331
    terms_of_use:             http://www.climex-project.org/sites/default/fil...
    title:                    The ClimEx CRCM5 Large Ensemble
    type:                     RCM
    EXTRA_DIMENSION.string1:  1
```

```
Maximim 1-day precipitation `lazy` output ..
```

```
<xarray.DataArray 'rx1day' (realization: 50, time: 145)>
dask.array<where, shape=(50, 145), dtype=float32, chunksize=(1, 50), chunktype=numpy.
↪ndarray>
Coordinates:
    rlat         float64 0.365
    rlon         float64 16.12
  * realization  (realization) |S64 b'historical-r1-r10i1p1' ... b'historical...
    lat          float32 45.45
    lon          float32 -73.7
  * time         (time) object 1955-01-01 00:00:00 ... 2099-01-01 00:00:00
Attributes:
    units:          mm/day
    cell_methods:   time: mean time: maximum over days
    history:        [2023-05-24 17:44:17] rx1day: RX1DAY(pr=pr, freq='YS') wi...
    standard_name:  lwe_thickness_of_precipitation_amount
```

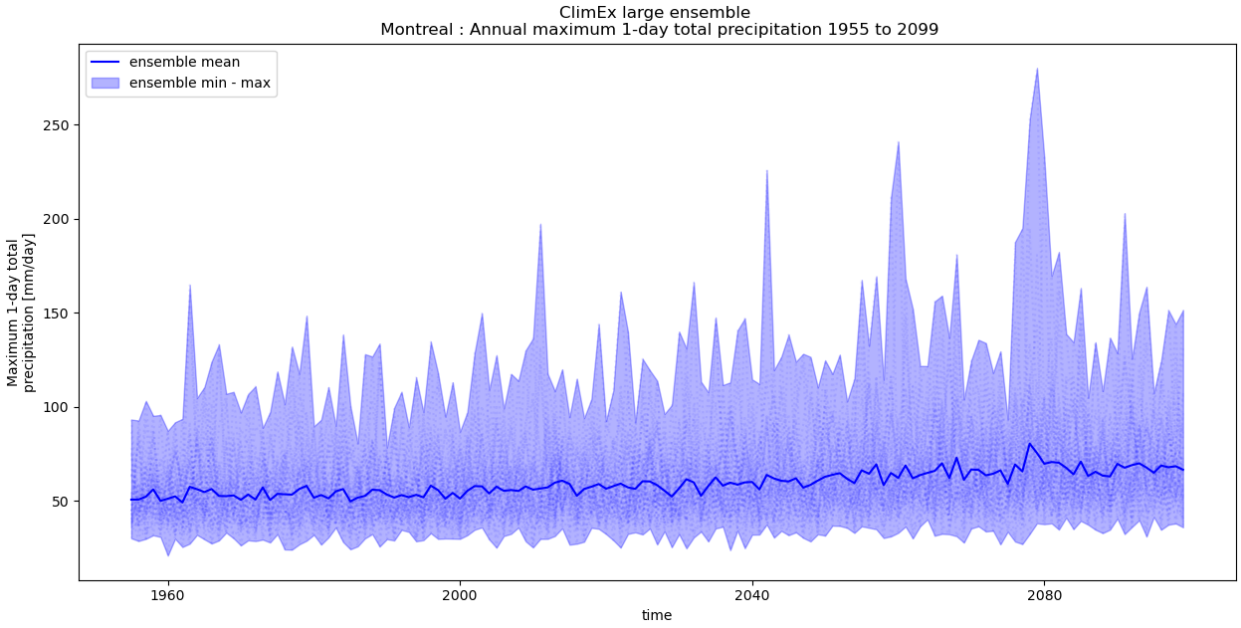
(continues on next page)

(continued from previous page)

```
    long_name:      Maximum 1-day total precipitation
    description:    Annual maximum 1-day total precipitation
```

```
# NBVAL_IGNORE_OUTPUT

# Compute the annual max 1-day precipitation
dask_kwargs = dict(
    n_workers=6,
    threads_per_worker=6,
    memory_limit="4GB",
    local_directory="/notebook_dir/writable-workspace/tmp",
    silence_logs=logging.ERROR,
)
with Client(**dask_kwargs) as client:
    # dashboard available at client.dashboard_link
    clear_output()
    display(
        HTML(
            f'<div class="alert alert-info"> Consult the client <a href="{client.
↪dashboard_link}" target="_blank"><strong><em><u>Dashboard</u></em></strong></a> or the
↪<strong><em>Dask jupyter sidebar</em></strong> to follow progress ... </div>'
        )
    )
    out = out.load()

clear_output()
```

```
out_ens = xens.ensemble_mean_std_max_min(out.to_dataset())
fig = plt.figure(figsize=(15, 7))
ax = plt.subplot(1, 1, 1)
h1 = out.plot.line(ax=ax, x="time", color="b", linestyle=":", alpha=0.075, label=None)
h2 = plt.fill_between(
    x=out_ens.time.values,
    y1=out_ens.rx1day_max,
    y2=out_ens.rx1day_min,
    color="b",
    alpha=0.3,
    label="ensemble min - max",
)
h3 = plt.plot(
    out_ens.time.values, out_ens.rx1day_mean, color="b", label="ensemble mean"
)
handles, labels = ax.get_legend_handles_labels()
leg = ax.legend(handles[::-1], labels[::-1])
tit = plt.title(
    f"ClimEx large ensemble \n Montreal : {out.description} {out_ens.time.dt.year[0].
↪values} to {out_ens.time.dt.year[-1].values}"
)
```



1.2.9 Computing indices on weather forecasts

The PAVICS data catalog includes the latest weather forecast from the Global Ensemble Prediction System (GEPS) from Environment and Climate Change Canada. For the 20 members in the ensemble, two variables are available (precipitation and air temperature), every 3 hours for the first 8 days of the forecast, then every 6 hours for the following 8 days.

This notebook shows how to access the forecast data and compute climate indices using xclim. The first step is to open the catalog and get the URL to the data.

```
import warnings

import numba

warnings.simplefilter("ignore", category=numba.core.errors.NumbaDeprecationWarning)

import intake
import xarray as xr
import xclim

# fmt: off
forecast = "https://pavics.ouranos.ca/catalog/forecast.json"  # TEST_USE_PROD_DATA
# fmt: on

cat = intake.open_esm_datastore(forecast)
cat.df.head()
```

```
  license_type                                          title  \
0   permissive  Global Ensemble Prediction System (GEPS) - ECCC

                                institution  member  \
0  Canadian Meteorological Service - Montreal     NaN
```

(continues on next page)

(continued from previous page)

```
            variable_name                                  variable_long_name  \
0  ['pr', 'tas', 'member']  ['depth of water-equivalent precipitation', '2...

                                                 path
0  https://pavics.ouranos.ca/twitcher/ows/proxy/t...
```

```
# NBVAL_IGNORE_OUTPUT
url = cat.df.path[0]
ds = xr.open_dataset(url)
ds
```

```
<xarray.Dataset> Size: 4GB
Dimensions:  (lon: 720, lat: 361, time: 97, member: 20)
Coordinates:
  * lon      (lon) float64 6kB 0.0 0.5 1.0 1.5 2.0 ... 358.0 358.5 359.0 359.5
  * lat      (lat) float64 3kB -90.0 -89.5 -89.0 -88.5 ... 88.5 89.0 89.5 90.0
    reftime  datetime64[ns] 8B ...
  * time     (time) datetime64[ns] 776B 2024-04-16 ... 2024-05-02
  * member   (member) float32 80B 1.0 2.0 3.0 4.0 5.0 ... 17.0 18.0 19.0 20.0
Data variables:
    pr       (member, time, lat, lon) float32 2GB ...
    tas      (member, time, lat, lon) float32 2GB ...
Attributes: (12/14)
    GRIB_edition:            2
    GRIB_centre:             cwao
    GRIB_centreDescription:  Canadian Meteorological Service - Montreal
    GRIB_subCentre:          0
    Conventions:             CF-1.7
    institution:             Canadian Meteorological Service - Montreal
    ...                      ...
    abstract:                Global ensemble forecasts are made twice a day u...
    dataset_description:     https://weather.gc.ca/grib/grib2_ens_geps_e.html
    dataset_id:              GEPS
    type:                    forecast
    license_type:            permissive
    license:                 https://open.canada.ca/en/open-government-licenc...
```

As the name suggests, GEPS is a global forecast, but to keep things simple, here we only pick one grid cell near Montréal. The following graph shows the time series of both precipitation and temperature. The member can be selected using the slider on the right of the graphics.

```
import hvplot.xarray

mtl = ds.sel(lon=45.5, lat=360 - 73.5, method="nearest")
mtl.pr.hvplot(x="time", width=300) + mtl.tas.hvplot(x="time", width=300)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[3], line 1
----> 1 import hvplot.xarray
```

(continues on next page)

(continued from previous page)

```
      3 mtl = ds.sel(lon=45.5, lat=360 - 73.5, method="nearest")
      4 mtl.pr.hvplot(x="time", width=300) + mtl.tas.hvplot(x="time", width=300)

ModuleNotFoundError: No module named 'hvplot'
```

Note that precipitation is cumulative starting from 0 at the beginning of the forecast. To get daily precipitation amounts, we need to first compute the 3- or 6-hourly precipitation, then sum daily.

```
from xclim.core.units import amount2rate

with xr.set_options(keep_attrs=True):
    pr_delta = mtl.pr.diff(dim="time")
    pr = pr_delta.resample(time="D").sum()
    # The units are mm, but for use with xclim, it's better to indicate this is a␣
↪precipitation rate (mm/d).
    pr = amount2rate(pr, out_units="mm/d")
    # To avoid xclim warnings, we also fix some attributes to reflect the fact that␣
↪precipitation is now a mean flux.
    pr.attrs["cell_methods"] = "time: mean"
    pr.attrs["standard_name"] = "precipitation_flux"

# It's then possible to compute xclim indicators, but of course, only on frequencies␣
↪smaller than 16 days.
wetdays = xclim.atmos.wetdays(pr, freq="7D")
```

For temperature, here we'll convert the time series to daily mean values. Once this is done, xclim indicators can be computed easily. Here, we compute heating and cooling degree days over periods of 7 days.

```
tas = mtl.tas.resample(time="D").mean(keep_attrs=True)
hdd = xclim.atmos.heating_degree_days(tas, freq="7D").dropna(dim="time")
cdd = xclim.atmos.cooling_degree_days(tas, freq="7D").dropna(dim="time")
```

```
hdd.hvplot.hist(groupby="time", legend=False, width=300) + cdd.hvplot.hist(
    groupby="time", legend=False, width=300
)
```

```
:Layout
   .DynamicMap.I  :DynamicMap   [time]
      :NdOverlay   [Element]
         :Histogram   [heating_degree_days]   (heating_degree_days_count)
   .DynamicMap.II :DynamicMap   [time]
      :NdOverlay   [Element]
         :Histogram   [cooling_degree_days]   (cooling_degree_days_count)
```

1.2.10 Working with the ECCC GeoAPI to access weather station data

Environment and Climate Change Canada (ECCC) hosts a data server compatible with the GeoAPI standard. This notebook shows how to send requests for daily climate station data and display the results.

Climate stations

The server holds different *collections*, and requests are made to a particular collection. Here we'll start with the `climate-station` collection, which holds metadata about available stations, but no actual meteorological data. Useful queryables fields in this collection include `DLY_FIRST_DATE` and `DLY_LAST_DATE`, `ENG_PROV_NAME`, `LATITUDE`, `LONGITUDE` and `ELEVATION` and `STATION_NAME`, among many others.

Creating a request to the server for data

Let's start by showing a map of all available stations locations in New-Brunswick. To do so, we first need to compose a URL request. The request includes the address of the server, the collection, then a query to filter results.

```
import os

os.environ["USE_PYGEOS"] = "0"  # force use Shapely with GeoPandas

import urllib

import geopandas as gpd
from urlpath import URL

# Compose the request
host = URL("https://api.weather.gc.ca")
climate_stations = host / "collections" / "climate-stations" / "items"
url = climate_stations.with_query({"ENG_PROV_NAME": "NOVA-SCOTIA"})
print(url)

# Send the request to the server
resp = url.get()
resp
```

```
https://api.weather.gc.ca/collections/climate-stations/items?ENG_PROV_NAME=NOVA-SCOTIA
```

```
<Response [200]>
```

The response from the server is a `Response` class instance. What we're interested in is the content of this response, which in this case is a geoJSON file.

```
# NBVAL_IGNORE_OUTPUT

resp.content[:100]
```

```
b'{"type":"FeatureCollection","features":[{"type":"Feature","properties":{"STN_ID":6289,
↪"STATION_NAME"'
```

We'll open the geoJSON using geopandas. We have a few options to do this:

- Load the response' content using `json.load`, then create GeoDataFrame using the `from_features` class method;
- Save the response content to a file on disk, then open using `geopandas.read_file`;
- Save the response in an in-memory file using `StringIO`;
- Call `geopandas.read_file(url)` to let geopandas handle the data download.

Here we'll use the last option, as it's the simplest. Note that the first method ignores the feature `id`, which seems to create problems with visualization with folium below.

```
# NBVAL_IGNORE_OUTPUT

# The first approach would look like this:
# import json
# stations = gpd.GeoDataFrame.from_features(json.loads(resp.content))

with urllib.request.urlopen(url=str(url)) as req:
    stations = gpd.read_file(filename=req, engine="pyogrio")
stations.head()
```

```
ERROR 1: PROJ: proj_create_from_database: Open of /home/docs/checkouts/readthedocs.org/
↪user_builds/pavics-sdi/conda/latest/share/proj failed
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
```

(continues on next page)

(continued from previous page)

```
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
```

```
        id  STN_ID       STATION_NAME PROV_STATE_TERR_CODE ENG_PROV_NAME  \
0  8200100    6289    ANNAPOLIS ROYAL                   NS   NOVA SCOTIA
1  8200400    6298       BARRIE BROOK                   NS   NOVA SCOTIA
2  8200575    6302            BEDFORD                   NS   NOVA SCOTIA
3  8200579    6303      BIG INTERVALE                   NS   NOVA SCOTIA
4  8200630    6309  CAMBRIDGE STATION                   NS   NOVA SCOTIA

     FRE_PROV_NAME COUNTRY   LATITUDE  LONGITUDE TIMEZONE  ... HLY_FIRST_DATE  \
0  NOUVELLE-ÉCOSSE     CAN  444500000 -653100000      AST  ...            NaT
1  NOUVELLE-ÉCOSSE     CAN  453900000 -612600000      AST  ...            NaT
2  NOUVELLE-ÉCOSSE     CAN  444400000 -634000000      AST  ...            NaT
3  NOUVELLE-ÉCOSSE     CAN  465000000 -603700000      AST  ...            NaT
4  NOUVELLE-ÉCOSSE     CAN  450400000 -643700000      AST  ...            NaT

  HLY_LAST_DATE DLY_FIRST_DATE DLY_LAST_DATE MLY_FIRST_DATE MLY_LAST_DATE  \
0           NaT     1914-04-01    2007-06-04     1914-01-01    2006-02-01
1           NaT     1950-02-01    1972-01-31     1950-01-01    1972-12-01
2           NaT     1954-12-01    1966-05-31     1955-01-01    1966-12-01
3           NaT     1984-09-01    1992-12-31     1984-01-01    1992-12-01
4           NaT     1973-10-01    1978-09-30     1973-01-01    1978-12-01

   HAS_MONTHLY_SUMMARY  HAS_NORMALS_DATA HAS_HOURLY_DATA  \
0                    Y                 N               N
1                    Y                 N               N
2                    Y                 N               N
3                    Y                 N               N
4                    Y                 N               N

                     geometry
0  POINT (-65.51667 44.75000)
1  POINT (-61.43333 45.65000)
2  POINT (-63.66667 44.73333)
3  POINT (-60.61667 46.83333)
4  POINT (-64.61667 45.06667)

[5 rows x 34 columns]
```

Filter stations

Now let's say we want to filter the stations that were in operations for at least 50 years. What we'll do is create a new column `n_days` and filter on it.

```
# NBVAL_IGNORE_OUTPUT

import pandas as pd

# Create a datetime.Timedelta object from the subtraction of two dates.
delta = pd.to_datetime(stations["DLY_LAST_DATE"]) - pd.to_datetime(
    stations["DLY_FIRST_DATE"]
)

# Get the number of days in the time delta
stations["n_days"] = delta.apply(lambda x: x.days)

# Compute condition
over_50 = stations["n_days"] > 50 * 365.25

# Index the data frame using the condition
select = stations[over_50]
select.head()
```

```
         id  STN_ID     STATION_NAME PROV_STATE_TERR_CODE ENG_PROV_NAME  \
0   8200100    6289  ANNAPOLIS ROYAL                   NS   NOVA SCOTIA
9   8201410    6336           DEMING                   NS   NOVA SCOTIA
18  8203400    6399      MALAY FALLS                   NS   NOVA SCOTIA
34  8205698    6485           SYDNEY                   NS   NOVA SCOTIA
39  8206300    6506        WHITEHEAD                   NS   NOVA SCOTIA

      FRE_PROV_NAME COUNTRY   LATITUDE  LONGITUDE TIMEZONE  ...  \
0   NOUVELLE-ÉCOSSE     CAN  444500000 -653100000      AST  ...
9   NOUVELLE-ÉCOSSE     CAN  451259007 -611040090      AST  ...
18  NOUVELLE-ÉCOSSE     CAN  445900000 -622900000      AST  ...
34  NOUVELLE-ÉCOSSE     CAN  460900000 -601200000      AST  ...
39  NOUVELLE-ÉCOSSE     CAN  451300000 -611100000      AST  ...

         HLY_LAST_DATE DLY_FIRST_DATE DLY_LAST_DATE MLY_FIRST_DATE  \
0                  NaT     1914-04-01    2007-06-04     1914-01-01
9                  NaT     1956-10-01    2011-12-31     1956-01-01
18 2000-08-31 23:00:00     1950-02-01    2000-08-31     1950-01-01
34                 NaT     1870-01-01    1941-03-31     1870-01-01
39                 NaT     1883-12-01    1960-06-30     1883-01-01

   MLY_LAST_DATE HAS_MONTHLY_SUMMARY  HAS_NORMALS_DATA  HAS_HOURLY_DATA  \
0     2006-02-01                   Y                 N                N
9     2006-02-01                   Y                 Y                N
18    2000-08-01                   Y                 N                N
34    1941-12-01                   Y                 N                N
39    1960-12-01                   Y                 N                N

                     geometry n_days
```

(continues on next page)

(continued from previous page)

```
0   POINT (-65.51667 44.75000)  34032
9   POINT (-61.17780 45.21639)  20179
18  POINT (-62.48333 44.98333)  18474
34  POINT (-60.20000 46.15000)  26021
39  POINT (-61.18333 45.21667)  27970

[5 rows x 35 columns]
```

Map the data

We can then simply map the locations of station with at least 50 years of data using the `explore` method. This will display an interactive base map and overlay the station locations, where on a station marker will display this station's information.

On top of this map, we'll add controls to draw a rectangle. To use the drawing tool, click on the square on the left hand side menu, and the click and drag to draw a rectangle over the area of interest. Once that's done, click on the Export button on the right of the map. This will download a file called `data.geojson`

```
from folium.plugins import Draw

# Add control to draw a rectangle, and an export button.
draw_control = Draw(
    draw_options={
        "polyline": False,
        "poly": False,
        "circle": False,
        "polygon": False,
        "marker": False,
        "circlemarker": False,
        "rectangle": True,
    },
    export=True,
)

# The map library Folium chokes on columns including time stamps, so we first select the␣
↪data to plot.
m = select[["geometry", "n_days"]].explore("n_days")
draw_control.add_to(m)
m
```

```
<folium.folium.Map at 0x7f67aa9ab940>
```

Filter stations using bounding box

Next, we'll use the bounding box drawn on the map to select a subset of stations. We first open the `data.geojson` file downloaded to disk, create a shapely object and use it filter stations.

```
# NBVAL_IGNORE_OUTPUT

# Adjust directory if running this locally.
# rect = gpd.read_file("~/Downloads/data.geojson")

# Here we're using an existing file so the notebook runs without user interaction.
rect = gpd.read_file(filename="./data.geojson", engine="pyogrio")

# Filter stations DataFrame using bbox
inbox = select.within(rect.loc[0].geometry)

print("Number of stations within subregion: ", sum(inbox))
sub_select = select[inbox]
sub_select.head()
```

```
Number of stations within subregion:  9
```

```
          id  STN_ID  STATION_NAME PROV_STATE_TERR_CODE ENG_PROV_NAME  \
9    8201410    6336        DEMING                   NS   NOVA SCOTIA
34   8205698    6485        SYDNEY                   NS   NOVA SCOTIA
39   8206300    6506     WHITEHEAD                   NS   NOVA SCOTIA
118  8201000    6329  COLLEGEVILLE                   NS   NOVA SCOTIA
138  8205600    6481    STILLWATER                   NS   NOVA SCOTIA

       FRE_PROV_NAME COUNTRY   LATITUDE  LONGITUDE TIMEZONE  ...  \
9    NOUVELLE-ÉCOSSE     CAN  451259007 -611040090      AST  ...
34   NOUVELLE-ÉCOSSE     CAN  460900000 -601200000      AST  ...
39   NOUVELLE-ÉCOSSE     CAN  451300000 -611100000      AST  ...
118  NOUVELLE-ÉCOSSE     CAN  452900000 -620100000      AST  ...
138  NOUVELLE-ÉCOSSE     CAN  451100000 -620000000      AST  ...

    HLY_LAST_DATE DLY_FIRST_DATE DLY_LAST_DATE MLY_FIRST_DATE MLY_LAST_DATE  \
9             NaT     1956-10-01    2011-12-31     1956-01-01    2006-02-01
34            NaT     1870-01-01    1941-03-31     1870-01-01    1941-12-01
39            NaT     1883-12-01    1960-06-30     1883-01-01    1960-12-01
118           NaT     1916-06-01    2016-09-30     1916-01-01    2006-02-01
138           NaT     1915-12-01    1979-10-31     1915-01-01    1979-12-01

    HAS_MONTHLY_SUMMARY  HAS_NORMALS_DATA  HAS_HOURLY_DATA  \
9                     Y                 Y                N
34                    Y                 N                N
39                    Y                 N                N
118                   Y                 Y                N
138                   Y                 N                N

                      geometry n_days
9    POINT (-61.17780 45.21639)  20179
34   POINT (-60.20000 46.15000)  26021
```

(continues on next page)

(continued from previous page)

```
39   POINT (-61.18333 45.21667)  27970
118  POINT (-62.01667 45.48333)  36646
138  POINT (-62.00000 45.18333)  23345

[5 rows x 35 columns]
```

Request meteorological data

Now we'll make a request for actual meteorological data from the stations filtered above. For this, we'll use the Daily Climate Observations collection (`climate-daily`). Here, we're picking just one station but we could easily loop on each station.

```
# NBVAL_IGNORE_OUTPUT

coll = host / "collections" / "climate-daily" / "items"
station_id = "8201410"

# Restricting the number of entries returned to keep things fast.
url = str(coll.with_query({"CLIMATE_IDENTIFIER": station_id, "limit": 365}))
print("Request: ", url)
with urllib.request.urlopen(url=str(url)) as req:
    data = gpd.read_file(filename=req, engine="pyogrio")
data.head()
```

```
Request:  https://api.weather.gc.ca/collections/climate-daily/items?CLIMATE_
↪IDENTIFIER=8201410&limit=365
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
```

```
                   id MAX_TEMPERATURE_FLAG SNOW_ON_GROUND_FLAG  \
0   8201410.1996.6.13                 None                None
1  8201410.1993.12.17                 None                None
2   8201410.1967.11.6                 None                None
3   8201410.2003.1.14                 None                None
4   8201410.2009.1.30                 None                None

  TOTAL_RAIN_FLAG  MIN_TEMPERATURE MAX_REL_HUMIDITY  SNOW_ON_GROUND  \
0            None             9.0             None             0.0
1            None            -2.5             None             0.0
2            None             3.9             None             0.0
3            None            -6.0             None            45.0
4            None            -5.5             None             5.0

  SPEED_MAX_GUST_FLAG HEATING_DEGREE_DAYS_FLAG MEAN_TEMPERATURE_FLAG  ...  \
0                None                     None                  None  ...
1                None                     None                  None  ...
```

(continues on next page)

(continued from previous page)

```
2                None                      None                  None  ...
3                None                      None                  None  ...
4                None                      None                  None  ...

   MIN_REL_HUMIDITY_FLAG MAX_TEMPERATURE  COOLING_DEGREE_DAYS  \
0                   None            12.5                  0.0
1                   None             0.5                  0.0
2                   None             8.9                  0.0
3                   None            -3.5                  0.0
4                   None            -1.0                  0.0

  HEATING_DEGREE_DAYS MAX_REL_HUMIDITY_FLAG TOTAL_SNOW DIRECTION_MAX_GUST  \
0                 7.2                  None        0.0                None
1                19.0                  None        0.0                None
2                11.6                  None        0.0                None
3                22.8                  None        0.0                None
4                21.3                  None        0.0                None

                    ID  DIRECTION_MAX_GUST_FLAG                    geometry
0   8201410.1996.6.13                     None  POINT (-61.17780 45.21639)
1  8201410.1993.12.17                     None  POINT (-61.17780 45.21639)
2   8201410.1967.11.6                     None  POINT (-61.17780 45.21639)
3   8201410.2003.1.14                     None  POINT (-61.17780 45.21639)
4   8201410.2009.1.30                     None  POINT (-61.17780 45.21639)

[5 rows x 36 columns]
```

We can also send a request for data inside a bounding box at a specific date.

```
bbox = rect.iloc[0].geometry.bounds
print("Bounding box: ", bbox)
url = str(
    coll.with_query(
        {
            "bbox": str(bbox).strip("()"),
            "LOCAL_DATE": "2000-01-01 00:00:00",
            "limit": 100,
        }
    )
)
with urllib.request.urlopen(url=str(url)) as req:
    snapshot = gpd.read_file(filename=req, engine="pyogrio")
```

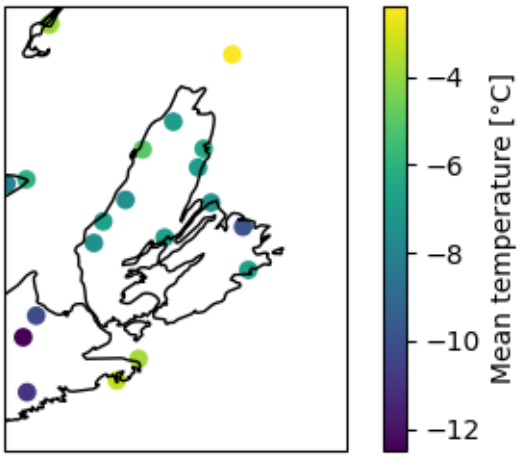
```
Bounding box:  (-62.186675, 44.78125, -59.123882, 47.53125)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
```

```
import cartopy
from cartopy import crs as ccrs
from matplotlib import pyplot as plt

# Create map projection
proj = ccrs.PlateCarree()

# If using another projection, remember you'll need to reproject the snapshot's␣
↪coordinates.
# snapshot.to_crs(proj, inplace=True)

# Create figure and axes
fig = plt.figure(figsize=(5, 3))
ax = fig.add_subplot(projection=proj)

# Set the map extent to the bounding box and draw the coastlines
ax.set_extent([bbox[0], bbox[2], bbox[1], bbox[3]])
ax.coastlines()

# Plot markers color-coded according to the temperature recorded.
ax = snapshot.plot(column="MEAN_TEMPERATURE", ax=ax, cmap=plt.cm.viridis, legend=True)

# Add a label to the colorbar
cax = ax.figure.axes[-1]
cax.set_ylabel("Mean temperature [°C]")
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/cartopy/io/__init__.py:241: DownloadWarning: Downloading: https://
↪naturalearth.s3.amazonaws.com/10m_physical/ne_10m_coastline.zip
  warnings.warn(f'Downloading: {url}', DownloadWarning)
```

```
Text(448.07777777777784, 0.5, 'Mean temperature [°C]')
```



Another useful filter is on dates and times. Let's say we only want data in a given period, we simply create a request with the `datetime` argument and a `/` separating the start and end dates. You may leave the start or end date open-ended using `..` instead of a date time string.

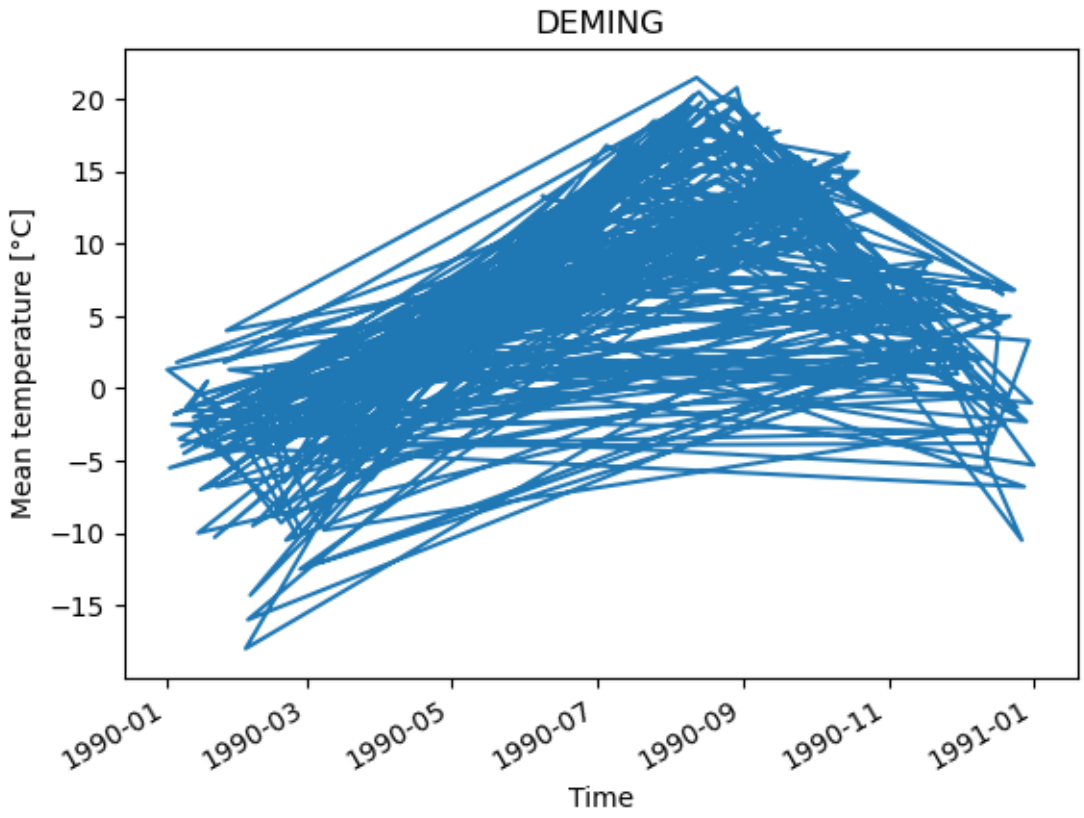
```
url = str(
    coll.with_query(
        {
            "CLIMATE_IDENTIFIER": station_id,
            "datetime": "1990-01-01 00:00:00/1991-01-01 00:00:00",
        }
    )
)
print(url)
with urllib.request.urlopen(url=str(url)) as req:
    gdf = gpd.read_file(filename=req, engine="pyogrio")
```

```
https://api.weather.gc.ca/collections/climate-daily/items?CLIMATE_IDENTIFIER=8201410&
↪datetime=1990-01-01+00%3A00%3A00%2F1991-01-01+00%3A00%3A00
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
```

```
# Convert the datetime string to a datetime object
gdf["LOCAL_DATE"] = pd.to_datetime(gdf["LOCAL_DATE"])

# Create a time series out of the column for mean temperature
ts = gdf.set_index("LOCAL_DATE")["MEAN_TEMPERATURE"]
```

```
# Plot the time series
ax = ts.plot()
ax.set_xlabel("Time")
ax.set_ylabel("Mean temperature [°C]")
ax.set_title(gdf.iloc[0]["STATION_NAME"])
plt.show()
```



Adjusted and Homogenized Canadian Climate Data (AHCCD)

The Adjusted and Homogenized Canadian Climate Data (AHCCD) datasets from ECCC are climate station data adjusted to account for discontinuities in the record, such as instrument relocation. The collections related to these datasets are `ahccd-stations` for station metadata, `ahccd-annual`, `ahccd-monthly` and `ahccd-seasonal` for temporally aggregated time series, and `ahccd-trends` for trends computed on the data.

Now, unfortunately, the fields for these datasets are different from those of the climate stations… One strategy to find out what keywords are accepted is to make a query with no filter except for `limit=1`. Another is to go to the collection search page (click on the link printed below), and inspect the column names.

