OGC Testbed-16

Data Access and Processing Engineering Report

Publication Date: 2021-01-13

Approval Date: 2020-12-14

Submission Date: 2020-11-20

Reference number of this document: OGC 20-016

Reference URL for this document: http://www.opengis.net/doc/PER/t16-D005

Category: OGC Public Engineering Report

Editor: Panagiotis (Peter) A. Vretanos

Title: OGC Testbed-16: Data Access and Processing Engineering Report

---

OGC Public Engineering Report

Table of Contents

1. Subject 10
2. Executive Summary 11
  2.1. Business statement 11
  2.2. Goals 11
  2.3. Summary of DAPA evaluation 11
  2.4. Data formats 12
  2.5. Future work 13
  2.6. DAPA and Processing in the OGC 13
3. Standard and/or Domain Working Group review 15
  3.1. Overview 15
    3.1.1. EO Exploitation Platform Domain Working Group 15
    3.1.2. Environmental Data Retrieval API SWG 16
    3.1.3. OGC API - Processes SWG 16
    3.1.4. Citizen Scientist DWG 16
4. Document contributor contact points 17
  4.1. Foreword 17
5. References 18
6. Terms and definitions 19
  6.1. Abbreviated terms 20
7. Overview 21
8. Previous work 22
  8.1. Introduction 22
  8.2. APIs 22
    8.2.1. openEO 22
    8.2.2. GeoTrellis 22
    8.2.3. GeoAPI 22
  8.3. OGC APIs 23
  8.4. Data encoding 23
9. Use Cases 24
  9.1. Overview 24
  9.2. General use cases 24
    9.2.1. Use case 1 - Data Retrieval 24
    9.2.2. Use case 2 - Data Processing 25
  9.3. Use-Case 3: API Evaluation 25
  9.4. Participant use cases 26
    9.4.1. Introduction 26
    9.4.2. DLR Use cases 26
      9.4.2.1. Use case 1: Data extraction services for volcano / ozone monitoring 26

9.4.2.1.1. Overview . . . 26
9.4.2.1.2. Use case 1.1 pseudo-api . . . 27
9.4.2.1.3. Use case 1.2 pseudo-api . . . 27
9.4.2.1.4. Use case 1.3 pseudo-api . . . 28
9.4.2.1.5. Use case 1.4 pseudo-api . . . 28
9.4.2.1.6. Examples . . . 29
9.4.2.2. Using case 2: Timescan processor . . . 30
9.4.2.2.1. Overview . . . 30
9.4.2.2.2. Use case 2.1 pseudo-api . . . 31
9.4.2.2.3. Use case 2.2 pseudo-api . . . 31
9.4.2.2.4. Use case 2.3 pseudo-api . . . 31
9.4.2.2.5. Example (pseudo graphic) . . . 32
9.4.2.3. Using case 3: Urban monitoring . . . 32
9.4.2.3.1. Overview . . . 32
9.4.2.3.2. Use case 3.1 pseudo-api . . . 33
9.4.2.3.3. Examples . . . 33
9.4.2.4. Data processing pipeline . . . 35
9.4.3. Terradue Use Cases . . . 36
9.4.4. GMU use cases . . . 38
9.4.4.1. Introduction . . . 38
9.4.4.2. Workflow . . . 39
9.4.5. CRIM Use cases: Gridded climate data . . . 39
9.4.5.1. Dataset selection . . . 39
9.4.5.2. Time and weather . . . 39
9.4.5.3. Location . . . 40
9.4.5.4. Time . . . 40
9.4.5.5. Processing . . . 40
10. Jupyter Notebooks . . . 41
10.1. Overview . . . 41
10.2. Jupyter Notebook 1 (Geomatys - D107) . . . 41
10.2.1. Overview . . . 41
10.2.2. Source code . . . 42
10.2.3. Description . . . 42
10.2.3.1. Operation of Jupyter Notebook 1 . . . 42
10.2.3.2. Lessons learned . . . 42
10.2.4. Model . . . 43
10.2.5. Usage . . . 45
10.2.5.1. Service discovery and loading . . . 45
10.2.5.2. Post-processing . . . 46
10.2.5.3. Multi-language API . . . 48
10.2.6. Limitations . . . 48

10.3. Jupyter Notebook 2 (GeoSolutions - D108) . . . 49
10.3.1. Overview . . . 49
10.3.2. Source . . . 49
10.3.3. URL paths and parameters . . . 49
10.3.4. Outputs . . . 49
10.3.5. Collections . . . 50
10.3.6. Other differences . . . 50
10.4. Jupyter Notebook 3 (Terradue - D109) . . . 51
10.4.1. Overview . . . 51
10.4.2. Source code . . . 51
10.4.3. Description . . . 51
11. DAPA Endpoints . . . 58
11.1. Overview . . . 58
11.2. Resource model . . . 58
11.3. API Endpoint 1 (EOX - D165) . . . 58
11.3.1. Usage . . . 58
11.3.2. Datasets . . . 58
11.3.3. API Documentation . . . 59
11.3.4. Deployment Infrastructure . . . 59
11.3.5. Implementation Details . . . 59
11.4. API Endpoint 2 (interactive instruments - D166) . . . 59
11.4.1. The data . . . 60
11.4.2. The API . . . 60
11.4.3. The API definition . . . 60
11.4.4. General OGC API resources in the API . . . 60
11.4.5. Access to the raw data . . . 61
11.4.6. DAPA overview . . . 64
11.4.6.1. Phase 1: Data access . . . 64
11.4.6.1.1. Spatial sampling geometry . . . 64
11.4.6.1.2. Temporal sampling geometry . . . 65
11.4.6.1.3. Observed properties . . . 65
11.4.6.2. Phase 2: Data aggregation and processing . . . 66
11.4.7. The DAPA landing page: Access information about the available data retrieval patterns . . . 66
11.4.7.1. Request . . . 66
11.4.7.2. Response in JSON . . . 67
11.4.8. Access information about the available variables . . . 69
11.4.8.1. Request . . . 69
11.4.8.2. Response in JSON . . . 69
11.4.9. Retrieve a time series for selected variables at a position . . . 70
11.4.9.1. Sample requests . . . 71

11.4.9.1.1. Using coordinates for the location, requesting data for an instant . . . 71
11.4.9.1.2. Using coordinates for the location, requesting data for an interval (time series) . . . 71
11.4.9.1.3. Using a reference for the location, requesting data for an instant . . . 71
11.4.9.1.4. Using a reference for the location, requesting data for an interval (time series) . . . 71
11.4.9.2. Sample responses . . . 71
11.4.9.2.1. CSV . . . 71
11.4.9.2.2. GeoJSON . . . 72
11.4.9.3. Example use in python . . . 74
11.4.10. Retrieve a time series for selected variables for each station in an area . . . 75
11.4.10.1. Sample requests . . . 76
11.4.10.1.1. Using a bounding box, requesting data for an instant . . . 76
11.4.10.1.2. Using a polygon, requesting data for an instant . . . 76
11.4.10.1.3. Using a reference for the area, requesting data for an instant . . . 76
11.4.10.1.4. Using a reference for the area, requesting data for an interval . . . 76
11.4.10.2. Sample responses . . . 76
11.4.10.2.1. CSV . . . 76
11.4.10.2.2. GeoJSON . . . 79
11.4.11. Retrieve a time series for selected variables for each station in an area and resample the observations to a time series for each cell in a 2D grid . . . 83
11.4.11.1. Sample request . . . 83
11.4.11.2. Sample responses . . . 84
11.4.11.2.1. CSV . . . 84
11.4.11.2.2. GeoTIFF . . . 84
11.4.11.2.3. GeoJSON . . . 85
11.4.12. Retrieve a time series for selected variables at a position and apply functions on the values for each variable . . . 86
11.4.12.1. Sample request . . . 86
11.4.12.2. Sample responses . . . 87
11.4.12.2.1. CSV . . . 87
11.4.12.2.2. GeoJSON . . . 87
11.4.13. Retrieve a time series for selected variables for each station in an area and apply functions on the values of each time series . . . 87
11.4.13.1. Sample request . . . 88
11.4.13.2. Sample responses . . . 88
11.4.13.2.1. CSV . . . 88
11.4.13.2.2. GeoJSON . . . 88
11.4.13.3. Example use in python . . . 89
11.4.14. Retrieve a time series for selected variables for each station in an area and apply functions on the values of each time step . . . 91

11.4.14.1. Sample request . . . 91
11.4.14.2. Sample responses . . . 91
11.4.14.2.1. CSV . . . 91
11.4.14.2.2. GeoJSON . . . 91
11.4.15. Retrieve a time series for selected variables for each station in an area and apply functions on all values . . . 94
11.4.15.1. Sample request . . . 94
11.4.15.2. Sample responses . . . 95
11.4.15.2.1. CSV . . . 95
11.4.15.2.2. GeoJSON . . . 95
11.4.16. Retrieve a time series for selected variables for each station in an area, resample the observations to a time series in a 2D grid and on the values of each time series . . . 96
11.4.16.1. Sample request . . . 96
11.4.16.2. Sample responses . . . 96
11.4.16.2.1. CSV . . . 96
11.4.16.2.2. GeoTIFF . . . 97
11.4.16.2.3. GeoJSON . . . 97
11.5. API Endpoint 3 (terradue - D167) . . . 97
11.5.1. /collections . . . 99
11.5.2. /collections/{collection_id}/variables . . . 99
11.5.3. /collections/{collection_id}/processes/position:retrieve . . . 100
11.5.4. /collections/{collection_id}/processes/position:aggregate-time: . . . 100
11.5.5. /collections/{collection_id}/processes/area:retrieve . . . 101
11.5.6. /collections/{collection_id}/processes/area:aggregate-space . . . 101
11.5.7. /collections/{collection_id}/processes/area:aggregate-time-space . . . 102
11.5.8. /collections/{collection_id}/processes/area:aggregate-time . . . 103
11.5.9. /geothesaurus - location . . . 103
12. Data Formats . . . 105
12.1. Overview . . . 105
13. API Evaluation . . . 106
13.1. Overview . . . 106
13.2. Evaluation workshop . . . 106
13.2.1. Overview . . . 106
13.2.2. Summary agenda . . . 106
13.2.3. Detailed agenda . . . 107
13.2.4. Evaluators . . . 109
13.2.5. Evaluation Questions . . . 109
13.2.6. EO Applications Lifecycle Diagram . . . 110
13.3. Evaluation of Expert 1 (EOX - D110) . . . 110
13.3.1. Introduction . . . 110
13.3.2. Evaluation . . . 111

13.3.3. Feedback responses . . . 111
13.3.4. Summary and Recommendations . . . 114
13.4. Evaluation of Expert 2 (GMU - D111) . . . 114
13.4.1. Introduction . . . 114
13.4.2. Evaluation . . . 115
13.4.3. The R Notebook . . . 116
13.4.3.1. About R notebooks . . . 116
13.4.3.2. Introduction . . . 116
13.4.3.3. DAPA Benefits and Further Considerations . . . 116
13.5. Evaluation of Expert 3 (CRIM - D112) . . . 116
13.5.1. Introduction . . . 116
13.5.2. Evaluation . . . 117
13.5.2.1. Evaluation use case . . . 117
13.5.2.2. Evaluation along the proposed axes . . . 117
13.5.2.3. Learning curve . . . 117
13.5.2.4. Richness of accessible functionality . . . 117
13.5.2.5. Amount of code needed to execute some common analyses . . . 118
13.5.2.6. Recommendations . . . 118
13.5.3. Evaluation questions from the participants . . . 119
13.5.4. Conclusion . . . 120
Appendix A: Comparing DAPA and ADES . . . 121
A.1. Overview . . . 121
A.2. Discovery . . . 122
A.3. Process description . . . 122
A.4. Encoding . . . 122
A.5. Job Control . . . 122
A.6. Invocation . . . 123
A.7. Responses . . . 123
A.8. Examples . . . 123
A.9. Process `position` . . . 123
A.10. Process `area` . . . 131
A.11. A KVP-encoding for execute ADES processes . . . 143
A.12. Execution endpoint . . . 143
A.13. Basic form of the URL . . . 144
A.14. Process Inputs (<input value>) . . . 144
A.15. Process outputs (<output reference>) . . . 145
A.16. Examples . . . 146
Appendix B: Revision History . . . 148

Chapter 1. Subject

This OGC Testbed-16 Engineering Report (ER) describes the work performed in the `Data Access and Processing API` (DAPA) thread.

The primary goal of the DAPA thread is to develop methods and apparatus that simplify access to, processing of, and exchange of environmental and Earth Observation (EO) data from an end-user perspective. This ER presents:

- The use cases participants proposed to guide the development of the client and server components deployed during the testbed.
- An abstract description of a resource model that binds a specific function to specific data and also provides a means of expressing valid combinations of data and processes.
- A description of each DAPA endpoint developed and deployed during the testbed.
- A description of the client components that interact with the deployed DAPA endpoints.
- End-user (i.e. data scientist) feedback concerning the ease-of-use of the

NOTE This ER does not cover the specific details of the Data Access and Processing API. A complete description of the API can be found in the Data Access and Processing API Engineering Report [http://fix.this.link].

Chapter 2. Executive Summary

2.1. Business statement

2.2. Goals

In the past, the provider-centric view has defined data retrieval mechanisms for geospatial content. This view has resulted in a collection of "stove-pipe" API specifications organized around resource types that, from an end-user perspective, are not well integrated or easy to invoke. For example, OGC defines a Features API in the OGC API - Feature [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html] standard for accessing vector feature data and a Processes API in the OGC API - Processes [https://htmlpreview.github.io/?https://github.com/opengeospatial/wps-rest-binding/blob/master/docs/18-062.html] candidate standard for invoking geo-processes over the Web. Deployed by a single provider, however, which processes can operate on which collection of data is not obvious. Furthermore, invocation of processes on the data is not straight forward or user-friendly. Invoking a process through the Processes [https://htmlpreview.github.io/?https://github.com/opengeospatial/wps-rest-binding/blob/master/docs/18-062.html] API requires the creation of a JSON-encoded execution request which is then sent to an execution endpoint using the HTTP POST method.

By contrast, the DAPA developed in this thread tightly couples a process to the data collection(s) upon which it can operate. The DAPA also uses a simple invocation pattern that makes use of the HTTP GET method with query parameters. This is familiar to anyone who has followed a link in a web page. As a result, a DAPA request can be easily embedded in many environments including a Jupyter notebook or within HTML pages that can be dynamically generated to from DAPA endpoints as just another output format (i.e. text/html). HTML pages provide a convenient way for end-uses to navigate DAPA resources and invoke processes using just a web browser. In the DAPA thread both environments, Jupyter notebooks and HTML pages were used to test and evaluate the DAPA endpoints provided by participants.

2.3. Summary of DAPA evaluation

In order to gauge the ease-of-use of the API, a multi-day workshop was organized involving earth observation and data scientists familiar with the use of Jupyter notebooks but not with DAPA. The purpose of the workshop was to have the scientists use the DAPA and then provide evaluation reports. The primary evaluation criteria were:

- Learning curve.
- Richness of functionality.
- Ease of use (i.e. how much code is required to make a DAPA request)

The evaluation reports are included in the API evaluation clause of this ER and the main points are summarized here:

- The DAPA succeeded in satisfying all the primary evaluation criteria:
  - Simple to learn.

    - Provided a reasonable set of functionality.
    - Simple to use and embed into a Jupyter notebook.
- The evaluations also described a number of recommendations to enrich the functionality of the API including:
    - Provide an easy-to-use interface to learn how to use the API.
    - Richer metadata via links to describe aspects of the API such as query parameters, metadata about the associated collection, example invocations, etc.
    - Add up/downsampling capability.
    - Polygon filtering rather than just BBOX.
    - Some means to combine different data sets in one request.
    - GDAL-compatible data formats and direct access via GDAL Virtual File Systems.
    - More user-friendly output formats (CSV, PNG, etc.).
    - Data streaming especially binary for raster outputs.
    - HTTP POST request for parameterizable queries
    - A complete, harmonized API description in OpenAPI to allow interactive clients and Jupyter widgets to be created.

2.4. Data formats

The work by the participants in this thread focused primarily on the development of the DAPA with respect to the use cases articulated by participants and described in the Participant use cases section of this ER. Due to time and resource constraints, less attention was paid to performing an analysis of output data formats. In order to get a sense of the data formats used in the DAPA thread, the following table presents, based on the type of data, what output formats the participants' DAPA endpoints generated.

Type of data desired	Output Formats
Coverage	GeoTIFF [http://docs.opengeospatial.org/is/19-008r4/19-008r4.html]
Imagery	GeoTIFF [http://docs.opengeospatial.org/is/19-008r4/19-008r4.html]
Vector Features	GeoJSON [https://tools.ietf.org/html/rfc7946]
Tabular Data	CSV [https://tools.ietf.org/html/rfc4180] or XARRAY [https://xarray.pydata.org/en/stable/data-structures.html] as JSON [https://tools.ietf.org/html/rfc7159]
Scalar Values	Plain text or JSON [https://tools.ietf.org/html/rfc7159]

2.5. Future work

Below are listed possible future work items that might be considered in relation to the DAPA:

- Consider adding the ability to negotiate output formats.
- Consider defining a set of possible output data type structures (e.g. data cube, coverage, time server, multi-dimensional data set, feature or simple value).
    - Define standard encodings for each output data type structure identified (e.g. GeoTIFF, GeoJSON).
    - If they don't already exist, create MIME type or MIME type templates that capture the valid combinations of data structure and encoding format(s).
- Consider adding query parameters to the API that support:
    - Spatial and temporal (aggregation) resolution on which the processing should be executed,
    - Pre-filtering of collection items that go into the processing function,
    - On-the-fly transformation of attributes (e.g. `&fields=NDVI=(B04-B08)/(B04+B08),NDBI=(B01-B02)/(B01+B02)`),
    - Cross-reference of values from other collections (e.g. `&fields=(NDVI=(B04-B08)/(B04+B08))*(external_collection:CLOUD_MASK)`).
- Explore adding support for HTTP POST for invoking DATA processes.
- Considering adding an up/downsampling capability of data to control the volume of data being processes (see similar feature in Google Earth Engine).
- Investigate DAPA extensibility.
    - More broadly, consider convergence of DAPA with OGC API - Processes / ADES (see Comparing ADES and DAPA).
- Enhanced and interactive documentation about all aspects of a DAPA deployment (e.g. paths, parameters, test queries, etc.).
- The ability to combine different datasets in one request (e.g. fusion of Sentinel-1 and Sentinel-2 for a specific point in time)
- Consider borrowing some capabilities from Google Earth Engine.

2.6. DAPA and Processing in the OGC

Processing within the OGC can be viewed as a spectrum where at one end there is a set of specialized processes deemed to be generally important for the geo-spatial community and at the other end there is the set of geo-processing modules that needs to be deployed and executed on the Web.

Mapping and routing are examples of generally important processes that are distinguished within the OGC by having dedicated APIs defined those processes.

At the other end of the spectrum, the OGC has defined a Processes API, the OGC API - Processes [https://htmlpreview.github.io/?https://github.com/opengeospatial/wps-rest-binding/blob/master/docs/18-062.html]

API, that enables any process to be deployed on the Web and invoked in a standardized way with standardized job control for long-running processes and standardized interfaces for retrieving processing results.

Between these two end points exists a set of processing requirements that satisfy the needs of specific communities of interest such as meteorology, pollution monitoring, and so forth. There are important considerations such as providing a rich, integrated functionally but also being easy to use. DAPA sits in this middle ground and provides APIs that encapsulate the set of functionalities required by the community by specifying an easy-to-learn and easy-to-use API. In other words, an API designed from the perspective of the end-user of the community of interest and not the data provider.

Chapter 3. Standard and/or Domain Working Group review

3.1. Overview

The task participants believed that the work of the Data Access and Process API task of the OGC Testbed-16 is relevant to work being done in the OGC SWGs/DWGs listed below. A request for review of this ER by the SWG/DWG members was forwarded by the editor.

3.1.1. EO Exploitation Platform Domain Working Group

The purpose of the EO Exploitation Platform Working Group is to foster interoperability (i.e. the pre-requisites for successful interaction between platform components and across platforms) among Exploitation Platforms and their components. To this end, the working group will act as an open form for discussion and documentation of interoperability requirements for the domain in order to drive OGC standards evolution towards better support for the use cases of the EO Exploitation Platforms.

The increasingly huge amount of new EO satellite data and In-Situ data every day has incentivized the creation of several web accessible platforms that enables scientists and commercial operators to use such data without the need to download content and have in-house Information Technology infrastructure to manage the large volume of data.

EO Exploitation platforms have been independently developed by public organization or commercial companies, but all share a common set of functionalities:

- Cataloguing and searching;
- Storage and access;
- Visualization;
- Data processing and analysis; and
- User Authentication, Authorization, and Accounting.

These platforms, however, have many different implementations, with interfaces and data formats which often hampers interoperability among platforms. In fact, the next step in this data exploitation revolution is to link the platforms together creating an ecosystem. This ecosystem is potentially across different administrative domains, in which each platform can contribute with the offered services to the implementation of more complex use cases.

The work in this Testbed 16 thread focused on the data process and analysis aspects of EO exploitation platforms. The particpants believe that the outcomes of this testbed would be of interest to users of EO exploitation platforms. Specifically the work of the testbed is to develop data access and processing interfaces that are simpler to use and thus simpler to learn and more easily integrated into the kinds of tools that EO exploitation platform users might use (e.g. using Jupyter notebooks to perform some analysis).

3.1.2. Environmental Data Retrieval API SWG

The purpose of the Environmental Data Retrieval (EDR) API SWG is to standardize several APIs, defined using OpenAPI (Version 3), to retrieve various common data patterns from a relatively persistent data store. The data patterns could include, but are not restricted to, data at a point in space and time, time series at a point, data along a trajectory, which may be 2, 3, or 4 dimensional, and covering a specified polygon or rectangular tile. The APIs will enable service users to retrieve resources over a discrete sampling geometry, created by the service in response to a standardized query pattern.

The work of the Testbed-16 Data Access and Processing task was closely aligned with the stated goals of the SWG. The participants believe that the outcomes of this testbed task could help inform the design of the API components that the SWG is endeavouring to specify.

3.1.3. OGC API - Processes SWG

The purpose of the OGC API - Processes SWG is to design a Web API that enables the execution of computing processes and the retrieval of metadata describing their purpose and functionality. Typically, these processes combine raster, vector, coverage and/or point cloud data with well-defined algorithms to produce new raster, vector, coverage and/or point cloud information.

In the current design of the Processes API, executing a process involves creating a JSON document that contains the process inputs and POSTing that document to the execution endpoint of the process. From an end-user perspective, this workflow is more complicated than may be necessary. Since one of the goals of the testbed task was to design a simpler API for invoking processes, the participants believe that this work could inform the design of an alternate invocation of WPS processes using KVP/GET method rather than the current JSON/POST method.

Furthermore, the work being done in Testbed-16 for binding processes to specific datasets and to advertise which combinations of data and processes are valid would be of interest to the SWG.

3.1.4. Citizen Scientist DWG

There are a large and increasing number of citizen science projects active around the world involving the public in environmental monitoring and other scientific research activities. The OGC Citizen Science DWG is motivated to support citizen science by providing a forum for increasing understanding and demonstration of the benefits brought by the use of open standards and best practices. This DWG will support the development of improved interoperability arrangements for the citizen science community.

The work of the testbed to simplify the interfaces for accessing and processing data from the end-user perspective is directly related to the goal of the DWG to improve interoperability arrangements for the citizen science community.

Chapter 4. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Contacts

Name	Organization	Role
Fabrice Brito	Terradue	Contributor
Jonas Eberle	DLR	Contributor
Pedro Gonçalves	Terradue	Contributor
Torsten Heinen	DLR	Contributor
David Landry	CRIM	Contributor
Clemens Portele	interactive instruments	Contributor
Panagiotis (Peter) A. Vretanos	CubeWerx Inc.	Editor

4.1. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Chapter 5. References

The following normative documents are referenced in this document.

- OGC: OGC 17-069r3, OGC® API - Features - Part 1: Core Standard version 1.0, 2019 [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html]
- OGC: OGC 19-008r4, OGC GeoTIFF Standard version 1.1, 2019 [http://docs.opengeospatial.org/is/19-008r4/19-008r4.html]
- OGC: OGC 10-090r3, OGC Network Common Data Form (NetCDF) Core Encoding Standard version 1.0, 2011 [http://portal.opengeospatial.org/files/?artifact_id=43732]
- Ecma International: ECMA-404, The Json Data Interchange Syntax [https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf]
- OGC: OGC 19-072, OGC® API - Common - Part 1: Core (draft) [http://docs.opengeospatial.org/DRAFTS/20-024.html]
- OGC: OGC 18-062, OGC API - Processes - Part 1: Core (draft) [https://htmlpreview.github.io/?https://github.com/opengeospatial/wps-rest-binding/blob/master/docs/18-062.html]
- OGC: OGC 20-044, OGC API - Processes - Part 2: Transactions (draft) [https://github.com/opengeospatial/wps-rest-binding/blob/master/extensions/transactions/standard/20-044.adoc]
- OGC: OGC 19-086r2, OGC API - Environmental Data Retrieval Standard (draft) [http://docs.opengeospatial.org/DRAFTS/19-086.html]

Chapter 6. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] and the clause titled "Terms and Definitions" of OGC 19-072 [http://docs.opengeospatial.org/DRAFTS/19-072.html] shall apply. In addition, the following terms and definitions apply.

- **area**

  region specified with a geographic envelope that may have a vertical dimension

- **collection**

  a body of resources that belong or are used together; an aggregate, set, or group of related resources

- **dataset**

  collection of data, published or curated by a single agent, and available for access or download in one or more serializations or formats

- **endpoint**

  the specific digital location where requests for information are sent to retrieve the digital resource that exists there

- **input | argument | query parameter**

  refers to data provided to a process; a process input is an identifiable item

- **LiDAR**

  Light Detection and Ranging — a common method for acquiring point clouds through aerial, terrestrial, and mobile acquisition methods.

- **location**

  identifiable geographic place

- **notebook**

  notebook is an electronic file that runs in a web browser and contains both programming code and text descriptions

- **output**

  data returned as a result of applying a process to a collection

- **position**

  a place specified with a geographic point

- **process**

  a function that for each input returns a corresponding output; within this engineering report inputs are also referred to as process arguments or query parameters

- **variable | field | observed property**

  a variable is the named representation of a property that has been observed; for example precipitation or temperature

6.1. Abbreviated terms

- ADES Application Deployment and Execution Service
- API Application Programming Interface
- CQL Common Query Language
- CRS Coordinate Reference System
- CSV Comma Separated Values
- DAPA Data Access and Processing API
- IDE Integrated Development Environment
- WKT Well Known Text

Chapter 7. Overview

The Previous work section discusses several developments and existing best practices for both APIs and data encodings that informed the work does in this thread.

The Use cases section discusses the use cases used to guide the design and implementation of the components developed for the DAPA thread.

The Jupyter notebooks section provides a brief introduction about Jupyter notebook technology and a detailed description of the notebooks developed in this thread.

The API endpoints section describes the DAPA endpoints developed by each of the participants. This section does not describe the API itself which is described in the OGC Testbed-16: Data Access and Processing API ER [http://fix.me].

The Data formats section provides a survey of which data formats work best in which situations of data retrieval informed by the different scenarios and user groups addressed in this ER.

The API evaluation section presents the end-user evaluations of the API gather during the X-day Evaluation workshop.

Chapter 8. Previous work

8.1. Introduction

The Data Access and Processing API development takes into account several developments and existing best practices for both APIs and data encodings.

8.2. APIs

8.2.1. openEO

The volume of Earth Observation data has grown so large that moving such data to a local machine for processing is no longer feasible. Instead of moving the data to local processing, the trend is now to store large repositories of Earth Observation data in the cloud or compute back-ends, and move the processing software to the data. The results of this processing on the cloud can then be browsed remotely or downloaded as desired.

In order to enable this "move the process to the data" concept, the openEO organization developed an API that connects clients such as R, Python and JavaScript to big Earth observation cloud back-ends in a simple and unified way. The main objectives of the project are the following concepts:

- Simplicity for clients (rather than data providers)
  - Many end-users use Python or R to analyze data and JavaScript to develop web applications. Analyzing large amounts of EO imagery should be equally simple, and seamlessly integrate with existing workflows.
- Unification
  - A common API makes it easier to validate and reproduce processing results across cloud back-ends.
  - A common API makes it easier to provision processing across in a coordinated way different back ends.
  - A common API makes it easier to compare cloud back-ends in term of capability and costs.

8.2.2. GeoTrellis

GeoTrellis is a geographic data processing engine for high performance applications. It is implemented as a Scala [https://scala-lang.org/] library and framework that uses Apache Spark [https://spark.apache.org/] to work with raster data and supports many Map Algebra [https://en.wikipedia.org/wiki/Map_algebra] operations as well as vector to raster or raster to vector operations.

8.2.3. GeoAPI

The OGC GeoAPI Implementation Standard defines the GeoAPI library. GeoAPI provides a set programming interfaces for geospatial applications. In a series of packages or modules, GeoAPI 3.0 defines interfaces for metadata handling and for geodetic referencing (map projections). The

GeoAPI interfaces closely follow the abstract models published collaboratively by ISO in its 19100 series of documents and the OGC in its abstract and implementation specifications. GeoAPI provides an interpretation and adaptation of these standards to match the expectations of Java or Python programmers. C

8.3. OGC APIs

These APIs are complemented by a set of emerging OGC API standards to handle geospatial data and processes. The OGC API family of (mostly emerging) standards is organized by resource type. So far, OGC API - Features has been released as a standard that specifies the fundamental API building blocks for interacting with features. The spatial data community uses the term 'feature' for things in the real world that are of interest. OGC API standards define modular API building blocks to spatially enable Web APIs in a consistent way. The OpenAPI specification is used to define and document the API building blocks.

8.4. Data encoding

On the data encoding side, there are several existing standards that are frequently being used for Earth observation, environmental, ecological, or climate data. These include NetCDF, GeoTIFF, HDF, GML/Observation and Measurements or variations thereof, or increasingly the use of JSON encoded data. Testbed-16 explored existing solutions as well as emerging specifications and provide recommendations with focus on the end-user, i.e. data or earth scientist.

The OGC-ESIP Coverage and Processing API Sprint at the ESIP Winter meeting in January 2020 performed an analysis on coverages beyond the current OGC WCS standard’s capabilities. This effort took into account various elements that needed to be developed for an API approach based on the abstract specifications for Coverages and Processing as well as OPeNDAP, GeoXarray/Zarr, R-spatial and other modern software development environments. The Geospatial Coverages Data Cube Community Practice document describes community practices for Geospatial Coverage Data Cubes as implemented by multiple communities and running as operational systems.

Chapter 9. Use Cases

9.1. Overview

This clause describes the use cases that guided the design and implementation of the components developed for the Testbed 16 DAPA task.

This chapter is organized into two sections. The first section describes the general use cases from the end-user perspective. Based on the general use case, the second section describes the Testbed participant use cases that guided the implementations of the DAPA endpoints.

9.2. General use cases

9.2.1. Use case 1 - Data Retrieval

For this use case the typical user is a developer.

The user wants to access geospatial data for a specific area in a simple function call. The function call identifies the data and allows the user to define the discrete sampling geometry. Valid geometries include:

- Point locations (x, y, and optional z),
- Bounding-box
- Polygon

All geometries are provided either in-line or by reference, as illustrated by the following examples:

- In-line: https://web.api/collections/temperature?location=50.11159,8.68248
- By reference: https://web.api/collections/temperature?location=https://other.api/collections/cities/items?name=Frankfurt

Specifying sampling geometries by reference also supports the use of an OGC API - Feature [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html] endpoint as shown in the example above where the value of the `location` parameter is an OGC API - Feature invocation to fetch the feature, and thus the geometry, of the city for Frankfurt.

Users need the ability to access the original data. The use of the term "original" is a little ambiguous in this context as data often undergoes some process on its way from original access to the final product. As an example, imagine a digital temperature sensor. The actual reading performed in the sensor is some form of electricity, but the value provided at the sensor interface is 21°Celsius (approximately 69.8°Fahrenheit). Thus, some form of calibration curve has been applied to the original reading, which might not be available at all. In this case, the value 21°Celsius can be considered as “original”. The same principles apply to satellite data. The original raw data readings are often not accessible. Instead, the data undergoes some correction process before being made available. Higher product levels may include orthorectification or re-gridding processes. In any case, data providers need to provide a description of the performed processing together with the actual data. In addition, data should be available as "raw" as possible.

End-users want to retrieve all data that exists within the provided target geometry. In the case of polygon geometries, the end-user would receive all data that intersects that polygon. In the case of point geometries, the end-user would retrieve the value exactly at that point.

In addition, end-users have the option to define the (interpolation) method for value generation. If no option is selected, the Web API indicates how a given value was produced. Testbed-16 participants developed a set of frequently used production options, including for example:

- “original value”
- “interpolation method”
- “re-gridding”

or any combination thereof. This use case differentiates the following data requests:

- Synopsis/Time-Averaged Map: The end-user wants to retrieve data for a single point in time or as an average value over a time period. The figure below is an example of visualized time-averaged data for a number of sampling locations.
- Area-Averaged Time Series: The end-user wants to retrieve a single value that averages all data in the target geometry for each time step. The figure below is an example of visualized area-averaged data for a number of time steps.
- Time Series: The end-user wants to retrieve the full time series for each data point. The figure below is an example of visualized full time series data set that includes a number of time steps.

Testbed-16 participants explored these use-cases in combination with additional processing steps. For example, the end-user requests synoptic, map, or time series data, that is interpolated to a grid.

9.2.2. Use case 2 - Data Processing

Testbed-16 participants explored simple data processing functions. These include calculations for:

- The minimum value in a result set.
- The maximum value in a result set.
- Average value in a result set.

These values are for any given data retrieval subset as accessible in the Data Retrieval use cases.

9.3. Use-Case 3: API Evaluation

The third use-case is orthogonal to the first two. This use case does not add any additional requirements on the API itself, but evaluates the API from an end-user point of view.

This third use case was implemented during a full-day workshop with several data and earth scientists who were invited to evaluate the DAPA API regarding:

- Learning curve to use the API.
- Richness of accessible functionality.
- Amount of code needed to execute some common analyses.

The goal of the workshop was to allow the API developers and endpoint providers to further refine the API and increase ease-of-use based on the feedback provided by the scientists.

9.4. Participant use cases

9.4.1. Introduction

This section describes the detailed use cases articulated by the Testbed 16 participants, based on the general use cases, that guided the development of thread clients and end-points.

9.4.2. DLR Use cases

9.4.2.1. Use case 1: Data extraction services for volcano / ozone monitoring

9.4.2.1.1. Overview

As a service provider, the user want to retrieve the { average | max } value of { Sulphur dioxide | ozone } in a specific region { volcano | pole } over a specific time period { yesterday | last week } in order to determine if the value is higher than a threshold. If the value is higher, a notification is sent and the creation of a { animation | chart } of the { particle dissemination | ozone concentration) is triggered for each successive day until the value falls below the threshold.

- Use case inputs:
  - Sentinel-5p products (e.g., L3 Sulphur dioxide, L3 Ozone).
- Expected outputs:
  - Datacube: Ozone concentration for each time instant.
  - Animation: Ozone concentration for each time instant color-coded and encoded video.
  - Coverage: Sulphur dioxide / ozone concentration for time period(s).
  - Chart: Timeseries of ozone concentration over bbox|point|polygon.
  - Feature: Timeseries of ozone concentration over bbox|point|polygon.
- Internal data processing
  - None, only temporal and spatial aggregation of data.
- Requirements for the API
  - Select collection (e.g., Sentinel-5p L3 Ozone).
  - Filter on area of interest (e.g., Frankfurt area).
  - Filter on time range to be processed (e.g., “last week”).
  - Define interval for temporal aggregation (e.g., none, all data, weekly).
  - Define method for temporal aggregation (e.g., mean, min, max, sd).
  - For Chart: Define method for spatial aggregation (e.g., mean).

9.4.2.1.2. Use case 1.1 pseudo-api

Retrieve Sulphur dioxide concentration over area g1 in the last month as datacube (x,y,z,t) encoded as CoverageJSON [https://covjson.org/spec/]. CoverageJSON, is a data format for describing "coverage" data in JSON. The primary intended purpose of the format is to enable data transfer between servers and web browsers, to support the development of interactive, data-driven web applications.

```
web.api
    collection=S5P
    properties=SO2
    subset=(<GEO>, BBOX(g1))
    subset=(<TIME>, TODAY-P1M/TODAY)
        process=NONE
    output_type=DATACUBE
    output_format=CoverageJSON

==>

{
  Type: Coverage
  Domain:
    type: Grid
    axes: x,y,z,t
  Parameters:
    sulphur_dioxide
  Ranges:
    axisNames: [x,y,z,t]
    sulphur_dioxide: [values]
}
```

9.4.2.1.3. Use case 1.2 pseudo-api

Retrieve Sulphur dioxide concentration over area g1 on a specific day as a coverage encoded as GeoTIFF.

```
web.api
    collection=S5P
    properties=SO2
    subset=(<GEO>, BBOX(g1))
    subset=(<TIME>, TODAY)
        process=( <SO2>, max(<TIME>) )
    output_type=COVERAGE
    output_format=geotiff

==>

PROJ: geo
Meta:
 Time=TODAY
O3: values
```

9.4.2.1.4. Use case 1.3 pseudo-api

Retrieve maximum Sulphur dioxide concentration over area g1 in the last month as timeseries table encoded in csv.

```
web.api
    collection=S5P
    properties=SO2
    subset=(<GEO>, BBOX(g1)) | (<GEO>, POLYGON(local:POI:anak-krakatau)
    subset=(<TIME>, TODAY-P1M/TODAY)
    process=(<SO2>, max(<GEO>) )
    output_type=TIMESERIES
    output_format=csv

==>

TIME,SO2
2000-01-01,2
2000-01-02,3
2000-01-04,10
...
```

9.4.2.1.5. Use case 1.4 pseudo-api

Retrieve maximum Sulphur dioxide concentration over area g1/ref:volcano at t1=2000-01-01 as simple value encoded in a simple GeoJSON feature.

```
web.api
    collection=S5P
    properties=SO2
    subset=(<GEO>, BBOX(g1)) | (<GEO>, POLYGON(http://external/anak_krakau))
    subset=(<TIME>, 2000-01-01)
    process=(<SO2>, max(<GEO>)
    process=(<SO2>, max(<TIME>)
    output_type=FEATURE
    output_format=GeoJSON

==>

{
  feature.1: {poly: [anak_krakau], time: 2000-01-01, SO2: value }
}
```

9.4.2.1.6. Examples

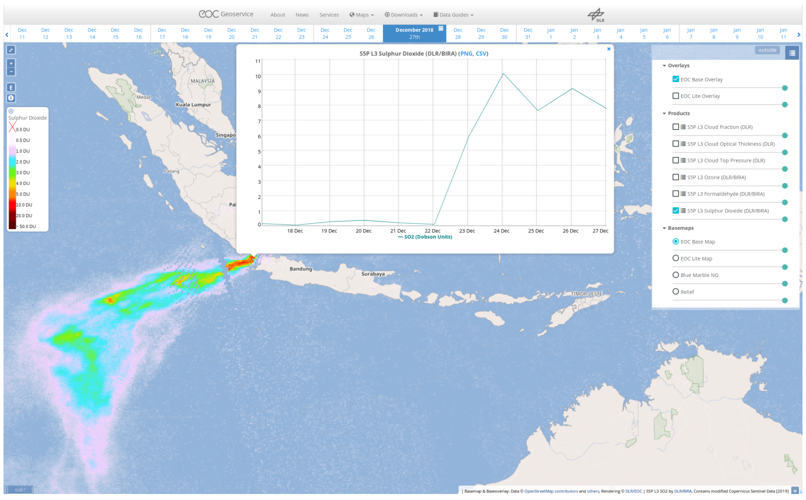


Figure 1. S5p L3 Sulphur Dioxide (DLR/BIRA)

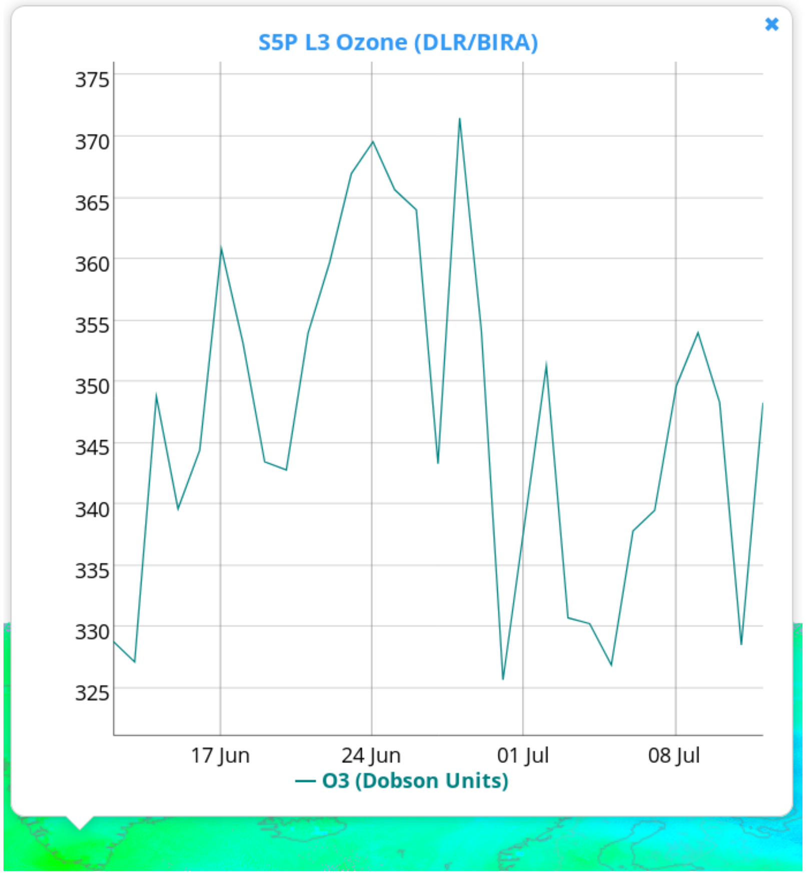


Figure 2. S5p L3 Ozone (DLR/BIRA)

9.4.2.2. Using case 2: Timescan processor

9.4.2.2.1. Overview

A process developer wants to generate and evaluate a settlement footprint mask based on the build-up index (NDBI) for different regions and time periods. The settlement footprints for different time period are combined in a map to show the settlement growth over time. Each optical image needs to be:

a) Corrected with a cloud mask b) Reprojected to an common (global) grid c) Resampled to the highest resolution d) Normalized with the build-up index (S2: (B11–B8A)/(B11+B8A, LS: (MIR1–NIR)/(MIR1+NIR)).