```
# NBVAL_IGNORE_OUTPUT

# The url to query station metadata - this should behave similarly as `climate-stations`
ahccd_stations = host / "collections" / "ahccd-stations" / "items"
url = ahccd_stations.with_query({"limit": 1})
print(ahccd_stations)
with urllib.request.urlopen(url=str(url)) as req:
    gpd.read_file(filename=req, engine="pyogrio")
```

```
https://api.weather.gc.ca/collections/ahccd-stations/items
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
```

(continues on next page)

(continued from previous page)

```
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
```

So if we want to see the stations in Yukon, we'd have to query with the `province__province` keyword… Now how do you know what code to use for provinces? One solution is to go again to the collection search page, zoom on the area of interest and click on the check box to "Only show items by map view", then inspect the results.

```
# NBVAL_IGNORE_OUTPUT

url = ahccd_stations.with_query({"province__province": "YT"})
with urllib.request.urlopen(url=str(url)) as req:
    gpd.read_file(filename=req, engine="pyogrio")
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
```

Let's pick the Dawson station (`2100LRP`), which seems to have a long record. Again, use the trick above to see which fields are accepted.

```
# NBVAL_IGNORE_OUTPUT

ahccd_mon = host / "collections" / "ahccd-monthly" / "items"
url = ahccd_mon.with_query({"station_id__id_station": "2100LRP"})
with urllib.request.urlopen(url=str(url)) as req:
    mts = gpd.read_file(filename=req, engine="pyogrio")
mts.head()
```

```
               id pressure_sea_level_units__pression_niveau_mer_unite  \
0  2100LRP.1983.03                                                hPa
1  2100LRP.1921.03                                                hPa
2  2100LRP.1992.06                                                hPa
3  2100LRP.1938.03                                                hPa
4  2100LRP.1963.09                                                hPa

  period_value__valeur_periode  lon__long  \
0                          Mar    -139.13
1                          Mar    -139.13
2                          Jun    -139.13
3                          Mar    -139.13
```

(continues on next page)

(continued from previous page)

```
4                          Sep    -139.13

  wind_speed_units__vitesse_vent_unites province__province  \
0                                   kph                 YT
1                                   kph                 YT
2                                   kph                 YT
3                                   kph                 YT
4                                   kph                 YT

  total_precip_units__precip_totale_unites station_id__id_station  \
0                                       mm                2100LRP
1                                       mm                2100LRP
2                                       mm                2100LRP
3                                       mm                2100LRP
4                                       mm                2100LRP

   temp_max__temp_max wind_speed__vitesse_vent  ...     date  \
0               -6.8                     None  ...  1983-03
1               -8.8                     None  ...  1921-03
2               22.1                     None  ...  1992-06
3               -4.5                     None  ...  1938-03
4               12.1                     None  ...  1963-09

  snow_units__neige_unites period_group__groupe_periode  \
0                       mm                      Monthly
1                       mm                      Monthly
2                       mm                      Monthly
3                       mm                      Monthly
4                       mm                      Monthly

  temp_min_units__temp_min_unites  rain_units__pluie_unites  \
0                               C                        mm
1                               C                        mm
2                               C                        mm
3                               C                        mm
4                               C                        mm

   temp_max_units__temp_max_unites identifier__identifiant  \
0                                C         2100LRP.1983.03
1                                C         2100LRP.1921.03
2                                C         2100LRP.1992.06
3                                C         2100LRP.1938.03
4                                C         2100LRP.1963.09

  pressure_sea_level__pression_niveau_mer temp_mean__temp_moyenne  \
0                                    None                   -15.7
1                                    None                   -16.8
2                                    None                    14.2
3                                    None                   -13.7
4                                    None                     6.5

                     geometry
```

(continues on next page)

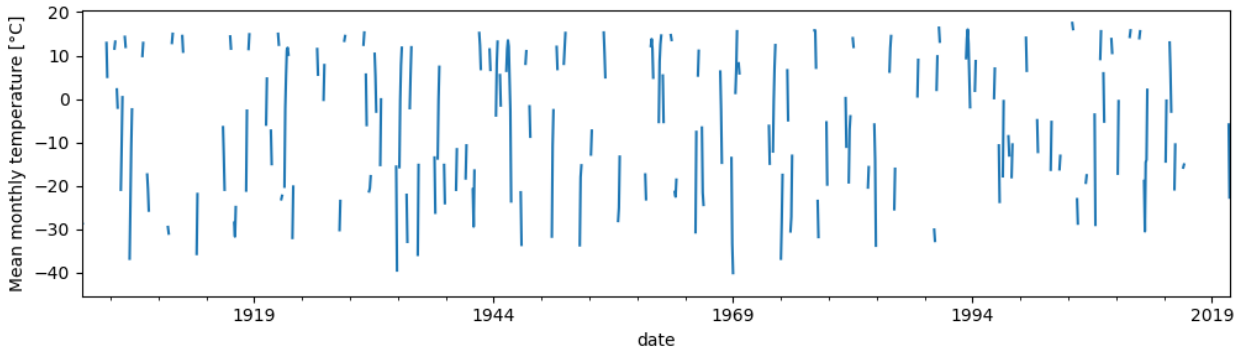
(continued from previous page)

```
0  POINT (-139.13000 64.07000)
1  POINT (-139.13000 64.07000)
2  POINT (-139.13000 64.07000)
3  POINT (-139.13000 64.07000)
4  POINT (-139.13000 64.07000)

[5 rows x 28 columns]
```

Now let's plot the mean temperature time series. Note that the server does not necessarily return a continuous time series, and when plotting time series with gaps, matplotlib just draws a continuous line between values. To convey the presence of missing values, here we'll use the `asfreq("MS")` method to fill-in gaps in the time series with explicit missing values.

```
# Set the DataFrame index to a datetime object and sort it
mts.set_index(pd.to_datetime(mts["date"]), inplace=True)
mts.sort_index(inplace=True)

# Convert the temperature to a continuous monthly time series (so missing values are␣
↪visible in the graphic)
tas = mts["temp_mean__temp_moyenne"].asfreq("MS")

# Mask missing values
tas.mask(tas < -300, inplace=True)

tas.plot(figsize=(12, 3))
plt.ylabel("Mean monthly temperature [°C]")
```

```
Text(0, 0.5, 'Mean monthly temperature [°C]')
```



1.2.11 Compute climate indicators on weather station data from ECCC's API

Environment and Climate Change Canada (ECCC) offers an API to access daily weather station data. This notebook focus is on converting the JSON response from the server to an `xarray.Dataset`. At that point, it's then easy to compute numerous climate indicators on the daily time series using xclim.

```
# NBVAL_IGNORE_OUTPUT

import os
```

(continues on next page)

(continued from previous page)

```
os.environ["USE_PYGEOS"] = "0"  # force use Shapely with GeoPandas

import warnings

import numba

warnings.simplefilter("ignore", category=numba.core.errors.NumbaDeprecationWarning)

import urllib.request

import cf_xarray as cfxr
import geopandas as gpd
from urlpath import URL

# Compose the request
host = URL("https://api.weather.gc.ca/")
api = host / "collections" / "climate-daily" / "items"
url = api.with_query(
    {
        "datetime": "1840-03-01 00:00:00/2021-06-02 00:00:00",
        "STN_ID": "10761",
        "sortby": "LOCAL_DATE",
    }
)

# Use geopandas to convert the json output to a GeoDataFrame.
with urllib.request.urlopen(url=str(url)) as req:
    gdf = gpd.read_file(filename=req, engine="pyogrio")
gdf.sort_index().head()
```

```
ERROR 1: PROJ: proj_create_from_database: Open of /home/docs/checkouts/readthedocs.org/
↪user_builds/pavics-sdi/conda/latest/share/proj failed
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/pyogrio/geopandas.py:49: FutureWarning: errors='ignore' is deprecated␣
↪and will raise in a future version. Use to_datetime without passing `errors` and catch␣
↪exceptions explicitly instead
  res = pd.to_datetime(ser, **datetime_kwargs)
```

```
                  id MAX_TEMPERATURE_FLAG SNOW_ON_GROUND_FLAG TOTAL_RAIN_FLAG  \
0  7024745.1994.7.26                 None                None               M
1  7024745.1994.7.27                 None                None               M
2  7024745.1994.7.28                 None                None               M
3  7024745.1994.7.29                 None                None               M
4  7024745.1994.7.30                 None                None               M

   MIN_TEMPERATURE  MAX_REL_HUMIDITY SNOW_ON_GROUND SPEED_MAX_GUST_FLAG  \
0             17.3               NaN           None                None
1             15.6               NaN           None                None
2             17.0               NaN           None                None
3             16.4               NaN           None                None
4             18.0               NaN           None                None
```

(continues on next page)

(continued from previous page)

```
  HEATING_DEGREE_DAYS_FLAG MEAN_TEMPERATURE_FLAG  ...  MIN_REL_HUMIDITY_FLAG  \
0                     None                  None  ...                     None
1                     None                  None  ...                     None
2                     None                  None  ...                     None
3                     None                  None  ...                     None
4                     None                  None  ...                     None

  MAX_TEMPERATURE  COOLING_DEGREE_DAYS HEATING_DEGREE_DAYS  \
0            26.7                  4.0                 0.0
1            23.5                  1.6                 0.0
2            21.2                  1.1                 0.0
3            27.0                  3.7                 0.0
4            23.7                  2.9                 0.0

  MAX_REL_HUMIDITY_FLAG TOTAL_SNOW  DIRECTION_MAX_GUST                 ID  \
0                  None       None                 0.0  7024745.1994.7.26
1                  None       None                 0.0  7024745.1994.7.27
2                  None       None                 0.0  7024745.1994.7.28
3                  None       None                 0.0  7024745.1994.7.29
4                  None       None                 0.0  7024745.1994.7.30

   DIRECTION_MAX_GUST_FLAG                    geometry
0                     None  POINT (-73.57917 45.50474)
1                     None  POINT (-73.57917 45.50474)
2                     None  POINT (-73.57917 45.50474)
3                     None  POINT (-73.57917 45.50474)
4                     None  POINT (-73.57917 45.50474)

[5 rows x 36 columns]
```

The next step is to convert the GeoDataFrame object into an `xarray.Dataset`, to do so, we've created some utilities packaged in `cf-xarray` to convert point geometries (the stations' locations) into CF-compliant coordinates. The function below bundles a number of operations to prepare the data for further analysis.

```
def preprocessing(gdf):
    """Convert geojson data from the ECCC weather API into a CF-compliant dataset.

    - Rename variables names CF standard names
    - Assign units to variables
    - Mask values with a flag
    - Convert wind directions in tens of degrees to degrees
    - Fill gaps in time series with NaNs

    Parameters
    ----------
    gdf : pandas.GeoDataFrame
      Data from the `api.weather.gc.ca` service.

    Returns
    -------
    xr.Dataset
```

(continues on next page)

(continued from previous page)

```
    Dataset complete with units and CF-compliant temporal and spatial coordinates.

    Notes
    -----
    DIRECTION_MAX_GUST is only reported if the maximum gust speed exceeds 29 km/h.
    A value of 0 or 360 means wind blowing from the geographic north, and 90 from the␣
↪east.

    """
    import pandas as pd
    import xarray as xr

    # Convert timestamp to datetime object
    gdf["time"] = gdf["LOCAL_DATE"].astype("datetime64[ns]")

    # Drop useless variables
    gdf = gdf.drop(["LOCAL_DATE", "LOCAL_YEAR", "LOCAL_MONTH", "LOCAL_DAY"], axis=1)

    # Convert to xarray.Dataset
    ds = gdf.set_index("time").to_xarray()

    # Convert geometries to CF - creates a features dimension
    ds = cfxr.geometry.reshape_unique_geometries(ds)
    coords = cfxr.shapely_to_cf(ds.geometry, grid_mapping="longitude_latitude")
    coords = coords.drop_vars(["geometry_container", "x", "y"])
    ds = xr.merge([ds.drop_vars("geometry"), coords])

    # Rename `features` dimension to `station`
    ds = ds.rename(features="station")

    # Mask values with a flag then drop flag variable
    for key in ds.data_vars:
        if key.endswith("_FLAG"):
            valid = ds[key].isnull()
            name = key.replace("_FLAG", "")
            ds[name] = ds[name].where(valid)
            ds = ds.drop_vars(key)

    # Convert wind gust from tens of degrees to degrees
    if "DIRECTION_MAX_GUST" in ds.data_vars:
        ds["DIRECTION_MAX_GUST"] *= 10

    # Assign units to variables
    # TODO: Add long_name and description from https://climate.weather.gc.ca/glossary_e.
↪html
    attrs = {
        "MEAN_TEMPERATURE": {
            "units": "degC",
            "standard_name": "air_temperature",
            "cell_methods": "time: mean within days",
        },
        "MIN_TEMPERATURE": {
```

(continues on next page)

(continued from previous page)

```
            "units": "degC",
            "standard_name": "air_temperature",
            "cell_methods": "time: min within days",
        },
        "MAX_TEMPERATURE": {
            "units": "degC",
            "standard_name": "air_temperature",
            "cell_methods": "time: max within days",
        },
        "TOTAL_PRECIPITATION": {
            "units": "mm/day",
            "standard_name": "precipitation_flux",
        },
        "TOTAL_RAIN": {
            "units": "mm/day",
        },
        "TOTAL_SNOW": {"units": "mm/day", "standard_name": "solid_precipitation_flux"},
        "SNOW_ON_GROUND": {"units": "cm", "standard_name": "surface_snow_thickness"},
        "DIRECTION_MAX_GUST": {
            "units": "degree",
            "standard_name": "wind_gust_from_direction",
        },
        "SPEED_MAX_GUST": {
            "units": "km/h",
            "standard_name": "wind_speed_of_gust",
            "cell_methods": "time: max within days",
        },
        "COOLING_DEGREE_DAYS": {"units": "degC days"},
        "HEATING_DEGREE_DAYS": {"units": "degC days"},
        "MIN_REL_HUMIDITY": {
            "units": "%",
            "standard_name": "relative_humidity",
            "cell_methods": "time: min within days",
        },
        "MAX_REL_HUMIDITY": {
            "units": "%",
            "standard_name": "relative_humidity",
            "cell_methods": "time: max within days",
        },
    }

    for key in ds.data_vars:
        ds[key].attrs.update(attrs.get(key, {}))

    # Try to squeeze arrays of identical values along the time dimension
    for key in ["STATION_NAME", "CLIMATE_IDENTIFIER", "PROVINCE_CODE"]:
        ds[key] = squeeze_if_unique(ds[key], "time")

    # Reindex over continuous time series
    ts = pd.date_range(ds.time[0].values, ds.time[-1].values)
    return ds.reindex(time=ts)
```

(continues on next page)

(continued from previous page)

```
def squeeze_if_unique(da, dim):
    """Squeeze dimension out of DataArray if all values are identical or masked.

    If a value is replicated along the time dimension, this function will return a
    DataArray defined only over the dimensions where values vary. If the resulting
    object is a scalar, it is converted to a global attribute.

    Parameters
    ----------
    da : xr.DataArray
      Input values.
    dim : str
      Dimension to squeeze if possible.
    """
    import numpy as np

    def n_unique(arr):
        return len(set(np.unique(arr)) - {""})

    if da.dtype == object:
        # Check if possible to squeeze along `dim`
        n = np.apply_along_axis(
            n_unique,
            da.get_axis_num(dim),
            da.fillna("").values,
        )

        if (n == 1).all():
            return da.max(dim)  # Should avoid returning the null value.

        return da

    # else : int, float or datetime
    da_filled = da.ffill(dim).bfill(dim)
    if (da_filled.isel({dim: 0}) == da_filled).all():
        return da_filled.isel({dim: 0}, drop=True)

    return da
```

```
# NBVAL_IGNORE_OUTPUT


# Convert the GeoDataFrame to an xarray.Dataset
# Different stations are aligned along the `station` dimension
ds = preprocessing(gdf)
ds
```

```
<xarray.Dataset> Size: 136kB
Dimensions:              (station: 1, time: 1061)
Coordinates:
```

(continues on next page)

(continued from previous page)

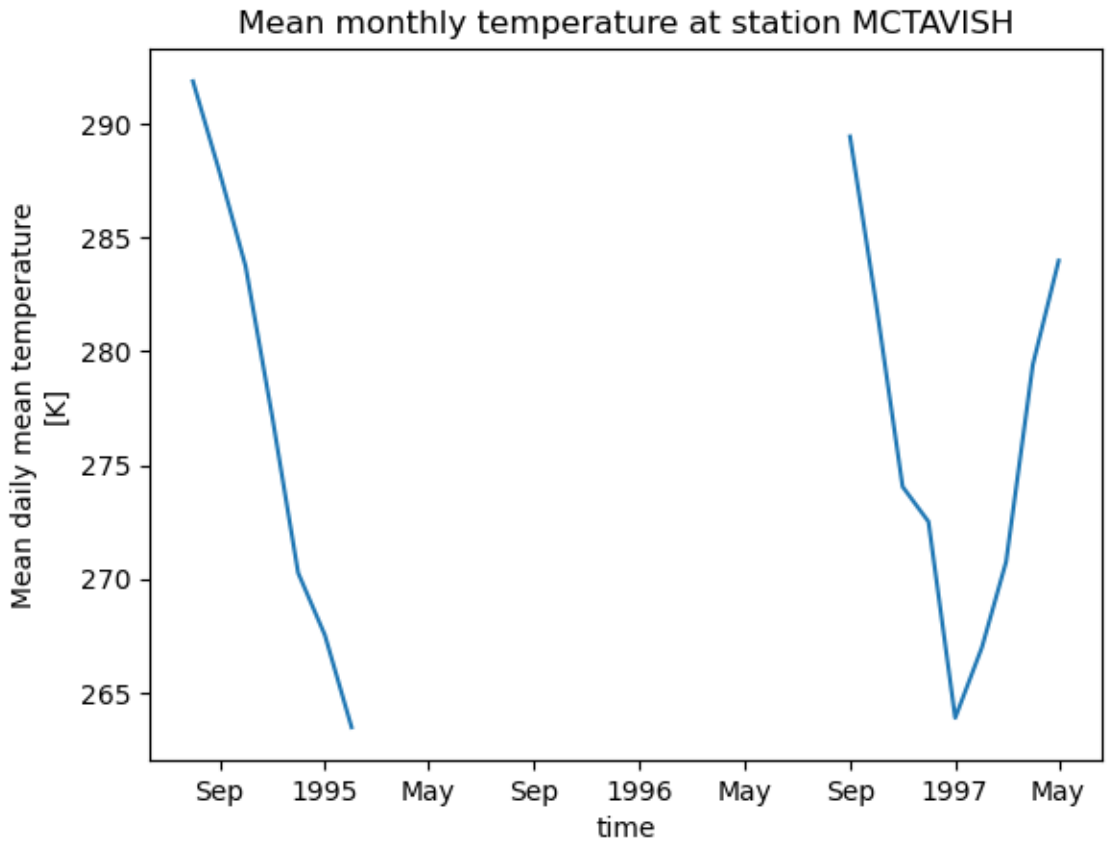
```
  * station              (station) int64 8B 0
  * time                 (time) datetime64[ns] 8kB 1994-07-26 ... 1997-06-20
    lon                  (station) float64 8B -73.58
    lat                  (station) float64 8B 45.5
Data variables: (12/18)
    id                   (station, time) object 8kB '7024745.1994.7.26' ... '...
    MIN_TEMPERATURE      (station, time) float64 8kB 17.3 15.6 ... 15.7 15.6
    MAX_REL_HUMIDITY     (station, time) float64 8kB nan nan nan ... 99.0 74.0
    SNOW_ON_GROUND       (station, time) object 8kB None None None ... None None
    CLIMATE_IDENTIFIER   (station) object 8B '7024745'
    TOTAL_PRECIPITATION  (station, time) float64 8kB 4.5 1.2 0.0 ... 6.8 1.0 0.0
    ...                   ...
    MAX_TEMPERATURE      (station, time) float64 8kB 26.7 23.5 ... 25.0 25.9
    COOLING_DEGREE_DAYS  (station, time) float64 8kB 4.0 1.6 1.1 ... 0.0 2.4 2.8
    HEATING_DEGREE_DAYS  (station, time) float64 8kB 0.0 0.0 0.0 ... 0.8 0.0 0.0
    TOTAL_SNOW           (station, time) object 8kB nan nan nan ... nan nan nan
    DIRECTION_MAX_GUST   (station, time) float64 8kB 0.0 0.0 0.0 ... 0.0 0.0 0.0
    ID                   (station, time) object 8kB '7024745.1994.7.26' ... '...
```

Computing climate indicators

In the next cell, we compute the monthly mean temperature from the daily observations. By default, `xclim` is very strict about missing values, marking any month with one missing value as entirely missing. Here we'll use the WMO recommendation for missing data, where a month is considered missing if there are 11 days or more missing, or 5 consecutive missing values.

```
import xclim

with xclim.set_options(check_missing="wmo"):
    mtas = xclim.atmos.tg_mean(ds.MEAN_TEMPERATURE, freq="MS")
```

```
from matplotlib import pyplot as plt

name = ds.STATION_NAME.isel(station=0).values
mtas.isel(station=0).plot()
plt.title(f"Mean monthly temperature at station {name}")
plt.show()
```



1.2.12 Web Coverage Service - Accessing GeoMet data using owslib

In this notebook we'll connect to Environment Canada's GeoMet service and fetch data using the WCS standard.

```
%matplotlib inline

import matplotlib.pyplot as plt
import xarray as xr
from owslib.wcs import WebCoverageService
```

```
# NBVAL_IGNORE_OUTPUT
wcs_url = "http://geo.weather.gc.ca/geomet/?lang=en&service=WCS"

# Connect to service
wcs = WebCoverageService(wcs_url, version="2.0.1")
print(wcs.identification.title)

# List some of the content available
sorted(list(wcs.contents.keys()))[:10]
```

```
MSC GeoMet - GeoMet-Weather 2.26.2
```

```
['CIOPS-East_2km_MixedLayerThickness',
 'CIOPS-East_2km_SeaIceAreaFraction',
 'CIOPS-East_2km_SeaIceCompressiveStrength',
 'CIOPS-East_2km_SeaIceDivergence',
 'CIOPS-East_2km_SeaIceInternalPressure',
 'CIOPS-East_2km_SeaIceShear',
 'CIOPS-East_2km_SeaIceSnowTemp',
 'CIOPS-East_2km_SeaIceSnowVol',
 'CIOPS-East_2km_SeaIceVol',
 'CIOPS-East_2km_SeaSfcHeight']
```

Now let's get some information about a given layer, here the salinity.

```
layerid = "OCEAN.GIOPS.3D_SALW_0000"
temp = wcs[layerid]
# Title
print("Layer title :", temp.title)
# bounding box
print("BoundingBox :", temp.boundingBoxWGS84)
# supported data formats - we'll use geotiff
print("Formats :", temp.supportedFormats)
# grid dimensions
print("Grid upper limits :", temp.grid.highlimits)
```

```
Layer title : None
BoundingBox : None
Formats : ['image/tiff', 'image/x-aaigrid', 'image/netcdf', 'application/x-grib2',
↪'image/png', 'image/jpeg', 'image/png; mode=8bit', 'image/vnd.jpeg-png', 'image/vnd.
↪jpeg-png8']
```

```
Grid upper limits : ['1799', '849']
```

To request data, we need to call the `getCoverage` service, which requires us specifying the geographic projection, the bounding box, the resolution and format of the output. With GeoMet 2.0.1, we can now get layers in the netCDF format.

```
format_wcs = "image/netcdf"
bbox_wcs = temp.boundingboxes[0]["bbox"]  # Get the entire domain
crs_wcs = temp.boundingboxes[0]["nativeSrs"]  # Coordinate system
w = int(temp.grid.highlimits[0])
h = int(temp.grid.highlimits[1])

print("Format:", format_wcs)
print("Bounding box:", bbox_wcs)
print("Projection:", crs_wcs)
print(f"Resolution: {w} x {h}")

output = wcs.getCoverage(
    identifier=[
        layerid,
    ],
    crs=crs_wcs,
    bbox=bbox_wcs,
```

(continues on next page)

(continued from previous page)

```
    width=w,
    height=h,
    format=format_wcs,
)
```

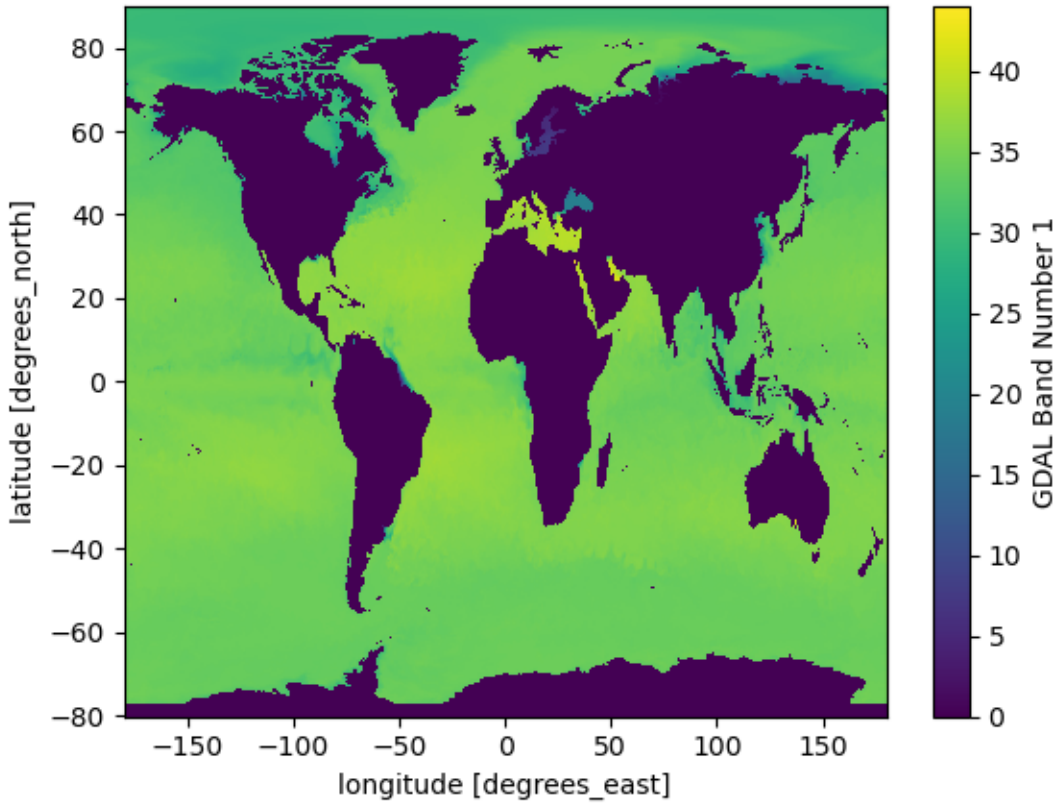
```
Format: image/netcdf
Bounding box: (-80.1, -180.0, 89.9, 180.0)
Projection: http://www.opengis.net/def/crs/EPSG/0/4326
Resolution: 1799 x 849
```

We then save the output to disk, open it normally using xarray and plot it's variable.

```
fn = layerid + ".nc"
with open(fn, "wb") as fh:
    fh.write(output.read())
```

```
ds = xr.open_dataset(fn)
print(ds.data_vars)
ds.Band1.plot()
plt.show()
```

```
Data variables:
    crs      |S1 1B ...
    Band1    (lat, lon) float32 6MB ...
```



1.2.13 Web Feature Service - Accessing region countours saved on a GeoServer

In this example, we're going to look at some layers that are currently accessible on our instance of GeoServer. With WFS, we can see what is available, collect the layers we want by using a query, download the results in geoJSON, and visualize them using `geopandas`.

We begin by loading the libraries needed for parsing and downloading from WFS and for opening and visualizing the results

```
%matplotlib inline

import os

os.environ["USE_PYGEOS"] = "0"  # force use Shapely with GeoPandas

import geopandas as gpd

# Import WFS from owslib
from owslib.wfs import WebFeatureService
```

We start by making a connection to the PAVICS instance we have locally on our server. Using WFS, we can very quickly see the contents, which are the layers and the workspaces they're located with (ie: TravisTest, scratchTJS). These layer names act as dictionaries for

```
# NBVAL_IGNORE_OUTPUT

wfs_url = "https://pavics.ouranos.ca/geoserver/wfs"  # TEST_USE_PROD_DATA

# Connect to GeoServer WFS service.
wfs = WebFeatureService(wfs_url, version="2.0.0")

# Print the list of available layers
sorted(wfs.contents.keys())
```

```
['TravisTest:NE_Admin_Level0',
 'TravisTest:mrc_poly',
 'TravisTest:region_admin_poly',
 'public:CANOPEX_5797_basinBoundaries',
 'public:CANVEC_hydro_waterbodies',
 'public:CanVec_Rivers',
 'public:CanVec_WaterBodies',
 'public:HydroLAKES_points',
 'public:HydroLAKES_poly',
 'public:USGS_HydroBASINS_lake_ar_lev12',
 'public:USGS_HydroBASINS_lake_na_lev12',
 'public:canada_admin_boundaries',
 'public:decamillenial_flood_CC',
 'public:gaspesie_mrc',
 'public:global_admin_boundaries',
 'public:ne_10m_populated_places',
 'public:quebec_admin_boundaries',
 'public:quebec_health_regions',
 'public:quebec_mrc_boundaries',
 'public:quebec_muni_boundaries',
```

(continues on next page)

(continued from previous page)

```
 'public:routing_1kmLakes_07',
 'public:routing_1kmLakes_08',
 'public:routing_1kmLakes_09',
 'public:routing_1kmLakes_10',
 'public:routing_1kmLakes_11',
 'public:routing_1kmLakes_12',
 'public:routing_allLakes_07',
 'public:routing_allLakes_08',
 'public:routing_allLakes_09',
 'public:routing_allLakes_10',
 'public:routing_allLakes_11',
 'public:routing_allLakes_12',
 'public:test_regions',
 'public:test_regions_lambert',
 'public:usa_admin_boundaries',
 'public:wshed_bound_n1',
 'public:wshed_bound_n2',
 'public:wshed_bound_n3']
```

More information about each layer is stored in the `contents` dictionary.

```
# NBVAL_IGNORE_OUTPUT

sorted_layer_ids = list(sorted(wfs.contents.keys()))
canada_admin_boundaries_index = sorted_layer_ids.index("public:canada_admin_boundaries")

for layerID in sorted_layer_ids[
    canada_admin_boundaries_index - 1 : canada_admin_boundaries_index + 2
]:
    layer = wfs[layerID]
    print("Layer ID:", layerID)
    print("Title:", layer.title)
    print("Boundaries:", layer.boundingBoxWGS84, "\n")
```

```
Layer ID: public:USGS_HydroBASINS_lake_na_lev12
Title: USGS_HydroBASINS_lake_na_lev12
Boundaries: (-180.0, -90.0, 180.0, 90.0)

Layer ID: public:canada_admin_boundaries
Title: Canada Administrative Boundaries
Boundaries: (-141.01807315799994, 41.681435425000075, -52.61940850399992, 83.
↪13550252400006)

Layer ID: public:decamillenial_flood_CC
Title: decamillenial_flood_CC
Boundaries: (-180.0, -90.0, 180.0, 90.0)
```

We can then perform a `GetFeatures` call using the layer name as a target. This returns an IOstream that can be written as a geoJSON file, a common file format for vector data served throughout the web. To reduce the download size, we'll only fetch the features (here polygons), intersecting a small region defined by a bounding box.

```
layer_id = "public:canada_admin_boundaries"
```

(continues on next page)

(continued from previous page)

```
meta = wfs.contents[layer_id]
print(meta.title)

# Get the actual data
data = wfs.getfeature(
    typename="public:canada_admin_boundaries",
    bbox=(-74.5, 45.2, -73, 46),
    outputFormat="JSON",
)

# Write to file
fn = "output.geojson"
with open(fn, "wb") as fh:
    fh.write(data.read())
```

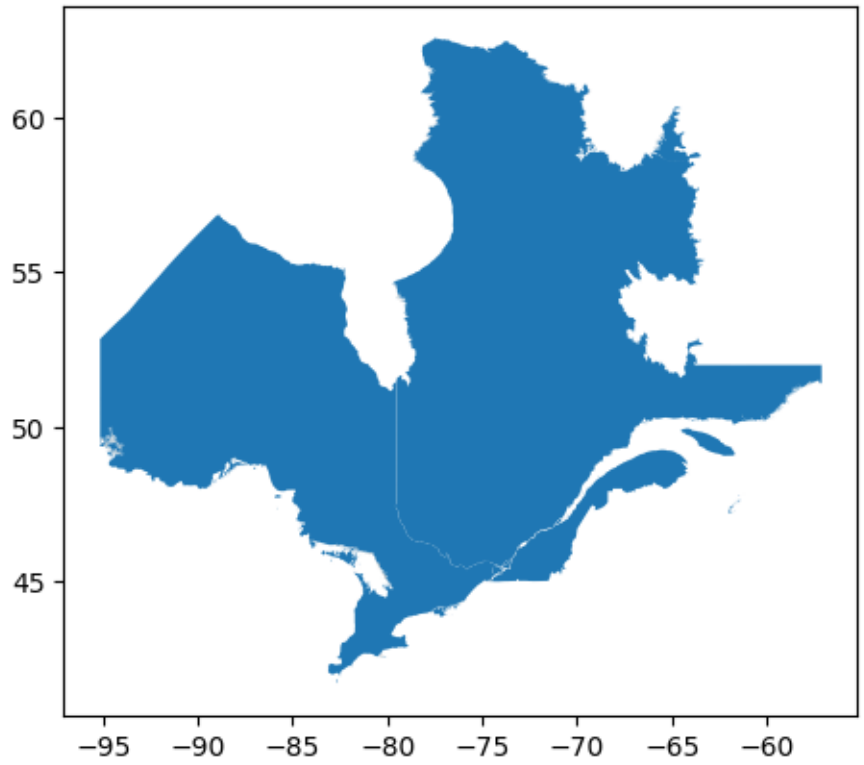
```
Canada Administrative Boundaries
```

Once the geoJSON file is downloaded, we can either open it with a GIS application or we can read the features using geopandas.

```
layers = gpd.read_file(fn)
layers.plot()
```

```
ERROR 1: PROJ: proj_create_from_database: Open of /home/docs/checkouts/readthedocs.org/
↪user_builds/pavics-sdi/conda/latest/share/proj failed
```

```
<Axes: >
```



1.2.14 Working with Web Processing Service with Python and OWSLib

In this notebook, you will interact with a Web Processing Service (WPS) server using the OWSLib library, a client for OGC Web Services. More specifically, you'll learn how to list available processes, get information about a process, and execute a process. If you need more information than provided here, refer to the OWSLib documentation.

Connect to server and list processes

Here is how to establish a connection to a WPS server and list the available processes. We first connect to the server, then send a `getCapabilities` request to the server, which answers with a list of all the processes it hosts. This allows the client to populate its `processes` attribute.

```
from owslib.wps import WebProcessingService

# Connect to the server
wps = WebProcessingService('https://pavics.ouranos.ca/twitcher/ows/proxy/finch')

print(wps.identification.title)

# Send a `getCapabilities` request to populate the list of processes
wps.getcapabilities()

# Print out the identifiers and abstract of five processes
for process in wps.processes[:5]:
    print(f"{process.identifier} \t : {process.abstract} \n")
```

```
Finch
```

```
humidex          : The humidex indicates how hot the air feels to an average person,␣
↪accounting for the effect of humidity. It can be loosely interpreted as the equivalent␣
↪perceived temperature when the air is dry.

heat_index          : Perceived temperature after relative humidity is taken into␣
↪account ([Blazejczyk2012]_). The index is only valid for temperatures above 20degC.

tg          : We assume a symmetrical distribution for the temperature and retrieve the␣
↪average value as Tg = (Tx + Tn) / 2

wind_speed_from_vector          : Computes the magnitude and angle of the wind vector␣
↪from its northward and eastward components, following the meteorological convention␣
↪that sets calm wind to a direction of 0deg and northerly wind to 360deg.

wind_vector_from_speed          : Compute the eastward and northward wind components␣
↪from the wind speed and direction.
```

Get information about a single process

Assume we're interested in using the `humidex` process, we first need to know which arguments this process expects. To get this information, we must first send a `describeProcess` request to the server for the process description.

```
process = wps.describeprocess('humidex')
print(process.title, " : ", process.abstract)
print ([x.identifier for x in process.dataInputs])
```

```
Humidex index.  :  The humidex indicates how hot the air feels to an average person,␣
↪accounting for the effect of humidity. It can be loosely interpreted as the equivalent␣
↪perceived temperature when the air is dry.
['tas', 'tdps', 'hurs', 'check_missing', 'missing_options', 'cf_compliance', 'data_
↪validation', 'variable', 'output_name', 'output_format', 'csv_precision']
```

Each process has a list of `dataInput` objects, each with an individual title and abstract, a list of supported data types, constraints on the minimum and maximum of values it can take, and optionally a default value. For example, the `tas` argument is mandatory (`minOccurs` is set to 1).

```
tas = process.dataInputs[0]
print(tas.abstract)
print(tas.minOccurs)
```

```
NetCDF Files or archive (tar/zip) containing netCDF files. Mean surface temperature.
1
```

Similarly, each process' has a list of `processOutput` object with descriptive fields, in addition to methods to retrieve the actual data from the server once the process has been executed.

```
from owslib.wps import ComplexDataInput

url = "https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/fileServer/birdhouse/
```

(continues on next page)

(continued from previous page)

```
↪testdata/xclim/cmip5/tas_Amon_CanESM2_rcp85_r1i1p1_200701-200712.nc"

exec = wps.execute("tg_mean", inputs=[
         ("tas", ComplexDataInput(url)),
         ("data_validation", "log")],
         mode="sync")

print(exec.status)
```

```
ProcessSucceeded
```

Retrieving the execution results

In synchronous mode, once the execution has completed the client automatically fetches the response. The response is an XML document that either stores the results, or links to the results, depending on how the process and server are configured.

```
from lxml import etree

print(etree.tostring(exec.response).decode())
```

```
<wps:ExecuteResponse xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.
↪opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.
↪w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 ..
↪/wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US"␣
↪serviceInstance="http://localhost:5000/wps?request=GetCapabilities&amp;amp;service=WPS
↪" statusLocation="">
    <wps:Process wps:processVersion="0.1">
        <ows:Identifier>tg_mean</ows:Identifier>
        <ows:Title>Mean of daily average temperature.</ows:Title>
        <ows:Abstract>Resample the original daily mean temperature series by taking the␣
↪mean over each period.</ows:Abstract>
        </wps:Process>
    <wps:Status creationTime="2024-04-16T18:28:58Z">
        <wps:ProcessSucceeded>PyWPS Process Mean of daily average temperature. finished</
↪wps:ProcessSucceeded>
        </wps:Status>
        <wps:ProcessOutputs>
                <wps:Output>
            <ows:Identifier>output</ows:Identifier>
            <ows:Title>Result</ows:Title>
            <ows:Abstract>The format depends on the 'output_format' input parameter.</
↪ows:Abstract>
            <wps:Reference href="https://pavics.ouranos.ca/wpsoutputs/finch/2f137ee2-
↪fc1f-11ee-ac94-0242ac130003/out.nc" mimeType="application/x-netcdf" encoding="base64"␣
↪schema=""/>
                </wps:Output>
                <wps:Output>
            <ows:Identifier>output_log</ows:Identifier>
            <ows:Title>Logging information</ows:Title>
```

(continues on next page)

(continued from previous page)

```
            <ows:Abstract>Collected logs during process run.</ows:Abstract>
            <wps:Reference href="https://pavics.ouranos.ca/wpsoutputs/finch/2f137ee2-
↪fc1f-11ee-ac94-0242ac130003/log.txt" mimeType="text/plain" encoding="" schema=""/>
                </wps:Output>
                <wps:Output>
            <ows:Identifier>ref</ows:Identifier>
            <ows:Title>Link to all output files</ows:Title>
            <ows:Abstract>Metalink file storing all references to output files.</
↪ows:Abstract>
                        <wps:Data>
                    <wps:ComplexData mimeType="application/metalink+xml; version=4.0"␣
↪encoding="" schema="metalink/4.0/metalink4.xsd">&lt;?xml version="1.0" encoding="UTF-8
↪"?&gt;
&lt;metalink xmlns="urn:ietf:params:xml:ns:metalink"&gt;
    &lt;published&gt;2024-04-16T18:28:58Z&lt;/published&gt;
    &lt;generator&gt;PyWPS/4.5.2&lt;/generator&gt;

    &lt;file name="out.nc"&gt;
        &lt;identity&gt;out&lt;/identity&gt;
        &lt;size&gt;91126&lt;/size&gt;
        &lt;metaurl mediatype="application/x-netcdf"&gt;https://pavics.ouranos.ca/
↪wpsoutputs/finch/30740cf2-fc1f-11ee-ac94-0242ac130003/out.nc&lt;/metaurl&gt;
        &lt;publisher name="Finch" url="http://localhost:5000/wps"/&gt;
    &lt;/file&gt;

&lt;/metalink&gt;</wps:ComplexData>
                        </wps:Data>
                </wps:Output>
        </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

In the case of the `tg_mean` process, the results are stored in a netCDF file, which is returned as a `ComplexDataOutput` object. It is stored on the server, so the response only contains the URL to this output file.

```
out = exec.processOutputs[0]

# The link to the output stored on the server
print(out.reference)
```

```
https://pavics.ouranos.ca/wpsoutputs/finch/2f137ee2-fc1f-11ee-ac94-0242ac130003/out.nc
```

You may retrieve the data directly with `out.retrieveData()`, in the case above you'll get the bytes of the netCDF result file, or save the results to a file on disk using `exec.getOutput(out.identifier, filepath="path/to/file")`.

1.2.15 Working with owslib's WMS interface

OWSlib is a Python package for client programming with OGC web service interface standards. In this tutorial we'll work with the WMS interface.

```
from owslib.wms import WebMapService
```

Web Mapping Service

We start by fetching a map using the WMS protocol. We first instantiate a `WebMapService` object using the address of the NASA server, then browse through its content.

```
wms = WebMapService("https://neo.gsfc.nasa.gov/wms/wms")
print("Title: ", wms.identification.title)
print("Type: ", wms.identification.type)
print("Operations: ", [op.name for op in wms.operations])
print("GetMap options: ", wms.getOperationByName("GetMap").formatOptions)
wms.contents.keys()
```

```
Traceback (most recent call last):

  File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/IPython/core/interactiveshell.py:3577 in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[2], line 1
    wms = WebMapService("https://neo.gsfc.nasa.gov/wms/wms")

  File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/owslib/wms.py:50 in WebMapService
    return wms111.WebMapService_1_1_1(

  File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/owslib/map/wms111.py:74 in __init__
    self._capabilities = reader.read(self.url, timeout=self.timeout)

  File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/owslib/map/common.py:69 in read
    return etree.fromstring(raw_text)

  File src/lxml/etree.pyx:3264 in lxml.etree.fromstring

  File src/lxml/parser.pxi:1989 in lxml.etree._parseMemoryDocument

  File src/lxml/parser.pxi:1876 in lxml.etree._parseDoc

  File src/lxml/parser.pxi:1164 in lxml.etree._BaseParser._parseDoc

  File src/lxml/parser.pxi:633 in lxml.etree._ParserContext._handleParseResultDoc

  File src/lxml/parser.pxi:743 in lxml.etree._handleParseResult
```

(continues on next page)

(continued from previous page)

```
  File src/lxml/parser.pxi:672 in lxml.etree._raiseParseError

  File <string>:1
XMLSyntaxError: Start tag expected, '<' not found, line 1, column 1
```

The `content` is a dictionary holding metadata for each layer. We'll print some of the metadata' title for a couple of layers to see what's in it.

```
for key in [
    "MOD14A1_M_FIRE",
    "CERES_LWFLUX_M",
    "ICESAT_ELEV_G",
    "MODAL2_M_CLD_WP",
    "MOD_143D_RR",
]:
    print(wms.contents[key].title)
```

```
Active Fires (1 month - Terra/MODIS)
Outgoing Longwave Radiation (1 month)
Greenland / Antarctica Elevation
Cloud Water Content (1 month - Terra/MODIS)
True Color (1 day - Terra/MODIS)
```

We'll select the true color Earth imagery from Terra/MODIS. Let's check out some of its properties. We can also pretty print the full abstract with HTML.

```
# NBVAL_IGNORE_OUTPUT

from IPython.core.display import HTML

name = "MOD_143D_RR"
layer = wms.contents[name]
print("Abstract: ", layer.abstract)
print("BBox: ", layer.boundingBoxWGS84)
print("CRS: ", layer.crsOptions)
print("Styles: ", layer.styles)
print("Timestamps: ", layer.timepositions)
HTML(layer.parent.abstract)
```

```
Abstract:  None
BBox:  (-180.0, -90.0, 180.0, 90.0)
CRS:  ['EPSG:4326']
Styles:  {}
Timestamps:  ['2006-09-01/2006-09-14/P1D', '2006-09-17/2006-10-10/P1D', '2006-10-12/2006-
↪11-18/P1D', '2006-11-21/2007-08-16/P1D', '2007-08-18', '2007-08-20/2007-09-11/P1D',
↪'2007-09-15/2007-12-30/P1D', '2008-01-01/2008-06-12/P1D', '2008-06-14', '2008-06-16/
↪2008-07-12/P1D', '2008-07-14/2008-09-17/P1D', '2008-09-19', '2008-09-22/2008-10-17/P1D
↪', '2008-10-19/2008-10-22/P1D', '2008-10-28/2008-12-02/P1D', '2008-12-04/2008-12-20/P1D
↪', '2008-12-23/2008-12-30/P1D', '2009-01-01/2009-01-20/P1D', '2009-01-22/2009-04-19/P1D
↪', '2009-04-23/2009-07-05/P1D', '2009-07-08/2009-12-30/P1D', '2010-01-01/2010-07-16/P1D
↪', '2010-07-18/2010-12-07/P1D', '2010-12-09/2010-12-30/P1D', '2011-01-01/2011-01-25/P1D
↪', '2011-01-27/2011-03-19/P1D', '2011-03-21/2011-07-23/P1D', '2011-07-27/2011-08-27/P1D
```

(continues on next page)

(continued from previous page)

```
↪', '2011-08-30/2011-12-13/P1D', '2011-12-15/2012-02-19/P1D', '2012-02-21/2013-12-01/P1D
↪', '2013-12-04/2018-03-12/P1D', '2018-03-14/2018-05-16/P1D', '2018-05-18/2018-09-17/P1D
↪', '2018-09-19/2022-05-04/P1D', '2022-05-06/2022-12-21/P1D', '2022-12-23/2023-09-28/P1D
↪', '2023-10-01/2023-10-17/P1D']
```

```
<IPython.core.display.HTML object>
```

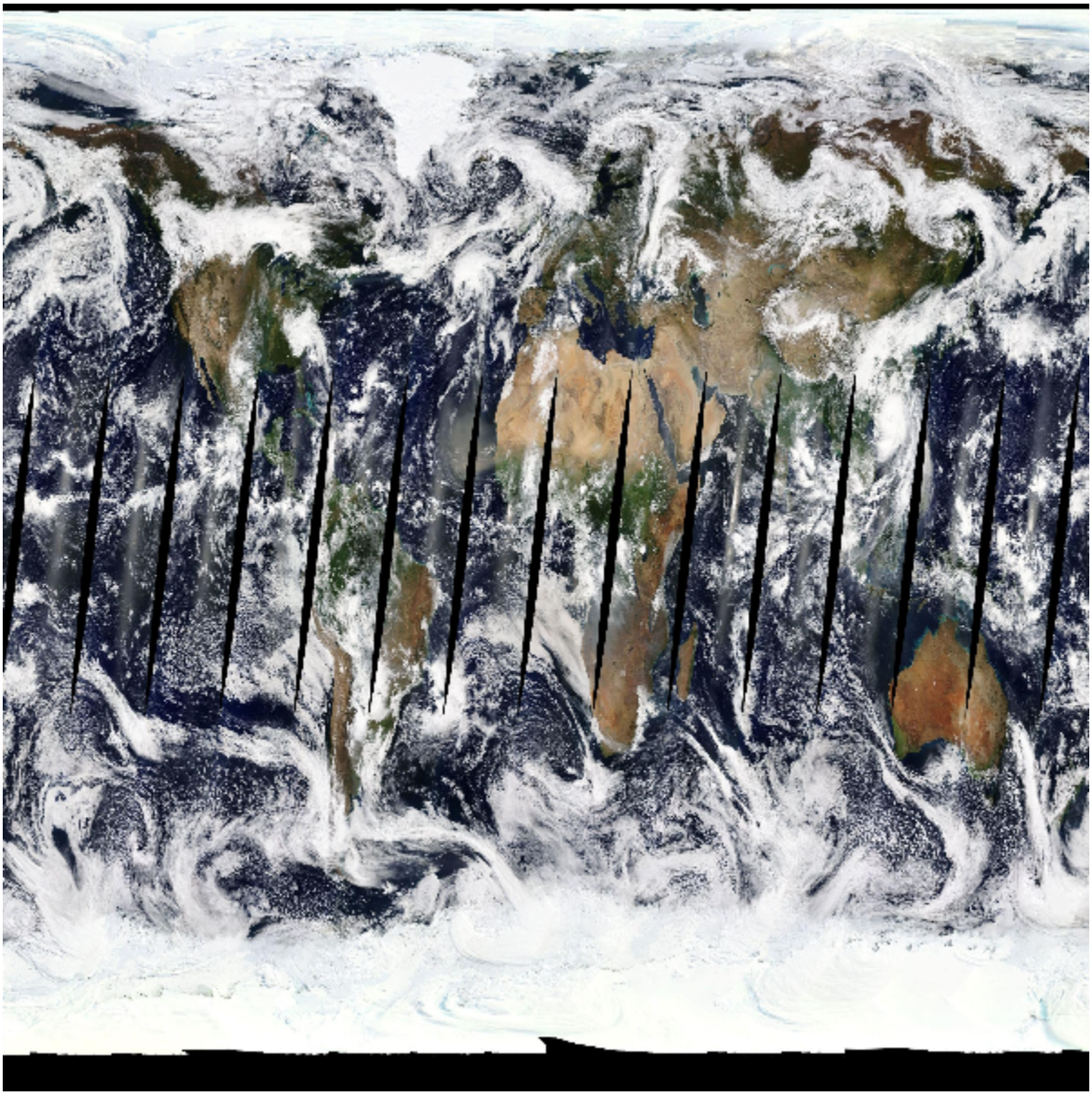
Getting the image data

Now let's get the image ! The response we're getting is a `ResponseWrapper` object, we need to `read` its content to get the actual bytes for the png file. Let's first display the raw image, then try to map it onto a projection of the Earth.

```
response = wms.getmap(
    layers=[
        name,
    ],
    styles=["rgb"],
    bbox=(-180, -90, 180, 90),  # Left, bottom, right, top
    format="image/png",
    size=(600, 600),
    srs="EPSG:4326",
    time="2018-09-16",
    transparent=True,
)
response
```

```
<owslib.util.ResponseWrapper at 0x7fa543b4fdf0>
```

```
from IPython.display import Image

Image(response.read())
```

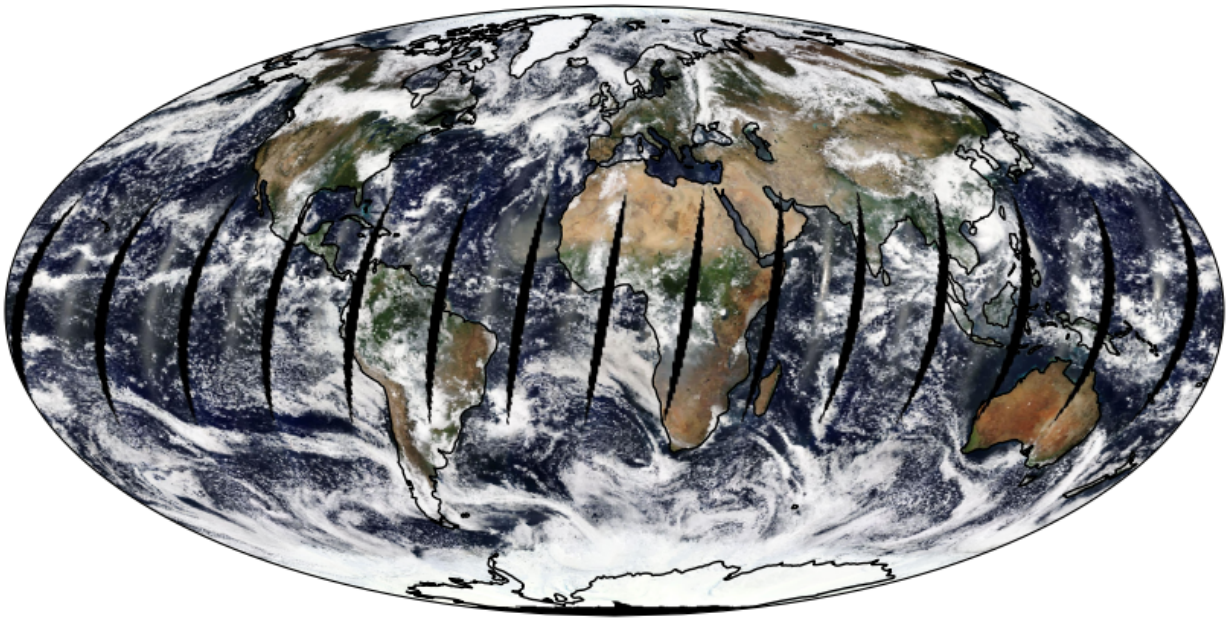


Plotting the image on a map

Using the `cartopy` library, we'll overlay the image on a map of the Earth. Since Matplotlib's `imread` function expects a file-like object, we'll mimic a file object in memory using the `io.BytesIO` function.

```
import io
import warnings

import cartopy
import matplotlib.pyplot as plt

image = io.BytesIO(response.read())
data = plt.imread(image)
```

```
warnings.filterwarnings("ignore", category=cartopy.io.DownloadWarning)
fig = plt.figure(figsize=(8, 6))
ax = fig.add_axes([0, 0, 1, 1], projection=cartopy.crs.Mollweide())
ax.imshow(
    data,
    origin="upper",
    extent=(-180, 180, -90, 90),
    transform=cartopy.crs.PlateCarree(),
)
ax.coastlines()
plt.show()
```



1.2.16 Regridding climate data with xESMF

A common element of climate data workflows is regridding, or reprojection, of model data unto more standard grids, or simply unto another dataset's grid. The powerful ESMF program, written in FORTRAN, has long been a reference in the matter. The xESMF python package provides an easy to use high-level API for using ESMF's methods. This notebook shows some examples of common regridding operations.

Regridding with `xESMF` is usually a two step process:

1. Create a `Regridder` objects from two datasets, defining the input and the output grids. This compute a weights mask which can, if needed, be saved to a netCDCF file.
2. Regrid a DataArray or Dataset by calling the `Regridder` with it. As the weights have already been computed, it reuses them for all time slices, which allows much better performance than, for example, interpolation using `scipy.interpolation.interpn`.

```
# NBVAL_IGNORE_OUTPUT
import geopandas as gpd  # isort: skip
from owslib.wfs import WebFeatureService  # isort: skip
```

(continues on next page)

(continued from previous page)

```
import warnings

# ipykernel_launcher.py:1: DeprecationWarning: xclim.subset is deprecated in xclim v0.19.
↪1-beta. Please take note that xclim presently exposes the 'clisops' library subsetting␣
↪API via `from clisops.core import subset`.
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Other utilities for style and clean examples
import copy
import json

import cf_xarray as cfxr
import matplotlib.pyplot as plt
import shapely
import xarray as xr
import xesmf as xe
from clisops.core.subset import subset_bbox  # For subsetting
from xclim.testing import open_dataset  # For opening xclim's test data

#  A colormap with grey where the data is missing
cmap = copy.copy(plt.cm.get_cmap("viridis"))
cmap.set_bad("lightgray")
```

```
WARNING:pint.util:Redefining 'percent' (<class 'pint.delegates.txt_defparser.plain.
↪UnitDefinition'>)
```

```
WARNING:pint.util:Redefining '%' (<class 'pint.delegates.txt_defparser.plain.
↪UnitDefinition'>)
```

```
WARNING:pint.util:Redefining 'year' (<class 'pint.delegates.txt_defparser.plain.
↪UnitDefinition'>)
```

```
WARNING:pint.util:Redefining 'yr' (<class 'pint.delegates.txt_defparser.plain.
↪UnitDefinition'>)
```

```
WARNING:pint.util:Redefining 'C' (<class 'pint.delegates.txt_defparser.plain.
↪UnitDefinition'>)
```

```
WARNING:pint.util:Redefining 'd' (<class 'pint.delegates.txt_defparser.plain.
↪UnitDefinition'>)
```

```
WARNING:pint.util:Redefining 'h' (<class 'pint.delegates.txt_defparser.plain.
↪UnitDefinition'>)
```

```
WARNING:pint.util:Redefining 'degrees_north' (<class 'pint.delegates.txt_defparser.plain.
↪UnitDefinition'>)
```

```
WARNING:pint.util:Redefining 'degrees_east' (<class 'pint.delegates.txt_defparser.plain.
↪UnitDefinition'>)
```

```
WARNING:pint.util:Redefining 'degrees' (<class 'pint.delegates.txt_defparser.plain.
↪UnitDefinition'>)
```

```
WARNING:pint.util:Redefining '[speed]' (<class 'pint.delegates.txt_defparser.plain.
↪DerivedDimensionDefinition'>)
```

Simple example : Bilinear regridding from model to obs

Our input in this example is a year of monthly sea ice concentration data from a CanESM5 run for CMIP6. It lies on an irregular grid defined by `latitude` and `longitude` coordinates. We'll interpolate the sea ice concentration to a regular observational grid from Natural Resources Canada.