The resulting single EO scenes can then be aggregated over time periods (e.g. yearly or five years) with basic statistical properties min, max, mean, slope, sd, and number of observations.

- Use case inputs:
  - Sentinel-2 L2A collection.
- Expected Outputs:
  - [UC2.1] Datacube: N coverages for N time intervals representing indices (e.g. NDVI/NDBI).
  - [UC2.2] Coverage: temporal aggregated indices within time interval as min, mean, max, slope, sd.
  - [UC2.3] Chart to plot area of indices greater than threshold (per time interval).
- Requirements for API:

- Select collection (e.g., Sentinel-2 L2A).
- Filter on area of interest (e.g., Frankfurt area).
- Filter on mask (e.g. settlements).
- Filter on time range to be processed (e.g., 2016-01-01/2019-12-31).
- Filter on metadata properties (e.g., cloud coverage, orbit direction).
- Define processing function to be conducted for each scene: Multiple indices to be calculated (e.g., NDBI, NDVI, NDWI).
- Define interval for temporal aggregation (e.g., P1Y for yearly aggregated data).
- Define method(s) for temporal aggregation (e.g., min, max, mean, slope, sd, count).
- For chart output: Define method(s) for spatial aggregation (e.g., min, max, mean, slope, sd).

9.4.2.2.2. Use case 2.1 pseudo-api

```
web.api
    collection=S2
    properties=L2A
    subset=(<GEO>, POLY(frankfurt))
    subset=(<TIME>, 2016-01-01/2019-12-31)
    subset=(<RANGE>, [B1,B2,B3])
    subset=(<META>, cql(eoCloudCover<0.2))
    process=(<NDBI>, math(<B1>-<B2>/<B1>+<B2>)
    output_type=<DATACUBE>
    output_format=STAC
```

9.4.2.2.3. Use case 2.2 pseudo-api

```
web.api
    collection=S2
    properties=L2A
    subset=(<GEO>, POLY(frankfurt))
    subset=(<TIME>, 2016-01-01/2019-12-31)
    subset=(<RANGE>, [B1,B2,B3])
    subset=(<META>, cql(eoCloudCover<0.2))
    process=(<NDBI>, math(<B1>-<B2>/<B1>+<B2>)
    process=(<NDBI>, avg(<TIME>, linear)
    output_type=<COVERAGE>
    output_format=CoverageJSON
```

9.4.2.2.4. Use case 2.3 pseudo-api

```
web.api
    collection=S2
    properties=L2A
    subset=(<GEO>, POLY(frankfurt))
    subset=(<TIME>, 2016-01-01/2019-12-31)
    subset=(<RANGE>, [B1,B2,B3])
    subset=(<META>, cql(eoCloudCover<0.2))
    process=(<NDBI>, math(<B1>-<B2>/<B1>+<B2>)
    process=(<NDBI>, avg(<TIME>, linear)
    process=(<NDBI>, area(<NDBI>, <NDBI> >= 0.5)
    output_type=<TIMESERIES>
    output_format=CSV
```

9.4.2.2.5. Example (pseudo graphic)

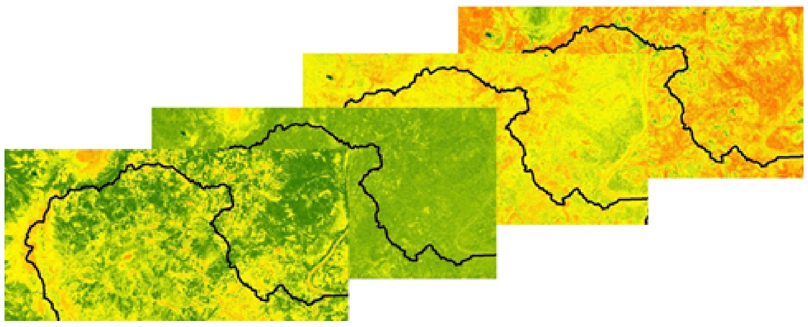


Figure 3. Index calculation

9.4.2.3. Using case 3: Urban monitoring

9.4.2.3.1. Overview

A user wants to identify the greenness of a city and its suburbs (based on NDVI) in relation to the population over time in order to determine the urban green per capita such as covered by SDG 11.7. For different cities there is a map showing statistical properties (min, max, mean, slope, sd, number of observations) of the vegetation index over distinct time periods. Additionally there is a chart showing the a) green surface percentage in relation to the overall urban surface in km2 and b) green surface km2 per capita. Furthermore these values shall be correlated to the higher-level topology (suburb → city → state → country), e.g. as deviation to higher-level topology mean.

Greenness can be replaced with impervious surface (based on NDBI) for SDG 11.3.1 to show the settlement footprint over time.

- Inputs:
  - Sentinel-2 L2A collection.
  - World Settlement Footprint (WSF) collection.
- Expected outputs:
  - Coverage: binary masks representing settlement/build-up areas for different time periods.
  - Map: color-coded settlement/build-up areas for different time periods.
  - Chart: settlement size in km2 (Y) and capita per time period (X).

    - Chart: settlement growth rate (Y) per time period (X).
    - Chart: size of green surfaces in km2 and percentage(s) (Y) in relation to built-up areas and capita per time period (X).
- Internal data processing:
    - NDVI calculation aggregated spatially (area of interest) and temporally (time interval) with greenness masking.
- Requirements for API for Sentinel-2 L2A collection:
    - Select collection (e.g., Sentinel-2 L2A).
    - Filter on area of interest (e.g., Frankfurt area).
    - Filter on time range to be processed (e.g., 2016-01-01/2019-12-31).
    - Define processing function to be conducted for each scene: NDVI band math.
    - Define interval for temporal aggregation (e.g., P1Y for yearly aggregated data).
    - Define method for temporal aggregation (e.g., temporal mean).
    - Define method and property for spatial aggregation (e.g., spatial mean).

9.4.2.3.2. Use case 3.1 pseudo-api

```
web.api
    collection=WSF #World Settlement Footprint
    properties=EVO #Evolution (one binary mask per 5 years)
    subset=(<GEO>, POLY(frankfurt))
    subset=(<TIME>, 2015)
    process=(<WSF_POP>, sum(<GPW:POP>, <WSF:EVO> >= 0 )
    output_type=<COVERAGE>
    output_format=GeoTIFF
```

9.4.2.3.3. Examples

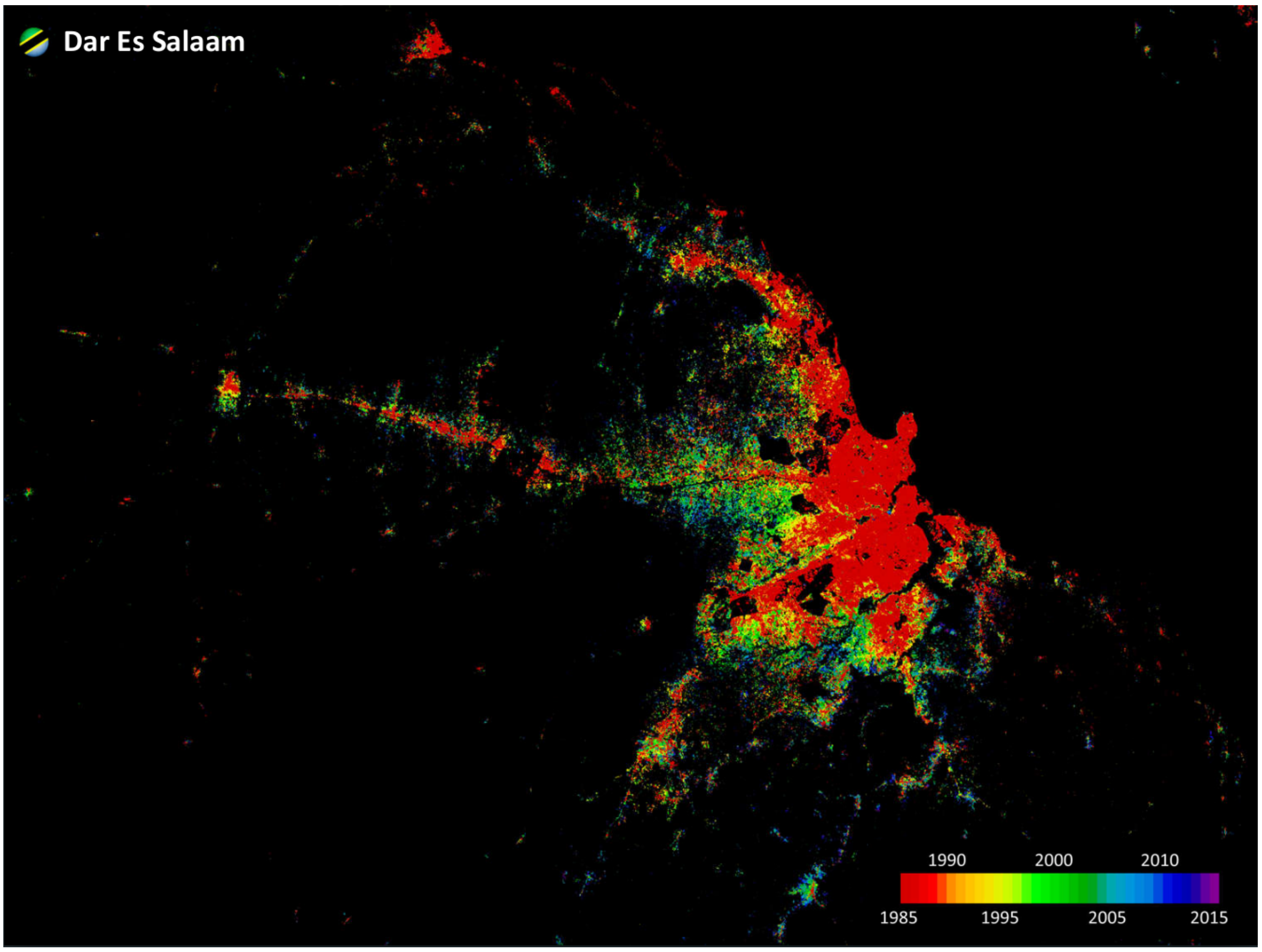


Figure 4. WSF evolution Dar Es Salaam

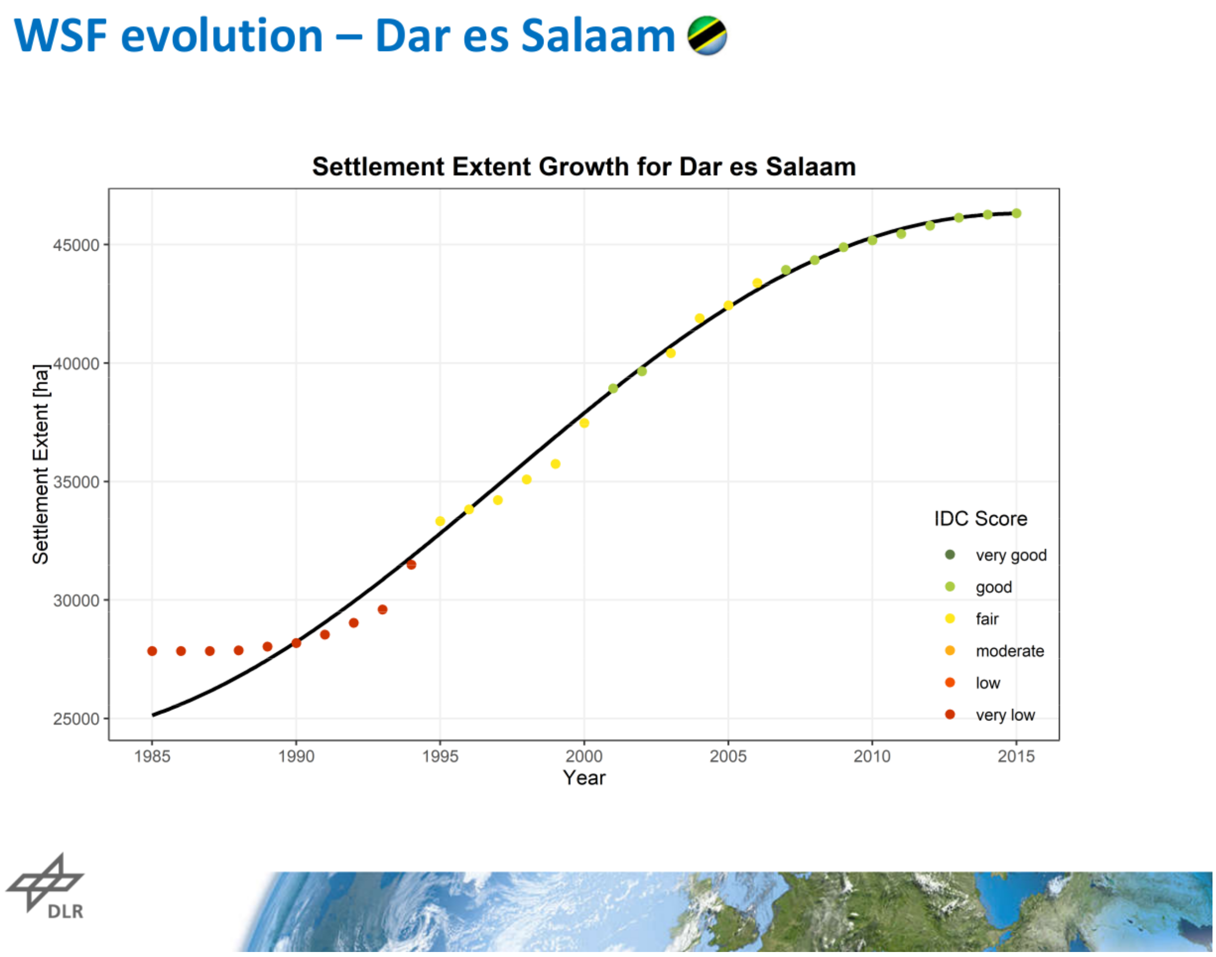


Figure 5. Settlement Extent Growth for Dar es Salaam

9.4.2.4. Data processing pipeline

For the above use cases, the processing steps can be broken into three distinct phases:

1. Filter: Select a subset of the overall data pool for further processing.
2. Transform: Transform the data with well-defined processes based on the previous data subset.
3. Encode: Encode the processed data into a specific format suitable of the output type.

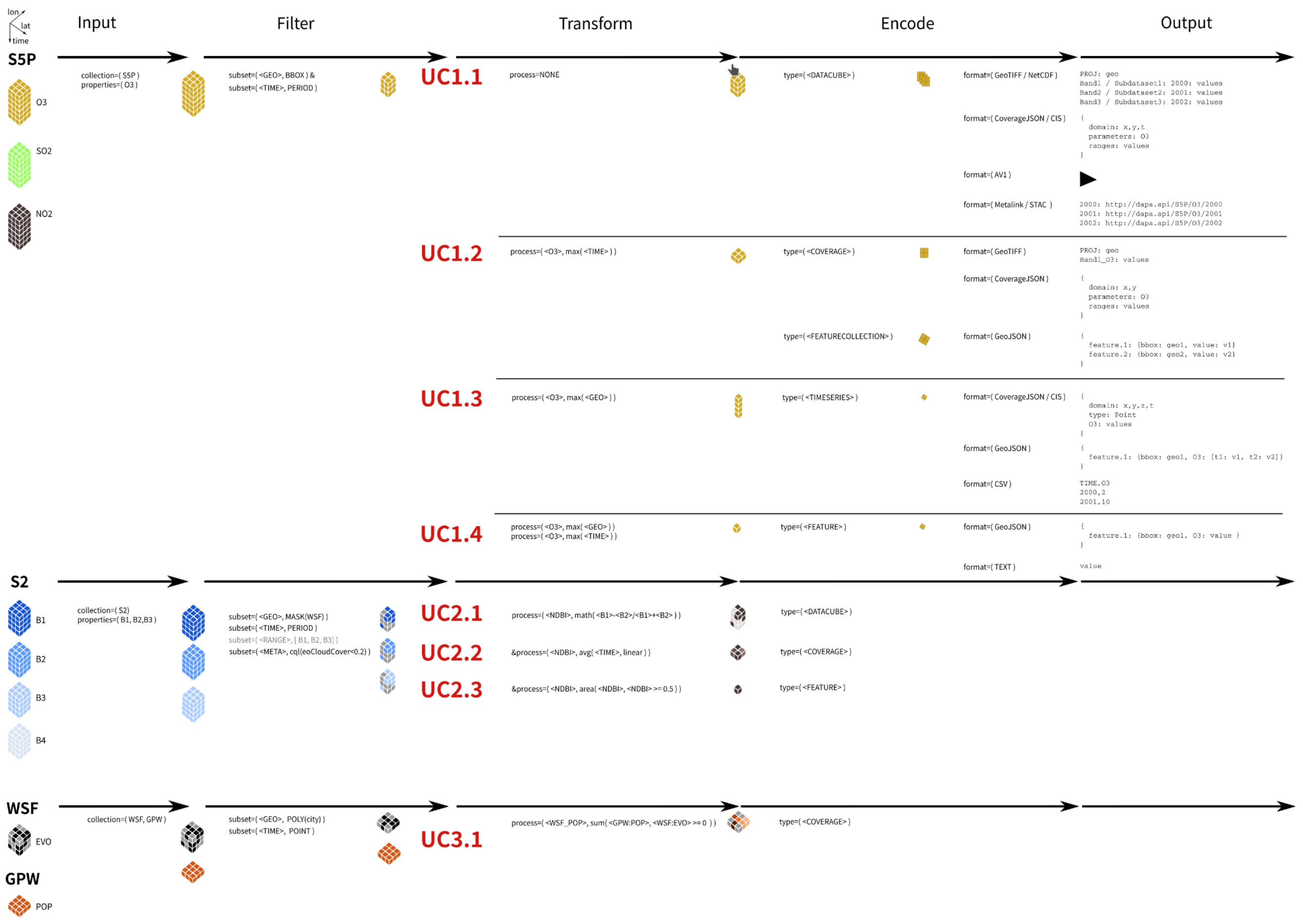


Figure 6. Data processing pipeline

9.4.3. Terradue Use Cases

This use case focus on the DAPA API approach for gridded data covers with Sentinel-5P tropospheric NO2 column density and supports the DAPA API evaluation by several data scientists or earth scientists regarding:

- Learning curve to use the API
- Richness of accessible functionality
- Amount of code needed to execute some common analyses

The typical use cases under evaluation are:

- A data scientist wants to discovery the available collections
- A data scientist wants to discovery the available variables for a given collection
- A data scientist wants to retrieve geospatial data for a variable in a collection in a specific area
- A data scientist wants to retrieve data for a variable in a collection in a single point in time or as an average value over a time period
- A data scientist wants to retrieve a single value for a variable in a collection that averages all data in the target geometry for each time step
- A data scientist wants to retrieve the full time series of a variable in a collection for each data point

- A data scientist wants to calculate the minimum, maximum, and average values of a variable in a collection for any given data retrieval subset

These user stories were the starting point for defining the DAPA API methods and are mapped according to the table below:

User Story	DAPA API
Discover the available collections	/collections
Discover the available variables in a given collection	/collections/{}/variables
Retrieve data in a specific area	/collections/{}/processes/area:retrieve
Retrieve data in a specific area aggregated over a time period *	/collections/{}/processes/area:aggregate-time
Retrieve data for each time step over a time period aggregated over a specific area *	collections/{}/processes/area:aggregate-space
Retrieve a single value in a specific area and for a time period *	collections/{}/processes/area:aggregate-time-space

(*) aggregate functions allows the request of minimum, maximum, and average values of a variable in a collection for any given data retrieval subset

This use case focuses on data from the Copernicus Sentinel-5P satellite and exposes the monthly (from May 2018 to April 2020) averaged nitrogen dioxide concentrations over a portion of Europe. Concentrations of short-lived pollutants, such as nitrogen dioxide, are indicators of changes in economic slowdowns and are comparable to changes in emissions.