The input data

```
# NBVAL_IGNORE_OUTPUT

# The input test data is hosted on the Ouranos THREDDS
url = "https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/testdata/
↪xclim/cmip6/sic_SImon_CCCma-CanESM5_ssp245_r13i1p2f1_2020.nc"
ds_in = xr.open_dataset(url)
ds_in
```

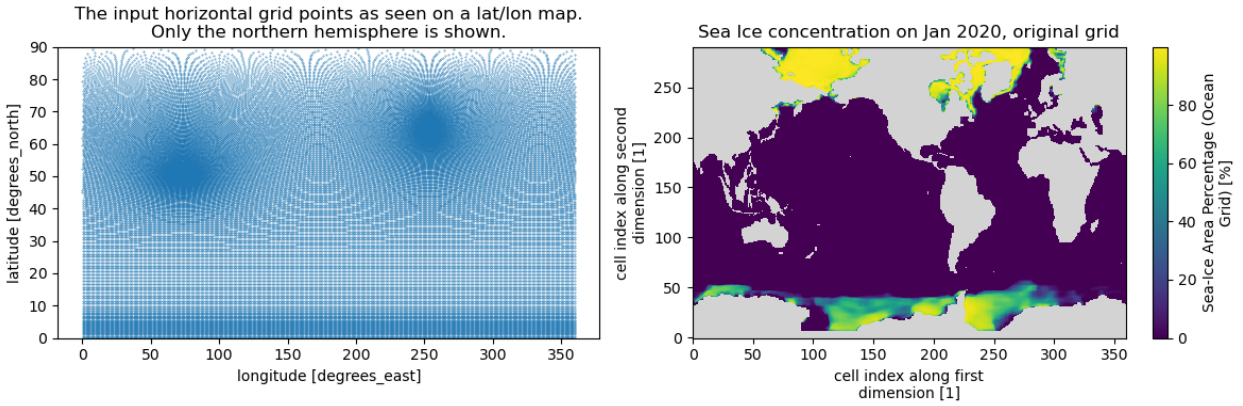
```
<xarray.Dataset> Size: 14MB
Dimensions:             (time: 12, bnds: 2, j: 291, i: 360, vertices: 4)
Coordinates:
  * time                (time) object 96B 2020-01-16 12:00:00 ... 2020-12-16 ...
  * j                   (j) int32 1kB 0 1 2 3 4 5 6 ... 285 286 287 288 289 290
  * i                   (i) int32 1kB 0 1 2 3 4 5 6 ... 354 355 356 357 358 359
    type                |S64 64B ...
    latitude            (j, i) float64 838kB ...
    longitude           (j, i) float64 838kB ...
Dimensions without coordinates: bnds, vertices
Data variables:
    time_bnds           (time, bnds) object 192B ...
    vertices_latitude   (j, i, vertices) float64 3MB ...
    vertices_longitude  (j, i, vertices) float64 3MB ...
    siconc              (time, j, i) float32 5MB ...
    areacello           (j, i) float32 419kB ...
Attributes: (12/56)
    CCCma_model_hash:               fc4bb7db954c862d023b546e19aec6c588bc0552
    CCCma_parent_runid:             p2-his13
    CCCma_pycmor_hash:              26c970628162d607fffd14254956ebc6dd3b6f49
    CCCma_runid:                    p2-s4513
    Conventions:                    CF-1.7 CMIP-6.2
    YMDH_branch_time_in_child:      2015:01:01:00
    ...                             ...
    license:                        CMIP6 model data produced by The Governm...
    cmor_version:                   3.5.0
    tracking_id:                    hdl:21.14100/9e4f804b-c161-44fa-acd1-c2e...
    DODS.strlen:                    64
```

(continues on next page)

(continued from previous page)

```
    DODS.dimName:                       maxStrlen64
    DODS_EXTRA.Unlimited_Dimension:  time
```

```
# Let's look at the grid shape itself and the data for one time step
fig, axs = plt.subplots(ncols=2, figsize=(12, 4))

axs[0].scatter(x=ds_in.longitude.values, y=ds_in.latitude.values, s=0.1)
axs[0].set_title(
    "The input horizontal grid points as seen on a lat/lon map.\nOnly the northern␣
↪hemisphere is shown."
)
axs[0].set_ylim(0, 90)
axs[0].set_ylabel(f"latitude [{ds_in.latitude.units}]")
axs[0].set_xlabel(f"longitude [{ds_in.longitude.units}]")
ds_in.siconc.isel(time=0).plot(ax=axs[1], cmap=cmap)
axs[1].set_title("Sea Ice concentration on Jan 2020, original grid")
fig.tight_layout()
```



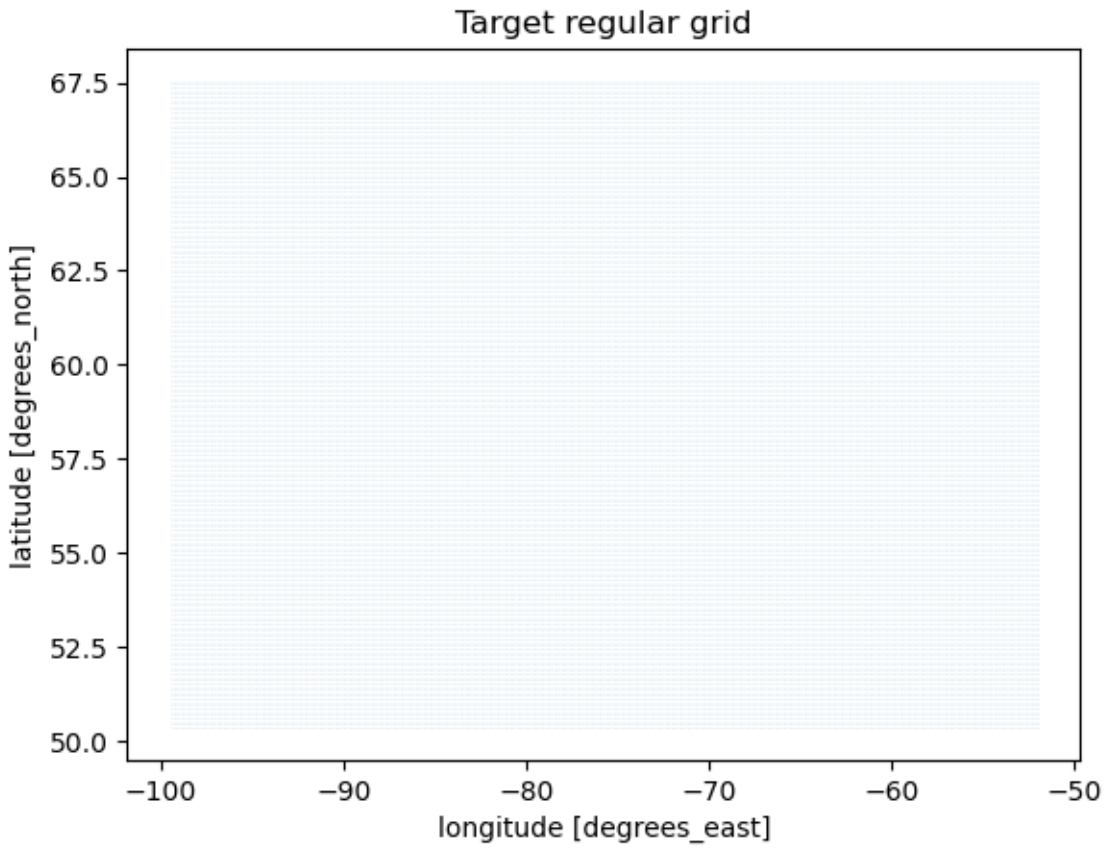
The output grid

The NRCAN observations' dataset uses a simple rectangular lat/lon grid over Canada at about 10km resolution. To reduce computation time for this example, we'll first crop the grid to include only Hudson Bay and the Labrador Sea.

```
# NBVAL_IGNORE_OUTPUT

url_obs = "https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/datasets/gridded_
↪obs/nrcan_v2.ncml"

# For this example, we're not interested in the observation data, only its underlying␣
↪grid, so we'll select a single time step.
ds_obs = xr.open_dataset(url_obs).sel(time="1993-05-20").drop("time")

# Subset over the Hudson Bay and the Labrador Sea for the example
bbox = dict(lon_bnds=[-99.5, -41.92], lat_bnds=[50.35, 67.61])
ds_tgt = subset_bbox(ds_obs, **bbox)
ds_tgt
```

```
<xarray.Dataset> Size: 1MB
Dimensions:  (lat: 207, lon: 570)
Coordinates:
  * lat      (lat) float32 828B 67.54 67.46 67.38 67.29 ... 50.54 50.46 50.38
  * lon      (lon) float32 2kB -99.46 -99.38 -99.29 ... -52.21 -52.13 -52.04
Data variables:
    tasmin   (lat, lon) float32 472kB ...
    tasmax   (lat, lon) float32 472kB ...
    pr       (lat, lon) float32 472kB ...
Attributes: (12/15)
    Conventions:          CF-1.5
    title:                NRCAN ANUSPLIN daily gridded dataset : version 2
    history:              Fri Jan 25 14:11:15 2019 : Convert from original fo...
    institute_id:         NRCAN
    frequency:            day
    abstract:             Gridded daily observational dataset produced by Nat...
    ...                   ...
    dataset_id:           NRCAN_anusplin_daily_v2
    version:              2.0
    license_type:         permissive
    license:              https://open.canada.ca/en/open-government-licence-c...
    attribution:          The authors provide this data under the Environment...
    citation:             Natural Resources Canada ANUSPLIN interpolated hist...
```

```
ds_tgt.cf.plot.scatter(x="longitude", y="latitude", s=0.1)
plt.title("Target regular grid");
```



xESMF relies on the useful cf_xarray package to infer which variables are the latitude and longitude points. It will automatically know to use `longitude` and `latitude` on the datasets because their attributes are correctly set, as `ds.cf.describe()` shows:

```
# NBVAL_IGNORE_OUTPUT

ds_in.cf.describe()
```

```
Coordinates:
             CF Axes: * X: ['i']
                      * Y: ['j']
                      * T: ['time']
                        Z: n/a

     CF Coordinates:   longitude: ['longitude']
                       latitude: ['latitude']
                      * time: ['time']
                        vertical: n/a

      Cell Measures:   area, volume: n/a

     Standard Names:   area_type: ['type']
                       latitude: ['latitude']
                       longitude: ['longitude']
                      * time: ['time']
```

(continues on next page)

(continued from previous page)

```
              Bounds:   n/a

       Grid Mappings:   n/a

Data Variables:
       Cell Measures:   area: ['areacello']
                        volume: n/a

      Standard Names:   cell_area: ['areacello']
                        sea_ice_area_fraction: ['siconc']

              Bounds:   T: ['time_bnds']
                        latitude: ['vertices_latitude']
                        longitude: ['vertices_longitude']
                        time: ['time_bnds']

       Grid Mappings:   n/a
```

If those attributes were **not** set, we would need to rename the coordinates to `lon` and `lat`, xESMF's default's coordinate names.

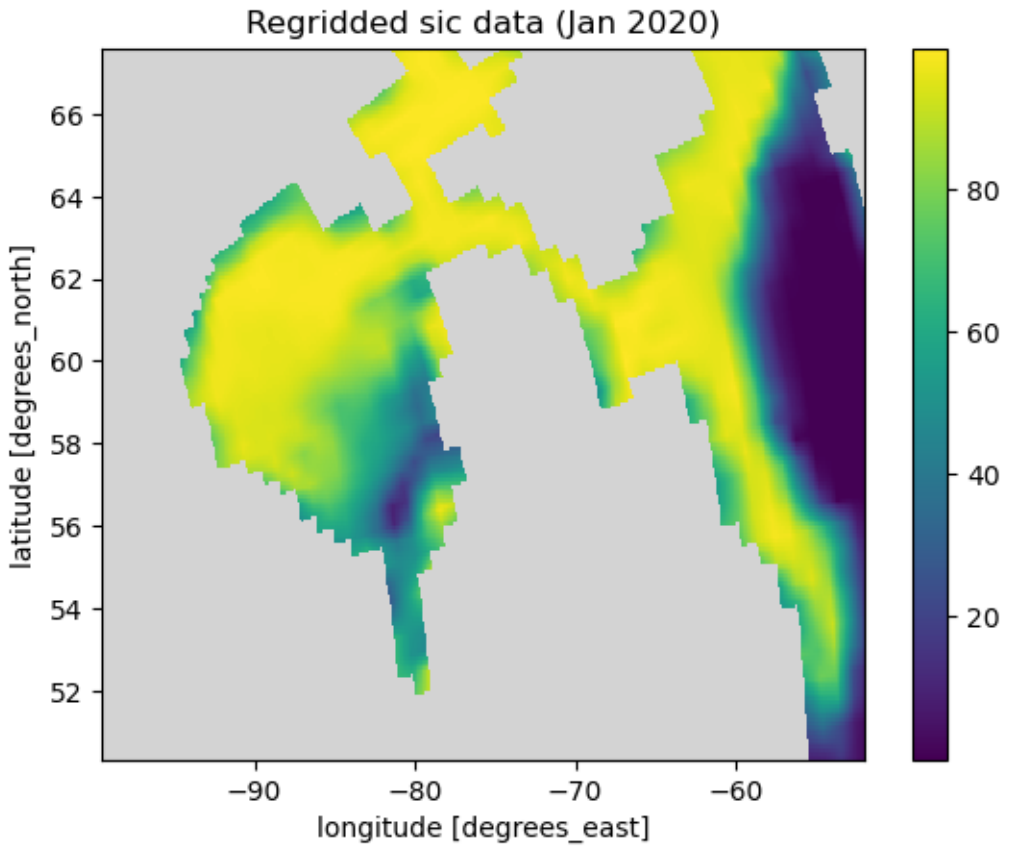
Regridding input data unto the output grid

First we create the regridding object, using the "bilinear" method, and then simply call it with the array that we want regridded (here `siconc`).

```
reg_bil = xe.Regridder(ds_in, ds_tgt, "bilinear")
reg_bil  # Show information about the regridding
```

```
xESMF Regridder
Regridding algorithm:       bilinear
Weight filename:            bilinear_291x360_207x570.nc
Reuse pre-computed weights? False
Input grid shape:           (291, 360)
Output grid shape:          (207, 570)
Periodic in longitude?      False
```

```
# xesmf/frontend.py:476: FutureWarning: ``output_sizes`` should be given in the ``dask_
↪gufunc_kwargs`` parameter. It will be removed as direct parameter in a future version.
warnings.filterwarnings("ignore", category=FutureWarning)

# Apply the regridding weights to the input sea ice concentration data
sic_bil = reg_bil(ds_in.siconc)

# Plot the results
sic_bil.isel(time=0).plot(cmap=cmap)
plt.title("Regridded sic data (Jan 2020)");
```



The output now has the same grid as the target! The regridding operation was broadcasted along the non spatial dimensions (here `time`), so that all time steps were regridded using the same pre-computed weights.

Second example : Conservative regridding and reusing weights

xESMF provides the following regridding methods : "bilinear", "conservative", "conservative_normed", "nearest_s2d", "nearest_d2s" and "patch" (see method descriptions). Conservative methods preserve areal averages, and for these methods we need to provide the coordinates of the grid cells' corners rather than the coordinates at the cells center.

Untangling corners definitions

Before we go further, it's worth highlighting differences between xESMF's description of corner coordinates and how the same information is stored in CF-compliant files.

For an `N x M` lon/lat grid, xESMF expects an array with one element more than the coordinates. For example, on a regular grid, the corner of point at `lon[0]` are given by `lon_b[0]` and `lon_b[1]`. However, in a typical CF-compliant file, *grid corner* information is in an array of shape (N, 2) typically called `lon_bounds` and `lat_bounds`. Thus, the western and eastern corners of point at `lon[0]` are given by `lon_corners[0, 0]` and `lon_corners[0, 1]`.

The `cf_xarray` package differentiates the two concepts by naming the CF-compliant one "bounds" and the xESMF one "vertices". However, CF conventions sometime uses vertices and bound interchangeably, and in our model dataset, the `vertices_longitude` variable stores corners according to the "bounds" definition… We will nevetheless stick with `cf_xarray`'s nomenclature in the following.

The table below summarizes the difference between the two versions:

	bounds	vertices
CF-compliant	Yes	No
Shape (regular grid)	(N, 2)	(N+1,)
Shape (irregular grid)	(Nx, Ny, 4)	(Nx+1, Ny+1)

Computing the corners

The corners of regular grids (1D lat/lon) are infered automatically if not given. This will be the case for our `ds_tgt` dataset.

For irregular grids, `xESMF` will check for variables `lon_b` and `lat_b`, or try automatic detection with the help of `cf_xarray`. If they are found, it uses `cf_xarray`'s method to convert from the CF-compliant "bounds" to the required "vertices" syntax. However, a small bug in xESMF 0.5.2 prevents use from using this feature with our model dataset. We will convert the corner variables ourselves from the CF-compliant format we have to the format xESMF expects.

```
# Get the bounds variable and convert them to "vertices" format
# Order=none, means that we do not know if the bounds are listed clockwise or␣
↪counterclockwise, so we ask cf_xarray to try both.
lat_corners = cfxr.bounds_to_vertices(ds_in.vertices_latitude, "vertices", order=None)
lon_corners = cfxr.bounds_to_vertices(ds_in.vertices_longitude, "vertices", order=None)
ds_in_crns = ds_in.assign(lon_b=lon_corners, lat_b=lat_corners)
```

Regridding

The regridding process is as simple as above now that `ds_in_crns` contains the corner coordinates (`lon_b`, `lat_b`). Here we also pass a filename, so that the weights are saved to disk and can be reused (see below).

```
%%time
reg_cons = xe.Regridder(
    ds_in_crns, ds_tgt, "conservative", filename="conservative_regridder.nc"
)
print(reg_cons)

# Regrid as before
sic_cons = reg_cons(ds_in_crns.siconc)
```

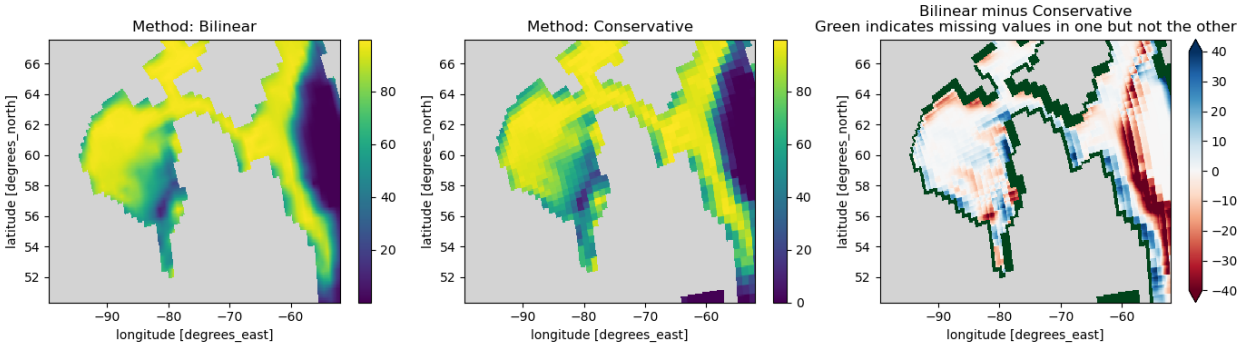
```
xESMF Regridder
Regridding algorithm:       conservative
Weight filename:            conservative_regridder.nc
Reuse pre-computed weights? False
Input grid shape:           (291, 360)
Output grid shape:          (207, 570)
Periodic in longitude?      False
CPU times: user 3.74 s, sys: 40.2 ms, total: 3.78 s
Wall time: 3.79 s
```

```
# Now let's look at the results
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(14, 4))
```

(continues on next page)

(continued from previous page)

```
sic_bil.isel(time=0).plot(ax=axs[0], cmap=cmap)
axs[0].set_title("Method: Bilinear")

sic_cons.isel(time=0).plot(ax=axs[1], cmap=cmap)
axs[1].set_title("Method: Conservative")

# A divergent colormap with gray on missing values
cmap_div = copy.copy(plt.cm.get_cmap("RdBu"))
cmap_div.set_bad("lightgray")
(sic_bil - sic_cons).isel(time=0).plot(ax=axs[2], cmap=cmap_div, vmin=-40, vmax=40)
diff_NaNs = (sic_bil.isnull() ^ sic_cons.isnull()).isel(time=0)
diff_NaNs.where(diff_NaNs).plot(
    cmap=plt.cm.Greens, ax=axs[2], vmin=0, add_colorbar=False
)
axs[2].set_title(
    "Bilinear minus Conservative\nGreen indicates missing values in one but not the other
↪"
)
fig.tight_layout()
```



As we can see, "bilinear" regridding results in a smooth output field, while "conservative" results preserves the original data's coarser resolution. In the last panel, the green cells show that the two methods have different missing values results. In our case of increasing resolution, there will often be more missing values when using "bilinear". The next example explains how xESMF can explictely manage missing values. But before, we look at the reusability of the weights generated by xESMF.

Reusing weights

The weights of the previous regridding have been written to disk. We can simply reuse them by specifying that filename and passing `reuse_weights=True`. You'll notice how faster the process is, as we don't compute the weights again.

```
%%time
reg_bis = xe.Regridder(
    ds_in_crns,
    ds_tgt,
    "conservative",
    reuse_weights=True,
    filename="conservative_regridder.nc",
)
```

(continues on next page)

(continued from previous page)

```
print(reg_bis)

# Regrid as before
sic_bis = reg_bis(ds_in_crns.siconc)
```

```
xESMF Regridder
Regridding algorithm:       conservative
Weight filename:            conservative_regridder.nc
Reuse pre-computed weights? True
Input grid shape:           (291, 360)
Output grid shape:          (207, 570)
Periodic in longitude?      False
CPU times: user 127 ms, sys: 23.9 ms, total: 151 ms
Wall time: 150 ms
```

Third example : Regridding and masks

By defaut, xESMF doesn't handle missing values in a special way, so when they are present in the input data they often bleed into the regridded field, especially when decreasing resolution. This example demonstrates this bleeding effect and how it can be mitigated using masks.

We will use a global model dataset and try to regrid the NRCAN observation unto the global grid, thus decreasing the resolution.

Target grid and mask

The target grid will be the CanESM2 model grid, but with the ocean masked. In the following, we fetch both the "tasmin" data for the same date as the obs and the "sftlf" mask so we can obtain a land mask (land fraction above 0.25).

```
# NBVAL_IGNORE_OUTPUT

# Model data for tasmin
ds_tgt = xr.open_dataset(
    "https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/cccma/CanESM2/
↪historical/day/atmos/r1i1p1/tasmin/tasmin_day_CanESM2_historical_r1i1p1_18500101-
↪20051231.nc"
)
# Land-sea fraction
ds_sftlf = xr.open_dataset(
    "https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/cccma/CanESM2/
↪historical/fx/atmos/r0i0p0/sftlf/sftlf_fx_CanESM2_historical_r0i0p0.nc"
)
ds_tgt = ds_tgt.sel(time="1993-05-20").drop("time")  # Extract same day as obs
ds_tgt = ds_tgt.rename(bnds="bounds")  # Small fix for xESMF 0.5.2
ds_tgt["tasmin"] = ds_tgt.tasmin.where(
    ds_sftlf.sftlf > 0.25
)  # Mask tasmin data that is over the ocean
ds_tgt
```

```
<xarray.Dataset> Size: 37kB
Dimensions:    (time: 1, bounds: 2, lat: 64, lon: 128)
Coordinates:
  * lat        (lat) float64 512B -87.86 -85.1 -82.31 ... 82.31 85.1 87.86
  * lon        (lon) float64 1kB 0.0 2.812 5.625 8.438 ... 351.6 354.4 357.2
    height     float64 8B ...
Dimensions without coordinates: time, bounds
Data variables:
    time_bnds  (time, bounds) object 16B ...
    lat_bnds   (lat, bounds) float64 1kB ...
    lon_bnds   (lon, bounds) float64 2kB ...
    tasmin     (time, lat, lon) float32 33kB 208.3 207.9 207.4 ... nan nan nan
Attributes: (12/32)
    institution:                    CCCma (Canadian Centre for Climate Model...
    institute_id:                   CCCma
    experiment_id:                  historical
    source:                         CanESM2 2010 atmosphere: CanAM4 (AGCM15i...
    model_id:                       CanESM2
    forcing:                        GHG,Oz,SA,BC,OC,LU,Sl,Vl (GHG includes C...
    ...                             ...
    title:                          CanESM2 model output prepared for CMIP5 ...
    parent_experiment:              pre-industrial control
    modeling_realm:                 atmos
    realization:                    1
    cmor_version:                   2.5.4
    DODS_EXTRA.Unlimited_Dimension:  time
```

```
# NBVAL_IGNORE_OUTPUT

# Input grid and data : reuse ds_obs (NRCAN but without the subsetting)
ds_in = ds_obs[["tasmin"]]
ds_in
```

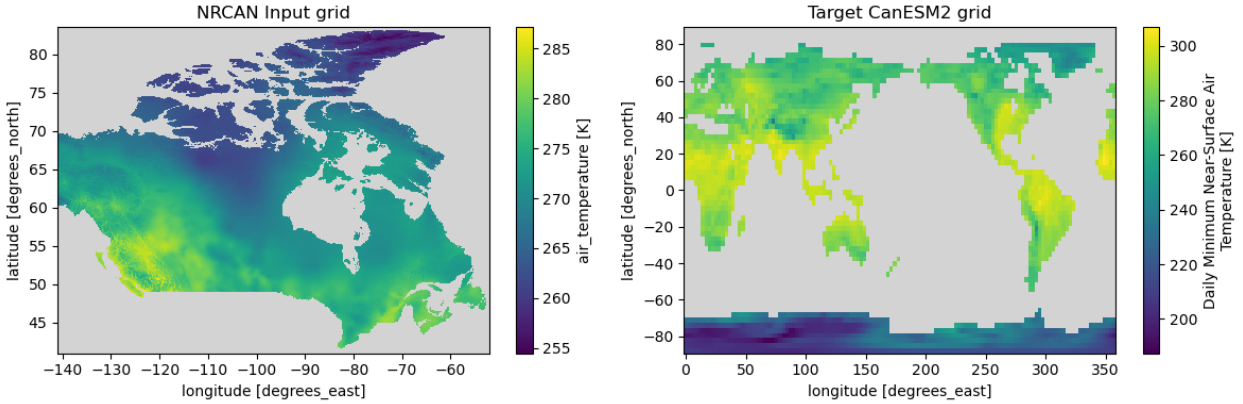
```
<xarray.Dataset> Size: 2MB
Dimensions:  (lat: 510, lon: 1068)
Coordinates:
  * lat      (lat) float32 2kB 83.46 83.38 83.29 83.21 ... 41.21 41.12 41.04
  * lon      (lon) float32 4kB -141.0 -140.9 -140.8 ... -52.21 -52.13 -52.04
Data variables:
    tasmin   (lat, lon) float32 2MB ...
Attributes: (12/15)
    Conventions:          CF-1.5
    title:                NRCAN ANUSPLIN daily gridded dataset : version 2
    history:              Fri Jan 25 14:11:15 2019 : Convert from original fo...
    institute_id:         NRCAN
    frequency:            day
    abstract:             Gridded daily observational dataset produced by Nat...
    ...                   ...
    dataset_id:           NRCAN_anusplin_daily_v2
    version:              2.0
    license_type:         permissive
    license:              https://open.canada.ca/en/open-government-licence-c...
```

(continues on next page)

(continued from previous page)

```
    attribution:          The authors provide this data under the Environment...
    citation:             Natural Resources Canada ANUSPLIN interpolated hist...
```

```
fig, axs = plt.subplots(ncols=2, figsize=(12, 4))

ds_in.tasmin.plot(ax=axs[0], cmap=cmap)
axs[0].set_title("NRCAN Input grid")

ds_tgt.tasmin.plot(ax=axs[1], cmap=cmap)
axs[1].set_title("Target CanESM2 grid")
fig.tight_layout()
```



Default regridding - No mask handling

We first naïvely try the regridding exactly as before. Here we use the "conservative_normed" method, the reason is explained at the end of the example.

```
reg_nomask = xe.Regridder(ds_in, ds_tgt, "conservative_normed")
print(reg_nomask)
tasmin_nomask = reg_nomask(ds_in.tasmin)
```

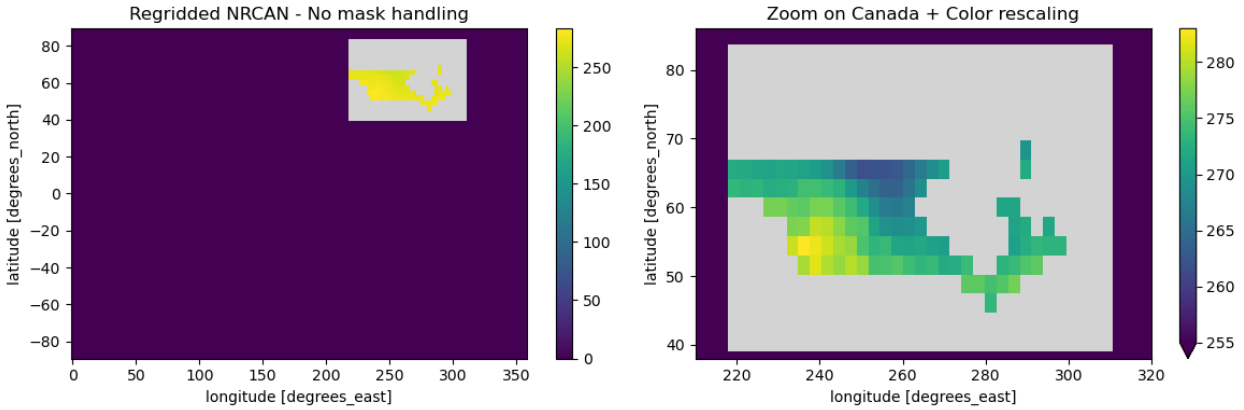
```
xESMF Regridder
Regridding algorithm:       conservative_normed
Weight filename:            conservative_normed_510x1068_64x128.nc
Reuse pre-computed weights? False
Input grid shape:           (510, 1068)
Output grid shape:          (64, 128)
Periodic in longitude?      False
```

```
fig, axs = plt.subplots(ncols=2, figsize=(12, 4))

tasmin_nomask.plot(ax=axs[0], cmap=cmap)
axs[0].set_title("Regridded NRCAN - No mask handling")

tasmin_nomask.plot(ax=axs[1], cmap=cmap, vmin=255)
axs[1].set_xlim(210, 320)
```

(continues on next page)

(continued from previous page)

```
axs[1].set_ylim(38, 86)
axs[1].set_title("Zoom on Canada + Color rescaling")
fig.tight_layout()
```



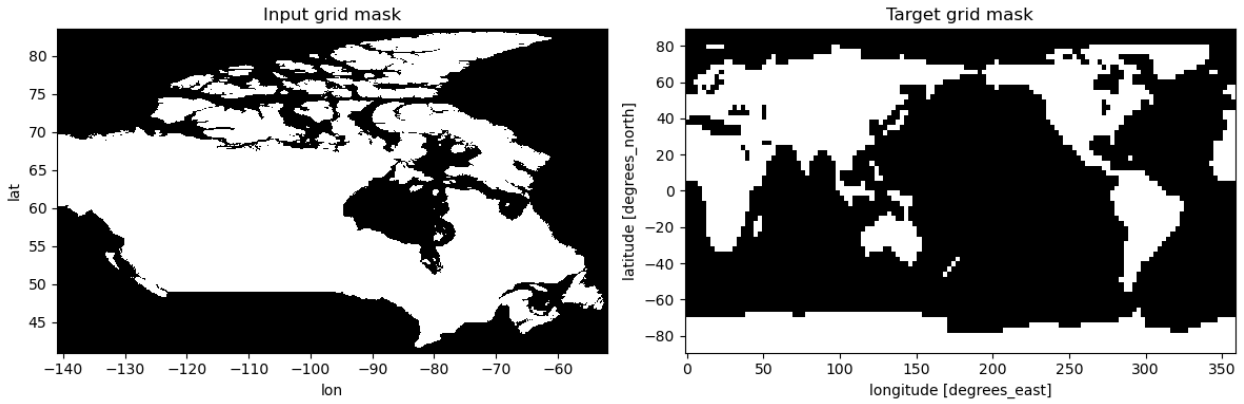
This ugly result is the default behaviour of xESMF when no mask information is passed :

1. A single missing value in the input suffices so that the target (coarser) grid cell is marked as missing. This erased all the Canadian Arctic Archipelago and most points near the sea in general.
2. Grid points outside the input grid are filled with 0s instead of NaNs.

To resolve this, we pass as binary mask to xESMF. xESMF will then exclude the masked values from the computation, and this way the small islands in the Canadian Arctic Archipelago won't be hidden by missing values. It will also activate a mode where values outside the input grid are marked as missing (NaN), which is usually more useful.

Note that ESMF masks defined as `True` where data is valid, and `False` where it is missing. The variable must be named `mask` to get picked up by xESMF.

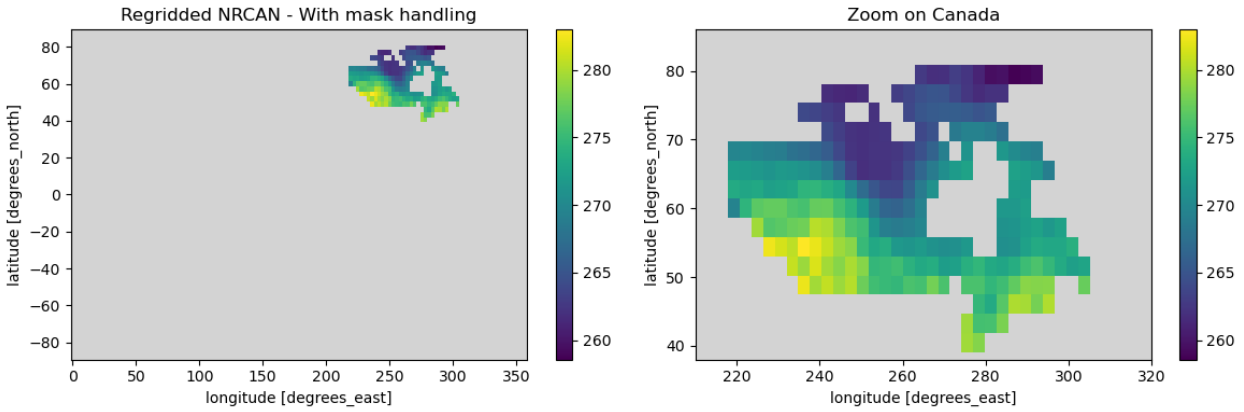
```
# Define the masks and assign them as variables for both the input and output datasets.
in_mask = ds_in.tasmin.notnull()
ds_in_mask = ds_in.assign(mask=in_mask)

tgt_mask = ds_tgt.tasmin.isel(time=0).notnull()
ds_tgt_mask = ds_tgt.assign(mask=tgt_mask)

fig, axs = plt.subplots(ncols=2, figsize=(12, 4))
in_mask.plot(ax=axs[0], cmap=plt.cm.binary_r, add_colorbar=False)
tgt_mask.plot(ax=axs[1], cmap=plt.cm.binary_r, add_colorbar=False)
axs[0].set_title("Input grid mask")
axs[1].set_title("Target grid mask")
fig.tight_layout()
```




```
reg_mask = xe.Regridder(ds_in_mask, ds_tgt_mask, "conservative_normed")
reg_mask  # Show information about the regriddingaa
```

```
xESMF Regridder
Regridding algorithm:       conservative_normed
Weight filename:            conservative_normed_510x1068_64x128.nc
Reuse pre-computed weights? False
Input grid shape:           (510, 1068)
Output grid shape:          (64, 128)
Periodic in longitude?      False
```

```
tasmin_mask = reg_mask(ds_in_mask.tasmin)
fig, axs = plt.subplots(ncols=2, figsize=(12, 4))

tasmin_mask.plot(ax=axs[0], cmap=cmap)
axs[0].set_title("Regridded NRCAN - With mask handling")

tasmin_mask.plot(ax=axs[1], cmap=cmap)
axs[1].set_xlim(210, 320)
axs[1].set_ylim(38, 86)
axs[1].set_title("Zoom on Canada")
fig.tight_layout()
```

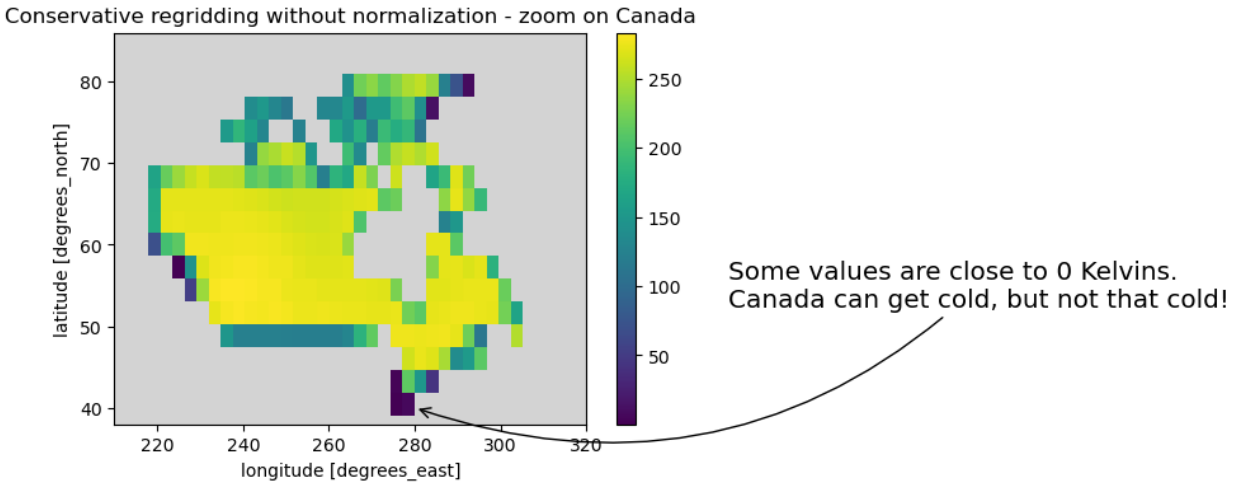


Much better! As expected, grid cells near the sea are kept and points outside the input grid are marked as missing.