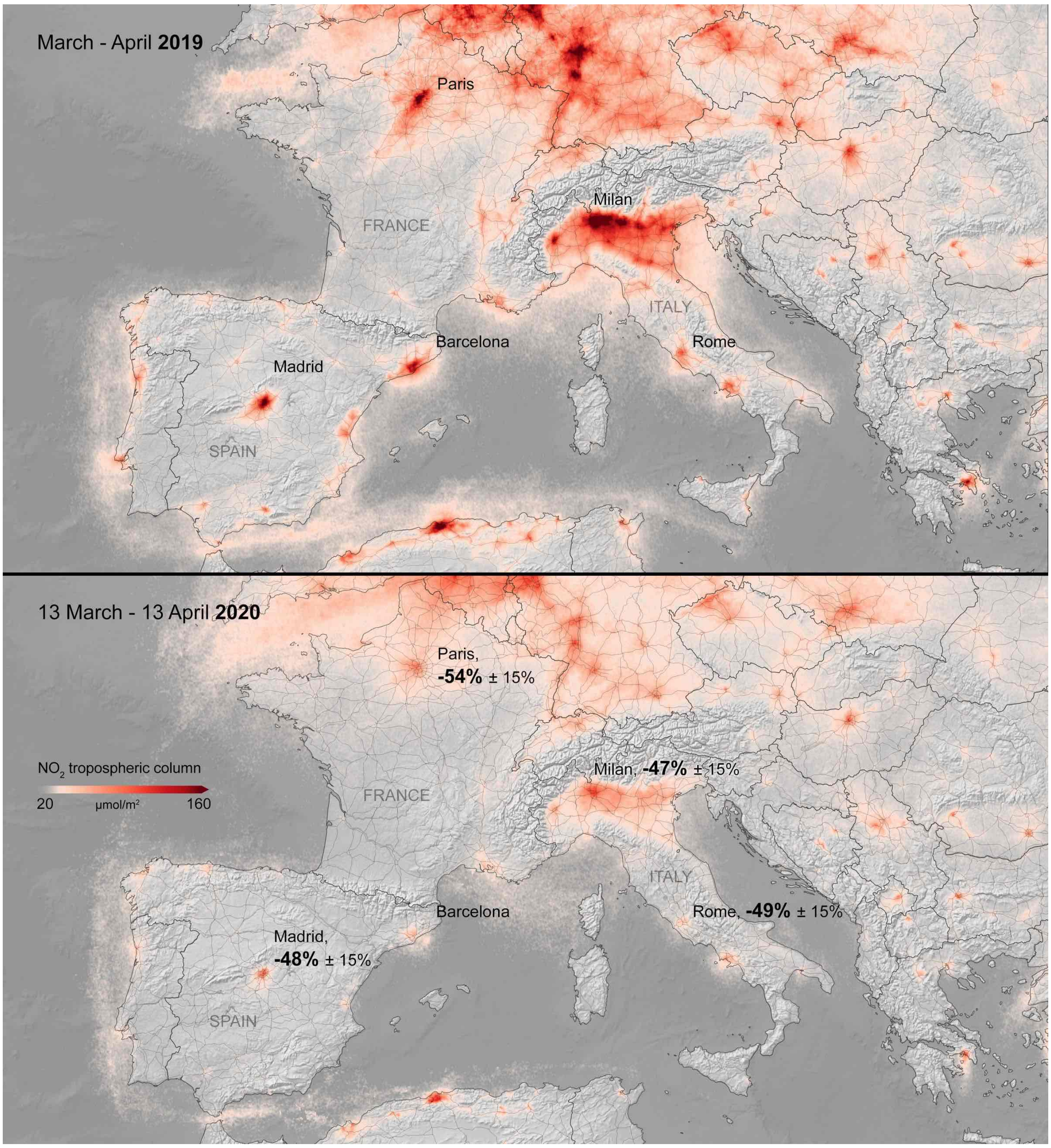


Figure 7. NO2 tropospheric column March/April 2019 versus March/April 2020

9.4.4. GMU use cases

9.4.4.1. Introduction

An analyst wants to find the residential areas prone to flooding. Data sets needed:

1. Digital elevation model,
2. Water level (Mean Sea level – MSL),
3. Land use raster.

9.4.4.2. Workflow

1. The sampling of raster data sets:
   a. Based on administrative polygons. For the U.S. states, counties, zip code. Since these polygons are standard, they should be incorporated as a sampling parameter of the API. If standard data sets are available for the rest of the world, they should be incorporated as well.
   b. Based on the value: e.g. we want to filter out only residential areas and we need to filter land use raster that has values 21, 22, 23, 24.
2. Raster calculator:
   a. Basic math expressions (+,-,*,/) – e.g. the conversion of DEM values from m to feet,
   b. Adding or subtracting two raster data sets – subtraction of the mean water level raster from the DEM raster for calculation of water depth.
   Ideally, these simple and common processing functions, as well as sampling which can be done based on publicly available administrative vector polygons, should be available through the API.
3. Labor division

The Web API is expected to handle data sampling and retrieval (item 1a). The rest should be done in a Jupyter notebook. Example notebook - work in progress.

Datasets are listed in the R workbook below.

Example in R: https://rpubs.com/scvetojevic/635933

Updated R workbook that uses EOX DAPA implementation as a source for DEM for Broward county: https://rpubs.com/scvetojevic/660301

9.4.5. CRIM Use cases: Gridded climate data

This use case describes the data access requirements of a gridded climate dataset. This use case is heavily inspired by the Canadian Climate Data Portal.

9.4.5.1. Dataset selection

Climate datasets often output many variables, which share the same grid and temporality. The API should allow a user to specify both the Dataset (climate model) and the variable (Temperature, Precipitation, Wind etc).

Weather and climate dataset sometimes extend the notion of *datacube* by adding two dimensions: a z axis (elevation) and a forecast emission time.

9.4.5.2. Time and weather

For weather datasets, there are actually two dimensions related to time: The lead time and the forecast time. The forecast time relates to when the forecast was begun. A forecast time of 2020-01-01 00:00 indicates that the best data available at that moment was used, and started the weather model. The lead time relates to how far in the future the weather model has stepped. A lead time of

6 hours indicates that we simulated 6 hours of physics, starting at the forecast time. The validity time is equal to the forecast time + the lead time.

There are many ways to represent this in an API. Using the validity time as the regular time axis and letting the user specify the lead time as a parameter would avoid confusion.

9.4.5.3. Location

Two useful ways of specifying the query location: point and area queries are envisioned.

- Point query over a gridded dataset.

Querying the dataset given a point coordinate should be possible. The API should then query the nearest point on the grid, or maybe interpolate nearby grid points.

- Area query over a gridded dataset.

The easiest way to designate an area over a gridded dataset is to make a rectangular query. The participants believe this should be possible through the API. More specific queries, for instance using polygons, should also possible.

9.4.5.4. Time

Specifying the beginning and the end of the time period that the query covers should be possible.

9.4.5.5. Processing

This case can be completely covered with aggregation and filtering functionalities.

- Aggregation

The API should allow aggregations over two axes: time and space.

The time axis should allow for aggregations such as mean, median (or arbitrary quartiles), min, max, sum. Asking for daily, monthly, yearly aggregations or any plausible temporality should be possible.

For the spatial axis, mean, min and max seem to make sense (e.g. mean monthly temperature over the Montreal Metropolitan area).

- Filtering

A common use case for the Climate Data portal is to make Boolean queries over the data. For instance, the user might want to count the number of days with temperature over 30 degrees in a year. Consequently, the API should allow for Boolean operations over the data (greater than, less).

Combining the binary operations with an aggregation operation should be possible. This enables counting the number of days where the condition was true.

Chapter 10. Jupyter Notebooks

10.1. Overview

A Jupyter notebook is an electronic file that runs in a web browser and contains both programming code and text descriptions. Jupyter notebooks are organized into **cells** of which there are three types:

- Code cells
- Output cells
- Markdown cells

Code cells contain blocks of code that can be executed. Many different programming languages are supported with Python being the most popular. In a Jupyter notebook, chunks of code ranging from one line to many lines can be run individually and in any order without running all of the code in all the code cells in the Jupyter notebook.

The output from running a code cell is shown in an output cell. Output cells can contain generated charts, plots, command line output, images and much more.

The markdown cells of a Jupyter notebook contain explanations and clarifications of the programming code in the markdown format. Markdown sections of a Jupyter notebook can include formatting to make text bold, italic, form tables and lists, show code listings and render images. They can also contain embedded charts, plots, images, videos and links.

A Jupyter notebook might be viewed as an integrated development environment (IDE) but this is not quite the case. A Jupyter notebook strikes a balance between being a simple text editor and a full-featured IDE. As such, Jupyter notebooks provide a quick and streamlined way for problem-solvers to prototype code and share their programming solutions with team members, supervisors, customer and other stakeholders. Jupyter notebooks open quickly and quickly produce output. Data exploration, data cleaning and plot building are examples of what can accomplished in a Jupyter notebook easier and quicker than can be produced using a text editor or an IDE.

This clause provides details of the Jupyter notebooks developed for the DAPA thread. The purpose of the notebooks is to interact with the DAPA endpoints in order to test and evaluate the DAPA API primarily for ease of use from an end user’s perspective.

10.2. Jupyter Notebook 1 (Geomatys - D107)

10.2.1. Overview

The primary function of this notebook is to exercise each processing endpoint offered by a service that implements the DAPA. This is done by crawling the machine-readable API description that each service must offer.

10.2.2. Source code

The source code for this notebook is available here: TBD

10.2.3. Description

10.2.3.1. Operation of Jupyter Notebook 1

The notebook depends on the following static endpoints being available as described in the OGC Testbed-16: Data Access and Processing API ER [http://fix.me]:

Endpoint	Description
/	The service landing page.
/api	Provides the OpenAPI [http://spec.openapis.org/oas/v3.0.3] document that described the API. Adding the query parameter `f=json` returns the API description encoded in JSON.
/collections	Provides a list of available collections

Processing endpoint discover proceeds as follows:

- The '/collections' endpoint is accessed to get the list of available collections.
- The OpenAPI document is accessed as JSON via the `/api` path to search for processing endpoints.
  - For the GET method, search each endpoint to determine if it provides a supported output type; this notebook expects application/geo+json.
  - Group the identified endpoints by collection identifier using the relative path from the landing page. This provides a list of processing endpoints for each collection.
  - Determine the set of query parameters each processing endpoint expects by interrogating the `components/parameters` object.
  - Use this information to construct a query specification.

10.2.3.2. Lessons learned

This section describes the lessons learned from building this Jupyter notebook.

DAPA service discovery capabilities could be enhanced and simplified by standardizing a few static root endpoints. In this context, endpoint standardization means:

- Endpoint path
- Endpoint characteristics:
  - Method (GET, POST, etc.),
  - Request variables (path variables, query parameters and/or request body),
  - Response content definition

Listing of necessary endpoints:

- Collection listing (could be collection "research" for better user experience / improved discovery).
- For each collection:
  - Specification/metadata: extents, description, etc.,
  - Processing/derivation capabilities.
  - For each process:
    - Query parameters,
    - Output definition: mime-type, variables.

Notes

- Endpoints should reflect purely abstract concepts, and above statements should remain true whatever protocol is used for user/provider communication.
- Variable definitions are placed on a per process basis, but maybe could be defined on a per collection basis (depending on processing complexity allowed).

10.2.4. Model

Internal components of the client attempt to map specific DAPA concepts to more generic ISO, OGC or open-source APIs. To model resources, the Apache SIS [https://sis.apache.org] API was used which was bound to OGC GeoAPI [https://www.ogc.org/standards/geoapi] interfaces to define core concepts such as naming and metadata. In Apache SIS a resource can be either:

- A **Dataset**: Direct handle over either a coverage of collection of features (or something else like a point cloud). For simplicity, specific concepts are not shown in the diagram.
- An **Aggregate**: That represents a group of resource. This is like a node in a tree of resources.

A DAPA client is made of custom implementations to ease usage. The client represents an **aggregate** of collections. Each collection is itself an **aggregate** composed solely of products. Products are specialization of **datasets** for DAPA processing/derivation/download endpoints. They are derivable through the specification of a set of query parameters.

```
@startuml

note "ISO-19103 model of a name" as names
note "ISO-19115 model of a metadata sheet" as md_n1
note "OGC filter encoding to subset/derivate\nEx: BBOX, Property < Literal, etc." as
filter_n1
note "In the future, OpenGIS filters could be mapped to DAPA service capabilities.\nIt
would allow delegation of tasks to service through usage of an open
specification.\nFor now, only specific dapa query delegates work to web-service" as
delegate
note "Based on OpenAPI description of an endpoint parameter" as params

package org.opengis {
    package util {
          interface GenericName

          names .. GenericName
    }

    package metadata {
        interface Metadata

        md_n1 .. Metadata
    }

    package geometry {
        interface Envelope
    }

    package filter {
        interface Filter
        interface Expression

        filter_n1 .. Filter
    }
}

package org.apache.sis {
    package storage {
        interface Resource

        interface Aggregate extends Resource

        interface DataSet extends Resource

        Resource "0" o-up- "1" GenericName

        Envelope "0" o-- "1" DataSet

        Resource o-up- Metadata
        Aggregate "1" o-- "n" Resource
    }
}

package com.geomatys.tb16.dapa {

    class Client extends Aggregate {
      serviceUri : URI
    }

    class Collection extends Aggregate

    class Product extends DataSet {
        subset(parameters : Query) : Product
        subset(ogcFilter: Filter) : Product
    }

    class Query {
        get(param: String) : Any?
        set(param: String, value: Any?)
    }

    class ParameterDescriptor {
        name : String
        description : String?
        required : Boolean
    }

    Collection "1" o-- "n" Product
    Query "0" o-- "n" ParameterDescriptor
    Product "0" o-- "1" Query

    delegate..Product
    params..ParameterDescriptor
}

@enduml
```

Note: Even when trying to create an interoperability layer based on open-source specifications, the current client engine is based upon Geomatys proprietary/closed-source tool **Examind Datacube**.

10.2.5. Usage

10.2.5.1. Service discovery and loading

- Connect to the web-service by providing URI to its landing page:

```
from examind import dapa
client=dapa.connect("<service_url>")
# IPython output displays service metadata
client
```

- Select a collection to work with:

```
myCollection=client.select("<collection-id>")
myCollection
```

- Choose a dataset endpoint:

```
datasetEndpoint=myCollection.select("<endpoint>")
datasetEndpoint
```

- Check parameters to fill:

```
q=datasetEndpoint.query()
```

- Configure dataset computing/download:

```
query["limit"]= 5
query["parameter2"]= "value"
```

- Compute dataset:

```
dataset=datasetEndpoint.subset(query)
# Display first records of computed dataset
dataset[0:5]
```

10.2.5.2. Post-processing

The choice of developing the client using the Geomatys **Examind Datacube** product allows usage of a number of features:

- Binding of collection of OGC simple features to both Pandas and Dask dataframes

```
dataset=datasetEndpoint.subset(query)
# Load dataset results in-memory into Pandas
pandas_dataframe = dataset[::]
# Create a Dask dataframe that will load results by chunk
dask_dataframe = dataset.as_dask()
```

- Ability to use OGC GeoAPI [https://www.ogc.org/standards/geoapi] filter interfaces to derivate/filter datasets

```python
from examind import filters

# Filter builder
ff=filters.factory()

# It is possible to create filters based on property values
value_filter = ff.greater(ff.property("<property_name>"), ff.literal(1.0))

# or can ask for specific regions of interest
alaska      = ff.bbox(-180, 50, -150, 80, "CRS:84")
antartic    = ff.bbox(-180, -90, 180, -70, "CRS:84")

# Filter chaining is possible using OR and AND filters.
bbox_filter    = ff.or_(antartic,alaska)
chained_filter = ff.and_(bbox_filter, value_filter)

# Apply filtering on base dataset:
subset=dataset.subset(chained_filter)
```

- Tooling to add columns in a dataframe representing measures from an EO dataset

```python
from examind import dapa
from examind import datasets
from examind import operations as ops

# Load target datasets
dapa_dataset = dapa.connect("<service_uri>")\
               .select("<collection>")\
               .select("<product>")
eo_dataset = datasets.open("<eo_dataset>")

# Derive dapa dataframe, add a column that will extract EO information at each feature
location.
point_crossing_dataset = ops.from_dataset(dapa_dataset)\
                            .add(eo_dataset)\
                            .build()
```

- Ability to create static renderings of datasets using SLD files for symbology customization

```
from examind import dapa
from examind import datasets
from examind import display

# Open datasets
dapa_dataset = dapa.connect("<service_uri>")\
               .select("<collection>")\
               .select("<product>")
eo_dataset = datasets.open("<eo_dataset>")

# Configure display
scene = display.Scene()
# This layer is the background. Symbology is configured through second argument, which
must be a valid SLD style.
scene.add(eo_dataset, "path/to/sld/style")
# Add another layer on top, using a default style. Then, display configured scene.
scene.add(dapa_dataset)\
     .draw()
```

10.2.5.3. Multi-language API

Apart from Dask/Pandas binding, all cited tools above are available in three different programming language: **Java, Kotlin-JVM, Python**. As an example, below is a filtering snippet with Kotlin:

```
import com.examind.datacube.interop.kt.filters.subset

// Filter factory implicitly used as context for subset operation
val subset = dataset.subset {
    // Usage of infix notation allow to create pseudo-operators for a better
lisibility
    property("<property_name>") greaterThan 1.0
    and
    (
      bbox(-180, -90, 180, -70, "CRS:84")
      or
      bbox(-180, 50, -150, 80)
    )
}
```

10.2.6. Limitations

The notebook develpoed during the Testbed is still incomplete in the following ways:

- Handles only simple parameters for DAPA queries (text, numbers).
- Limited testing using the interactive instruments server.
- Only works with GeoJSON datasets.

10.3. Jupyter Notebook 2 (GeoSolutions - D108)

10.3.1. Overview

GeoSolutions and DLR created a notebook to compare the Terradue, EOX and GeoSolutions DAPA API endpoints. The purpose of this notebook was to compare various aspects for each of the APIs and support the harmonization of the API specification.

Generally, both APIs share the same approach and overall semantics as described in OGC Testbed-16: Data Access and Processing API Engineering Report [http://fix.me] but differ in some aspects as described in this section.

10.3.2. Source

The source code for this notebook can be found here: https://gitlab.com/ogceo/t16-dapa-dlr/-/blob/master/DAPA_Comparison.ipynb.

A live link of the notebook can be found here: https://mybinder.org/v2/gl/ogceo%2Ft16-dapa-dlr/master?urlpath=lab/tree/DAPA_Comparison.ipynb

10.3.3. URL paths and parameters

The following table compares the URL paths and parameters for each endpoint.

Aspect	**EOX endpoint**	**Terradue/GeoSolutions endpoint**
URL Paths	path contains the data output type (cube, area, timeseries, value) and aggregation input geometry (position or area)	paths contains the aggregation input geometry (position or area) and aggregation dimension (time or space)
URL Parameters	spatial filter = bbox or point temporal filter = time property filter = fields transformation = aggregate output type (embedded in path) = `cube`, `area`, `timeseries` or `value`	spatial filter = location temporal filter = datetime property filter = variable transformation = function output type (embedded in path) = `position:` or `area:`

10.3.4. Outputs

All APIs encode output differently. The specific details can be found in the Jupyter notebook but are summarized here:

- Metadata requests to the collection and variable endpoints all return JSON but the schema of the returned documents are different.
- For data or processing requests,
  - Terradue uses a pure JSON output. It is unclear whether the response JSON is conforming to some standard JSON schema or is using a bespoke schema.