Normalization for conservative regridding

The "conservative_normed" method includes information about the missing values in the final normalization of the data. On the other hand, the "conservative" method normalizes using the total area of the target cell, no matter how many input grid points were valid. The following figure shows how that can lead to large biases in the data near the boundaries. Indeed, in the example below, the temperatures reach values close to 0 Kelvins near the boundaries.

```
reg_mask_cons = xe.Regridder(ds_in_mask, ds_tgt_mask, "conservative")
tasmin_mask_cons = reg_mask_cons(ds_in_mask.tasmin)

fig, ax = plt.subplots(figsize=(6, 4))
tasmin_mask_cons.plot(cmap=cmap, ax=ax)
ax.set_xlim(210, 320)
ax.set_ylim(38, 86)
ax.set_title("Conservative regridding without normalization - zoom on Canada")
ax.annotate(
    "Some values are close to 0 Kelvins.\nCanada can get cold, but not that cold!",
    (280, 40),
    xytext=(1.3, 0.3),
    xycoords="data",
    textcoords="axes fraction",
    fontsize="x-large",
    arrowprops=dict(arrowstyle="->", connectionstyle="arc3, rad=-0.3"),
);
```



Fourth example : Averaging over polygons

Because the conservative regridding method preserves areal averages, we can use xESMF to compute *exact* averages over polygons. We call it "exact" because is takes into account partial overlaps between the gridcells and the shapes, including potential holes. While it is fast and powerful, this polygon averaging functionality is new in xESMF and still lacks some features, like missing values handling and performance issues with high-resolution polygons.

The following example grabs some polygon shapes from PAVICS' Geoserver and averages the NRCAN data over them.

Define polygon shapes

This example fetches all MRC of Québec and then only selects 10 large ones.

```
wfs_url = "https://pavics.ouranos.ca/geoserver/wfs"  # TEST_USE_PROD_DATA
# # Connect to GeoServer WFS service.
wfs = WebFeatureService(wfs_url, version="2.0.0")
# Get the json as a binary stream
# Here we select Quebec's MRCs polygons
# We select only a few properties
data = wfs.getfeature(
    typename="public:quebec_mrc_boundaries",
    # bbox=(-93.1, 41.1, -75.0, 49.6),
    outputFormat="json",
    propertyname=["the_geom", "MRS_NM_MRC"],
)

# Load into a GeoDataFrame by reading the json on-the-fly
shapes_all = gpd.GeoDataFrame.from_features(json.load(data))
# Just for simplicity, let's take 10 large MRCs
shapes_all["AREA"] = shapes_all.area
shapes = shapes_all.sort_values("AREA").iloc[-20:-10].set_index("MRS_NM_MRC")
```

Validate and simplify shapes

High resolution polygons might slow down the creation of the xESMf averager object. Here we ensure polygons are simplified to a resolution 50x times finer than the input data. This should have a minimal impact on the output while still improving performance.

As it is the case here, downloaded polygons sometime have topological problems which can be tested with `shapes.is_valid`. Simplifying polygons sometimes help overcome these issues: here, we simplify with a tolerance of 1/100th of the grid size. Another workaround for self-intersections is to call `shapes.buffer(0)`.

```
# NBVAL_IGNORE_OUTPUT
shapes.is_valid.all()
```

```
False
```

```
# This is only to show the decrease in size


def count_points(elem):
    def _count(poly):
        return len(poly.exterior.coords) + sum(
            len(hole.coords) for hole in poly.interiors
        )

    if isinstance(elem, shapely.geometry.MultiPolygon):
        return sum(_count(poly) for poly in elem.geoms)
    return _count(elem)
```

(continues on next page)

(continued from previous page)

```
# Count the total number of nodes in the shapes:
print(
    "Total number of nodes in the raw shapes : ",
    shapes.geometry.apply(count_points).sum(),
)

min_grid_size = float(
    min(abs(ds_in.lat.diff("lat")).min(), abs(ds_in.lon.diff("lon")).min())
)
print(
    f"Minimal grid size [°] of input ds: {min_grid_size:0.3f}, we will simplify to a␣
↪tolerance of {min_grid_size / 100:0.5f}"
)

# Simplify geometries
shapes_simp = shapes.copy()
shapes_simp["geometry"] = shapes.simplify(min_grid_size / 100).buffer(0)

print(
    "Total number of nodes in the simplified shapes : ",
    shapes_simp.geometry.apply(count_points).sum(),
)
assert shapes_simp.buffer(0).is_valid.all()
```

```
Total number of nodes in the raw shapes :  166813
Minimal grid size [°] of input ds: 0.083, we will simplify to a tolerance of 0.00083
```

```
Total number of nodes in the simplified shapes :  7232
```

Averaging over each polygon

Performing the spatial average is as simple as regridding. We first construct a `SpatialAverager` object from the input grid and polygons, then call it with the data to average. Note that xESMf expects a list of shapes, so we pass the `shapes.geometry` series (and not the `GeoDataFrame` itself).

The returned DataArray was averaged along its spatial (lat/lon) dimensions and the average over the different shapes are along the new `geom` dimension, which is in the same order as the initial GeoDataframe.

The current missing value handling in xESMF's SpatialAverager is very strict and we can see here how the 3 MRCs that overlap with ocean cells of the data (where tasmin is NaN) are flagged as missing (NaN).

```
# NBVAL_IGNORE_OUTPUT
savg = xe.SpatialAverager(ds_in, shapes_simp.geometry)
tn_avg = savg(ds_in.tasmin)
tn_avg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/
↪site-packages/xesmf/frontend.py:1220: UserWarning: `polys` contains large (> 1°)␣
↪segments. This could lead to errors over large regions. For a more accurate average,␣
↪segmentize (densify) your shapes with  `shapely.segmentize(polys, 1)`
  self._check_polys_length(polys)
```

```
<xarray.DataArray (geom: 10)> Size: 40B
array([      nan,       nan, 274.64337, 275.4841 ,       nan, 276.23572,
       274.05746, 276.86087, 275.4918 , 276.8586 ], dtype=float32)
Coordinates:
    lon      (geom) float64 80B -66.01 -63.91 -77.09 ... -73.26 -76.76 -73.98
    lat      (geom) float64 80B 49.21 48.29 46.43 46.91 ... 48.81 48.15 48.04
Dimensions without coordinates: geom
Attributes:
    regrid_method:  conservative
```

Merging polygon features' properties into the result

In the previous results, the polygons are indexed along the `geom` dimension, but we'd like to have the region names and properties.

```
# NBVAL_IGNORE_OUTPUT

# Set coordinates of "geom" to the shapes index
tn_avg["geom"] = shapes_simp.index.values

# Get a Dataset of properties from the dataframe
# Drop the geometries (we don't want them), convert to xarray and rename the index so it␣
↪matches the one in tn_avg
props = shapes_simp.drop(columns=["geometry"]).to_xarray().rename(MRS_NM_MRC="geom")

# Assign all properties as "auxiliary" coordinates
tn_avg = tn_avg.assign_coords(**props.data_vars)
tn_avg
```

```
<xarray.DataArray (geom: 10)> Size: 40B
array([      nan,       nan, 274.64337, 275.4841 ,       nan, 276.23572,
       274.05746, 276.86087, 275.4918 , 276.8586 ], dtype=float32)
Coordinates:
    lon      (geom) float64 80B -66.01 -63.91 -77.09 ... -73.26 -76.76 -73.98
    lat      (geom) float64 80B 49.21 48.29 46.43 46.91 ... 48.81 48.15 48.04
  * geom     (geom) object 80B 'La Haute-Gaspésie' ... 'La Tuque'
    AREA     (geom) float64 80B 1.415 1.543 1.654 1.673 ... 2.364 3.307 3.573
Attributes:
    regrid_method:  conservative
```

Or, on the contrary, we could want to merge the averaged data to the dataframe instead.

```
# NBVAL_IGNORE_OUTPUT
shapes_data = shapes_simp.copy()
shapes_data["tasmin"] = tn_avg.to_series()
shapes_data
```

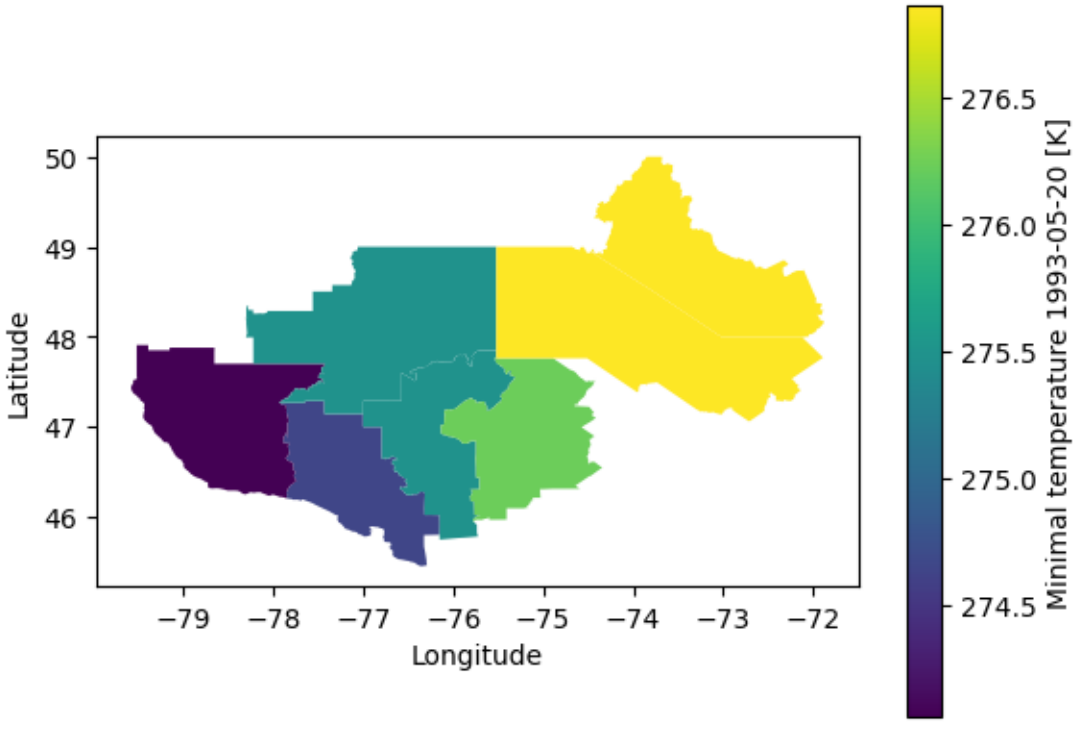
```
                                                        geometry  \
MRS_NM_MRC
La Haute-Gaspésie         POLYGON ((-65.19560 49.22960, -65.18720 49.099...
Le Rocher-Percé           POLYGON ((-62.99910 48.62500, -62.99910 47.157...
```

(continues on next page)

(continued from previous page)

```
Pontiac                    POLYGON ((-77.93130 47.26920, -77.64730 47.269...
La Vallée-de-la-Gatineau   POLYGON ((-76.27030 47.68990, -76.27000 47.692...
La Haute-Côte-Nord         POLYGON ((-69.50820 49.99830, -69.51050 49.997...
Antoine-Labelle            POLYGON ((-75.53860 47.76330, -74.88940 47.762...
Témiscamingue              MULTIPOLYGON (((-77.57920 47.44240, -77.58630 ...
Le Domaine-du-Roy          POLYGON ((-73.68260 49.99730, -73.68440 49.997...
La Vallée-de-l'Or          MULTIPOLYGON (((-77.57920 47.44240, -77.57290 ...
La Tuque                   POLYGON ((-74.67630 48.99960, -74.62850 48.967...

                               AREA      tasmin
MRS_NM_MRC
La Haute-Gaspésie          1.414604         NaN
Le Rocher-Percé            1.542547         NaN
Pontiac                    1.653789  274.643372
La Vallée-de-la-Gatineau   1.672640  275.484100
La Haute-Côte-Nord         1.800530         NaN
Antoine-Labelle            1.923892  276.235718
Témiscamingue              2.282582  274.057465
Le Domaine-du-Roy          2.363714  276.860870
La Vallée-de-l'Or          3.306656  275.491791
La Tuque                   3.572838  276.858612
```

```
# Now we can plot easily the results as a choropleth map!
ax = shapes_data.plot(
    "tasmin", legend=True, legend_kwds={"label": "Minimal temperature 1993-05-20 [K]"}
)
ax.set_ylabel("Latitude")
ax.set_xlabel("Longitude");
```



1.2.17 General workflow demonstration

Setup and configuration

```
# # If you have JupyterLab, you will need to install the JupyterLab extension:
# # !jupyter labextension install @jupyter-widgets/jupyterlab-manager jupyter-leaflet

# On a normal notebook server, the extension needs only to be enabled
# !jupyter nbextension enable --py --sys-prefix ipyleaflet
```

```
import os

os.environ["USE_PYGEOS"] = "0"  # force use Shapely with GeoPandas

import ipyleaflet
from birdy import WPSClient
from siphon.catalog import TDSCatalog

# we will use these coordinates later to center the map on Canada
canada_center_lat_lon = (52.4292, -93.2959)
```

PAVICS URL configuration

```
# Ouranos production server
pavics_url = "https://pavics.ouranos.ca"

finch_url = os.getenv("WPS_URL")
if not finch_url:
    finch_url = f"{pavics_url}/twitcher/ows/proxy/finch/wps"
```

Getting user input from a map widget

Notes about ipyleaflet

ipyleaflet is a **"A Jupyter / Leaflet bridge enabling interactive maps in the Jupyter notebook"**

This means that the interactions with graphical objects on the map and in python are synchronized. The documentation is at: https://ipyleaflet.readthedocs.io

```
leaflet_map = ipyleaflet.Map(
    center=canada_center_lat_lon,
    basemap=ipyleaflet.basemaps.Stamen.Terrain,
    zoom=4,
)

initial_marker_location = canada_center_lat_lon
marker = ipyleaflet.Marker(location=initial_marker_location, draggable=True)
leaflet_map.add_layer(marker)

leaflet_map
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/xyzservices/lib.py:44, in Bunch.__getattr__(self, key)
     43 try:
---> 44     return self.__getitem__(key)
     45 except KeyError as err:

KeyError: 'Stamen'

The above exception was the direct cause of the following exception:

AttributeError                            Traceback (most recent call last)
Cell In[4], line 3
      1 leaflet_map = ipyleaflet.Map(
      2     center=canada_center_lat_lon,
----> 3     basemap=ipyleaflet.basemaps.Stamen.Terrain,
      4     zoom=4,
      5 )
      7 initial_marker_location = canada_center_lat_lon
      8 marker = ipyleaflet.Marker(location=initial_marker_location, draggable=True)

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/xyzservices/lib.py:46, in Bunch.__getattr__(self, key)
     44     return self.__getitem__(key)
     45 except KeyError as err:
---> 46     raise AttributeError(key) from err

AttributeError: Stamen
```

If you move the marker on the map, it will update the `marker.location` variable. Also, if you update `marker.location` manually (`marker.location = (45.44, -90.44)`) it will also move on the map.

```
marker.location = (45.55, -72.44)
```

```
print(initial_marker_location)
print(marker.location)
```

```
(52.4292, -93.2959)
[45.55, -72.44]
```

Helper function to get user input

```
import ipyleaflet
import ipywidgets as widgets
from IPython.display import display


def get_rectangle():
    canada_center = (52.4292, -93.2959)
    m = ipyleaflet.Map(
```

(continues on next page)

(continued from previous page)

```
        center=canada_center,
        basemap=ipyleaflet.basemaps.Stamen.Terrain,
        zoom=4,
    )

    # Create a new draw control
    draw_control = ipyleaflet.DrawControl()

    # disable some drawing inputs
    draw_control.polyline = {}
    draw_control.circlemarker = {}
    draw_control.polygon = {}

    draw_control.rectangle = {
        "shapeOptions": {
            "fillColor": "#4ae",
            "color": "#4ae",
            "fillOpacity": 0.3,
        }
    }

    output = widgets.Output(layout={"border": "1px solid black"})

    rectangle = {}

    # set drawing callback
    def callback(control, action, geo_json):
        if action == "created":
            # note: we can't close the map or remove it from the output
            # from this callback. The map keeps the focus, and the
            # jupyter keyboard input is messed up.
            # So we set it very thin to make it disappear :)
            m.layout = {"max_height": "0"}
            with output:
                print("*User selected 1 rectangle*")
                rectangle.update(geo_json)

    draw_control.on_draw(callback)

    m.add_control(draw_control)

    with output:
        print("Select a rectangle:")
        display(m)

    display(output)

    return rectangle
```

The user wants to select a rectangle on a map, and get a GeoJSON back

```
# NBVAL_IGNORE_OUTPUT


rectangle = get_rectangle()
```

```
Output(layout=Layout(border_bottom='1px solid black', border_left='1px solid black',␣
↪border_right='1px solid b...
```

```
#  GeoJSON with custom style properties
if not len(rectangle):
    # Use the default region of Greater Montreal Area
    rectangle = {
        "type": "Feature",
        "properties": {
            "style": {
                "stroke": True,
                "color": "#4ae",
                "weight": 4,
                "opacity": 0.5,
                "fill": True,
                "fillColor": "#4ae",
                "fillOpacity": 0.3,
                "clickable": True,
            }
        },
        "geometry": {
            "type": "Polygon",
            "coordinates": [
                [
                    [-74.511948, 45.202296],
                    [-74.511948, 45.934852],
                    [-72.978537, 45.934852],
                    [-72.978537, 45.202296],
                    [-74.511948, 45.202296],
                ]
            ],
        },
    }
rectangle
```

```
{'type': 'Feature',
 'properties': {'style': {'stroke': True,
   'color': '#4ae',
   'weight': 4,
   'opacity': 0.5,
   'fill': True,
   'fillColor': '#4ae',
   'fillOpacity': 0.3,
   'clickable': True}},
 'geometry': {'type': 'Polygon',
```

(continues on next page)

(continued from previous page)

```
  'coordinates': [[[-74.511948, 45.202296],
    [-74.511948, 45.934852],
    [-72.978537, 45.934852],
    [-72.978537, 45.202296],
    [-74.511948, 45.202296]]]}}
```

Get the maximum and minimum bounds

```
import geopandas as gpd

rect = gpd.GeoDataFrame.from_features([rectangle])
bounds = rect.bounds
bounds
```

```
        minx       miny       maxx       maxy
0 -74.511948  45.202296 -72.978537  45.934852
```

Calling wps processes

For this example, we will **subset** a dataset with the user-selected bounds, and launch a **heat wave frequency** analysis on it.

```
finch = WPSClient(finch_url, progress=False)
```

```
help(finch.subset_bbox)
```

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

subset_bbox(resource=None, lon0=0.0, lon1=360.0, lat0=-90.0, lat1=90.0, start_date=None,␣
↪end_date=None, variable=None) method of birdy.client.base.WPSClient instance
    Return the data for which grid cells intersect the bounding box for each input␣
↪dataset as well as the time range selected.

    Parameters
    ----------
    resource : ComplexData:mimetype:`application/x-netcdf`, :mimetype:`application/x-ogc-
↪dods`
        NetCDF files, can be OPEnDAP urls.
    lon0 : float
        Minimum longitude.
    lon1 : float
        Maximum longitude.
    lat0 : float
        Minimum latitude.
    lat1 : float
        Maximum latitude.
    start_date : string
        Initial date for temporal subsetting. Can be expressed as year (%Y), year-month (
```

(continues on next page)

(continued from previous page)

```
↪%Y-%m) or year-month-day(%Y-%m-%d). Defaults to first day in file.
    end_date : string
        Final date for temporal subsetting. Can be expressed as year (%Y), year-month (
↪%Y-%m) or year-month-day(%Y-%m-%d). Defaults to last day in file.
    variable : string
        Name of the variable in the NetCDF file.

    Returns
    -------
    output : ComplexData:mimetype:`application/x-netcdf`
        netCDF output
    ref : ComplexData:mimetype:`application/metalink+xml; version=4.0`
        Metalink file storing all references to output files.
```

Subset tasmin and tasmax datasets

```
# gather data from the PAVICS data catalogue
catalog = "https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/catalog/datasets/gridded_
↪obs/catalog.xml"  # TEST_USE_PROD_DATA

cat = TDSCatalog(catalog)
data = cat.datasets[0].access_urls["OPENDAP"]
data
```

```
'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/datasets/gridded_obs/nrcan_
↪v2.ncml'
```

```
# NBVAL_IGNORE_OUTPUT

# If we want a preview of the data, here's what it looks like:
import xarray as xr

ds = xr.open_dataset(data, chunks="auto")
ds
```

```
<xarray.Dataset>
Dimensions:  (lat: 510, lon: 1068, time: 24837)
Coordinates:
  * lat      (lat) float32 83.46 83.38 83.29 83.21 ... 41.29 41.21 41.12 41.04
  * lon      (lon) float32 -141.0 -140.9 -140.8 -140.7 ... -52.21 -52.13 -52.04
  * time     (time) datetime64[ns] 1950-01-01 1950-01-02 ... 2017-12-31
Data variables:
    tasmin   (time, lat, lon) float32 dask.array<chunksize=(3375, 69, 144), meta=np.
↪ndarray>
    tasmax   (time, lat, lon) float32 dask.array<chunksize=(3375, 69, 144), meta=np.
↪ndarray>
    pr       (time, lat, lon) float32 dask.array<chunksize=(3375, 69, 144), meta=np.
↪ndarray>
Attributes: (12/15)
```

(continues on next page)

(continued from previous page)

```
    Conventions:            CF-1.5
    title:                  NRCAN ANUSPLIN daily gridded dataset : version 2
    history:                Fri Jan 25 14:11:15 2019 : Convert from original fo...
    institute_id:           NRCAN
    frequency:              day
    abstract:               Gridded daily observational dataset produced by Nat...
    ...                     ...
    dataset_id:             NRCAN_anusplin_daily_v2
    version:                2.0
    license_type:           permissive
    license:                https://open.canada.ca/en/open-government-licence-c...
    attribution:            The authors provide this data under the Environment...
    citation:               Natural Resources Canada ANUSPLIN interpolated hist...
```

```
lon0 = float(bounds.minx)
lon1 = float(bounds.maxx)
lat0 = float(bounds.miny)
lat1 = float(bounds.maxy)

result_tasmin = finch.subset_bbox(
    resource=data,
    variable="tasmin",
    lon0=lon0,
    lon1=lon1,
    lat0=lat0,
    lat1=lat1,
    start_date="1958-01-01",
    end_date="1958-12-31",
)
```

```
# NBVAL_IGNORE_OUTPUT

# wait for process to complete before running this cell (the process is async)
tasmin_subset = result_tasmin.get().output
tasmin_subset
```

```
'https://pavics.ouranos.ca/wpsoutputs/finch/d00d6fa4-b40a-11ee-acf8-0242ac130007/nrcan_
↪v2_sub.ncml'
```

```
result_tasmax = finch.subset_bbox(
    resource=data,
    variable="tasmax",
    lon0=lon0,
    lon1=lon1,
    lat0=lat0,
    lat1=lat1,
    start_date="1958-01-01",
    end_date="1958-12-31",
)
```

```
# NBVAL_IGNORE_OUTPUT

# wait for process to complete before running this cell (the process is async)
tasmax_subset = result_tasmax.get().output
tasmax_subset
```

```
'https://pavics.ouranos.ca/wpsoutputs/finch/d1151ece-b40a-11ee-acf8-0242ac130007/nrcan_
↪v2_sub.ncml'
```

Launch a heat wave frequency analysis with the subsetted datasets

```
help(finch.heat_wave_frequency)
```

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

heat_wave_frequency(tasmin=None, tasmax=None, thresh_tasmin='22.0 degC', thresh_tasmax=
↪'30 degC', window=3, freq='YS', check_missing='any', missing_options=None, cf_
↪compliance='warn', data_validation='raise', variable=None, output_name=None, output_
↪format='netcdf', csv_precision=None, output_formats=None) method of birdy.client.base.
↪WPSClient instance
    Number of heat waves over a given period. A heat wave is defined as an event where␣
↪the minimum and maximum daily temperature both exceeds specific thresholds over a␣
↪minimum number of days.

    Parameters
    ----------
    tasmin : ComplexData:mimetype:`application/x-netcdf`, :mimetype:`application/x-ogc-
↪dods`
        NetCDF Files or archive (tar/zip) containing netCDF files. Minimum surface␣
↪temperature.
    tasmax : ComplexData:mimetype:`application/x-netcdf`, :mimetype:`application/x-ogc-
↪dods`
        NetCDF Files or archive (tar/zip) containing netCDF files. Maximum surface␣
↪temperature.
    thresh_tasmin : string
        The minimum temperature threshold needed to trigger a heatwave event.
    thresh_tasmax : string
        The maximum temperature threshold needed to trigger a heatwave event.
    window : integer
        Minimum number of days with temperatures above thresholds to qualify as a␣
↪heatwave.
    freq : {'YS', 'MS', 'QS-DEC', 'AS-JUL'}string
        Resampling frequency.
    check_missing : {'any', 'wmo', 'pct', 'at_least_n', 'skip', 'from_context'}string
        Method used to determine which aggregations should be considered missing.
    missing_options : ComplexData:mimetype:`application/json`
        JSON representation of dictionary of missing method parameters.
    cf_compliance : {'log', 'warn', 'raise'}string
        Whether to log, warn or raise when inputs have non-CF-compliant attributes.
    data_validation : {'log', 'warn', 'raise'}string
```

(continues on next page)

(continued from previous page)

```
        Whether to log, warn or raise when inputs fail data validation checks.
    variable : string
        Name of the variable in the NetCDF file.
    output_name : string
        Filename of the output (no extension).
    output_format : {'netcdf', 'csv'}string
        Choose in which format you want to receive the result. CSV actually means a zip␣
↪file of two csv files.
    csv_precision : integer
        Only valid if output_format is CSV. If not set, all decimal places of a 64 bit␣
↪floating precision number are printed. If negative, rounds before the decimal point.

    Returns
    -------
    output : ComplexData:mimetype:`application/x-netcdf`, :mimetype:`application/zip`
        The format depends on the 'output_format' input parameter.
    output_log : ComplexData:mimetype:`text/plain`
        Collected logs during process run.
    ref : ComplexData:mimetype:`application/metalink+xml; version=4.0`
        Metalink file storing all references to output files.
```

```
result = finch.heat_wave_frequency(
    tasmin=tasmin_subset,
    tasmax=tasmax_subset,
    thresh_tasmin="14 degC",
    thresh_tasmax="22 degC",
)
```

Get the output as a python object (xarray Dataset)

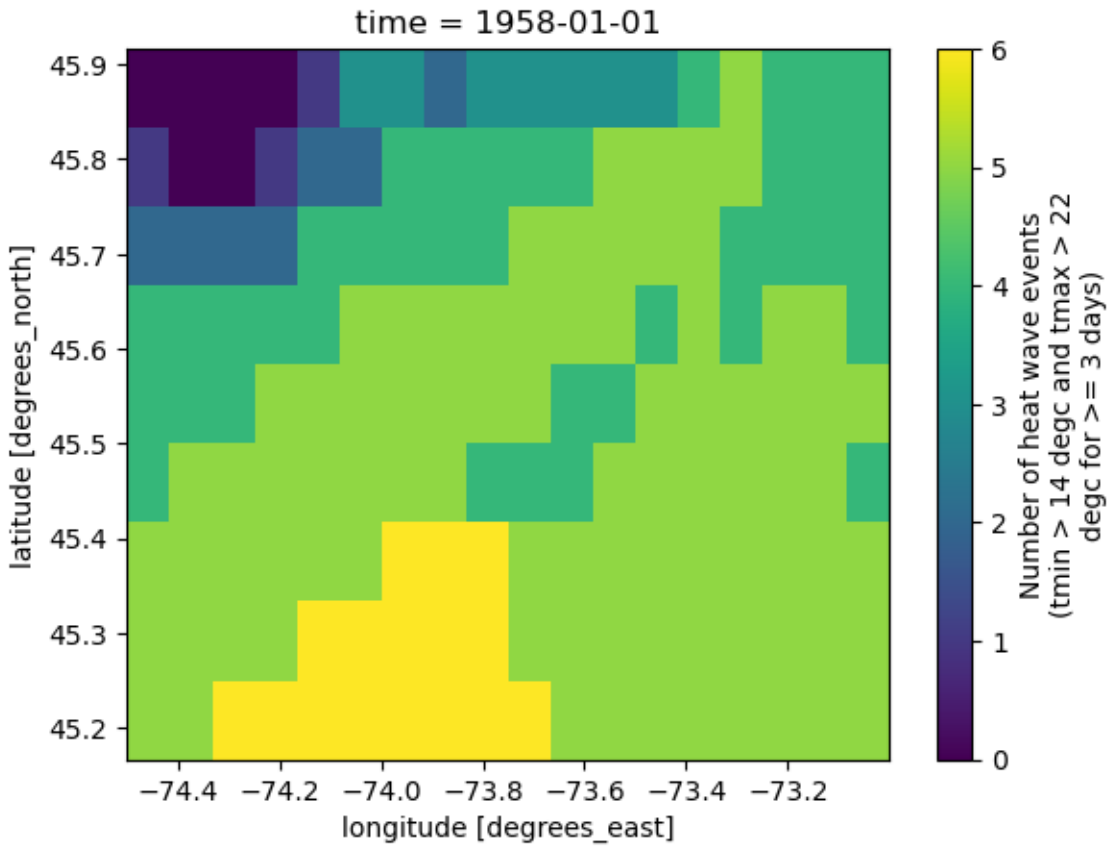
```
response = result.get(asobj=True)
```

```
Downloading to /tmp/tmpqet24mhs/out.nc.
```

```
ds = response.output
```

```
ds.heat_wave_frequency.plot()
```

```
<matplotlib.collections.QuadMesh at 0x7fccd44f0df0>
```



Streaming the data and showing on a map

The results are stored in a folder that is available to thredds, which provides a multiple services to access its datasets. In the case of large outputs, the user could view ths results of the analysis through an OPeNDAP service, so only the data to be shown is downloaded, and not the whole dataset.

Get the OPeNDAP url from the 'wpsoutputs'

```
from urllib.parse import urlparse

output_url = result.get().output
print("output_url = ", output_url)
parsed = urlparse(output_url)
output_path = parsed.path.replace("wpsoutputs", "wps_outputs")
print("output_path = ", output_path)

output_thredds_url = (
    f"https://{parsed.hostname}/twitcher/ows/proxy/thredds/dodsC/birdhouse{output_path}"
)
print("output_thredds_url = ", output_thredds_url)
```

```
output_url =  https://pavics.ouranos.ca/wpsoutputs/finch/d2117e94-b40a-11ee-acf8-
↪0242ac130007/out.nc
output_path =  /wps_outputs/finch/d2117e94-b40a-11ee-acf8-0242ac130007/out.nc
output_thredds_url =  https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/
↪birdhouse/wps_outputs/finch/d2117e94-b40a-11ee-acf8-0242ac130007/out.nc
```

This time, we will be using `hvplot` to build our figure. This tool is a part of the `holoviz` libraries and adds an easy interface to xarray datasets. Since `geoviews` and `cartopy` are also installed, we can simply pass `geo=True` and a choice for `tiles` to turn the plot into a map. If multiple times were present in the dataset, a slider would appear, letting the user choose the slice. As we are using an OPeNDAP link, only the data that needs to be plotted (the current time slice) is downloaded.

```
# NBVAL_IGNORE_OUTPUT
import hvplot.xarray

dsremote = xr.open_dataset(output_thredds_url)
dsremote.hvplot.quadmesh(
    "lon",
    "lat",
    "heat_wave_frequency",
    geo=True,
    alpha=0.8,
    frame_height=540,
    cmap="viridis",
    tiles="CartoLight",
)
```

```
/opt/conda/envs/birdy/lib/python3.9/site-packages/geoviews/operation/__init__.py:14:␣
↪HoloviewsDeprecationWarning: 'ResamplingOperation' is deprecated and will be removed␣
↪in version 1.17, use 'ResampleOperation2D' instead.
  from holoviews.operation.datashader import (
```

```
:DynamicMap   [time]
   :Overlay
      .Tiles.I    :Tiles   [x,y]
      .QuadMesh.I :QuadMesh   [lon,lat]   (heat_wave_frequency)
```

1.2.18 Deprecated Notebooks

Warning: These notebooks are marked for deprecation. Their workflow could still be working temporarily, but they are expected to fail in a near future as their features will be replaced by other mechanisms.

PAVICS catalog search

To find files that meet constraints, PAVICS offer a process called `pavicssearch` that searches through a catalog for files matching user-defined criteria. The information for each file is scraped from the attributes of each netCDF file.

```
import collections
```

(continues on next page)

(continued from previous page)

```
from birdy import WPSClient

url = "https://pavics.ouranos.ca/twitcher/ows/proxy/catalog/wps"
wps = WPSClient(url)
help(wps.pavicsearch)
```

```
---------------------------------------------------------------------------
ServiceException                          Traceback (most recent call last)
Cell In[1], line 6
      3 from birdy import WPSClient
      5 url = "https://pavics.ouranos.ca/twitcher/ows/proxy/catalog/wps"
----> 6 wps = WPSClient(url)
      7 help(wps.pavicsearch)

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/birdy/client/base.py:147, in WPSClient.__init__(self, url, processes,␣
↪converters, username, password, headers, auth, verify, cert, progress, version, caps_
↪xml, desc_xml, language, lineage, **kwds)
    133 self._wps = WebProcessingService(
    134     url,
    135     version=version,
   (...)
    143     **kwds,
    144 )
    146 try:
--> 147     self._wps.getcapabilities(xml=caps_xml)
    148 except ServiceException as e:
    149     if "AccessForbidden" in str(e):

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/owslib/wps.py:277, in WebProcessingService.getcapabilities(self, xml)
    275     self._capabilities = reader.readFromString(xml)
    276 else:
--> 277     self._capabilities = reader.readFromUrl(
    278         self.url, headers=self.headers)
    280 log.debug(element_to_string(self._capabilities))
    282 # populate the capabilities metadata obects from the XML tree

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/owslib/wps.py:546, in WPSCapabilitiesReader.readFromUrl(self, url, username,␣
↪password, headers, verify, cert)
    539 def readFromUrl(self, url, username=None, password=None,
    540                 headers=None, verify=None, cert=None):
    541     """
    542     Method to get and parse a WPS capabilities document, returning an␣
↪elementtree instance.
    543
    544     :param str url: WPS service base url, to which is appended the HTTP␣
↪parameters: service, version, and request.
    545     """
--> 546     return self._readFromUrl(url,
    547                              {'service': 'WPS', 'request':
```

(continues on next page)

(continued from previous page)

```
    548                                 'GetCapabilities', 'version': self.version},
    549                             self.timeout,
    550                             username=username, password=password,
    551                             headers=headers, verify=verify, cert=cert)

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/owslib/wps.py:503, in WPSReader._readFromUrl(self, url, data, timeout, method,
↪ username, password, headers, verify, cert)
    501     # split URL into base url and query string to use utility function
    502     spliturl = request_url.split('?')
--> 503     u = openURL(spliturl[0], spliturl[
    504                 1], method='Get', username=self.auth.username, password=self.
↪auth.password,
    505                 headers=headers, verify=self.auth.verify, cert=self.auth.cert,␣
↪timeout=self.timeout)
    506     return etree.fromstring(u.read())
    508 elif method == 'Post':

File ~/checkouts/readthedocs.org/user_builds/pavics-sdi/conda/latest/lib/python3.10/site-
↪packages/owslib/util.py:212, in openURL(url_base, data, method, cookies, username,␣
↪password, timeout, headers, verify, cert, auth)
    209 req = requests.request(method.upper(), url_base, headers=headers, **rkwargs)
    211 if req.status_code in [400, 401]:
--> 212     raise ServiceException(req.text)
    214 if req.status_code in [404, 500, 502, 503, 504]:    # add more if needed
    215     req.raise_for_status()

ServiceException: <?xml version="1.0" encoding="utf-8"?>
<ExceptionReport version="1.0.0"
    xmlns="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">
    <Exception exceptionCode="NoApplicableCode" locator="NotAcceptable">
        <ExceptionText>Request failed: HTTPConnectionPool(host=&#x27;pavics.ouranos.ca&
↪#x27;, port=8086): Max retries exceeded with url: /pywps/wps?service=WPS&amp;
↪request=GetCapabilities&amp;version=1.0.0 (Caused by NewConnectionError(&#x27;&lt;
↪urllib3.connection.HTTPConnection object at 0x7fd750cf06d0&gt;: Failed to establish a␣
↪new connection: [Errno 111] Connection refused&#x27;))</ExceptionText>
    </Exception>
</ExceptionReport>
```

Potential search constraints are:

- project
- experiment
- model
- frequency
- variable
- variable_long_name

- units
- institute

Note that the *rip* label (realization, initialization, physics), e.g. r5i1p1, is missing from search facets.

The process returns an output dictionary storing the search facets of each file found, as well as a simple list of the links. Note that it is important to specify `type="File"`, otherwise the process will look for datasets, ie file aggregations. At the moment, very few aggregations are available on the PAVICS data server.

```
# NBVAL_IGNORE_OUTPUT

resp = wps.pavicsearch(
    constraints="variable:tasmax,project:CMIP5,experiment:rcp45,model:MPI-ESM-MR,
↪institute:MPI-M,frequency:mon",
    limit=100,
    type="File",
)
[result, files] = resp.get(asobj=True)
files
```

```
['https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/cmip5/MPI-M/MPI-
↪ESM-MR/rcp45/mon/atmos/r2i1p1/tasmax/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-210012.
↪nc',
 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/testdata/secure/
↪tasmax_Amon_MPI-ESM-MR_rcp45_r1i1p1_200601-200612.nc',
 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/testdata/
↪flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r1i1p1_200701-200712.nc',
 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/testdata/
↪flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc',
 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/testdata/secure/
↪tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc',
 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/cmip5/MPI-M/MPI-
↪ESM-MR/rcp45/mon/atmos/r3i1p1/tasmax/tasmax_Amon_MPI-ESM-MR_rcp45_r3i1p1_200601-210012.
↪nc',
 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/testdata/
↪flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r1i1p1_200601-200612.nc',
 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/testdata/secure/
↪tasmax_Amon_MPI-ESM-MR_rcp45_r1i1p1_200701-200712.nc',
 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/cmip5/MPI-M/MPI-
↪ESM-MR/rcp45/mon/atmos/r1i1p1/tasmax/tasmax_Amon_MPI-ESM-MR_rcp45_r1i1p1_200601-210012.
↪nc']
```

```
# NBVAL_IGNORE_OUTPUT

searchfile = [
    f
    for f in result["response"]["docs"]
    if f["resourcename"]
    == "birdhouse/testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-
↪200612.nc"
]
searchfile[0]
```

```
{'cf_standard_name': ['air_temperature'],
 'abstract': 'birdhouse/testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_
↪200601-200612.nc',
 'replica': False,
 'wms_url': 'https://pavics.ouranos.ca/twitcher/ows/proxy/ncWMS2/wms?SERVICE=WMS&
↪REQUEST=GetCapabilities&VERSION=1.3.0&DATASET=outputs/testdata/flyingpigeon/cmip5/
↪tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc',
 'keywords': ['air_temperature',
  'mon',
  'application/netcdf',
  'tasmax',
  'thredds',
  'CMIP5',
  'rcp45',
  'MPI-ESM-MR',
  'MPI-M'],
 'dataset_id': 'testdata.flyingpigeon.cmip5',
 'datetime_max': '2006-12-16T12:00:00Z',
 'id': '44b680cec0a7d4cc',
 'subject': 'Birdhouse Thredds Catalog',
 'category': 'thredds',
 'opendap_url': 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/
↪testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc',
 'title': 'tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc',
 'variable_palette': ['default'],
 'variable_min': [0],
 'variable_long_name': ['Daily Maximum Near-Surface Air Temperature'],
 'source': 'https://pavics.ouranos.ca//twitcher/ows/proxy/thredds/catalog.xml',
 'datetime_min': '2006-01-16T12:00:00Z',
 'score': 1.0,
 'variable_max': [1],
 'units': ['K'],
 'resourcename': 'birdhouse/testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_
↪r2i1p1_200601-200612.nc',
 'type': 'File',
 'catalog_url': 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/catalog/birdhouse/
↪testdata/flyingpigeon/cmip5/catalog.xml?dataset=birdhouse/testdata/flyingpigeon/cmip5/
↪tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc',
 'experiment': 'rcp45',
 'last_modified': '2018-12-21T15:13:38Z',
 'content_type': 'application/netcdf',
 '_version_': 1658705594373111809,
 'variable': ['tasmax'],
 'url': 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/fileServer/birdhouse/
↪testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc',
 'project': 'CMIP5',
 'institute': 'MPI-M',
 'frequency': 'mon',
 'model': 'MPI-ESM-MR',
 'latest': True,
 'fileserver_url': 'https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/fileServer/
↪birdhouse/testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-
↪200612.nc'}
```

```
for k in sorted(searchfile[0].keys()):
    # remove attributes that changes between different servers for the same file
    if k not in ["id", "last_modified", "_version_", "source"]:
        value = searchfile[0][k]
        valuesorted = (
            sorted(value)
            if (
                isinstance(value, collections.abc.Iterable)
                and not isinstance(value, str)
            )
            else value
        )
        print(f"{k}: {valuesorted}")
```

```
abstract: birdhouse/testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_
↪200601-200612.nc
catalog_url: https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/catalog/birdhouse/
↪testdata/flyingpigeon/cmip5/catalog.xml?dataset=birdhouse/testdata/flyingpigeon/cmip5/
↪tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc
category: thredds
cf_standard_name: ['air_temperature']
content_type: application/netcdf
dataset_id: testdata.flyingpigeon.cmip5
datetime_max: 2006-12-16T12:00:00Z
datetime_min: 2006-01-16T12:00:00Z
experiment: rcp45
fileserver_url: https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/fileServer/
↪birdhouse/testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-
↪200612.nc
frequency: mon
institute: MPI-M
keywords: ['CMIP5', 'MPI-ESM-MR', 'MPI-M', 'air_temperature', 'application/netcdf', 'mon
↪', 'rcp45', 'tasmax', 'thredds']
latest: True
model: MPI-ESM-MR
opendap_url: https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/
↪testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc
project: CMIP5
replica: False
resourcename: birdhouse/testdata/flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_
↪200601-200612.nc
score: 1.0
subject: Birdhouse Thredds Catalog
title: tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc
type: File
units: ['K']
url: https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/fileServer/birdhouse/testdata/
↪flyingpigeon/cmip5/tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc
variable: ['tasmax']
variable_long_name: ['Daily Maximum Near-Surface Air Temperature']
variable_max: [1]
variable_min: [0]
```

(continues on next page)

(continued from previous page)

```
variable_palette: ['default']
wms_url: https://pavics.ouranos.ca/twitcher/ows/proxy/ncWMS2/wms?SERVICE=WMS&
↪REQUEST=GetCapabilities&VERSION=1.3.0&DATASET=outputs/testdata/flyingpigeon/cmip5/
↪tasmax_Amon_MPI-ESM-MR_rcp45_r2i1p1_200601-200612.nc
```

1.2.19 Optional Notebooks

These following notebooks demonstrate *additional* and *optional* features of the PAVICS platform when corresponding Components or Optional Components are enabled.

PAVICS Web Processing Services using OGC-API integration with Weaver

When Weaver component is enabled, all WPS *birds* registered as process *providers* will be automatically accessible using OGC-API - Processes interface from the endpoint where Weaver is defined.

```
import json
import os
import time

import requests
import urllib3

WEAVER_TEST_FQDN = os.getenv(
    "WEAVER_TEST_FQDN", os.getenv("PAVICS_HOST", "pavics.ouranos.ca")
)
WEAVER_TEST_URL = os.getenv("WEAVER_TEST_URL", f"https://{WEAVER_TEST_FQDN}/weaver")
WEAVER_TEST_SSL_VERIFY = str(os.getenv("WEAVER_TEST_SSL_VERIFY", "true")).lower() in [
    "true",
    "1",
    "on",
    "yes",
]
WEAVER_TEST_DEFAULT_BIRDS = "finch, hummingbird, raven"
WEAVER_TEST_KNOWN_BIRDS = os.getenv(
    "WEAVER_TEST_KNOWN_BIRDS", WEAVER_TEST_DEFAULT_BIRDS
)
WEAVER_TEST_KNOWN_BIRDS = list(
    bird.strip() for bird in WEAVER_TEST_KNOWN_BIRDS.split(",")
)
WEAVER_TEST_DEFAULT_FILE = "/twitcher/ows/proxy/thredds/dodsC/birdhouse/testdata/ta_Amon_
↪MRI-CGCM3_decadal1980_r1i1p1_199101-200012.nc"
WEAVER_TEST_FILE = os.getenv(
    "WEAVER_TEST_FILE",
    f"https://{WEAVER_TEST_FQDN}{WEAVER_TEST_DEFAULT_FILE}",
)
WEAVER_TEST_WPS_OUTPUTS = f"https://{WEAVER_TEST_FQDN}/wpsoutputs"  # for validation

WEAVER_TEST_REQUEST_HEADERS = {
    "Accept": "application/json",
    "Content-Type": "application/json",
```

(continues on next page)

(continued from previous page)

```
}
WEAVER_TEST_REQUEST_XARGS = dict(
    headers=WEAVER_TEST_REQUEST_HEADERS, verify=WEAVER_TEST_SSL_VERIFY, timeout=5
)

if not WEAVER_TEST_SSL_VERIFY:
    urllib3.disable_warnings()

print("Variables:")
variables = [
    ("WEAVER_TEST_FQDN", WEAVER_TEST_FQDN),
    ("WEAVER_TEST_URL", WEAVER_TEST_URL),
    ("WEAVER_TEST_WPS_OUTPUTS", WEAVER_TEST_WPS_OUTPUTS),
    ("WEAVER_TEST_SSL_VERIFY", WEAVER_TEST_SSL_VERIFY),
    ("WEAVER_TEST_FILE", WEAVER_TEST_FILE),
    ("WEAVER_TEST_KNOWN_BIRDS", WEAVER_TEST_KNOWN_BIRDS),
    ("WEAVER_TEST_REQUEST_XARGS", WEAVER_TEST_REQUEST_XARGS),
]
max_len = max(len(var[0]) for var in variables) + 2
msg = f"  {{:{max_len}}}{{}}"
for var, val in variables:
    print(msg.format(var, val))


assert (
    len(WEAVER_TEST_KNOWN_BIRDS) >= 1
), "No test WPS provider provided in 'WEAVER_TEST_KNOWN_BIRDS'."
```

```
Variables:
  WEAVER_TEST_FQDN           pavics.ouranos.ca
  WEAVER_TEST_URL            https://pavics.ouranos.ca/weaver
  WEAVER_TEST_WPS_OUTPUTS    https://pavics.ouranos.ca/wpsoutputs
  WEAVER_TEST_SSL_VERIFY     True
  WEAVER_TEST_FILE           https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/
↪birdhouse/testdata/ta_Amon_MRI-CGCM3_decadal1980_r1i1p1_199101-200012.nc
  WEAVER_TEST_KNOWN_BIRDS    ['finch', 'hummingbird', 'raven']
  WEAVER_TEST_REQUEST_XARGS  {'headers': {'Accept': 'application/json', 'Content-Type':
↪'application/json'}, 'verify': True, 'timeout': 5}
```

Define some utility functions for displaying test results

```
def json_dump(_json):
    try:
        if isinstance(_json, str):
            _json = json.loads(_json)
        return json.dumps(_json, indent=2, ensure_ascii=False)
    except Exception:
        return str(_json)
```

(continues on next page)

(continued from previous page)

```
def json_print(_json):
    print(json_dump(_json))
```

Start with simple listing of registered WPS providers in Weaver

```
print("Listing WPS providers registered under Weaver...\n")

path = f"{WEAVER_TEST_URL}/providers"
query = {
    "detail": False,
    "check": False,
}  # skip pre-fetch to obtain results quickly (all checked in following cells)
resp = requests.get(path, params=query, **WEAVER_TEST_REQUEST_XARGS)
assert (
    resp.status_code == 200
), f"Error during WPS bird providers listing from [{path}]:\n{json_dump(resp.text)}"
body = resp.json()
json_print(body)

assert "providers" in body and len(
    body["providers"]
), f"Could not find Weaver WPS providers in response:\n{json_dump(body)}"
missing = []
for bird in sorted(WEAVER_TEST_KNOWN_BIRDS):
    if bird not in body["providers"]:
        missing.append(bird)
assert (
    not missing
), f"Could not find all expected Weaver WPS providers.\nMissing: [{missing}]\nExpected: [
↪{WEAVER_TEST_KNOWN_BIRDS}]"
bird_ids = body["providers"]
```

```
Listing WPS providers registered under Weaver...

{
  "checked": false,
  "providers": [
    "catalog",
    "finch",
    "flyingpigeon",
    "hummingbird",
    "malleefowl",
    "raven"
  ]
}
```

Obtain OGC-API converted WPS processes by Weaver from original WPS providers endpoints

For each registered provider, Weaver sends a *GetCapabilities* WPS request to the remote endpoint and parses the XML result in order to form the corresponding OGC-API JSON content.

```
print("Listing WPS provider processes converted to OGC-API interface by Weaver:\n")

process_locations = []
for bird in sorted(WEAVER_TEST_KNOWN_BIRDS):
    path = f"{WEAVER_TEST_URL}/providers/{bird}/processes"
    resp = requests.get(path, **WEAVER_TEST_REQUEST_XARGS)
    assert (
        resp.status_code == 200
    ), f"Error during WPS bird processes retrieval on: [{path}]\n[{json_dump(resp.text)}]
↪"
    body = resp.json()
    assert len(body["processes"]), f"WPS bird [{bird}] did not list any process!"
    for process in sorted(body["processes"], key=lambda p: p["id"]):
        process_desc_url = f"{path}/{process['id']}"
        process_locations.append(process_desc_url)
        print(" -", process_desc_url)
```

```
Listing WPS provider processes converted to OGC-API interface by Weaver:
```

```
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/average_polygon
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/base_flow_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/biologically_effective_
↪degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/blowing_snow
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/calm_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/cdd
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/cold_and_dry_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/cold_and_wet_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/cold_spell_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/cold_spell_duration_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/cold_spell_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/consecutive_frost_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/consecutive_frost_free_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/continuous_snow_cover_end
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/continuous_snow_cover_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/cool_night_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/cooling_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/corn_heat_units
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/cwd
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/days_over_precip_doy_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/days_over_precip_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/days_with_snow
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/degree_days_exceedance_date
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/dlyfrzthw
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/doy_qmax
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/doy_qmin
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/dry_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/dry_spell_frequency
```

(continues on next page)

(continued from previous page)

```
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/dry_spell_total_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/dtr
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/dtrmax
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/dtrvar
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/effective_growing_degree_
↪days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/empirical_quantile_mapping
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_cdd
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_cold_and_dry_
↪days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_cold_and_wet_
↪days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_cold_spell_
↪days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_cold_spell_
↪duration_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_cold_spell_
↪frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_consecutive_
↪frost_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_consecutive_
↪frost_free_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_cooling_
↪degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_corn_heat_
↪units
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_cwd
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_days_over_
↪precip_doy_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_days_over_
↪precip_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_degree_days_
↪exceedance_date
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_dlyfrzthw
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_dry_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_dry_spell_
↪frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_dry_spell_
↪total_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_dtr
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_dtrmax
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_dtrvar
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_effective_
↪growing_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_etr
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_first_day_
↪above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_first_day_
↪below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_fraction_
↪over_precip_doy_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_fraction_
```

(continues on next page)

(continued from previous page)

```
↪over_precip_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_freezethaw_
↪spell_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_freezethaw_
↪spell_max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_freezethaw_
↪spell_mean_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_freezing_
↪degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_freshet_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_frost_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_frost_free_
↪season_end
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_frost_free_
↪season_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_frost_free_
↪season_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_frost_season_
↪length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_growing_
↪degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_growing_
↪season_end
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_growing_
↪season_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_growing_
↪season_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_heat_wave_
↪frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_heat_wave_
↪index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_heat_wave_
↪max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_heat_wave_
↪total_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_heating_
↪degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_high_precip_
↪low_temp
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_hot_spell_
↪frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_hot_spell_
↪max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_ice_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_last_spring_
↪frost
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_liquid_
↪precip_ratio
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_liquidprcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_max_n_day_
↪precipitation_amount
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_max_pr_
```

(continues on next page)

(continued from previous page)

```
↪intensity
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_maximum_
↪consecutive_warm_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_prcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_prlp
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_prsn
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_rain_frzgr
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_rx1day
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_sdii
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_solidprcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tg
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tg10p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tg90p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tg_days_above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tg_days_below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tg_max
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tg_mean
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tg_min
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_thawing_
↪degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tn10p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tn90p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tn_days_above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tn_days_below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tn_max
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tn_mean
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tn_min
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tropical_
↪nights
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tx10p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tx90p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tx_days_above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tx_days_below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tx_max
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tx_mean
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tx_min
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_tx_tn_days_
↪above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_warm_and_dry_
↪days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_warm_and_wet_
↪days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_warm_spell_
↪duration_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_wet_prcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_wetdays
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_bbox_wetdays_prop
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_cdd
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_cold_
↪and_dry_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_cold_
↪and_wet_days
```

(continues on next page)

(continued from previous page)

```
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_cold_
↪spell_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_cold_
↪spell_duration_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_cold_
↪spell_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪consecutive_frost_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪consecutive_frost_free_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪cooling_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_corn_
↪heat_units
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_cwd
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_days_
↪over_precip_doy_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_days_
↪over_precip_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_degree_
↪days_exceedance_date
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪dlyfrzthw
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_dry_
↪days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_dry_
↪spell_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_dry_
↪spell_total_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_dtr
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_dtrmax
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_dtrvar
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪effective_growing_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_etr
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_first_
↪day_above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_first_
↪day_below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪fraction_over_precip_doy_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪fraction_over_precip_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪freezethaw_spell_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪freezethaw_spell_max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪freezethaw_spell_mean_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪freezing_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
```

(continues on next page)

(continued from previous page)

```
↪freshet_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_frost_
↪days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_frost_
↪free_season_end
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_frost_
↪free_season_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_frost_
↪free_season_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_frost_
↪season_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪growing_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪growing_season_end
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪growing_season_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪growing_season_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_heat_
↪wave_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_heat_
↪wave_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_heat_
↪wave_max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_heat_
↪wave_total_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪heating_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_high_
↪precip_low_temp
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_hot_
↪spell_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_hot_
↪spell_max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_ice_
↪days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_last_
↪spring_frost
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_liquid_
↪precip_ratio
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪liquidprcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_max_n_
↪day_precipitation_amount
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_max_pr_
↪intensity
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪maximum_consecutive_warm_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_prcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_prlp
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_prsn
```

(continues on next page)

(continued from previous page)

```
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_rain_
↪frzgr
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_rx1day
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_sdii
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪solidprcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tg
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tg10p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tg90p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tg_
↪days_above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tg_
↪days_below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tg_max
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tg_mean
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tg_min
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪thawing_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tn10p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tn90p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tn_
↪days_above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tn_
↪days_below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tn_max
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tn_mean
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tn_min
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪tropical_nights
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tx10p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tx90p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tx_
↪days_above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tx_
↪days_below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tx_max
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tx_mean
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tx_min
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_tx_tn_
↪days_above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_warm_
↪and_dry_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_warm_
↪and_wet_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_warm_
↪spell_duration_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_wet_
↪prcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_wetdays
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_grid_point_
↪wetdays_prop
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_cdd
```

(continues on next page)

(continued from previous page)

```
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_cold_and_
↪dry_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_cold_and_
↪wet_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_cold_
↪spell_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_cold_
↪spell_duration_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_cold_
↪spell_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_
↪consecutive_frost_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_
↪consecutive_frost_free_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_cooling_
↪degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_corn_heat_
↪units
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_cwd
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_days_over_
↪precip_doy_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_days_over_
↪precip_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_degree_
↪days_exceedance_date
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_dlyfrzthw
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_dry_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_dry_spell_
↪frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_dry_spell_
↪total_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_dtr
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_dtrmax
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_dtrvar
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_effective_
↪growing_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_etr
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_first_day_
↪above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_first_day_
↪below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_fraction_
↪over_precip_doy_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_fraction_
↪over_precip_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_
↪freezethaw_spell_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_
↪freezethaw_spell_max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_
↪freezethaw_spell_mean_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_freezing_
```

(continues on next page)

(continued from previous page)

```
↪degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_freshet_
↪start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_frost_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_frost_
↪free_season_end
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_frost_
↪free_season_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_frost_
↪free_season_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_frost_
↪season_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_growing_
↪degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_growing_
↪season_end
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_growing_
↪season_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_growing_
↪season_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_heat_wave_
↪frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_heat_wave_
↪index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_heat_wave_
↪max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_heat_wave_
↪total_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_heating_
↪degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_high_
↪precip_low_temp
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_hot_spell_
↪frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_hot_spell_
↪max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_ice_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_last_
↪spring_frost
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_liquid_
↪precip_ratio
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_
↪liquidprcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_max_n_day_
↪precipitation_amount
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_max_pr_
↪intensity
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_maximum_
↪consecutive_warm_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_prcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_prlp
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_prsn
```

(continues on next page)

(continued from previous page)

```
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_rain_frzgr
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_rx1day
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_sdii
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_
↪solidprcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tg
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tg10p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tg90p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tg_days_
↪above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tg_days_
↪below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tg_max
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tg_mean
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tg_min
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_thawing_
↪degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tn10p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tn90p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tn_days_
↪above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tn_days_
↪below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tn_max
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tn_mean
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tn_min
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tropical_
↪nights
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tx10p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tx90p
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tx_days_
↪above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tx_days_
↪below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tx_max
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tx_mean
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tx_min
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_tx_tn_
↪days_above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_warm_and_
↪dry_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_warm_and_
↪wet_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_warm_
↪spell_duration_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_wet_
↪prcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_wetdays
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ensemble_polygon_wetdays_
↪prop
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/etr
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/fire_season
```

(continues on next page)

(continued from previous page)

```
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/first_day_above
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/first_day_below
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/first_snowfall
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/fit
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/fraction_over_precip_doy_
↪thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/fraction_over_precip_thresh
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/freezethaw_spell_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/freezethaw_spell_max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/freezethaw_spell_mean_
↪length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/freezing_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/freq_analysis
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/freshet_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/frost_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/frost_free_season_end
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/frost_free_season_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/frost_free_season_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/frost_season_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/geoseries_to_netcdf
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/growing_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/growing_season_end
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/growing_season_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/growing_season_start
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/heat_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/heat_wave_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/heat_wave_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/heat_wave_max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/heat_wave_total_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/heating_degree_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/high_precip_low_temp
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/hot_spell_frequency
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/hot_spell_max_length
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/hourly_to_daily
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/huglin_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/humidex
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/hurs
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/hurs_fromdewpoint
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/huss_fromdewpoint
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/ice_days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/jetstream_metric_woollings
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/last_snowfall
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/last_spring_frost
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/latitude_temperature_index
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/liquid_precip_ratio
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/liquidprcptot
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/max_n_day_precipitation_
↪amount
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/max_pr_intensity
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/maximum_consecutive_warm_
↪days
 - https://pavics.ouranos.ca/weaver/providers/finch/processes/mean_radiant_temperature
```

(continues on next page)

(continued from previous page)

```
- https://pavics.ouranos.ca/weaver/providers/finch/processes/melt_and_precip_max
- https://pavics.ouranos.ca/weaver/providers/finch/processes/potential_
↪evapotranspiration
- https://pavics.ouranos.ca/weaver/providers/finch/processes/prcptot
- https://pavics.ouranos.ca/weaver/providers/finch/processes/prlp
- https://pavics.ouranos.ca/weaver/providers/finch/processes/prsn
- https://pavics.ouranos.ca/weaver/providers/finch/processes/rain_frzgr
- https://pavics.ouranos.ca/weaver/providers/finch/processes/rb_flashiness_index
- https://pavics.ouranos.ca/weaver/providers/finch/processes/rprctot
- https://pavics.ouranos.ca/weaver/providers/finch/processes/rx1day
- https://pavics.ouranos.ca/weaver/providers/finch/processes/sdii
- https://pavics.ouranos.ca/weaver/providers/finch/processes/sea_ice_area
- https://pavics.ouranos.ca/weaver/providers/finch/processes/sea_ice_extent
- https://pavics.ouranos.ca/weaver/providers/finch/processes/snd_max_doy
- https://pavics.ouranos.ca/weaver/providers/finch/processes/snow_cover_duration
- https://pavics.ouranos.ca/weaver/providers/finch/processes/snow_depth
- https://pavics.ouranos.ca/weaver/providers/finch/processes/snow_melt_we_max
- https://pavics.ouranos.ca/weaver/providers/finch/processes/snw_max
- https://pavics.ouranos.ca/weaver/providers/finch/processes/snw_max_doy
- https://pavics.ouranos.ca/weaver/providers/finch/processes/solidprcptot
- https://pavics.ouranos.ca/weaver/providers/finch/processes/stats
- https://pavics.ouranos.ca/weaver/providers/finch/processes/subset_bbox
- https://pavics.ouranos.ca/weaver/providers/finch/processes/subset_bbox_dataset
- https://pavics.ouranos.ca/weaver/providers/finch/processes/subset_grid_point_dataset
- https://pavics.ouranos.ca/weaver/providers/finch/processes/subset_gridpoint
- https://pavics.ouranos.ca/weaver/providers/finch/processes/subset_polygon
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tg
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tg10p
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tg90p
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tg_days_above
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tg_days_below
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tg_max
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tg_mean
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tg_min
- https://pavics.ouranos.ca/weaver/providers/finch/processes/thawing_degree_days
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tn10p
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tn90p
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tn_days_above
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tn_days_below
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tn_max
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tn_mean
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tn_min
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tropical_nights
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tx10p
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tx90p
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tx_days_above
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tx_days_below
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tx_max
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tx_mean
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tx_min
- https://pavics.ouranos.ca/weaver/providers/finch/processes/tx_tn_days_above
- https://pavics.ouranos.ca/weaver/providers/finch/processes/utci
```

(continues on next page)

(continued from previous page)

```
- https://pavics.ouranos.ca/weaver/providers/finch/processes/warm_and_dry_days
- https://pavics.ouranos.ca/weaver/providers/finch/processes/warm_and_wet_days
- https://pavics.ouranos.ca/weaver/providers/finch/processes/warm_spell_duration_index
- https://pavics.ouranos.ca/weaver/providers/finch/processes/water_budget
- https://pavics.ouranos.ca/weaver/providers/finch/processes/water_budget_from_tas
- https://pavics.ouranos.ca/weaver/providers/finch/processes/wet_prcptot
- https://pavics.ouranos.ca/weaver/providers/finch/processes/wetdays
- https://pavics.ouranos.ca/weaver/providers/finch/processes/wetdays_prop
- https://pavics.ouranos.ca/weaver/providers/finch/processes/wind_chill
- https://pavics.ouranos.ca/weaver/providers/finch/processes/wind_speed_from_vector
- https://pavics.ouranos.ca/weaver/providers/finch/processes/wind_vector_from_speed
- https://pavics.ouranos.ca/weaver/providers/finch/processes/windy_days
- https://pavics.ouranos.ca/weaver/providers/finch/processes/winter_storm
```

```
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/cchecker
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/cdo_bbox
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/cdo_copy
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/cdo_indices
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/cdo_inter_mpi
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/cdo_operation
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/cdo_sinfo
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/cfchecker
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/cmor_checker
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ensembles
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/qa_cfchecker
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/qa_checker
- https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/spotchecker
```

```
- https://pavics.ouranos.ca/weaver/providers/raven/processes/hydrobasins-select
- https://pavics.ouranos.ca/weaver/providers/raven/processes/nalcms-zonal-stats
- https://pavics.ouranos.ca/weaver/providers/raven/processes/nalcms-zonal-stats-raster
- https://pavics.ouranos.ca/weaver/providers/raven/processes/raster-subset
- https://pavics.ouranos.ca/weaver/providers/raven/processes/shape-properties
- https://pavics.ouranos.ca/weaver/providers/raven/processes/terrain-analysis
- https://pavics.ouranos.ca/weaver/providers/raven/processes/zonal-stats
```

Dispatched execution of Hummingbird WPS process

Here, we attempt running the `ncdump` process defined Hummingbird. This is analogous to the *WPS_example notebook*, but through the OGC-API interface provided by Weaver.