    - EOX relies on well-known formats such as GeoTIFF, NetCDF and CSV.
    - GeoSolutions encodes all output as vectorized DGGS zones in GeoJSON.

The ability to negotiate output formats was not investigated during this testbed and is left as a future work item. Future investigation should start by defining the set of possible output data type structures (e.g. data cube, coverage, time series, multi-dimensional data set, feature or simple value) and then define recommended standard encodings for each (e.g. GeoTIFF, GeoJSON). The result of doing this would be to:

- Reduce the number of permutations of possible output types for a specific collection and function (e.g. you cannot request a multi-dimensional datacube from one-dimensional collection).
- Provide a clear understanding, from the user's perspective, of what will be returned.

Specifying the output format alone is not sufficient since the structure of the content could be quite different, for example time series, multi-dimensional data set, feature data, or simple values.

The creation of MIME types or MIME type templates that capture the valid combinations of data structure and encoding format would be very useful to facilitate response negotiation.

These ideas were sketched out in the data pipeline illustration and ended up implemented in the EOX API as one of the URL path elements as described in the table above.

10.3.5. Collections

The structure of the collection resources is similar. However sub-resources use different names: `fields` for EOX vs. `variables` for Terradue and GeoSolutions. The specific resources are:

- EOX Collection: `S5PL2`; Fields: `NO2`, `SO2`.
- Terradue Collection: `S5P-NO2`; Variables: `troposheric_NO2_column_number_density`.
- GeoSolutions Collection: `S2-H3` and `S2-RPIX`; Variables: `B01`-`B12`, `NDVI`, `NDBI`, `NDWI`.

Collection metadata in all cases contained start/end date, but the temporal resolution is missing thus making it difficult to determine. For example, if the collection is a collection of discrete time instances or daily/monthly periods. The DGGS-based collection metadata additionally publishes the available spatial resolutions of the available grid zones.

10.3.6. Other differences

The actual API results are not comparable due to differences in the source data. The EOX endpoint uses original S5P L2 scenes. The Terradue endpoint uses gridded S5P monthly aggregates. The GeoSolutions endpoint uses Sentinel-2 scenes over Canberra stored in two different DGGS grid systems (H3 and rHealPix).

EOX `fields` parameter support simple algebra which is not supported in the Terradue and GeoSolutions endpoint.

Finally, all endpoints provide no capability to specify the temporal resolution of aggregation. For example, the APIs cannot be asked to provide all data between 2000 and 2001 aggregated as monthly means. Specification of spatial resolution is only possible in the GeoSolutions DAPA API through the use of the `resolution` parameter that selects the zoom level in which the aggregation is executed.

10.4. Jupyter Notebook 3 (Terradue - D109)

10.4.1. Overview

The primary function of this notebook was to compare NO2 tropospheric column between March/April 2019 and March/April 2020 using Sentinel-5P data products.

10.4.2. Source code

The source code for this notebook is available here: https://github.com/opengeospatial/t16-dapa-terradue-sentinel-5p-NO2

A live link that uses Binder [https://mybinder.org/] to execute the notebook can be found here: https://mybinder.org/v2/gl/terradue-ogctb16%2Fdapa%2Fd167-api-endpoint%2Ffastapi-dapa/master?urlpath=/proxy/8001/

10.4.3. Description

The operation of the notebook proceeds as follows:

The first step is to identify the available collections querying the DAPA endpoint.

```
query = '{}/collections'.format(dapa_endpoint)
r = requests.get(query)
r.json()
```

The response returns a JSON document

```
{
    "collections": [{
        "id": "S5P-NO2",
        "title": "Sentinel-5P NO2 column number density over Europe",
        "extent": {
            "spatial": {
                "bbox": [[-13, 57, 33, 33]],
                "crs": "EPSG:4326"
            },
            "temporal": {
                "interval": [["2018-05-01T00:00:00/2020-04-01T00:00:00"]]
            }
        }
    }]
}
```

The second step is to list the variables available on the selected collection.

```
collection = r.json()['collections'][0]['id']
query = '{}/collections/{}/variables'.format(dapa_endpoint, collection)
r = requests.get(query)
```

The returned JSON object identifies the available variables.

```
{"variables": [{
    "id": "tropospheric_NO2_column_number_density",
    "title": "Troposheric NO2 column number density",
    "uom": "umol/m2"
    }]
}
```

And then the variable name can be retrieved.

```
variable = r.json()['variables'][0]['id']
variable

'tropospheric_NO2_column_number_density'
```

With this information the user is able to query using a location name.

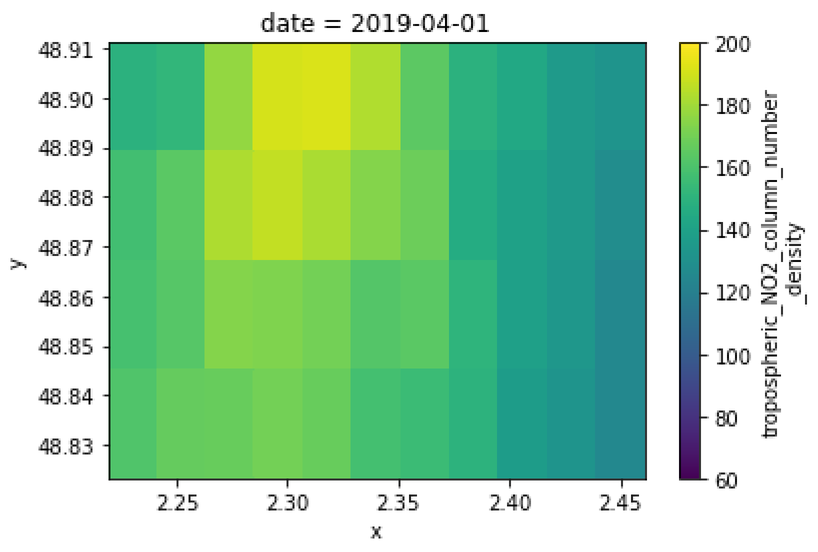
```
location = 'Paris, France'
query = '{}/collections/{}/processes/area:retrieve'.format( dapa_endpoint, collection
)
params = {'location': location, 'variable': variable}
r = requests.get(query, params=params)
r.json()
```

The xArray represents the obtained data.

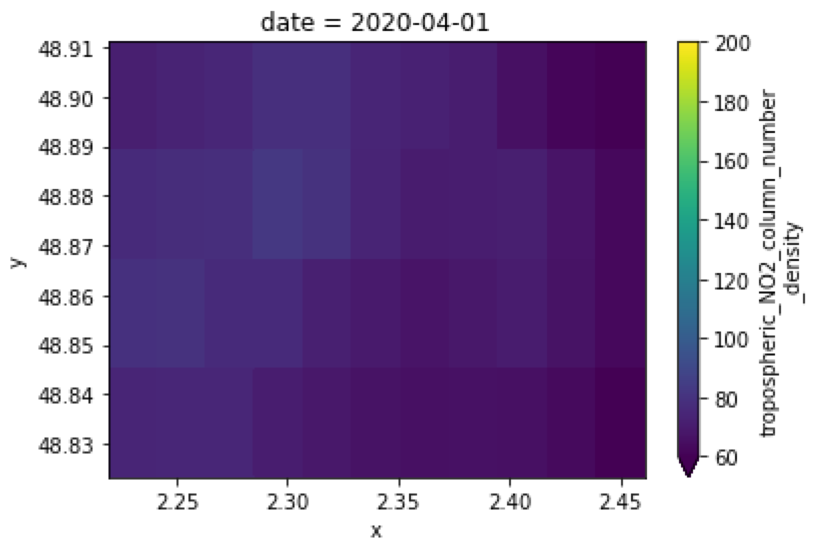
```
{
    "dims": ["date", "y", "x"],
    "attrs": {
        "transform": [0.02197265625,0.0,-13.0078125,0.0,-0.02197265625,57.01904296875
],
        "crs": "+init=epsg:4326",
        "res": [0.02197265625, 0.02197265625],
        "is_tiled": 0,
        "nodatavals": [9.969209968386869e+36],
        "scales": [1.0],
        "offsets": [0.0],
        "AREA_OR_POINT": "Area",
        "latitude#units": "degree_north",
        "longitude#units": "degree_east",
        "NC_GLOBAL#Conventions": "HARP-1.0",
        "NC_GLOBAL#datetime_start": "6695.090290335648",
        "NC_GLOBAL#datetime_stop": "6725.99574832176",
        "NC_GLOBAL#history": "2020-05-07T18:08:44Z [harp-1.10] harpmerge -a  ...
/workspace/s5pl3/s5p-l3-tropno2-month-201805.nc",
        "NETCDF_DIM_EXTRA": "{time}",
        "tropospheric_NO2_column_number_density#description": "tropospheric vertical
column of NO2"
        "tropospheric_NO2_column_number_density#units": "umol/m2"
    },
    "data": [[
        [87.39471435546875,92.51070404052734, ...
        93.9330825805664,95.68144226074219],
        ...
        [74.52445220947266,75.1952133178711, ...
        63.43209457397461,59.1694221496582]
    ]],
    "coords": {
        "date": {
            "dims": ["date"],
            "attrs": {},
            "data": [
                "2018-05-01T00:00:00",
                "2018-06-01T00:00:00",
                ...
                "2020-03-01T00:00:00",
                "2020-04-01T00:00:00"
            ]
        },
        "x": {
            "dims": ["x"],
            "attrs": {},
            "data": [2.230224609375,2.252197265625, 2.274169921875, ...
2.427978515625, 2.449951171875]
        },
        "y": {
            "dims": ["y"],
            "attrs": {},
            "data": [48.900146484375,48.878173828125, 48.856201171875,
48.834228515625]
        }
    },
    "name": "tropospheric_NO2_column_number_density"
}
```

The data can be retrieved to a different object and visualized.

```
area_retrieved = xr.DataArray.from_dict(r.json()).assign_coords(date=[np.datetime64(d)
for d in xr.DataArray.from_dict(r.json())['date'].values])

area_retrieved.values

array([
    [[ 87.39471436,  92.51070404,  98.10374451, ...,  94.32688141, 93.93308258,
95.68144226],
        ...
        [ 96.47353363, 100.71057892, 104.87663269, ..., 106.05647278, 104.1901474 ,
101.26545715]],
    ...,
    [[ 71.58709717,  73.33903503,  75.02533722, ...,  65.71749878, 61.98894119,
57.75169754],
        [ 74.52445221,  75.19521332,  75.29916382, ...,  65.9913559, 63.43209457,
59.16942215]]])

area_retrieved.sel(date='2019-04-01T00:00:00').plot(vmin=60, vmax=200)
```



```
area_retrieved.sel(date='2020-04-01T00:00:00').plot(vmin=60, vmax=200)
```

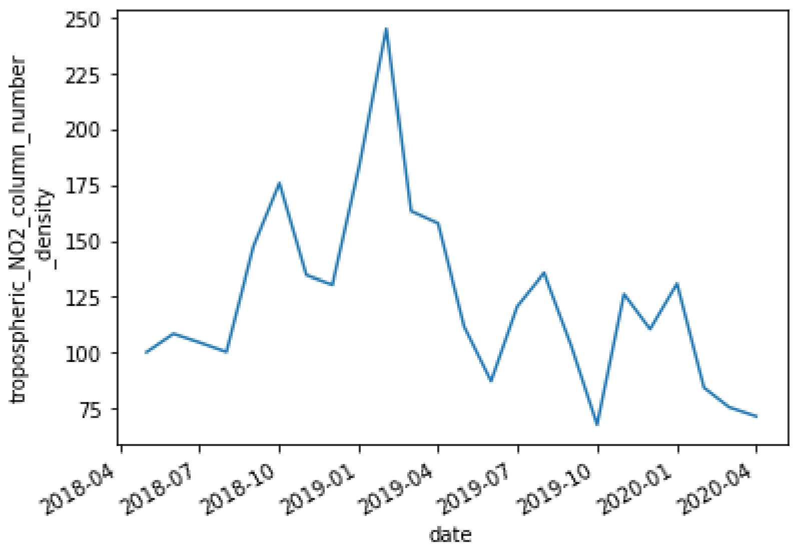


The request can be executed with a aggregation function over space.

```
query = '{}/collections/{}/processes/area:aggregate-space'.format(dapa_endpoint
,collection)
params = {'location': location, 'variable': variable, 'function': 'mean'}
r = requests.get(query, params=params)
r.json()

{'dims': ['date'],
 'attrs': {},
 'data': [100.1324234008789,108.42546844482422,
  ...
  75.42339324951172,71.36090850830078],
 'coords': {'date': {'dims': ['date'],
   'attrs': {},
   'data': ['2018-05-01T00:00:00',
    '2018-06-01T00:00:00',
    ...
    '2020-03-01T00:00:00',
    '2020-04-01T00:00:00']}},
 'name': 'tropospheric_NO2_column_number_density'}
```

And a plot can be made.

```
xr.DataArray.from_dict(r.json())
area_space_extracted = xr.DataArray.from_dict(r.json()).assign_coords(date=[np
.datetime64(d) for d in xr.DataArray.from_dict(r.json())['date'].values])
area_space_extracted.plot()
```



And with DAPA the request can also be executed with a aggregation function over space and time.

```
query = '{}/collections/{}/processes/area:aggregate-time-space'.format(dapa_endpoint
,collection)
params = {
    'location': location,
    'variable': variable,
    'function': 'mean'
    }
r = requests.get(query, params=params)
r.json()
{'mean': 124.00251770019531}
```

Chapter 11. DAPA Endpoints

11.1. Overview

This section describes the details of the DAPA endpoints. This section does not describe the API itself (see OGC Testbed-16: Data Access and Processing API ER [http://fix.me]) but rather the implementation details of each endpoint.

11.2. Resource model

An abstract description of a resource model that binds a specific function to specific data and also provides a means of expressing valid combinations of data and operations is now defined.

11.3. API Endpoint 1 (EOX - D165)

11.3.1. Usage

The DAPA API is implemented by extending the OGC-EDC application [https://eurodatacube.com/marketplace/apps/ogc-edc]. This application (app_) provides a layer of OGC services on top of Euro Data Cube (EDC). The source code [https://github.com/eurodatacube/ogc-edc] is available on GitHub in the Euro Data Cube [https://github.com/eurodatacube] organization.

The DAPA API itself is available as an app [https://eurodatacube.com/marketplace/apps/ogc-edc-tb16-dapa] from the EDC marketplace and can easily be deployed in any user's workspace. The source code is also available on GitHub in the tb16-dapa branch [https://github.com/eurodatacube/ogc-edc/tree/tb16-dapa] of the OGC-EDC app.

The DAPA API app description [https://eurodatacube.com/marketplace/apps/ogc-edc-tb16-dapa] on the EDC marketplace provides a detailed step-by-step guide as well as tutorial Jupyter notebooks to set up and use the DAPA service endpoint. In the end it is as simple as deploying the app from the marketplace in the before activated EDC EOxHub Workspace [https://eurodatacube.com/marketplace/infra/edc_eoxhub_workspace] after registration at Euro Data Cube [https://eurodatacube.com/register]. There is a free three-month promotional plan available to try out the platform.

Once the app is successfully deployed, the generated access URL can be used in the provided notebook tutorials in the JupyterLab environment [https://eurodatacube.com/register] of the user's workspace. The tutorials are available either from the EDC marketplace [https://eurodatacube.com/marketplace?tag=DAPA] or directly in the JupyterLab environment from where they can with one click be imported in the user's workspace for interactive usage.

11.3.2. Datasets

The DAPA API service endpoint is based on the Euro Data Cube environment and thus provides access to a wide range of datasets:

- Sentinel-1 GRD (full archive).

- Sentinel-2 L1C and L2A (full archive).
- Sentinel-3 OLCI and SLSTR (full archive).
- Sentinel-5P L2 (currently not supported).
- Landsat-8 L1C (ESA archive).
- MODIS.
- DEM.

Please note: The API might not work with every dataset. The API has been tested successfully with Sentinel-2, Sentinel-3, and the Digital Elevation Model (DEM).

11.3.3. API Documentation

The online DAPA API app description [https://eurodatacube.com/marketplace/apps/ogc-edc-tb16-dapa] provides a detailed API documentation which is not copied here. A technical API description using the Swagger UI is included in the app as well.

11.3.4. Deployment Infrastructure

The deployment as described above in the Usage section relies on the Euro Data Cube (EDC) platform. The platform is deployed in a Kubernetes cluster provided by the Amazon Web Services (AWS).

The EDC EOxHub Workspace is a dedicated namespace in the Kubernetes cluster with reserved resources according to the user subscription. By default, a JupyterLab environment is provided in each user workspace to interactively execute provided notebooks. Available apps can be deployed in the workspace and are available under custom URLs. This way each user has, for example, their private custom DAPA API endpoint running on the reserved resources. In particular this setup ensures that there is no interference with other users using the DAPA API.

11.3.5. Implementation Details

The DAPA API is implemented in the Python programming language (3.6) using the Flask web application framework. The DAPA API is built on the OGC API layer of EDC, and inherits all of its capabilities.

All requests to the provided endpoints are internally translated to API calls to the catalogue and processing API of EDC. Only minor processing is actually done in the DAPA App itself. To route these requests, extensive use of the OAuth and “requests” libraries is made.

11.4. API Endpoint 2 (interactive instruments - D166)

The endpoint was deployed using ldproxy [https://github.com/interactive-instruments/ldproxy], an implementation of several OGC API standards (most of them drafts) and a Reference Implementation for OGC API Features. For the testbed, an experimental extension to ldproxy was developed to support DAPA.

11.4.1. The data

Global Historical Climatology Network Daily (GHCN-D) [https://www.ncdc.noaa.gov/ghcnd-data-access] is a dataset from NOAA that contains daily observations over global land areas. The dataset contains station-based measurements from land-based stations worldwide, about two thirds of which are for precipitation measurement only. Other meteorological elements include, but are not limited to, daily maximum and minimum temperature, snowfall, and snow depth. The dataset is a composite of climate records from numerous sources that were merged together and subjected to a common suite of quality assurance reviews.

The 2019 data from the GHCN-D dataset was made available via the API. The API provided access to 115,082 weather stations and 34,093,913 observations.

11.4.2. The API

The endpoint provided a Web API consistent with the emerging OGC API family of standards [https://ogcapi.org/]. The API provided access to the weather observations in two ways:

- Access to the raw data, that is each weather station and observation, via the OGC API Features standard [https://www.ogc.org/standards/ogcapi-features].
- In addition, the DAPA endpoints provided additional options to retrieve the data with the goal to simplify the use of the data for end-users, for example, for use in Jupyter Notebooks. Data retrieval in DAPA is based on the selection of the observations by the spatial sampling geometry (point, area, or grid), the temporal sampling geometry (instant or interval), and the observed properties followed by optional post-processing to aggregate observations by space and/or time.

The landing page of the Web API was at https://t16.ldproxy.net/ghcnd.

Note that all readable URIs for this endpoint are not percent-encoded for better readability.

11.4.3. The API definition

The API was formally specified using OpenAPI [https://github.com/OAI/OpenAPI-Specification/blob/master/README.md], available as JSON, YAML and HTML. The HTML documentation also acted as an interactive client and could be used to play with the API.

11.4.4. General OGC API resources in the API

Consistent with OGC API practice, the API supported JSON and HTML for most endpoints.

Software clients including scripts in a Jupyter Notebook would typically request responses in JSON - either by including `application/json` or `application/geo+json` in the `Accept` header of the HTTP request or by adding a query parameter `f=json` to the request.

The HTML responses enabled to explore the API directly in the web browser without the need for a specific client.

The endpoint implements the standard resources for an API implementing OGC API standards sharing geospatial data:

- A landing page with basic information about the API and links to other resources in the API;
- A conformance declaration listing all the OGC API conformance classes that the API implemented;
- The spatial data resources available in the API are also known as "collections". In this case there are two such resources, the weather stations and the observations. Both collections consist of features;
- Each of the collections also has its own landing page that links to the feature data itself and to other information like schema information or the queryables (properties that can be used in queries).

11.4.5. Access to the raw data

This section lists some examples of how to access the raw observations directly as GeoJSON.

The API endpoint for these requests is:

```
https://t16.ldproxy.net/ghcnd/collections/observation/items
```

This request returns 10 observation features and a link to the next "page" of features (the link with `rel` set to `next`). This enables clients to page through the observations. The response is shown below but reduced to two observations.

Each observation has the point geometry of the weather station that recorded the observation and the following properties based on the Observation & Measurement standard:

- phenomenonTime: The day of the observation.
- observedProperty: The physical property that was observed, more on the available properties is described below.
- result: The observed numerical value.
- unitOfMeasurement: The unit of the value in result.
- locationCode: The code of the weather station that recorded the observation.
- locationName: The name of the weather station that recorded the observation.
- locationLink: The URI of the weather station in the API.

```
{
   "type": "FeatureCollection",
   "@context": "https://t16.ldproxy.net/ghcnd/collections/observation/context",
   "@type": "geojson:FeatureCollection",
   "links": [
      {
         "href": "https://t16.ldproxy.net/ghcnd/collections/observation/items?f=json",
         "rel": "self",
         "type": "application/geo+json",
         "title": "This document"
      },
      {
        "href": "https://t16.ldproxy.net/ghcnd/collections/observation/items?f=html",
        "rel": "alternate",
        "type": "text/html",
        "title": "This document as HTML"
      },
      {
        "href":
"https://t16.ldproxy.net/ghcnd/collections/observation/items?f=json&offset=10",
        "rel": "next",
        "type": "application/geo+json",
        "title": "Next page"
      }
    ],
    "numberReturned": 10,
    "timeStamp": "2020-07-25T11:43:19Z",
    "features": [
      {
        "type": "Feature",
        "@type": [
          "geojson:Feature",
          "sosa:Observation"
        ],
        "@id": "https://t16.ldproxy.net/ghcnd/collections/observation/items/1",
        "id": "1",
        "geometry": {
          "type": "Point",
          "coordinates": [
            -80.3111,
            27.3237
          ]
        },
        "properties": {
          "phenomenonTime": "2019-01-01",
          "observedProperty": "PRCP",
          "unitOfMeasurement": "0.1 mm",
          "result": 0,
          "locationCode": "US1FLSL0019",
          "locationLink":
"https://t16.ldproxy.net/ghcnd/collections/station/items/US1FLSL0019",
          "locationName": "PORT ST. LUCIE 4.0 NE"
        }
      }, ...
      {
        "type": "Feature",
        "@type": [
          "geojson:Feature",
          "sosa:Observation"
        ],
        "@id": "https://t16.ldproxy.net/ghcnd/collections/observation/items/10",
          "id": "10",
          "geometry": {
             "type": "Point",
             "coordinates": [
                -97.6697,
                39.5592
             ]
          },
          "properties": {
             "phenomenonTime": "2019-01-01",
             "observedProperty": "SNOW",
             "unitOfMeasurement": "mm",
             "result": 0,
             "locationCode": "USC00141761",
             "locationLink":
"https://t16.ldproxy.net/ghcnd/collections/station/items/USC00141761",
             "locationName": "CONCORDIA 1 W"
          }
       }
    ]
}
```

The response also includes a JSON-LD 1.1 context that relates the JSON values to definitions in the W3C Semantic Sensor Network Ontology [https://www.w3.org/TR/vocab-ssn/] and the GeoJSON vocabulary [https://geojson.org/geojson-ld/]. JSON-LD-aware clients can process the observation data and convert it to RDF.

Query parameters can be used to adjust the request and select specific observations. The following request selects the first 1000 precipitation observations in the border area of Germany, France and Luxembourg during August 2019:

```
https://t16.ldproxy.net/ghcnd/collections/observation/items?
bbox=6,48.5,8,50.5&
datetime=2019-08-01/2019-08-31&
observedProperty=PRCP&
limit=1000
```

The next request selects the maximum temperature at Berlin-Tegel airport on July 29, 2019, with the geometry in the response in the Web Mercator projection.

```
https://t16.ldproxy.net/ghcnd/collections/observation/items?
locationName=BERLIN-TEGEL&
datetime=2019-07-29&
observedProperty=TMAX&
crs=http://www.opengis.net/def/crs/EPSG/0/3857
```

11.4.6. DAPA overview

In addition to the access to the raw observation data, additional patterns to access and process the observation data are available via the DAPA endpoints. Data retrieval in DAPA is based on the selection of the observations by the spatial sampling geometry (point, area, or grid), the temporal sampling geometry (instant or interval), and the observed properties followed by optional post-processing to aggregate observations by space and/or time.

The endpoint provided by interactive instruments covered two use cases described in Use Cases: Data Retrieval and Data Processing. This endpoint processed the data in two phases, which are described in the next sections.

11.4.6.1. Phase 1: Data access

The first phase selected the observations to retrieve. This is in a way similar to the raw data access described above, but while the OGC API Features endpoints can be used with all kinds of features, the DAPA selection pattern is based on the knowledge that the set of all observations can be viewed as a spatio-temporal data cube.

The data was selected based on three aspects:

- The spatial sampling geometry (point, area, or grid),
- The temporal sampling geometry (instant or interval), and
- The observed properties / variables to retrieve.

The resulting data could be requested as CSV, GeoJSON or GeoTIFF (only for grid).

11.4.6.1.1. Spatial sampling geometry

Point

The observations at the location are retrieved. The results are in general interpolated from other observations. The result is a time series for each variable.

The location of the point can be provided in one of two ways:

The first is a Well-Known Text (WKT) point geometry in the parameter `coords`. Example: `POINT(6.9299617 50.000008)`. This was also the default point sampling geometry, if neither `coords` nor `coordsRef` is specified.

The second is a URI in the parameter `coordsRef` where the URI returns a GeoJSON feature, e.g. a request to an API that implements OGC API - Features [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html] or a request to the OpenStreetMap geocoding API [https://nominatim.org/release-docs/develop/api/Overview/]. If the feature geometry is not a point, the centroid is used.

Area

The observations in the area of interest are retrieved. The result is a time series for each variable and each weather station.

The polygon of the area can be provided in one of three ways:

The first is a bounding box in parameter `bbox` consisting of four numbers, separated by commas: minimum longitude, minimum latitude, maximum longitude, maximum latitude. Example: `6,48.5,8,50.5`. If neither `bbox`, `coords` nor `coordsRef` is specified this was also the default area sampling geometry,

The second is a polygon or multi-polygon geometry specified as Well Known Text (WKT) in the parameter `coords`.

The third is a URI in the parameter `coordsRef` where the URI returns a GeoJSON feature, e.g. a request to an API that implements OGC API features. If the feature geometry is not a polygon or multi-polygon, a buffer is added and the bounding box of the resulting geometry is used.

Grid

The observations in the grid area are retrieved and resampled to the spatio-temporal grid. The result is a time series for each variable and each grid cell.

The area of the grid can be provided in one of two ways:

The first is a bounding box in parameter `bbox` consisting of four numbers, separated by commas: minimum longitude, minimum latitude, maximum longitude, maximum latitude. Example: `6,48.5,8,50.5`. If neither `bbox`, `coords` nor `coordsRef` is specified this was also the default area sampling geometry.

The second is a URI in the parameter `coordsRef` where the URI returns a GeoJSON feature, e.g. a request to an API that implements OGC API features. The bounding box of the resulting geometry is used. For example URI see above.

The number of spatial grid cells is determined by parameters `width` and `height`. The default for `width` was `10`. The default for `height` was computed from the grid area and the `width` value.

11.4.6.1.2. Temporal sampling geometry

The temporal sampling geometry is always provided in the parameter `datetime`. Since the dataset consists of daily observations, the API not only supported time stamps, but also dates as values. Intervals are expressed according to ISO 8601. Open intervals are supported, too. Examples:

- `2019-07-29`: July 29,2019
- `2019-08-07/2019-08-11`: August 7-11, 2019 (this was also the default sampling geometry, if `datetime` was not provided)
- `../2019-02-28`: Until February 28, 2019 (since the first observation was on 2019-01-01 this is the same as `2019-01-01/2019-02-28`)
- `2019-11-15/..`: November 15, 2019, and later (since the last observation was on 2019-12-31 this is the same as `2019-11-15/2019-12-31`)

11.4.6.1.3. Observed properties

One or more observed properties or variables could be accessed. The values are provided as a comma-separated list in parameter `variables`.

The supported variables are:

- Precipitation in 0.1mm (`PRCP`)
- Snowfall in mm (`SNOW`)
- Snow depth in mm (`SNWD`)
- Maximum temperature in 0.1°C (`TMAX`)
- Minimum temperature in 0.1°C (`TMIN`)
- Average temperature in 0.1°C (`TAVG`)

The default are all variables, i.e. `PRCP,SNOW,SNWD,TMAX,TMIN,TAVG`.

11.4.6.2. Phase 2: Data aggregation and processing

This is an optional step. If selected, the observations that have been retrieved in phase 1 are aggregated for each variable

- Over time (for point, area or grid sampling geometries),
- Over the whole area (area sampling geometry only), or
- Both (area sampling geometry only).

Note that temporal aggregation is only meaningful if the temporal sampling geometry is an interval and not an instant.

If data is aggregated, the set of values for each variable the functions to apply are specified as a comma-separated list in parameter `functions`. The following functions are available:

- Minimum value (`min`)
- Maximum value (`max`)
- Average/mean value (`mean`)
- Standard deviation (`std-dev`)
- Number of values (`count`)
- Sum (`sum`)

The default value was: `min,max,mean`.

11.4.7. The DAPA landing page: Access information about the available data retrieval patterns

11.4.7.1. Request

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes
```

11.4.7.2. Response in JSON

The response lists the available endpoints to retrieve and process the observations in addition to the standard feature queries. The endpoints are described in the API definition and the links point to the specification of the operation in the OpenAPI definition with the available input parameters and the response schema. The URIs in the "ogc-dapa-endpoint-definition" links are JSON Pointers to the definition of the operation in the OpenAPI definition.

This page is also available as HTML.

```
{
  "title" : "Data retrieval patterns",
  "description" : "The following endpoints are available to retrieve and process the
observations in addition to the standard feature queries. The endpoints are described
in the API definition and the links point to the specification of the operation in the
OpenAPI definition with the available input parameters and the response schema.",
  "links" : [ {
    "rel" : "self",
    "type" : "application/json",
    "title" : "This document",
    "href" :
"http://localhost:7080/rest/services/ghcnd/collections/observation/processes?f=json"
  }, {
    "rel" : "alternate",
    "type" : "text/html",
    "title" : "This document as HTML",
    "href" :
"http://localhost:7080/rest/services/ghcnd/collections/observation/processes?f=html"
  }, {
    "rel" : "ogc-variables",
    "title" : "Observable properties included in this observation collection",
    "href" :
"http://localhost:7080/rest/services/ghcnd/collections/observation/variables"
  } ],
  "endpoints" : [ {
    "title" : "retrieve a time series for selected variables for each station in an
area",
    "description" : "This DAPA endpoint returns observation values at the selected
location (parameter `coords` or `coordsRef`) in the selected time interval or at the
selected time instant (parameter `datetime`).\n\nAll values in the time interval for
each requested variable (parameter `variables`) are aggregated and each of the
requested statistical functions (parameter `functions`) is applied to the aggregated
values.",
    "links" : [ {
      "rel" : "ogc-dapa-endpoint",
      "title" : "Execute the data retrieval pattern with the default parameters",
      "href" :
"http://localhost:7080/rest/services/ghcnd/collections/observation/processes/area"
    }, {
      "rel" : "ogc-dapa-endpoint-definition",
      "title" : "Detailed description in the API definition (OpenAPI 3.0)",
      "href" :
"http://localhost:7080/rest/services/ghcnd/api?f=json#/paths/~1collections~1observatio
n~1processes~1area"
    }, {
      "rel" : "ogc-dapa-endpoint-documentation",
      "title" : "Detailed description in the API documentation",
      "href" :
"http://localhost:7080/rest/services/ghcnd/api?f=html#/DAPA/get_collections_observatio
n_processes_area"
    } ],
    "id" : "area",
    "inputCollectionId" : "observation",
    "mediaTypes" : [ "application/geo+json", "text/csv" ],
    "externalDocs" : {
      "description" : "Example requests and responses",
      "url" : "https://github.com/cportele/ogc-api-dev/blob/master/dapa/06-area.md"
    }
  }, ... {
    "title" : "retrieve a time series for selected variables for each station in an
area, resample the observations to a time series in a 2D grid and apply functions on
the values of each time series",
    "description" : "This DAPA endpoint retrieves a time series for each station in an
area (parameter `box`, `coords` or `coordsRef`) in the selected time interval or at
the selected time instant (parameter `datetime`). Each time series contains daily
observation values for each selected variable (parameter `variables`) for which a
value has been observed at the station during the time interval.\n\nThe data is then
resampled to a time series for each cell in a 2D spatial grid and all values in each
time series are aggregated and each of the requested statistical functions (parameter
`functions`) is applied to the values.",
    "links" : [ {
      "rel" : "ogc-dapa-endpoint",
      "title" : "Execute the data retrieval pattern with the default parameters",
      "href" :
"http://localhost:7080/rest/services/ghcnd/collections/observation/processes/grid:aggr
egate-time"
    }, {
      "rel" : "ogc-dapa-endpoint-definition",
      "title" : "Detailed description in the API definition (OpenAPI 3.0)",
      "href" :
"http://localhost:7080/rest/services/ghcnd/api?f=json#/paths/~1collections~1observatio
n~1dapa~1grid:aggregate-time"
    }, {
      "rel" : "ogc-dapa-endpoint-documentation",
      "title" : "Detailed description in the API documentation",
      "href" :
"http://localhost:7080/rest/services/ghcnd/api?f=html#/DAPA/get_collections_observatio
n_processes_grid_aggregate_time"
    } ],
    "id" : "grid:aggregate-time",
    "inputCollectionId" : "observation",
    "mediaTypes" : [ "application/geo+json", "image/tiff", "text/csv" ],
    "externalDocs" : {
      "description" : "Example requests and responses",
      "url" : "https://github.com/cportele/ogc-api-dev/blob/master/dapa/07-grid.md"
    }
  } ]
}
```

11.4.8. Access information about the available variables

Variables are the properties that have been observed and are included in this collection of observations.

11.4.8.1. Request

```
https://t16.ldproxy.net/ghcnd/collections/observation/variables
```

11.4.8.2. Response in JSON

The response lists the available variables for which observations are available. It is also available as HTML.

```
{
  "links" : [ {
    "rel" : "self",
    "type" : "application/json",
    "title" : "This document",
    "href" : "https://t16.ldproxy.net/ghcnd/collections/observation/variables?f=json"
  }, {
    "rel" : "alternate",
    "type" : "text/html",
    "title" : "This document as HTML",
    "href" : "https://t16.ldproxy.net/ghcnd/collections/observation/variables?f=html"
  } ],
  "variables" : [ {
    "id" : "PRCP",
    "title" : "Precipitation",
    "uom" : "0.1 mm"
  }, {
    "id" : "SNOW",
    "title" : "Snowfall",
    "uom" : "mm"
  }, {
    "id" : "SNWD",
    "title" : "Snow depth",
    "uom" : "mm"
  }, {
    "id" : "TMAX",
    "title" : "Maximum temperature",
    "uom" : "0.1 °C"
  }, {
    "id" : "TMIN",
    "title" : "Minimum temperature",
    "uom" : "0.1 °C"
  }, {
    "id" : "TAVG",
    "title" : "Average temperature",
    "uom" : "0.1 °C"
  } ]
}
```

11.4.9. Retrieve a time series for selected variables at a position

This DAPA endpoint returns a time series with daily observation values at the selected location (parameter `coords` or `coordsRef`) in the selected time interval or at the selected time instant (parameter `datetime`).

The time series contains values for each selected variable (parameter `variables`) for which a value can be interpolated at the location.

11.4.9.1. Sample requests

11.4.9.1.1. Using coordinates for the location, requesting data for an instant

Precipitation and maximum/minimum temperatures on August 9, 2019, in Lieser, Germany.

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/position:retrieve?
coords=POINT(7.0218 49.9174)&
datetime=2019-08-09&
variables=TMAX,TMIN,PRCP
```

11.4.9.1.2. Using coordinates for the location, requesting data for an interval (time series)

Precipitation and maximum/minimum temperatures on 5 days in August 2019, in Lieser, Germany.

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/position:retrieve?
coords=POINT(7.0218 49.9174)&
datetime=2019-08-07/2019-08-11&
variables=TMAX,TMIN,PRCP
```

11.4.9.1.3. Using a reference for the location, requesting data for an instant

Precipitation and maximum/minimum temperatures on August 9, 2019, at a cadastral parcel in Bonn, Germany.

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/position:retrieve?
coordsRef=https%3A%2F%2Fwww.ldproxy.nrw.de%2Fkataster%2Fcollections%2Fflurstueck%2Fite
ms%2FDENW36AL10005X65FL&
datetime=2019-08-09&
variables=TMAX,TMIN,PRCP
```

11.4.9.1.4. Using a reference for the location, requesting data for an interval (time series)

Precipitation and maximum/minimum temperatures in August 2019 in Washington, DC, USA.

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/position:retrieve?
coordsRef=https%3A%2F%2Fnominatim.openstreetmap.org%2Fsearch%3Fq%3DWashington%26format
%3Dgeojson%26limit%3D1&
datetime=2019-08-01/2019-08-31&
variables=TMAX,TMIN,PRCP
```

11.4.9.2. Sample responses

11.4.9.2.1. CSV

At an instant:

```
phenomenonTime,TMAX,TMIN,PRCP
2019-08-09,261.97275,148.28244,175.72902
```

For an interval, a time series:

```
phenomenonTime,TMAX,TMIN,PRCP
2019-08-31,326.22485,186.5174,1.8554863E-5
2019-08-30,299.9434,164.96535,0.2183162
2019-08-29,290.19263,181.47484,10.744431
2019-08-28,275.73703,190.04454,0.39175823
2019-08-27,249.96811,179.19333,1.7285985E-5
2019-08-26,254.65437,169.74426,3.914662E-9
2019-08-25,266.80197,169.72357,0.6220044
2019-08-24,252.26584,169.91318,60.580723
2019-08-23,309.53717,216.67346,46.07018
2019-08-22,348.21063,229.43196,19.694708
2019-08-21,347.32162,218.79291,98.5237
2019-08-20,357.6917,230.24718,47.368004
2019-08-19,356.72153,237.50298,0.0032744503
2019-08-18,342.5798,236.04352,3.8299541
2019-08-17,321.73395,232.62738,0.0013890546
2019-08-16,310.34668,224.90352,12.868282
2019-08-15,308.08978,237.34647,4.08197
2019-08-14,320.06943,233.05789,35.683617
2019-08-13,319.316,233.87689,2.407364
2019-08-12,310.0867,189.36937,1.0333144E-4
2019-08-11,293.19348,183.55244,0.0
2019-08-10,316.1553,179.15393,0.0
2019-08-09,332.17712,229.48021,0.0049290378
2019-08-08,343.6015,212.32523,186.5498
2019-08-07,339.2283,223.82373,145.55368
2019-08-06,325.06027,210.4592,0.006794223
2019-08-05,323.88538,218.16284,6.199424E-5
2019-08-04,331.6571,234.15137,2.254841
2019-08-03,318.93088,222.72612,2.552398E-4
2019-08-02,332.15082,219.27402,0.0
2019-08-01,319.9032,211.88776,0.012168143
```

11.4.9.2.2. GeoJSON

At an instant:

```
{
  "type" : "FeatureCollection",
  "features" : [ {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.0218, 49.9174 ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-09",
      "TMAX" : 261.97275,
      "TMIN" : 148.28244,
      "PRCP" : 175.72902
    }
  } ]
}
```

For an interval, a time series:

```
{
  "type" : "FeatureCollection",
  "features" : [ {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.085770560342503, 50.71640207544834 ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-11",
      "TMAX" : 231.23917,
      "TMIN" : 132.63652,
      "PRCP" : 0.050290037
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.085770560342503, 50.71640207544834 ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-10",
      "TMAX" : 259.5085,
      "TMIN" : 159.90451,
      "PRCP" : 0.0731554
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.085770560342503, 50.71640207544834 ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-09",
      "TMAX" : 275.02377,
      "TMIN" : 149.30214,
      "PRCP" : 108.23
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.085770560342503, 50.71640207544834 ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-08",
      "TMIN" : 134.923,
      "PRCP" : 0.31049046,
      "TMAX" : 237.01088
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.085770560342503, 50.71640207544834 ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-07",
      "TMAX" : 246.96074,
      "TMIN" : 148.34018,
      "PRCP" : 35.399906
    }
  } ]
}
```

11.4.9.3. Example use in python

Below is a simple plot created from the time series data for August 2019 for a location using python.

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

baseUrl = 'https://t16.ldproxy.net/ghcnd/collections/observation/processes'
lon = '7.0218'
lat = '49.9174'
datetime = '2019-07-01/2019-08-31'
variables = 'TMIN,TMAX,PRCP'
# either read as GeoJSON into GeoPandas or as CSV into Pandas
# ghcnd =
gpd.read_file(baseUrl+'/position:retrieve?coords=POINT('+lon+'%20'+lat+')&datetime='+d
atetime+'&variables='+variables+'&f=json')
ghcnd = pd.read_csv(baseUrl+'/position:retrieve?coords=POINT('+lon+'%20'+lat+
')&datetime='+datetime+'&variables='+variables+'&f=csv')
pos = ghcnd
pos.index = pos.phenomenonTime
fig, ax = plt.subplots(figsize=(16, 8))
pos.TMAX.plot(label='max. Temperature [0.1°C]', legend='best').invert_xaxis()
pos.TMIN.plot(label='min. Temperature [0.1°C]', legend='best').invert_xaxis()
pos.PRCP.plot(kind='bar', label='Precipitation [0.1mm]', legend='best').invert_xaxis()
plt.show()
```