The process execution received by Weaver gets dispatched to the real WPS location. Weaver then monitors the process until completion and, once completed, returns the location where results can be retrieved.

```
assert (
    "hummingbird" in WEAVER_TEST_KNOWN_BIRDS
), "Hummingbird not specified within known WPS provider birds by Weaver. Cannot test␣
↪dispatched process execution..."

WEAVER_BIRD_URL = f"{WEAVER_TEST_URL}/providers/hummingbird"
```

(continues on next page)

(continued from previous page)

```
WEAVER_BIRD_PROCESS_URL = f"{WEAVER_BIRD_URL}/processes/ncdump"
assert (
    WEAVER_BIRD_PROCESS_URL in process_locations
), f"Could not find WPS bird process URL to test execution [{WEAVER_BIRD_PROCESS_URL}]."

print(f"Will run process: [{WEAVER_BIRD_PROCESS_URL}]")
```

```
Will run process: [https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/
↪ncdump]
```

First let's obtain the specific description of the test WPS process

This request will tell us the explicit details of the process such as its inputs, outputs, and other metadata. Weaver parses the results retrieved from the original WPS provider using *DescribeProcess* request to generate the corresponding outputs. Weaver also adds additional metadata when it can infer some missing details from returned description fields.

```
# NBVAL_IGNORE_OUTPUT
# ignore detailed description prone to changes, instead run a few basic manual␣
↪validations

print("Getting WPS process description...\n")

resp = requests.get(WEAVER_BIRD_PROCESS_URL, **WEAVER_TEST_REQUEST_XARGS)
assert (
    resp.status_code == 200
), f"Error getting WPS process description:\n[{json_dump(resp.text)}]"
body = resp.json()
json_print(body)

assert "hummingbird" in body["keywords"]
assert "wps-remote" in body["keywords"]
assert body["id"] == "ncdump"
```

```
Getting WPS process description...
```


```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://schemas.opengis.net/ogcapi/processes/part1/1.0/openapi/schemas/process.
↪yaml",
  "id": "ncdump",
  "title": "NCDump",
  "version": "4.4.1.1",
  "mutable": true,
  "description": "Run ncdump to retrieve NetCDF header metadata.",
  "keywords": [
    "hummingbird",
    "Hummingbird",
    "wps-remote"
  ],
  "metadata": [
```


(continues on next page)

(continued from previous page)


```
    {
      "title": "Birdhouse",
      "href": "http://bird-house.github.io/",
      "rel": "birdhouse"
    },
    {
      "title": "User Guide",
      "href": "http://birdhouse-hummingbird.readthedocs.io/en/latest/",
      "rel": "user-guide"
    }
  ],
  "inputs": {
    "dataset": {
      "title": "Dataset",
      "description": "Enter a URL pointing to a NetCDF file (optional)",
      "minOccurs": 0,
      "maxOccurs": 100,
      "schema": {
        "type": "array",
        "items": {
          "type": "string"
        },
        "minItems": 0,
        "maxItems": 100
      },
      "formats": [
        {
          "default": true,
          "mediaType": "application/x-netcdf"
        }
      ]
    },
    "dataset_opendap": {
      "title": "Remote OpenDAP Data URL",
      "description": "Or provide a remote OpenDAP data URL, for example: http://my.
↪opendap/thredds/dodsC/path/to/file.nc",
      "minOccurs": 0,
      "maxOccurs": 100,
      "schema": {
        "type": "array",
        "items": {
          "type": "string"
        },
        "minItems": 0,
        "maxItems": 100
      },
      "literalDataDomains": [
        {
          "default": true,
          "dataType": {
            "$id": "https://schemas.opengis.net/ogcapi/processes/part1/1.0/openapi/
↪schemas/nameReferenceType.yaml",
```


(continues on next page)

(continued from previous page)


```
            "name": "string"
          },
          "valueDefinition": {
            "anyValue": true
          }
        }
      ]
    }
  },
  "outputs": {
    "output": {
      "title": "NetCDF Metadata",
      "description": "NetCDF Metadata",
      "schema": {
        "type": "string"
      },
      "formats": [
        {
          "default": true,
          "mediaType": "text/plain"
        }
      ]
    }
  },
  "visibility": "public",
  "jobControlOptions": [
    "async-execute"
  ],
  "outputTransmission": [
    "reference",
    "value"
  ],
  "processDescriptionURL": "https://pavics.ouranos.ca/weaver/providers/hummingbird/
↪processes/ncdump",
  "processEndpointWPS1": "https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird?
↪service=WPS&request=DescribeProcess&version=1.0.0&identifier=ncdump",
  "executeEndpoint": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/
↪ncdump/jobs",
  "deploymentProfile": "http://www.opengis.net/profiles/eoc/wpsApplication",
  "links": [
    {
      "title": "Current process description.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump:4.
↪4.1",
      "type": "application/json",
      "rel": "self"
    },
    {
      "title": "Process definition.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump",
```


(continues on next page)

(continued from previous page)


```
      "type": "application/json",
      "rel": "process-meta"
    },
    {
      "title": "Process execution endpoint for job submission.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump/
↪execution",
      "type": "application/json",
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/execute"
    },
    {
      "title": "List of registered processes.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes",
      "type": "application/json",
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/processes"
    },
    {
      "title": "List of job executions corresponding to this process.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump/
↪jobs",
      "type": "application/json",
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/job-list"
    },
    {
      "title": "List of processes registered under the service.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes",
      "type": "application/json",
      "rel": "up"
    },
    {
      "title": "Tagged version of this process description.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump:4.
↪4.1",
      "type": "application/json",
      "rel": "working-copy"
    },
    {
      "title": "Most recent revision of this process.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump",
      "type": "application/json",
      "rel": "latest-version"
    },
    {
      "title": "Listing of all revisions of this process.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes?
```


(continues on next page)

(continued from previous page)

```
↪detail=false&revisions=true&process=ncdump",
      "type": "application/json",
      "rel": "version-history"
    },
    {
      "title": "Provider service description.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird",
      "type": "application/xml",
      "rel": "service"
    },
    {
      "title": "Provider service definition.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird",
      "type": "application/xml",
      "rel": "service-meta"
    },
    {
      "title": "Remote service description.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird?service=WPS&
↪request=GetCapabilities&version=1.0.0",
      "type": "application/xml",
      "rel": "service-desc"
    },
    {
      "title": "Remote process description.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird?service=WPS&
↪request=DescribeProcess&version=1.0.0&identifier=ncdump",
      "type": "application/xml",
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/process-desc"
    }
  ]
}
```

Submit the new process execution

Using OGC-API interface, WPS process execution are accomplished using a *Job*. That job will tell us the status location where we can monitor the process execution.

From the previous response, we can see that the process accepts many inputs and format variations. In this case, we are interested in the input named `dataset` to submit the file defined by `WEAVER_TEST_FILE`.

Following execution of the process, we expect to obtain a raw text data dump of the test file content. The location of the raw text file is expected be provided by output named `output` according to the process description.

```
print("Submitting process job with:")
print(f"  File:     [{WEAVER_TEST_FILE}]")
print(f"  Process:  [{WEAVER_BIRD_PROCESS_URL}]")
```

(continues on next page)

(continued from previous page)

```
data = {
    "mode": "async",  # This tells Weaver to run the process asynchronously, such that␣
↪we get non-blocking status location
    "response": "document",  # Type of status response (only this mode supported for the␣
↪time being)
    "inputs": [
        {
            "id": "dataset_opendap",  # Target input of the process
            # Note: even though this is an URL, the expected type is a 'string' (not a
↪'File')
            #       therefore, 'data' (or 'value') must be used instead of 'href'
            "data": WEAVER_TEST_FILE,
        }
    ],
    "outputs": [
        {
            "id": "output",  # Target output we want to retrieve
            "transmissionMode": "reference",  # Ask to provide the result as HTTP␣
↪reference
        }
    ],
}


# define a function to allow re-submitting later in case of error
def submit_job() -> str:
    _path = f"{WEAVER_BIRD_PROCESS_URL}/jobs"
    _resp = requests.post(_path, json=data, **WEAVER_TEST_REQUEST_XARGS)
    assert _resp.status_code in [
        200,
        201,
    ], f"Error during WPS job submission:\n{json_dump(resp.text)}"
    loc = _resp.headers.get("Location")
    assert loc, "Could not find status location URL"
    return loc


status_location = submit_job()
print(f"Job Status Location: [{status_location}]")
```

```
Submitting process job with:
  File:     [https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/birdhouse/
↪testdata/ta_Amon_MRI-CGCM3_decadal1980_r1i1p1_199101-200012.nc]
  Process:  [https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump]
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[7], line 38
     34     assert loc, "Could not find status location URL"
     35     return loc
---> 38 status_location = submit_job()
```

(continues on next page)

(continued from previous page)

```
     39 print(f"Job Status Location: [{status_location}]")

Cell In[7], line 29, in submit_job()
     27 _path = f"{WEAVER_BIRD_PROCESS_URL}/jobs"
     28 _resp = requests.post(_path, json=data, **WEAVER_TEST_REQUEST_XARGS)
---> 29 assert _resp.status_code in [
     30     200,
     31     201,
     32 ], f"Error during WPS job submission:\n{json_dump(resp.text)}"
     33 loc = _resp.headers.get("Location")
     34 assert loc, "Could not find status location URL"

AssertionError: Error during WPS job submission:
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://schemas.opengis.net/ogcapi/processes/part1/1.0/openapi/schemas/process.
↪yaml",
  "id": "ncdump",
  "title": "NCDump",
  "version": "4.4.1.1",
  "mutable": true,
  "description": "Run ncdump to retrieve NetCDF header metadata.",
  "keywords": [
    "hummingbird",
    "Hummingbird",
    "wps-remote"
  ],
  "metadata": [
    {
      "title": "Birdhouse",
      "href": "http://bird-house.github.io/",
      "rel": "birdhouse"
    },
    {
      "title": "User Guide",
      "href": "http://birdhouse-hummingbird.readthedocs.io/en/latest/",
      "rel": "user-guide"
    }
  ],
  "inputs": {
    "dataset": {
      "title": "Dataset",
      "description": "Enter a URL pointing to a NetCDF file (optional)",
      "minOccurs": 0,
      "maxOccurs": 100,
      "schema": {
        "type": "array",
        "items": {
          "type": "string"
        },
        "minItems": 0,
        "maxItems": 100
```

(continues on next page)

(continued from previous page)

```
      },
      "formats": [
        {
          "default": true,
          "mediaType": "application/x-netcdf"
        }
      ]
    },
    "dataset_opendap": {
      "title": "Remote OpenDAP Data URL",
      "description": "Or provide a remote OpenDAP data URL, for example: http://my.
↪opendap/thredds/dodsC/path/to/file.nc",
      "minOccurs": 0,
      "maxOccurs": 100,
      "schema": {
        "type": "array",
        "items": {
          "type": "string"
        },
        "minItems": 0,
        "maxItems": 100
      },
      "literalDataDomains": [
        {
          "default": true,
          "dataType": {
            "$id": "https://schemas.opengis.net/ogcapi/processes/part1/1.0/openapi/
↪schemas/nameReferenceType.yaml",
            "name": "string"
          },
          "valueDefinition": {
            "anyValue": true
          }
        }
      ]
    }
  },
  "outputs": {
    "output": {
      "title": "NetCDF Metadata",
      "description": "NetCDF Metadata",
      "schema": {
        "type": "string"
      },
      "formats": [
        {
          "default": true,
          "mediaType": "text/plain"
        }
      ]
    }
  },
```

(continues on next page)

(continued from previous page)


```
  "visibility": "public",
  "jobControlOptions": [
    "async-execute"
  ],
  "outputTransmission": [
    "reference",
    "value"
  ],
  "processDescriptionURL": "https://pavics.ouranos.ca/weaver/providers/hummingbird/
↪processes/ncdump",
  "processEndpointWPS1": "https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird?
↪service=WPS&request=DescribeProcess&version=1.0.0&identifier=ncdump",
  "executeEndpoint": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/
↪ncdump/jobs",
  "deploymentProfile": "http://www.opengis.net/profiles/eoc/wpsApplication",
  "links": [
    {
      "title": "Current process description.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump:4.
↪4.1",
      "type": "application/json",
      "rel": "self"
    },
    {
      "title": "Process definition.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump",
      "type": "application/json",
      "rel": "process-meta"
    },
    {
      "title": "Process execution endpoint for job submission.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump/
↪execution",
      "type": "application/json",
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/execute"
    },
    {
      "title": "List of registered processes.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes",
      "type": "application/json",
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/processes"
    },
    {
      "title": "List of job executions corresponding to this process.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump/
↪jobs",
      "type": "application/json",
```


(continues on next page)

(continued from previous page)


```
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/job-list"
    },
    {
      "title": "List of processes registered under the service.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes",
      "type": "application/json",
      "rel": "up"
    },
    {
      "title": "Tagged version of this process description.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump:4.
↪4.1",
      "type": "application/json",
      "rel": "working-copy"
    },
    {
      "title": "Most recent revision of this process.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/ncdump",
      "type": "application/json",
      "rel": "latest-version"
    },
    {
      "title": "Listing of all revisions of this process.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird/processes?
↪detail=false&revisions=true&process=ncdump",
      "type": "application/json",
      "rel": "version-history"
    },
    {
      "title": "Provider service description.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird",
      "type": "application/xml",
      "rel": "service"
    },
    {
      "title": "Provider service definition.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/weaver/providers/hummingbird",
      "type": "application/xml",
      "rel": "service-meta"
    },
    {
      "title": "Remote service description.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird?service=WPS&
↪request=GetCapabilities&version=1.0.0",
      "type": "application/xml",
```


(continues on next page)

(continued from previous page)

```
      "rel": "service-desc"
    },
    {
      "title": "Remote process description.",
      "hreflang": "en-CA",
      "href": "https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird?service=WPS&
↪request=DescribeProcess&version=1.0.0&identifier=ncdump",
      "type": "application/xml",
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/process-desc"
    }
  ]
}
```

Monitor execution until completion

Now, we wait until the process completes by periodically verifying the provided status location of the job. The job will be running asynchronously and will be gradually updated with progression and logging details.

Following job submission request, the `status` can be either `accepted` if it is still in queue pending execution, or already be `running`. Once the job completes, the `status` should indicate it was either `succeeded` or `failed`.

```
# NBVAL_IGNORE_OUTPUT
# ignore status updates of job monitoring

print("Waiting for job completion with pooling monitoring of its status...")

# Define a timeout to abandon this monitoring. Process is relatively quick and shouldn't␣
↪last too long.
# The process will be retried if failed to handle possible sporadic errors from the WPS␣
↪remote provider.
# Stops on first maximum timout/retry reached, whichever happens first.
timeout = 60
retries = 10
attempt = retries
delta = 5
body = {}
while timeout >= 0:
    resp = requests.get(status_location, **WEAVER_TEST_REQUEST_XARGS)
    assert (
        resp.status_code == 200
    ), f"Failed retrieving job status at location [{status_location}]"
    body = resp.json()
    timeout -= delta
    status = body["status"]
    if status in ["accepted", "running"]:
        print(f"Delay: {delta}s, Duration: {body['duration']}, Status: {status}")
        time.sleep(delta)
        continue
    if status in ["failed", "succeeded"]:
        print(f"Final job status: [{status}]")
        if status == "failed":
```

(continues on next page)

(continued from previous page)

```
            if attempt > 0:
                attempt -= 1
                retry_msg = f"{retries - attempt}/{retries}"
                print(f"Retrying execution... ({retry_msg})")
                status_location = submit_job()
                print(f"Job Status Location: [{status_location}] (retry: {retry_msg})")
                continue
            else:
                print(f"Final retry attempt reached ({retries}). Aborting.")
        break
    raise ValueError(f"Unhandled job status during monitoring: [{status}]")
assert timeout > 0, "Timeout reached. Process job submission never finished."

# note: don't assert the process success/failure yet, to retrieve more details in case it␣
↪failed
assert body and "status" in body, f"Could not retrieve job status [{status_location}]"
status = body["status"]
```

```
Waiting for job completion with pooling monitoring of its status...
Delay: 5s, Duration: 00:00:00, Status: running
Final job status: [failed]
Retrying execution... (1/10)
Job Status Location: [https://pavics.ouranos.ca/weaver/providers/hummingbird/processes/
↪ncdump/jobs/91b62b44-fb06-4be9-ad2b-43d5265d0048] (retry: 1/10)
Delay: 5s, Duration: 00:00:00, Status: accepted
Final job status: [succeeded]
```

Obtain job execution logs

Retrieve job logs listing execution steps accomplished by Weaver and the underlying process if it provided status messages. During job execution, Weaver attempts to collect any output the original WPS produces and integrates them within its own job logs in order to generate sequential chain of log events by each executed steps.

In case the job `failed` execution, this log will help us identify the cause of the problem. Otherwise, we will have a summary of processing steps.

NOTE:

> Job logs is a feature specific to Weaver that is not necessarily implemented by other implementations of OGC-API - Processes.

```
# NBVAL_IGNORE_OUTPUT
# ignore variable logs values that could easily change, only informative

print("Obtaining job logs from execution...")

path = f"{status_location}/logs"
resp = requests.get(path, **WEAVER_TEST_REQUEST_XARGS)
assert resp.status_code == 200, f"Failed to retrieve job logs [{path}]"
logs = resp.json()

log_lines = "\n".join(logs)
```

(continues on next page)

(continued from previous page)

```
assert len(logs) > 1
assert (
    status == "succeeded"
), f"Job execution was not successful. Status: [{status}]\nFull Logs:\n\n{log_lines}"
assert (
    "100%" in logs[-1] and "succeeded" in logs[-1]
), f"Log entry: [{logs[-1]}]\nFull Logs:\n\n{log_lines}"
print(f"Job logs retrieved from [{path}]:\n\n{log_lines}")
```

```
Obtaining job logs from execution...
Job logs retrieved from [https://pavics.ouranos.ca/weaver/providers/hummingbird/
↪processes/ncdump/jobs/91b62b44-fb06-4be9-ad2b-43d5265d0048/logs]:

[2023-09-01 16:39:07] INFO     [weaver.datatype.Job] 00:00:00   0% accepted   Job task␣
↪submitted for execution.
[2023-09-01 16:39:07] INFO     [weaver.datatype.Job] 00:00:00   0% running    Job␣
↪started.
[2023-09-01 16:39:07] INFO     [weaver.datatype.Job] 00:00:00   0% running    Job task␣
↪setup initiated.
[2023-09-01 16:39:07] INFO     [weaver.datatype.Job] 00:00:00   1% running    Job task␣
↪setup completed.
[2023-09-01 16:39:07] DEBUG    [weaver.datatype.Job] 00:00:00   2% running    Employed␣
↪WPS URL: [https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird]
[2023-09-01 16:39:07] INFO     [weaver.datatype.Job] 00:00:00   2% running    Execute␣
↪WPS request for process [ncdump]
[2023-09-01 16:39:08] INFO     [weaver.datatype.Job] 00:00:00   3% running    Fetching␣
↪job input definitions.
[2023-09-01 16:39:08] INFO     [weaver.datatype.Job] 00:00:00   4% running    Fetching␣
↪job output definitions.
[2023-09-01 16:39:08] INFO     [weaver.datatype.Job] 00:00:00   5% running    Starting␣
↪job process execution.
[2023-09-01 16:39:08] INFO     [weaver.datatype.Job] 00:00:00   5% running    Following␣
↪updates could take a while until the Application Package answers...
[2023-09-01 16:39:12] DEBUG    [weaver.datatype.Job] 00:00:04   6% running    Updated␣
↪job status location: [/data/wps_outputs/weaver/91b62b44-fb06-4be9-ad2b-43d5265d0048.
↪xml].
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04   7% running    Starting␣
↪monitoring of job execution.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04   8% running    [2023-09-
↪01 16:39:08] INFO     [weaver.processes.wps_package|ncdump]   1% running    Preparing␣
↪package logs done.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  10% running    [2023-09-
↪01 16:39:08] INFO     [weaver.processes.wps_package|ncdump]   2% running    Launching␣
↪package...
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  13% running    [2023-09-
↪01 16:39:08] WARNING  [weaver.processes.wps_package|ncdump] Visible application CWL␣
↪euid:egid [0:0]
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  16% running    [2023-09-
↪01 16:39:08] INFO     [cwltool] Resolved '/tmp/tmpw4u4bc2w/ncdump' to 'file:///tmp/
↪tmpw4u4bc2w/ncdump'
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  19% running    [2023-09-
↪01 16:39:09] INFO     [cwltool] ../../../../tmp/tmpw4u4bc2w/ncdump:1:1: Unknown hint␣
```

(continues on next page)

(continued from previous page)

```
↪file:///tmp/tmpw4u4bc2w/WPS1Requirement
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  22% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]   5% running    Loading␣
↪package content done.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  25% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]   6% running    Retrieve␣
↪package inputs done.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  27% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]   8% running    Convert␣
↪package inputs done.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  30% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]   9% running    Checking␣
↪package prerequisites... (operation could take a while depending on requirements)
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  33% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]   9% running    Package␣
↪ready for execution.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  36% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]  10% running    Running␣
↪package...
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  39% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]  10% running    Preparing␣
↪to launch package ncdump.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  42% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump] WPS-1 Package resolved␣
↪from requirement/hint: WPS1Requirement
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  44% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]  11% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - Preparing␣
↪process for remote execution.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  47% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]  14% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - Process␣
↪ready for execute remote process.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  50% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]  18% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - Staging␣
↪inputs for remote execution.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  53% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]  20% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - Preparing␣
↪inputs/outputs for remote execution.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  56% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]  22% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - Executing␣
↪remote process job.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  59% running    [2023-09-
↪01 16:39:09] INFO     [weaver.processes.wps_package|ncdump]  27% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - Monitoring␣
↪remote process job until completion.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  61% running    [2023-09-
↪01 16:39:11] INFO     [weaver.processes.wps_package|ncdump]  27% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - 0% accepted ␣
```

(continues on next page)

(continued from previous page)

```
↪ PyWPS Process ncdump accepted
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  64% running    [2023-09-
↪01 16:39:12] INFO     [weaver.processes.wps_package|ncdump]  82% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - 100%␣
↪succeeded  PyWPS Process NCDump finished
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  67% running    [2023-09-
↪01 16:39:12] INFO     [weaver.processes.wps_package|ncdump]  82% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - Retrieving␣
↪job results definitions.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  70% running    [2023-09-
↪01 16:39:12] INFO     [weaver.processes.wps_package|ncdump]  82% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - Retrieving␣
↪job output definitions from remote WPS-1 provider.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  73% running    [2023-09-
↪01 16:39:12] INFO     [weaver.processes.wps_package|ncdump]  86% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - Staging job␣
↪outputs from remote process.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  76% running    [2023-09-
↪01 16:39:12] INFO     [weaver.processes.wps_package|ncdump]  90% running    [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - Running␣
↪final cleanup operations before completion.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  78% running    [2023-09-
↪01 16:39:12] INFO     [weaver.processes.wps_package|ncdump]  95% succeeded  [provider:␣
↪https://pavics.ouranos.ca/twitcher/ows/proxy/hummingbird, step: ncdump] - Execution of␣
↪remote process execution completed successfully.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  81% running    [2023-09-
↪01 16:39:12] INFO     [weaver.processes.wps_package|ncdump]  95% running    Package␣
↪execution done.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  84% running    [2023-09-
↪01 16:39:12] INFO     [weaver.processes.wps_package|ncdump]  95% running    Nothing␣
↪captured from internal application logs.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  87% running    [2023-09-
↪01 16:39:12] INFO     [weaver.processes.wps_package|ncdump] Resolved WPS output␣
↪[output] as file reference: [/tmp/wps_workdir/weaver/pywps_process__ezcfmkz/nc_dump_
↪2QIB8z.txt]
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  90% running    [2023-09-
↪01 16:39:12] INFO     [weaver.processes.wps_package|ncdump]  98% running    Generate␣
↪package outputs done.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  93% running    [2023-09-
↪01 16:39:12] INFO     [weaver.processes.wps_package|ncdump] 100% succeeded  Package␣
↪complete.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  96% succeeded  Job␣
↪succeeded (status: Package complete.).
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04  98% succeeded  Job␣
↪succeeded.
[2023-09-01 16:39:12] INFO     [weaver.datatype.Job] 00:00:04 100% succeeded  Job task␣
↪complete.
```

Obtain the result location and output the data

When job is succeeded, the result endpoint under the corresponding job will provide the downloadable file references for each of the available output ID defined by the WPS process.

Since the sample NetCDF file provided as input is expected to be converted to raw text data, it can be displayed below.

```
# If execution succeeded, the results endpoint will return 200 with corresponding␣
↪references.
# Otherwise, 400 occurs because results were not produced due to failing job, and␣
↪requesting its outputs is an invalid request.
path = f"{status_location}/results"
resp = requests.get(path, **WEAVER_TEST_REQUEST_XARGS)
assert (
    resp.status_code == 200
), f"Failed to retrieve job results location [{path}]. Code: [{resp.status_code}]."
print("\nJob was successful! Retrieving result location...")
body = resp.json()

# Here, our target output ID is named 'output' according to the process description
output = body.get("output")
assert isinstance(
    output, dict
), f"Could not find result matching ID 'output' within:\n{json_dump(body)}"
href = output["href"]
assert isinstance(href, str) and href.startswith(
    WEAVER_TEST_WPS_OUTPUTS
), f"Output result location does not have expected reference format: [{href}]"
print(f"Result is located at: [{href}]\n")
assert href.endswith(".txt")

print("Fetching output contents...")
resp = requests.get(href)
print(f"\nNCDUMP 'output' result content:\n\n{resp.text}")
```

```
Job was successful! Retrieving result location...
Result is located at: [https://pavics.ouranos.ca/wpsoutputs/weaver/public/91b62b44-fb06-
↪4be9-ad2b-43d5265d0048/output/nc_dump_2QIB8z.txt]

Fetching output contents...

NCDUMP 'output' result content:

netcdf ta_Amon_MRI-CGCM3_decadal1980_r1i1p1_199101-200012.nc {
dimensions:
        time = UNLIMITED ; // (120 currently)
        bnds = 2 ;
        lat = 160 ;
        lon = 320 ;
        plev = 23 ;
variables:
        double time(time) ;
                time:bounds = "time_bnds" ;
```

(continues on next page)

(continued from previous page)

```
                time:units = "days since 1981-01-01" ;
                time:calendar = "standard" ;
                time:axis = "T" ;
                time:long_name = "time" ;
                time:standard_name = "time" ;
        double time_bnds(time, bnds) ;
        double plev(plev) ;
                plev:units = "Pa" ;
                plev:axis = "Z" ;
                plev:positive = "down" ;
                plev:long_name = "pressure" ;
                plev:standard_name = "air_pressure" ;
        double lat(lat) ;
                lat:bounds = "lat_bnds" ;
                lat:units = "degrees_north" ;
                lat:axis = "Y" ;
                lat:long_name = "latitude" ;
                lat:standard_name = "latitude" ;
        double lat_bnds(lat, bnds) ;
        double lon(lon) ;
                lon:bounds = "lon_bnds" ;
                lon:units = "degrees_east" ;
                lon:axis = "X" ;
                lon:long_name = "longitude" ;
                lon:standard_name = "longitude" ;
        double lon_bnds(lon, bnds) ;
        float ta(time, plev, lat, lon) ;
                ta:standard_name = "air_temperature" ;
                ta:long_name = "Air Temperature" ;
                ta:units = "K" ;
                ta:original_name = "T" ;
                ta:cell_methods = "time: mean (interval: 30 minutes)" ;
                ta:cell_measures = "area: areacella" ;
                ta:history = "2011-08-12T05:05:34Z altered by CMOR: replaced missing␣
↪value flag (-9.99e+33) with standard missing value (1e+20)." ;
                ta:missing_value = 1.e+20f ;
                ta:_FillValue = 1.e+20f ;
                ta:associated_files = "baseURL: http://cmip-pcmdi.llnl.gov/CMIP5/
↪dataLocation gridspecFile: gridspec_atmos_fx_MRI-CGCM3_decadal1980_r0i0p0.nc␣
↪areacella: areacella_fx_MRI-CGCM3_decadal1980_r0i0p0.nc" ;

// global attributes:
                :institution = "MRI (Meteorological Research Institute, Tsukuba, Japan)"␣
↪;
                :institute_id = "MRI" ;
                :experiment_id = "decadal1980" ;
                :source = "MRI-CGCM3 2011 atmosphere: GSMUV (gsmuv-110112, TL159L48);␣
↪ocean: MRI.COM3 (MRICOM-3_0-20101116, 1x0.5L51); sea ice: MRI.COM3; land: HAL (HAL_
↪cmip5_v0.31_04); aerosol: MASINGAR-mk2 (masingar_mk2-20110111_0203, TL95L48)" ;
                :model_id = "MRI-CGCM3" ;
                :forcing = "GHG, SA, Oz, LU, Sl, Vl, BC, OC (GHG includes CO2, CH4, N2O,␣
↪CFC-11, CFC-12, and HCFC-22)" ;
```

(continues on next page)

(continued from previous page)

```
                :parent_experiment_id = "N/A" ;
                :parent_experiment_rip = "N/A" ;
                :branch_time = 0. ;
                :contact = "Seiji Yukimoto (yukimoto@mri-jma.go.jp)" ;
                :history = "Output from /sharex3/cmip5/decadal1980/run-C3_decadal1980_
↪01a/grads/atm_avr_mon.ctl 2011-08-12T05:05:34Z CMOR rewrote data to comply with CF␣
↪standards and CMIP5 requirements." ;
                :references = "Model described by Yukimoto et al. (Technical Report of␣
↪the Meteorological Research Institute, 2011, 64, 83pp.)" ;
                :initialization_method = 1 ;
                :physics_version = 1 ;
                :tracking_id = "ce91e727-5f22-44fc-b24d-5bb53393ac69" ;
                :product = "output" ;
                :experiment = "10- or 30-year run initialized in year 1980" ;
                :frequency = "mon" ;
                :creation_date = "2011-08-12T05:05:34Z" ;
                :Conventions = "CF-1.4" ;
                :project_id = "CMIP5" ;
                :table_id = "Table Amon (26 July 2011) 976b7fd1d9e1be31dddd28f5dc79b7a1"␣
↪;
                :title = "MRI-CGCM3 model output prepared for CMIP5 10- or 30-year run␣
↪initialized in year 1980" ;
                :parent_experiment = "N/A" ;
                :modeling_realm = "atmos" ;
                :realization = 1 ;
                :cmor_version = "2.7.1" ;
                :DODS_EXTRA.Unlimited_Dimension = "time" ;
}
```

1.3 Processes

PAVICS offers a growing list of computational services through the Web Processing Services (WPS) standard. Each individual service is hosted by a thematic server. For example, the Finch server specializes in climate indicators, while the Raven server focuses on hydrological modeling and time-series analysis. A typical server will thus host a few dozens or more individual processes that can be combined into workflows. Indeed, each service typically operates on netCDF inputs and also yields netCDF outputs, making it possible to chain together multiple processes.