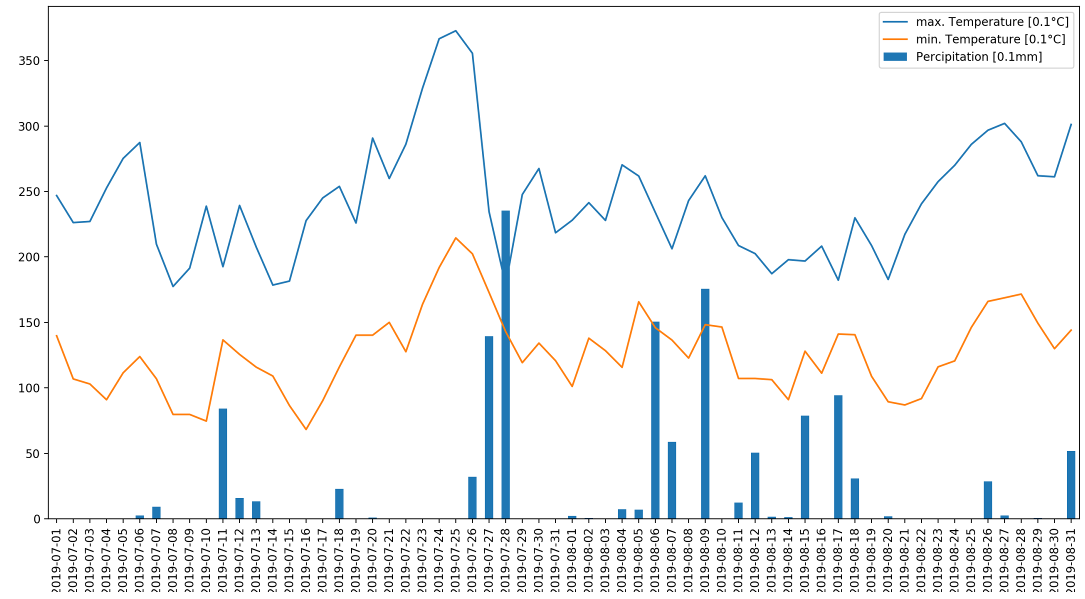


Figure 8. A time series for a position

11.4.10. Retrieve a time series for selected variables for each station in an area

This DAPA endpoint returns a time series for each station in an area (parameter `box`, `coords` or `coordsRef`) in the selected time interval or at the selected time instant (parameter `datetime`).

Each time series contains daily observation values for each selected variable (parameter `variables`) for which a value has been observed at the station during the time interval.

11.4.10.1. Sample requests

11.4.10.1.1. Using a bounding box, requesting data for an instant

The area is a rough bounding box of Germany:

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/area:retrieve?
bbox=5.8,47.2,15.1,55.1&
datetime=2019-08-09&
variables=TMAX,TMIN,PRCP
```

11.4.10.1.2. Using a polygon, requesting data for an instant

The same bounding box as above, but as a polygon:

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/area:retrieve?
coords=POLYGON((5.8 47.2,15.1 47.2,15.1 55.1,5.8 55.1,5.8 47.2))&
datetime=2019-08-09&
variables=TMAX,TMIN,PRCP
```

11.4.10.1.3. Using a reference for the area, requesting data for an instant

The area is the polygon of the German state North-Rhine Westphalia retrieved from another API using a reference:

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/area:retrieve?
coordsRef=https%3A%2F%2Fwww.ldproxy.nrw.de%2Fdvg%2Fcollections%2Fnw_dvg2_bld%2Fitems%2
Fnw_dvg2_bld.05000000&
datetime=2019-08-09&
variables=TMAX,TMIN,PRCP
```

11.4.10.1.4. Using a reference for the area, requesting data for an interval

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/area:retrieve?
coordsRef=https%3A%2F%2Fwww.ldproxy.nrw.de%2Fdvg%2Fcollections%2Fnw_dvg2_bld%2Fitems%2
Fnw_dvg2_bld.05000000&
datetime=2019-08-07/2019-08-11&
variables=TMAX,TMIN,PRCP
```

11.4.10.2. Sample responses

11.4.10.2.1. CSV

A time instant (observations in North-Rhine Westphalia):

```
longitude,latitude,locationCode,locationName,phenomenonTime,TMAX,TMIN,PRCP
7.888899803161621,51.57780075073242,GMM00010424,WERL,2019-08-09,267.0,136.0,15.0
6.535799980163574,51.8307991027832,GME00130294,BOCHOLT,2019-08-09,,,64.0
8.468099594116211,51.19029998779297,GME00111457,ALTASTENBERG,2019-08-
09,208.0,146.0,88.0
6.968299865722656,51.405601501464844,GME00121978,ESSEN-BREDENEY,2019-08-
09,254.0,165.0,3.0
6.88670015335083,51.87419891357422,GME00131182,BORKEN IN WESTFALEN,2019-08-
09,242.0,125.0,17.0
6.246699810028076,51.495601654052734,GME00122746,GELDERN-WALBECK,2019-08-
09,258.0,160.0,26.0
6.424200057983398,50.67559814453125,GME00126694,NIDEGGEN-SCHMIDT,2019-08-
09,259.0,144.0,101.0
7.341700077056885,51.33420181274414,GME00122146,GEVELSBERG-OBERBROKING,2019-08-
09,246.0,142.0,39.0
6.445000171661377,51.29079818725586,GME00129346,TONISVORST,2019-08-09,264.0,160.0,25.0
8.395299911499023,51.6343994140625,GME00125422,LIPPSTADT-BOKENFORDE,2019-08-
09,278.0,132.0,11.0
6.7916998863220215,50.71310043334961,GME00122002,WEILERSWIST-LOMMERSUM,2019-08-
09,272.0,146.0,76.0
8.839400291442871,51.78670120239258,GME00121078,BAD LIPPSPRINGE,2019-08-
09,266.0,126.0,5.0
9.112799644470215,51.50529861450195,GME00129886,WARBURG,2019-08-09,263.0,128.0,34.0
8.573100090026855,52.44810104370117,GME00127558,RAHDEN-VARL,2019-08-
09,276.0,105.0,12.0
6.659200191497803,50.82889938354492,GME00126766,NORVENICH (FLUGPLATZ),2019-08-
09,,,96.0
7.094200134277344,51.14080047607422,GME00128902,SOLINGEN-HOHENSCHEID,2019-08-
09,,,109.0
7.520299911499023,51.76810073852539,GME00131518,LUDINGHAUSEN-BROCHTRUP,2019-08-
09,276.0,119.0,0.0
7.158299922943115,50.86579895019531,GME00121042,KOLN-BONN,2019-08-09,266.0,159.0,129.0
6.701900005340576,51.5099983215332,GME00122098,DUISBURG-BAERL,2019-08-
09,274.0,173.0,12.0
6.095600128173828,51.76250076293945,GME00124606,KLEVE,2019-08-09,237.0,152.0,96.0
8.457799911499023,52.07279968261719,GME00131830,BIELEFELD-DEPPENDORF,2019-08-
09,269.0,93.0,2.0
5.983099937438965,50.98310089111328,NLE00109280,SCHINVELD,2019-08-09,,,31.0
7.978899955749512,51.46440124511719,GME00131974,ARNSBERG-NEHEIM,2019-08-
09,277.0,117.0,14.0
6.941699981689453,52.082801818847656,GMM00010309,AHAUS,2019-08-09,236.0,106.0,81.0
7.105800151824951,51.225799560546875,GME00130582,WUPPERTAL-BUCHENHOFEN,2019-08-
09,242.0,137.0,40.0
8.65060043334961,51.41559982299805,GME00131374,BRILON-THULEN,2019-08-
09,251.0,138.0,26.0
8.753100395202637,52.10559844970703,GME00128182,BAD SALZUFLEN,2019-08-
09,264.0,120.0,4.0
6.025000095367432,50.799198150634766,GME00120958,AACHEN-ORSBACH,2019-08-
09,265.0,164.0,77.0
8.156900405883789,51.25529861450195,GME00121966,ESLOHE,2019-08-09,245.0,118.0,49.0
8.035799980163574,51.1343994140625,GME00125230,LENNESTADT-THETEN,2019-08-
```

```
09,252.0,124.0,58.0
7.3719000816345215,50.84560012817383,GME00126562,NEUNKIRCHEN-SEELSCHEID-KRAWINK,2019-
08-09,250.0,156.0,125.0
6.104400157928467,51.04249954223633,GME00123574,HEINSBERG-SCHLEIDEN,2019-08-
09,248.0,163.0,34.0
7.405600070953369,51.59809875488281,GME00122134,WALTROP-ABDINGHOF,2019-08-
09,270.0,134.0,4.0
8.355299949645996,51.88890075683594,GME00123178,GUTERSLOH,2019-08-09,,,9.0
6.526700019836426,50.50279998779297,GME00124366,KALL-SISTIG,2019-08-
09,239.0,139.0,135.0
8.368900299072266,50.98500061035156,GME00127258,BAD-STUNZEL BERLEBURG,2019-08-
09,225.0,127.0,148.0
6.978300094604492,50.9906005859375,GME00125302,KOLN-STAMMHEIM,2019-08-
09,262.0,168.0,67.0
7.643099784851074,51.246700286865234,GME00125602,LUDENSCHEID,2019-08-
09,242.0,134.0,24.0
6.769999980926514,51.29690170288086,GME00102268,DUSSELDORF,2019-08-09,264.0,161.0,7.0
7.62939977645874,51.09080123901367,GME00122182,MEINERZHAGEN-REDLENDORF,2019-08-
09,232.0,119.0,93.0
9.271900177001953,51.867801666259766,GME00131242,LUGDE-PAENBRUCH,2019-08-
09,262.0,118.0,9.0
7.193900108337402,50.73640060424805,GME00130990,BONN-ROLEBER,2019-08-
09,275.0,157.0,127.0
7.699999809265137,52.13529968261719,GME00111430,MUENSTER/OSNABRUECK (AIRPORT),2019-08-
09,265.0,121.0,20.0
```

A time series for each station (the response has been shortened to four stations):

```
longitude,latitude,locationCode,locationName,phenomenonTime,TMAX,TMIN,PRCP
7.888899803161621,51.57780075073242,GMM00010424,WERL,2019-08-11,244.0,138.0,0.0
7.888899803161621,51.57780075073242,GMM00010424,WERL,2019-08-10,257.0,181.0,0.0
7.888899803161621,51.57780075073242,GMM00010424,WERL,2019-08-09,267.0,136.0,15.0
7.888899803161621,51.57780075073242,GMM00010424,WERL,2019-08-08,257.0,143.0,0.0
7.888899803161621,51.57780075073242,GMM00010424,WERL,2019-08-07,252.0,147.0,0.0
6.535799980163574,51.8307991027832,GME00130294,BOCHOLT,2019-08-11,,,0.0
6.535799980163574,51.8307991027832,GME00130294,BOCHOLT,2019-08-10,,,10.0
6.535799980163574,51.8307991027832,GME00130294,BOCHOLT,2019-08-09,,,64.0
6.535799980163574,51.8307991027832,GME00130294,BOCHOLT,2019-08-08,,,0.0
6.535799980163574,51.8307991027832,GME00130294,BOCHOLT,2019-08-07,,,0.0
...
7.193900108337402,50.73640060424805,GME00130990,BONN-ROLEBER,2019-08-
11,231.0,143.0,0.0
7.193900108337402,50.73640060424805,GME00130990,BONN-ROLEBER,2019-08-
10,256.0,180.0,0.0
7.193900108337402,50.73640060424805,GME00130990,BONN-ROLEBER,2019-08-
09,275.0,157.0,127.0
7.193900108337402,50.73640060424805,GME00130990,BONN-ROLEBER,2019-08-
08,232.0,146.0,0.0
7.193900108337402,50.73640060424805,GME00130990,BONN-ROLEBER,2019-08-
07,243.0,154.0,0.0
7.699999809265137,52.13529968261719,GME00111430,MUENSTER/OSNABRUECK (AIRPORT),2019-08-
11,240.0,166.0,0.0
7.699999809265137,52.13529968261719,GME00111430,MUENSTER/OSNABRUECK (AIRPORT),2019-08-
10,259.0,184.0,0.0
7.699999809265137,52.13529968261719,GME00111430,MUENSTER/OSNABRUECK (AIRPORT),2019-08-
09,265.0,121.0,20.0
7.699999809265137,52.13529968261719,GME00111430,MUENSTER/OSNABRUECK (AIRPORT),2019-08-
08,256.0,125.0,0.0
7.699999809265137,52.13529968261719,GME00111430,MUENSTER/OSNABRUECK (AIRPORT),2019-08-
07,257.0,165.0,0.0
```

11.4.10.2.2. GeoJSON

The response of the first request (time instant), shortened to two of many features:

```
{
  "type" : "FeatureCollection",
  "features" : [ {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 11.7856, 50.7294 ]
    },
    "properties" : {
      "locationCode" : "GME00126070",
      "locationName" : "SCHMIERITZ-WELTWITZ",
      "phenomenonTime" : "2019-08-09",
      "TMAX" : 297.0,
      "TMIN" : 127.0,
      "PRCP" : 0.0
    }
  }, ... {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 6.2589, 52.4342 ]
    },
    "properties" : {
      "locationCode" : "NLE00152479",
      "locationName" : "HEINO",
      "phenomenonTime" : "2019-08-09",
      "TMAX" : 221.0,
      "TMIN" : 108.0,
      "PRCP" : 53.0
    }
  } ]
}
```

Data for an interval, i.e. a time series at each station (the response is shortened to data for 2 stations):

```
{
  "type" : "FeatureCollection",
  "features" : [ {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.888899803161621, 51.57780075073242 ]
    },
    "properties" : {
      "locationCode" : "GMM00010424",
      "locationName" : "WERL",
      "phenomenonTime" : "2019-08-11",
      "TMAX" : 244.0,
      "TMIN" : 138.0,
      "PRCP" : 0.0
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.888899803161621, 51.57780075073242 ]
    },
    "properties" : {
      "locationCode" : "GMM00010424",
      "locationName" : "WERL",
      "phenomenonTime" : "2019-08-10",
      "TMAX" : 257.0,
      "TMIN" : 181.0,
      "PRCP" : 0.0
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.888899803161621, 51.57780075073242 ]
    },
    "properties" : {
      "locationCode" : "GMM00010424",
      "locationName" : "WERL",
      "phenomenonTime" : "2019-08-09",
      "TMIN" : 136.0,
      "PRCP" : 15.0,
      "TMAX" : 267.0
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.888899803161621, 51.57780075073242 ]
    },
    "properties" : {
      "locationCode" : "GMM00010424",
      "locationName" : "WERL",
      "phenomenonTime" : "2019-08-08",
      "TMIN" : 143.0,
      "PRCP" : 0.0,
      "TMAX" : 257.0
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.888899803161621, 51.57780075073242 ]
    },
      "properties" : {
        "locationCode" : "GMM00010424",
        "locationName" : "WERL",
        "phenomenonTime" : "2019-08-07",
        "TMIN" : 147.0,
        "PRCP" : 0.0,
        "TMAX" : 252.0
      }
    }, ... {
      "type" : "Feature",
      "geometry" : {
        "type" : "Point",
        "coordinates" : [ 7.699999809265137, 52.13529968261719 ]
      },
      "properties" : {
        "locationCode" : "GME00111430",
        "locationName" : "MUENSTER/OSNABRUECK (AIRPORT)",
        "phenomenonTime" : "2019-08-11",
        "TMIN" : 166.0,
        "PRCP" : 0.0,
        "TMAX" : 240.0
      }
    }, {
      "type" : "Feature",
      "geometry" : {
        "type" : "Point",
        "coordinates" : [ 7.699999809265137, 52.13529968261719 ]
      },
      "properties" : {
        "locationCode" : "GME00111430",
        "locationName" : "MUENSTER/OSNABRUECK (AIRPORT)",
        "phenomenonTime" : "2019-08-10",
        "TMIN" : 184.0,
        "PRCP" : 0.0,
        "TMAX" : 259.0
      }
    }, {
      "type" : "Feature",
      "geometry" : {
        "type" : "Point",
        "coordinates" : [ 7.699999809265137, 52.13529968261719 ]
      },
      "properties" : {
        "locationCode" : "GME00111430",
        "locationName" : "MUENSTER/OSNABRUECK (AIRPORT)",
        "phenomenonTime" : "2019-08-09",
        "TMIN" : 121.0,
        "PRCP" : 20.0,
        "TMAX" : 265.0
      }
    }, {
```

```
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.699999809265137, 52.13529968261719 ]
    },
    "properties" : {
      "locationCode" : "GME00111430",
      "locationName" : "MUENSTER/OSNABRUECK (AIRPORT)",
      "phenomenonTime" : "2019-08-08",
      "TMIN" : 125.0,
      "PRCP" : 0.0,
      "TMAX" : 256.0
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.699999809265137, 52.13529968261719 ]
    },
    "properties" : {
      "locationCode" : "GME00111430",
      "locationName" : "MUENSTER/OSNABRUECK (AIRPORT)",
      "phenomenonTime" : "2019-08-07",
      "TMIN" : 165.0,
      "PRCP" : 0.0,
      "TMAX" : 257.0
    }
  } ]
}
```

11.4.11. Retrieve a time series for selected variables for each station in an area and resample the observations to a time series for each cell in a 2D grid

This DAPA endpoint retrieves a time series for each station in an area (parameter `box`, `coords` or `coordsRef`) in the selected time interval or at the selected time instant (parameter `datetime`). Each time series contains daily observation values for each selected variable (parameter `variables`) for which a value has been observed at the station during the time interval.

The data is then resampled to a time series for each cell in a 2D spatial grid.