You'll find below links to services or service providers organized by topics. You may also use the Search bar, which will search through the index for all the main thematic servers.

1.3.1 Basic climate data analysis

The following processes concern basic climate data analysis, methods for formatting data to examine a specific region or time interval.

Spatial subsetting and averaging

Spatial subsetting processed are provided by Finch.

- `SubsetBboxProcess` Subset over a latitude-longitude bounding box.
- `SubsetGridPointProcess` Subset over a number of grid points.
- `SubsetPolygonProcess` Subset over a polygon.
- `AveragePolygonProcess` Average over an area defined by a polygon.

1.3.2 Climate indicators

PAVICS now relies on Finch to provide processes for climate indicators. The full list of available processes can be found here.

Todo: Take a systematic approach and link to other birds and libraries through intersphinx

1.4 Projects using PAVICS

Some Ouranos projects are using PAVICS as a mechanism to distribute results, code or documentation. Here you'll find links to project pages.

1.4.1 Flood Frequency Analysis and Dam Safety in the 21st Century Climate

Version en français

This project completed in March 2021 looked at how climate change information can be integrated into flood design values estimated by frequency analysis. The report includes maps showing the relative change in 1,000 and 10,000 return values for over 500 watersheds over Canada, comparing the 2080-2100 period to 1990-2010. The data underlying these figures is available online as a geospatial layer named `public:decamillenial_flood_CC` on the *Ouranos GeoServer <pavics.ouranos.ca/geoserver>*. This layer can be downloaded locally, streamed by GIS client (see instructions below) or accessed programmatically (see notebook).

For each watershed, the results include the following properties:

name
: Watershed name

watershed_area
: Surface area (km^2)

NSE
: Nash-Sutcliffe Efficiency of calibrated model

<gcm>_PWM_<t>_[<p>_]<kind>
: Climate change factor per unit area (mm/d/km^2) computed following these specifications:

  - `gcm`: Climate model, one of CESM1, CanESM2
  - *Probability Weighted Moments* calibration method
  - `t`: return period, one of 1000, 10000
  - `kind`: Factor operation, either additive (+), or multiplicative (*)

- p (*optional*): percentile from bootstrap parameter uncertainty assessment, one of 0.75, 0.9, 0.95 or 0.99. If not present, the result is the direct estimate from the PWM method on the full sample.

Warning: The ESRI Shapefile format limits field names to 10 characters. The column names above are thus truncated and replaced by an index (e.g. *CanESM2_18*), which complicates parsing. We recommend using the GeoPackage format when downloading the results.

QGIS Client Instructions

These instructions were written for QGIS 3.10

1. Add a layer using the Web Feature Services (WFS) standard *Layer -> Add Layer -> Add WFS Layer ...*
2. Click on *New*
3. Enter the name and address of the PAVICS GeoServer *Name*: PAVICS *URL*: https://pavics.ouranos.ca/geoserver/ows?version=1.1.0&
4. Click *OK*
5. Click *Connect*
6. In table, select `decamillenial_flood_CC`
7. Click *Add*, the layer will be downloaded and should appear in the *Layers* widget.
8. Click *Close*

Once the layer is available, you can access the various columns of the data table and display them.

1. Right click on `decamillenial_flood_CC` layer, select *Properties*
2. In left menu, select *Symbology*
3. In top right menu, select *Graduated*
4. On next line, pick the value to display.
5. Modify as needed the legend format, color ramp and number of classes.
6. Click *Classify*
7. Click *Apply* then *OK*.

1.4.2 Analyse de fréquence des crues et sécurité des barrages dans le climat du 21e siècle

Ce projet complété en mars 2021 s'intéresse à l'intégration des projections climatiques dans l'estimation des crues de conception estimée par analyse fréquentielle. Le *rapport* présente des cartes illustrant le changement relatif des crues de temps de retour 1:1,000 et 1:10,000 pour plus de 500 bassins versants au Canada, comparant la période 2080-2100 à celle de 1990-2010. Les données utilisées pour créer ces cartes sont disponibles sous forme de couches géospatiales (`public:decamillenial_flood_CC`) sur le *GeoServer d'Ouranos <pavics.ouranos.ca/geoserver>*. Ces couches peuvent être téléchargées manuellement, via un client SIG (voir instructions plus bas), ou accédées par une interface de programmation (voir notebook, en anglais).

Pour chaque bassin versant, le tableau de résultats incluent les colonnes suivantes:

name
: Nom du bassin versant

watershed_area
Superficie (km^2)

NSE
Nash-Sutcliffe Efficiency (NSE) du modèle hydrologique

<gcm>_PWM_<t>_[<p>_]<kind>
Facteur de changement climatique de la crue par unité de surface ($mm/d/km^2$) calculé selon les paramètres suivants:

- `gcm`: Modèle de climat, soit CESM1 ou CanESM2
- *Probability Weighted Moments* méthode de calibration des paramètres de la GEV
- `t`: période de retour, soit 1000 ou 10000
- `kind`: Type d'opération à effectuer pour appliquer le facteur de changement, soit une addition (+), ou une multiplication (*)
- `p` (*optional*): percentile estimé par *bootstrap* , l'un de 0.75, 0.9, 0.95 or 0.99. Si *p* est absent, les résultats sont une estimation directe des paramètres par PWM, sans *bootstrap*.

Warning: Le format *shapefile* limite les noms de colonnes à 10 caractères. Les noms de colonnes décrit plus haut sont donc tronqués et remplacés par un nombre (e.g. *CanESM2_18*), ce qui rend les résultats inintelligibles. On recommande utiliser le format *GeoPackage* pour télécharger les résultats.

Instructions pour QGIS

1. Ajouter une couche de type *Web Feature Services* (WFS) *Couche -> Ajouter une couche -> Ajouter une couche WFS* ...
2. Cliquer sur *Nouveau*
3. Entrer le nom et l'adresse URL du Geoserver de PAVICS *Name*: PAVICS *URL*: https://pavics.ouranos.ca/geoserver/ows?version=1.1.0&
4. Cliquer *OK*
5. Cliquer *Connexion*
6. Dans les résultats, selectionner `decamillenial_flood_CC`
7. Cliquer *Ajouter*, la couche devrait être téléchargée et apparaître dans le menu *Couches*.
8. Cliquer *Fermer*

Une fois la couche ajoutée, les différentes colonnes de la table de données peuvent être affichées.

1. Cliquer sur la couche `decamillenial_flood_CC` avec le bouton droit, sélectionner *Propriétés*
2. Dans le menu de gauche, sélectionner *Symbologie*
3. Dans le menu de droite, en haut complètement, sélectionner *Gradué*
4. À la ligne suivante choisissez la variable à illustrer.
5. Choisir au besoin le format de légende, la palette de couleur et le nombre de classes.
6. Cliquer sur *Classer*
7. Cliquer sur *Appliquer*, puis *OK*

Project' related notebooks

Projects' related notebooks

1.4.3 Programmatic access to geospatial layers from the deca-millennial flood project

The relative and absolute changes in 1,000 and 10,000 flood frequency values have been computed for over 500 watersheds using two climate projection large ensembles, CESM1 and CanESM2 (see *project overview* for more info). These results are available on the PAVICS GeoServer, and this notebook shows how to get access to those files from a programming environment.

```
%matplotlib inline

import json

import geopandas as gpd
from owslib.wfs import WebFeatureService

# Connect to Ouranos' GeoServer WFS service.
url = "https://pavics.ouranos.ca/geoserver/wfs"
wfs = WebFeatureService(url, version="2.0.0")

typename = "public:decamillenial_flood_CC"
layer = wfs.contents[typename]
print(layer.abstract)
```

```
Climate change factors for 1,000 and 10,000-year design floods estimated from the CESM1␣
↪and CanESM2 Large Ensembles and the GR4J-Cemaneige hydrological model over the period␣
↪2080-2100 compared to 1990-2010.
```

In the next cell, we'll download the data in the GeoJSON format, load it into a dictionary and instantiate a GeoDataFrame.

```
typename = "public:decamillenial_flood_CC"
txt = wfs.getfeature(typename=typename, outputFormat="json").read()
data = json.loads(txt.decode())
gdf = gpd.GeoDataFrame.from_features(data)
```

Let's take a look at the top of the table.

```
gdf.head()
```

```
                                            geometry       ID  \
0  POLYGON ((-111.39170 54.17100, -111.42500 54.1...  06CD002
1  POLYGON ((-134.94580 59.57930, -134.80000 59.6...  09AA014
2  POLYGON ((-130.19170 59.40850, -130.17920 59.4...  10AC004
3  POLYGON ((-126.10000 58.79600, -126.11250 58.8...  10BE007
4  POLYGON ((-130.50830 58.44600, -130.52500 58.4...  10AC003

                                        name  watershed_area       NSE  \
0            CHURCHILL RIVER ABOVE OTTER RAPIDS        119000.0  0.715358
1       FANTAIL RIVER AT OUTLET OF FANTAIL LAKE           717.0  0.758993
```

(continues on next page)

(continued from previous page)

```
2                      BLUE RIVER NEAR THE MOUTH          1700.0  0.818958
3  TROUT RIVER AT KILOMETRE 783.7 ALASKA HIGHWAY          1170.0  0.846326
4            DEASE RIVER AT OUTLET OF DEASE LAKE          1520.0  0.961293

   CESM1_PWM_T100_MUL  CESM1_PWM_T100_P75_MUL  CESM1_PWM_T100_P90_MUL  \
0          -57.343822              -54.547526              -51.714055
1           27.442331               31.577337               35.676450
2            8.734342               13.139980               17.049680
3            1.782831                6.310975               10.423199
4           12.827155               17.025618               20.555976

   CESM1_PWM_T100_P95_MUL  CESM1_PWM_T100_P99_MUL  ...  CanESM2_PWM_T1000_ADD  \
0              -49.651469              -46.376278  ...              -0.274774
1               38.155664               43.461187  ...               3.627438
2               19.928544               24.816201  ...               4.524804
3               13.551472               17.247709  ...              -0.826701
4               22.540481               26.720056  ...               2.612100

   CanESM2_PWM_T1000_P75_ADD  CanESM2_PWM_T1000_P90_ADD  \
0                  -0.220550                  -0.162439
1                   5.299869                   7.212052
2                   5.183105                   6.138951
3                   0.289358                   1.253341
4                   3.271549                   3.910924

   CanESM2_PWM_T1000_P95_ADD  CanESM2_PWM_T1000_P99_ADD  \
0                  -0.128182                  -0.068190
1                   8.076060                  10.182767
2                   6.809163                   7.634194
3                   1.857471                   2.806194
4                   4.246732                   4.884688

   CanESM2_PWM_T10000_ADD  CanESM2_PWM_T10000_P75_ADD  \
0               -0.138914                   -0.022957
1                1.583382                    5.487793
2                9.541766                   11.049454
3               -2.461317                   -0.095001
4                3.814419                    5.089299

   CanESM2_PWM_T10000_P90_ADD  CanESM2_PWM_T10000_P95_ADD  \
0                    0.136653                    0.224204
1                    9.107839                   11.806705
2                   13.842788                   15.958528
3                    1.757461                    3.090589
4                    6.404652                    7.192629

   CanESM2_PWM_T10000_P99_ADD
0                    0.422348
1                   15.833487
2                   18.720224
3                    4.684330
4                    7.993150
```

(continues on next page)

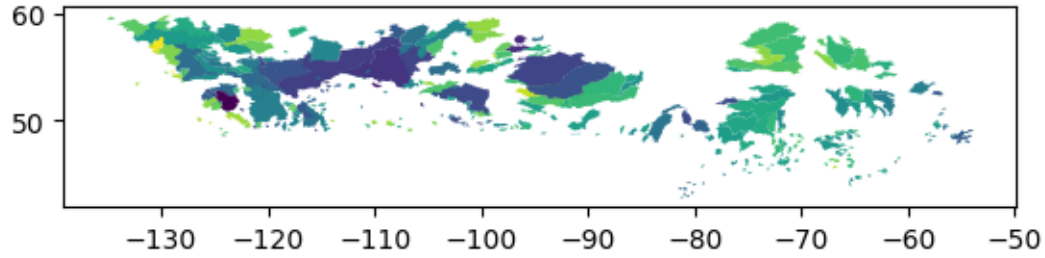
(continued from previous page)

```
[5 rows x 65 columns]
```

And let's create a basic plot for the values of one of the columns, here the relative change factor for the 100-year event estimated from the CanESM2 Large Ensemble, using the Probability Weighted Moment method to estimate GEV parameters.

```
col = "CanESM2_PWM_T100_MUL"
gdf.plot(column=col)
```

```
<Axes: >
```



1.5 Developer Documentation

1.5.1 Installation

Deployment instructions for system administrators are provided in the birdhouse-deploy documentation.

1.5.2 NetCDF data management

The PAVICS architecture relies on the Thematic Real-time Environmental Distributed Data Service (THREDDS) server to distribute netCDF files. This service allows for real-time collection and presentation of archived data and metadata using remote access protocols to bridge the gap between data providers and researchers.

The THREDDS Project is an Open Source initiative maintained by UCAR's Unidata Program. For more information on Unidata, see the Project Home Page. To learn more about THREDDS, view the Project Description on GitHub.

To better understand the way THREDDS integrates within PAVICS, see the System Architecture *Overview*.

Data preparation for inclusion in the platform

NetCDF files integrated in the PAVICS platform must follow the CF Conventions document: http://cfconventions.org/

For variables, the standard_name and units should follow the CF standard name table: http://cfconventions.org/standard-names.html

Adding files

NetCDF files can be added to the THREDDS Data Server by system administrators, simply by copying them to the directory used as a docker volume in `docker-compose.yml`. Contact pavics@ouranos.ca if there are datasets you'd like to see included.

Inspecting metadata

An essential requirement for a functional platform is that netCDF data stored in THREDDS has complete and uniform metadata. Work is in progress to validate metadata fields as part of the cataloguing process.

1.5.3 GeoServer administration

Before you begin

We strongly encourage that you create a `Workspace`. For more information on Workspaces and the data structure of GeoServer, refer to the official GeoServer Online Documentation.

Adding data to GeoServer

There are two possible methods for loading data sets into a store in GeoServer: using SCP/SSH or with QGIS GeoServer Explorer. Both require write access credentials to the GeoServer Administrator panel. Employing SCP/SSH is a more manual method and requires more configuration from within the GeoServer Administration portal while the QGIS GeoServer Explorer can save some time and provide an instantaneous result with less time spent setting layer properties.

The SCP/SSH method

Note: You must have write access permissions to the server-side GeoServer filesystem.

Folders can be loaded with vector and raster data in many formats and can be stored in the same parent folder.

- Begin by tunneling into the server:

```
ssh user@server
```

- Determine where the GeoServer data folder exists on your server. Navigate to this directory and create a new folder that will contain your new data sets:

```
mkdir GeoServer/DATASETS
```

- On your local terminal, navigate to your directory containing your data and run `scp` on the folder recursively:

```
scp -pr localdata user@server:/PATHTO/GeoServer/DATASETS/
```

- Login to the GeoServer Administration Panel and click on *Stores* in the sidebar.
- Click on *Add new Store*
- Specify the type of data you are adding (e.g. Shapefile, GeoTIFF, PostGIS DB, etc). Each option will allow you to load one such file at a time. If you already have a Workspace, you can specify to associate the data with it. If you are adding several Shapefiles, select the option for *Directory of Shapefiles*.

For more information on the Data Adding process from the GeoServer Administration Panel, see the GeoServer documentation.

Note: Click the *Enabled* box or uploaded layers won't be available!

The QGIS GeoServer Explorer method

Note: You must have a working installation of QGIS (2.14.x, 2.18.x) and access to the QGIS Plugin Manager. QGIS is multi-OS and available at QGIS.org.

Note: At the time of this writing, the newest point release of QGIS (3.0.x) does not support the QGIS GeoServer Explorer

- From the QGIS window menu, select *Plugins* then *Manage and Install Plugins*. From the plugin list, find "GeoServer Explorer" and click *Install Plugin*.
- Open your data layers in QGIS and name them accordingly.
- From the *Web* menu tab, select *GeoServer Explorer* and a new window will pop-up or appear below the processing toolbox.

Styling data layers

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

1.5.4 Integration tests

Notebooks found in in this repo are used as integration tests for the platform. That is, each time a change is made in the birdhouse-deploy repository, the notebooks are run in the same Python environment run by the JupyterLab server.

1.5.5 Building the docs

These docs are automatically built by ReadTheDocs, but can also be built locally.

To build the docs, grab a copy of the pavics-sdi repository on github:

```
git clone https://github.com/Ouranosinc/pavics-sdi.git
```

There are requirements (sphinx and a few extensions) that can be installed using *pip*:

```
pip install -r requirements.txt
```

After installing these libraries, you should be able to build the docs without errors:

```
cd pavics-sdi/docs
mkdir source/_static
make html
```

Translations

pavics-sdi is also being translated to French, and it's possible to add other languages. For example to add a German translation, run `sphinx-intl` from the `docs/` directory with the `de` locale:

```
sphinx-intl update -p build/locale -l de
```

This will create a `locale/de/LC_MESSAGES` folder storing `.po` files.

Translators will then be able to edit those `.po` files to translate the documentation content. Once that's done, the documentation can be compiled using:

```
make -e SPHINXOPTS="-D language='de'" html
```

A `make` command to build the french documentation has been created to facilitate building:

```
make html_fr
```

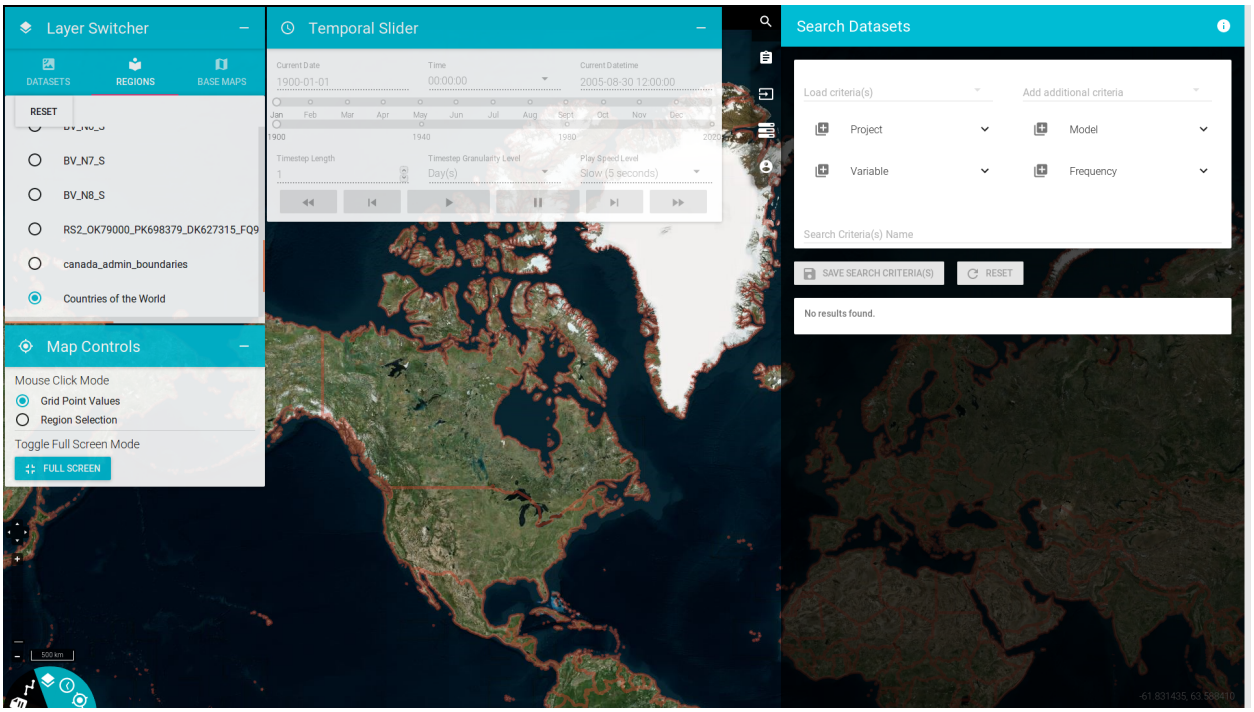
When the source documentation in english changes and the translation needs to be updated, run:

```
sphinx-intl update -p build/locale
```

edit the `.po` files and rebuild the documentation.

1.6 System Architecture

1.6.1 Overview



PAVICS is a Spatial Data Infrastructure (SDI) for climate data. It is composed of modular components that together provide access to data and a library of climate services. It is meant to facilitate climate data analysis for both researchers and climate service providers. PAVICS is not intended to be installed on individual computers, but rather on servers located close to data archives.

There are multiple building blocks composing the PAVICS SDI:

Birdhouse
: Web Processing Services (WPS) supporting data processing in the climate science community. It includes multiple sub-components:

  Birdhouse/Finch
  : A library of climate indicators.

Raven
: A WPS server for the extraction of watershed properties.

JupyterLab
: A notebook interface to demonstrate how WPS services can be used from a programming environment.

Magpie
: Authentication and authorization services.

Weaver
: Workflow Execution Management Service (EMS) and Application, Deployment and Execution Service (ADES) supporting legacy WPS services as well as OGC API - Processes REST bindings.

THREDDS
: netCDF data server.

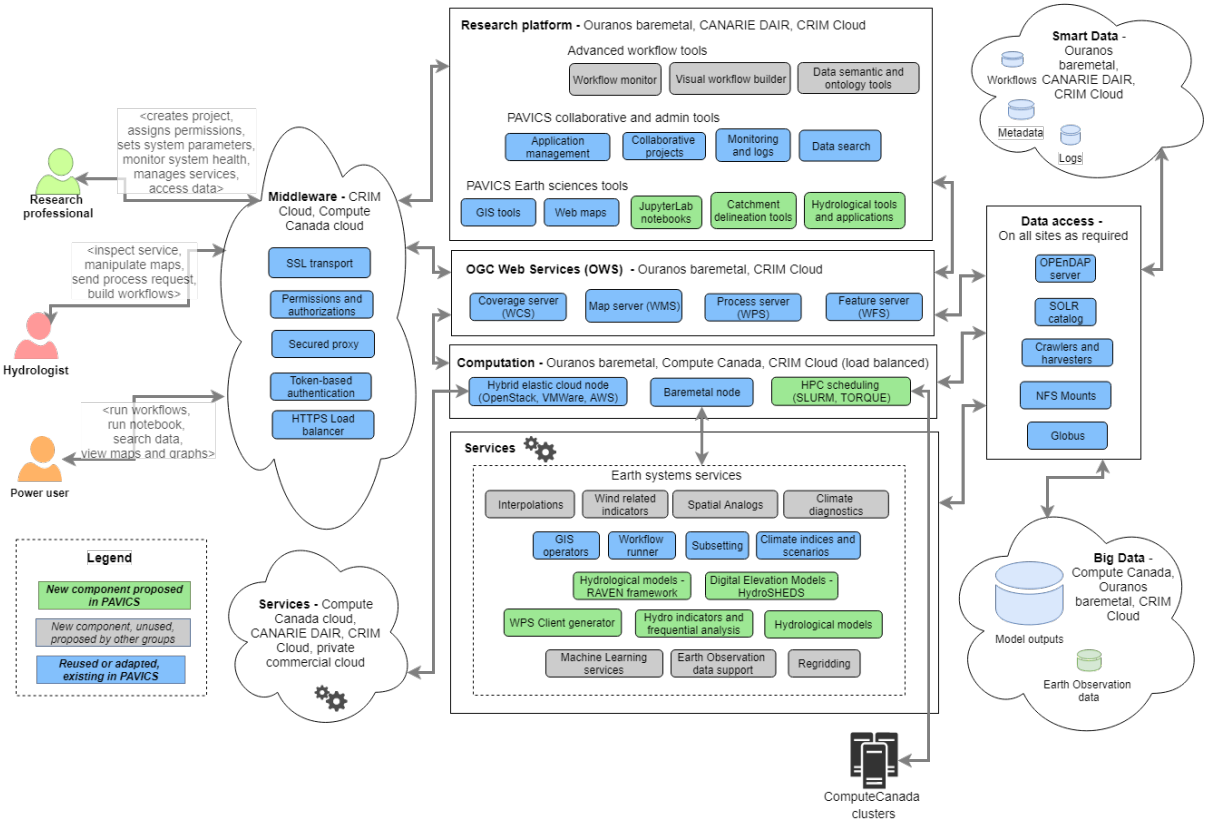
GeoServer
: Geospatial data server.

These components work together to offer users a seamless access to data and a suite of services that can convert raw climate data into useful information, graphics and tables.

Credits

PAVICS is led by Ouranos, a regional climatology research consortium, and CRIM, an informatics and software research institute, (both located in Montreal, Quebec, Canada) to provide climate scientists with a set of tools to acquire and analyze climate data. The project was initially funded by the CANARIE research software program, and has since benefited from contributions from the Open Geospatial Consortium, the Québec Ministry of Environment and Fight Against Climate Change, Environment and Climate Change Canada and the Canadian Foundation for Innovation.

1.6.2 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.

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.

Data Catalog

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. To improve file discoverability, we manage [intake-esm](https://github.com/intake/intake-esm) catalogs for each data category found within the dataset folder of THREDDS:

- biasadjusted
- climex
- cmip5
- forecast
- gridobs
- reanalysis
- stationobs

These catalogs are created by:

- walking through all the aggregated datasets found in the THREDDS `Datasets` folder;
- calling THREDDS NCML service on each dataset. This returns an XML file storing metadata;
- parsing the NCML metadata and creating a catalog description (`json`) and a data table (`csv`).

The resulting catalogs are hosted at https://pavics.ouranos.ca/catalog

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 Finch also includes processes to subset and average gridded data over bounding boxes or polygons.

Raven
: Provides tools for watershed properties extraction for hydrological modeling.

Virtually all individual processes ingest and return netCDF files (or OPeNDAP links for some processes), 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 PAVICS platform is not meant as a front-end, but still provides backend services to facilitate data visualization. It provides for each netCDF file in the THREDDS server a WMS endpoint that can be used to display netCDF fields over maps.

1.6.3 JupyterLab Interface

A JupyterLab instance runs on the PAVICS server at pavics.ouranos.ca/jupyter. The Python3 engine has a number of libraries pre-installed, making it easy to experiment with web processing services and netCDF files.

There is a demo account available for those interested in testing its capabilities. Contact the support email on the login screen to get the password. This demo account has limited computing resources, for security reasons. Any files created using the demo account are visible and modifiable by all users having access to the demo account.

For production usage, without computing resource limitation and with private user workspace, request your own user account using the support email on the login screen.

A number of tutorial notebooks are available in each user workspace to give a sense of how services can be used in scientific workflows. These notebooks are tested daily against the production server and kept up-to-date automatically.

1.7 Provenance

1.7.1 Software

The PAVICS platform relies on a number of independent components, each with its own release schedule. Each component that Ouranos and CRIM maintain is stored in an individual github repository, where every change of the master branch triggers a test suite run. Development occurs in code branches, and modifications are only merged after code reviews have been completed and all tests passed. Official releases are tagged after significant code changes and are authorized by each library's maintainer.

These releases are then deployed by updating the version of the docker image loaded by docker-compose in *birdhouse-deploy*. Relevant changes are described in file `CHANGES.md`. Pull requests trigger a test suite, where documentation and tutorial notebooks are executed against the platform and compare results with expected outputs.

Todo: Review by CRIM.

birdhouse-deploy: https://github.com/bird-house/birdhouse-deploy

1.8 Support

To report bugs, suggest improvements, or point to missing documentation, please post about them on our issue tracker.

If you want to get in touch with a human, email us at pavics@ouranos.ca and we'll be happy to help.

1.9 Release notes

The *pavics-sdi* repository only holds the documentation. Although all the individual components of the PAVICS platform are open-source, the code configuring, deploying and linking these components together is in a private repository. We are planning to remove all sensitive material from this private repository so it can be made public. In the meantime, these release notes are the public facing information tracking the development of the platform.

1.9.1 PAVICS-SDI

1.2.x (2019-06-17)

- Deploy Raven, an hydrological modeling WPS server.
- Moving towards continuous, automated deployment.
- Better config templating.
- Fix for large http requests.

1.1.0 (2019-05-06)

- Adds a JupyterLab interface to the platform. Only a public user is supported.

1.0.0 (2019-03-20)

- Deploy Finch, a library of climate indicators.
- This is a production release, as the backend is considered stable enough to do some actual work.

1.0.0-beta (2017-11-29)

- First official release of the PAVICS Spatial Data Infrastructure system in time for the Canarie demo. Contains all elements from the statement of work, but the frontend does not provide a comfortable user experience. This release is not production ready.

1.10 License

1.11 TODO

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/pavics-sdi/checkouts/latest/docs/source/dev/geoserver.rst, line 69.)

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/pavics-sdi/checkouts/latest/docs/source/processes/index.rst, line 16.)

Todo: Review by CRIM.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/checkouts/latest/docs/source/provenance/index.rst, line 12.)

Todo: Write tutorial on creating and launching workflows

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pavics-sdi/checkouts/latest/docs/source/tutorials/index.rst, line 9.)

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/pavics-sdi/checkouts/latest/docs/source/tutorials/searching.rst, line 7.)

1.12 Indices and tables

- genindex
- modindex
- search