11.4.11.1. Sample request

In the example we use a small grid (width = 20 cells, height is computed) and only request data at a time instant

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/grid:retrieve?
bbox=5.8,47.2,15.1,55.1&
datetime=2019-08-09&
variables=TMAX,TMIN,PRCP&
width=20
```

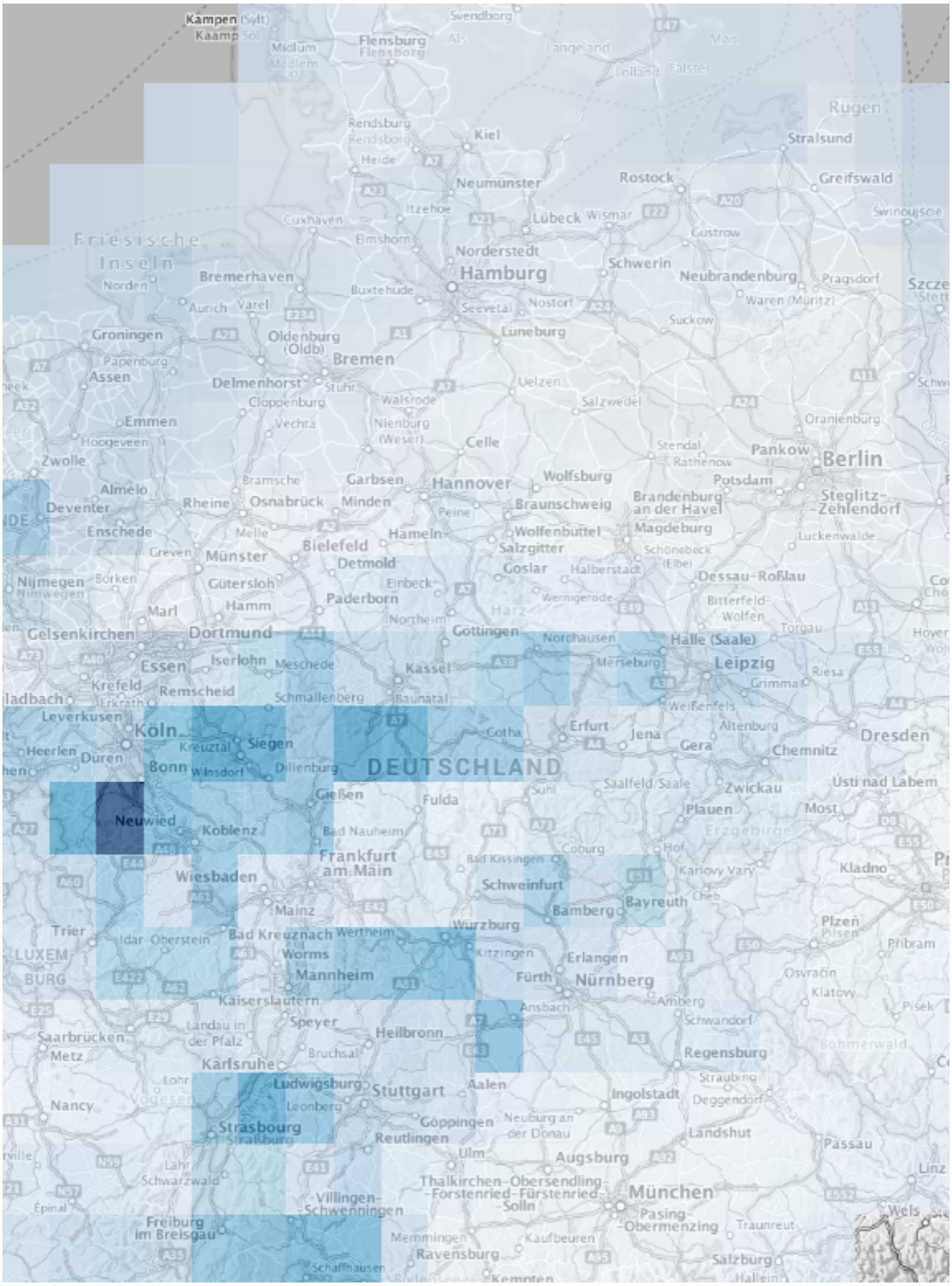
11.4.11.2. Sample responses

11.4.11.2.1. CSV

The response has been shortened to the first column:

```
longitude,latitude,phenomenonTime,TMAX,TMIN,PRCP
5.8,54.17058823529412,2019-08-09,208.3371,165.58093,
5.8,53.70588235294118,2019-08-09,212.30658,154.50684,69.43805
5.8,53.241176470588236,2019-08-09,216.48091,108.501335,44.1395
5.8,52.7764705882353,2019-08-09,221.22427,108.43177,38.235477
5.8,52.311764705882354,2019-08-09,229.13065,137.56642,114.42797
5.8,51.847058823529416,2019-08-09,246.9007,155.55235,73.69644
5.8,51.38235294117647,2019-08-09,253.04602,164.75665,24.300041
5.8,50.91764705882353,2019-08-09,255.91014,162.96167,106.63208
5.8,50.45294117647059,2019-08-09,257.63416,165.82362,81.62919
5.8,49.98823529411765,2019-08-09,265.50992,166.86095,63.125282
5.8,49.523529411764706,2019-08-09,274.08966,162.93968,45.84553
5.8,49.05882352941177,2019-08-09,280.13544,160.14008,19.011019
5.8,48.59411764705882,2019-08-09,284.58804,,12.559709
5.8,48.129411764705885,2019-08-09,286.05768,,34.865925
5.8,47.66470588235295,2019-08-09,281.6034,,62.27944
...
```

11.4.11.2.2. GeoTIFF



11.4.11.2.3. GeoJSON

The response is shortened to two cells.

```
{
  "type" : "FeatureCollection",
  "features" : [ {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 5.8, 50.91765 ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-09",
      "PRCP" : 26.634087
    }
  }, ... {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 14.610527, 52.77647 ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-09",
      "TMAX" : 263.56512,
      "PRCP" : 2.2810037
    }
  } ]
}
```

11.4.12. Retrieve a time series for selected variables at a position and apply functions on the values for each variable

This DAPA endpoint returns observation values at the selected location (parameter `coords` or `coordsRef`) in the selected time interval or at the selected time instant (parameter `datetime`).

All values in the time interval for each requested variable (parameter `variables`) are aggregated and each of the requested statistical functions (parameter `functions`) is applied to the aggregated values.

11.4.12.1. Sample request

Precipitation and maximum/minimum temperatures on 5 days in August 2019, in Lieser, Germany. The functions min(), max(), mean(), count() and std-dev() are applied to the time series.

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/position:aggregate-
time?
coords=POINT(7.0218 49.9174)&
datetime=2019-08-07/2019-08-11&
variables=TMAX,TMIN,PRCP&
functions=min,max,mean,count,std-dev
```

11.4.12.2. Sample responses

11.4.12.2.1. CSV

```
PRCP_count,PRCP_max,PRCP_mean,PRCP_min,PRCP_std-
dev,TMAX_count,TMAX_max,TMAX_mean,TMAX_min,TMAX_std-
dev,TMIN_count,TMIN_max,TMIN_mean,TMIN_min,TMIN_std-dev
5,175.7282,49.449417,0.1784924,74.594795,5,261.97256,230.02454,206.25876,23.499502,5,1
48.28212,132.24287,107.17521,17.294535
```

11.4.12.2.2. GeoJSON

```
{
  "type" : "FeatureCollection",
  "features" : [ {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 7.0218, 49.9174 ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-07/2019-08-11",
      "TMAX_mean" : 230.02454,
      "TMIN_std-dev" : 17.294535,
      "TMIN_max" : 148.28212,
      "TMIN_count" : 5,
      "TMAX_count" : 5,
      "PRCP_mean" : 49.449417,
      "TMAX_max" : 261.97256,
      "PRCP_min" : 0.1784924,
      "TMAX_min" : 206.25876,
      "PRCP_count" : 5,
      "TMAX_std-dev" : 23.499502,
      "TMIN_min" : 107.17521,
      "PRCP_std-dev" : 74.594795,
      "TMIN_mean" : 132.24287
    }
  } ]
}
```

11.4.13. Retrieve a time series for selected variables for each station in an area and apply functions on the values of each time series

This DAPA endpoint returns observation values for an area (parameter `bbox`, `coords` or `coordsRef`) in the selected time interval or at the selected time instant (parameter `datetime`).

All values for each requested variable (parameter `variables`) are aggregated for each weather station and each of the requested statistical functions (parameter `functions`) is applied to the aggregated values.

11.4.13.1. Sample request

Precipitation and maximum/minimum temperatures on 5 days in August 2019 over Germany. The functions min(), max(), mean(), count() and std-dev() are applied to the time series of each weather station in the area.

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/area:aggregate-time?
coords=POLYGON((5.8 47.2,15.1 47.2,15.1 55.1,5.8 55.1,5.8 47.2))&
datetime=2019-08-07/2019-08-11&
variables=TMAX,TMIN,PRCP&
function=min,max,mean,count,std-dev
```

11.4.13.2. Sample responses

11.4.13.2.1. CSV

The response is shortened to two stations.

```
longitude,latitude,locationCode,locationName,PRCP_count,PRCP_max,PRCP_mean,PRCP_min,PR
CP_std-dev,TMAX_count,TMAX_max,TMAX_mean,TMAX_min,TMAX_std-
dev,TMIN_count,TMIN_max,TMIN_mean,TMIN_min,TMIN_std-dev
11.785599708557129,50.729400634765625,GME00126070,SCHMIERITZ-
WELTWITZ,5,59.0,11.8,0.0,26.385603,5,297.0,256.2,204.0,36.16905,5,170.0,144.8,127.0,17
.626684
...
6.258900165557861,52.434200286865234,NLE00152479,HEINO,5,53.0,18.8,0.0,26.090227,5,235
.0,229.2,221.0,5.6302752,5,170.0,135.0,108.0,26.1247
```

11.4.13.2.2. GeoJSON

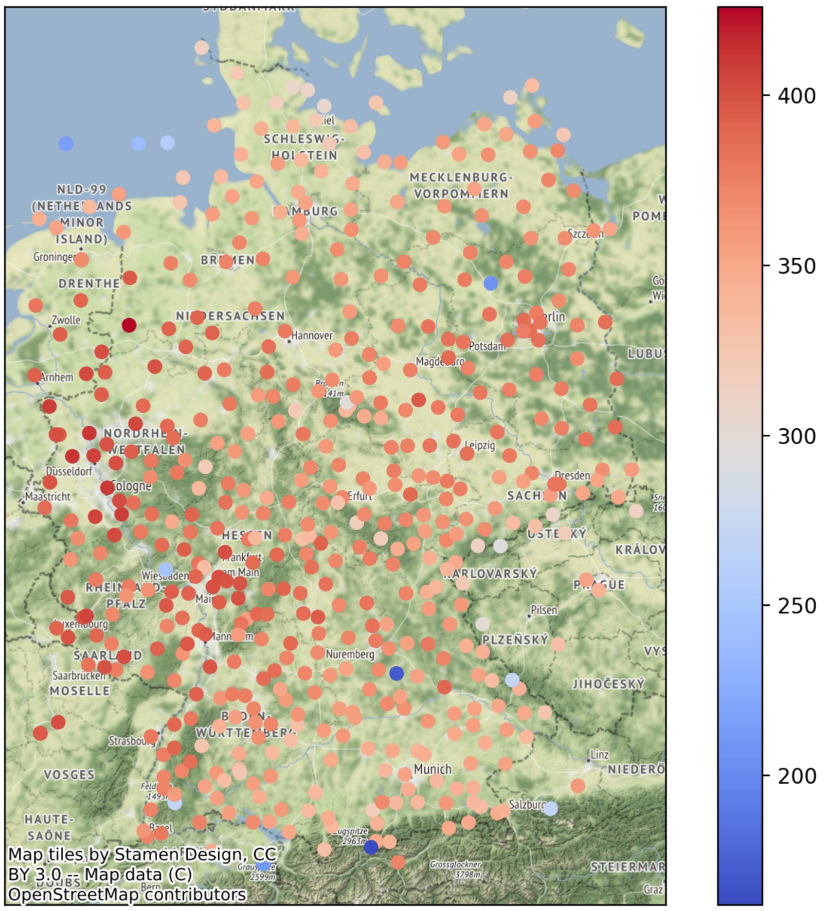
The response is shortened to two stations.

```
{
  "type" : "FeatureCollection",
  "features" : [ {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 11.785599708557129, 50.729400634765625 ]
    },
    "properties" : {
      "locationCode" : "GME00126070",
      "locationName" : "SCHMIERITZ-WELTWITZ",
      "phenomenonTime" : "2019-08-07/2019-08-11",
      "TMAX_mean" : 256.2,
      "TMIN_std-dev" : 17.626684,
      "TMIN_max" : 170.0,
      "TMIN_count" : 5,
      "TMAX_count" : 5,
      "TMAX_max" : 297.0,
      "PRCP_mean" : 11.8,
      "PRCP_min" : 0.0,
      "PRCP_count" : 5,
      "TMIN_min" : 127.0,
      "PRCP_std-dev" : 26.385603,
      "TMAX_min" : 204.0,
      "PRCP_max" : 59.0,
      "TMAX_std-dev" : 36.16905,
      "TMIN_mean" : 144.8
    }
  }, ... {
   "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 6.258900165557861, 52.434200286865234 ]
    },
    "properties" : {
      "locationCode" : "NLE00152479",
      "locationName" : "HEINO",
      "phenomenonTime" : "2019-08-07/2019-08-11",
      "TMAX_mean" : 229.2,
      "TMIN_std-dev" : 26.1247,
      "TMIN_max" : 170.0,
      "TMIN_count" : 5,
      "TMAX_count" : 5,
      "PRCP_mean" : 18.8,
      "TMAX_max" : 235.0,
      "PRCP_min" : 0.0,
      "TMAX_min" : 221.0,
      "PRCP_max" : 53.0,
      "PRCP_count" : 5,
      "TMIN_min" : 108.0,
      "PRCP_std-dev" : 26.090227,
      "TMAX_std-dev" : 5.6302752,
      "TMIN_mean" : 135.0
    }
  } ]
}
```

11.4.13.3. Example use in python

Below is a simple map created from the time series data for 2019 for a bounding box of Germany using python.

```
import geopandas as gpd
import matplotlib.pyplot as plt
import contextily as ctx

baseUrl = 'https://t16.ldproxy.net/ghcnd/collections/observation/processes'
bbox = '5.8,47.2,15.1,55.1'
datetime = '2019-01-01/2019-12-31'
variables = 'TMAX'
ghcnd = gpd.read_file(baseUrl+'/area:aggregate-time?bbox='+bbox+'&datetime='+datetime
+'&variables='+variables+'&f=json').to_crs(epsg=3857)
ax2 = ghcnd.plot(column='TMAX_max', legend='best', cmap='coolwarm')
ax2.tick_params(axis='both', which='both', left=False, right=False, bottom=False, top
=False, labelleft=False, labelbottom=False)
ctx.add_basemap(ax2)
plt.show()
```



Map tiles by Stamen Design, CC BY 3.0 -- Map data (C) OpenStreetMap contributors
400
350
300
250
200

Figure 9. Maximum temperatures in Germany in 2019

11.4.14. Retrieve a time series for selected variables for each station in an area and apply functions on the values of each time step

This DAPA endpoint returns a time series for an area (parameter `bbox`, `coords` or `coordsRef`) in the selected time interval or at the selected time instant (parameter `datetime`).

All values in the area for each requested variable (parameter `variables`) are aggregated for each time step and each of the requested statistical functions (parameter `functions`) is applied to the aggregated values.

11.4.14.1. Sample request

Precipitation and maximum/minimum temperatures on 5 days in August 2019 over Germany. The functions min(), max(), mean(), count() and std-dev() are applied to the values for each day.

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/area:aggregate-space?
coords=POLYGON((5.8 47.2,15.1 47.2,15.1 55.1,5.8 55.1,5.8 47.2))&
datetime=2019-08-07/2019-08-11&
variables=TMAX,TMIN,PRCP&
functions=min,max,mean,count,std-dev
```

11.4.14.2. Sample responses

11.4.14.2.1. CSV

```
phenomenonTime,PRCP_count,PRCP_max,PRCP_mean,PRCP_min,PRCP_std-
dev,TMAX_count,TMAX_max,TMAX_mean,TMAX_min,TMAX_std-
dev,TMIN_count,TMIN_max,TMIN_mean,TMIN_min,TMIN_std-dev
2019-08-
11,709,281.0,26.931225,0.0,45.49806,485,306.0,258.65854,115.0,27.334553,486,187.0,137.
17947,58.0,20.78836
2019-08-
10,709,229.0,12.449949,0.0,26.586481,485,305.0,245.46971,99.0,24.421343,486,206.0,157.
32607,61.0,20.63871
2019-08-
09,709,431.0,55.968887,0.0,56.121117,485,351.0,277.59433,134.0,34.76338,486,188.0,128.
55798,52.0,21.31758
2019-08-
08,709,439.0,5.9656353,0.0,25.032032,485,294.5,248.36009,67.0,23.795979,486,178.0,131.
79442,20.0,16.903353
2019-08-
07,709,490.0,63.96998,0.0,84.83652,485,285.0,225.75258,73.0,26.090208,486,177.53384399
414062,144.37961,27.0,16.778051
```

11.4.14.2.2. GeoJSON

```
{
  "type" : "FeatureCollection",
  "features" : [ {
    "type" : "Feature",
    "geometry" : {
      "type" : "Polygon",
      "coordinates" : [ [ [ 5.8, 47.2 ], [ 15.1, 47.2 ], [ 15.1, 55.1 ], [ 5.8, 55.1
], [ 5.8, 47.2 ] ] ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-11",
      "TMAX_mean" : 258.65854,
      "TMIN_std-dev" : 20.78836,
      "TMIN_max" : 187.0,
      "TMAX_max" : 306.0,
      "PRCP_min" : 0.0,
      "TMIN_count" : 486,
      "TMAX_count" : 485,
      "PRCP_mean" : 26.931225,
      "TMAX_min" : 115.0,
      "PRCP_max" : 281.0,
      "PRCP_count" : 709,
      "TMIN_min" : 58.0,
      "PRCP_std-dev" : 45.49806,
      "TMAX_std-dev" : 27.334553,
      "TMIN_mean" : 137.17947
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Polygon",
      "coordinates" : [ [ [ 5.8, 47.2 ], [ 15.1, 47.2 ], [ 15.1, 55.1 ], [ 5.8, 55.1
], [ 5.8, 47.2 ] ] ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-10",
      "TMAX_mean" : 245.46971,
      "TMIN_std-dev" : 20.63871,
      "TMIN_max" : 206.0,
      "TMIN_count" : 486,
      "TMAX_count" : 485,
      "PRCP_mean" : 12.449949,
      "TMAX_max" : 305.0,
      "PRCP_min" : 0.0,
      "TMAX_min" : 99.0,
      "PRCP_max" : 229.0,
      "TMIN_min" : 61.0,
      "PRCP_std-dev" : 26.586481,
      "TMAX_std-dev" : 24.421343,
      "TMIN_mean" : 157.32607
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Polygon",
      "coordinates" : [ [ [ 5.8, 47.2 ], [ 15.1, 47.2 ], [ 15.1, 55.1 ], [ 5.8, 55.1
], [ 5.8, 47.2 ] ] ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-09",
      "TMAX_mean" : 277.59433,
      "TMIN_std-dev" : 21.31758,
      "TMIN_max" : 188.0,
      "TMIN_count" : 486,
      "TMAX_count" : 485,
      "PRCP_mean" : 55.968887,
      "TMAX_max" : 351.0,
      "PRCP_min" : 0.0,
      "PRCP_count" : 709,
      "TMIN_min" : 52.0,
      "PRCP_std-dev" : 56.121117,
      "TMAX_min" : 134.0,
      "PRCP_max" : 431.0,
      "TMAX_std-dev" : 34.76338,
      "TMIN_mean" : 128.55798
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Polygon",
      "coordinates" : [ [ [ 5.8, 47.2 ], [ 15.1, 47.2 ], [ 15.1, 55.1 ], [ 5.8, 55.1
], [ 5.8, 47.2 ] ] ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-08",
      "TMAX_mean" : 248.36009,
      "TMIN_std-dev" : 16.903353,
      "TMIN_max" : 178.0,
      "TMIN_count" : 486,
      "TMAX_count" : 485,
      "PRCP_mean" : 5.9656353,
      "TMAX_max" : 294.5,
      "PRCP_min" : 0.0,
      "TMAX_std-dev" : 23.795979,
      "TMIN_mean" : 131.79442,
      "TMAX_min" : 67.0,
      "PRCP_max" : 439.0,
      "PRCP_count" : 709,
      "TMIN_min" : 20.0,
      "PRCP_std-dev" : 25.032032
    }
  }, {
    "type" : "Feature",
    "geometry" : {
      "type" : "Polygon",
      "coordinates" : [ [ [ 5.8, 47.2 ], [ 15.1, 47.2 ], [ 15.1, 55.1 ], [ 5.8, 55.1
], [ 5.8, 47.2 ] ] ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-07",
      "TMAX_mean" : 225.75258,
      "TMIN_std-dev" : 16.778051,
      "TMIN_max" : 177.53384,
      "TMIN_count" : 486,
      "TMAX_count" : 485,
      "PRCP_mean" : 63.96998,
      "TMAX_max" : 285.0,
      "PRCP_min" : 0.0,
      "PRCP_count" : 709,
      "TMIN_min" : 27.0,
      "PRCP_std-dev" : 84.83652,
      "TMAX_min" : 73.0,
      "PRCP_max" : 490.0,
      "TMAX_std-dev" : 26.090208,
      "TMIN_mean" : 144.37961
    }
  } ]
}
```

11.4.15. Retrieve a time series for selected variables for each station in an area and apply functions on all values

This DAPA endpoint returns observation values for an area (parameter `bbox`, `coords` or `coordsRef`) in the selected time interval or at the selected time instant (parameter `datetime`).

All values for each requested variable (parameter `variables`) are aggregated and each of the requested statistical functions (parameter `functions`) is applied to the aggregated values.

11.4.15.1. Sample request

Retrieve all observations in the state North-Rhine Westphalia in Germany for August 2019 and apply the functions min(), max(), mean(), count() and std-dev().

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/area:aggregate-space-
time?
coordsRef=https%3A%2F%2Fwww.ldproxy.nrw.de%2Fdvg%2Fcollections%2Fnw_dvg2_bld%2Fitems%2
Fnw_dvg2_bld.05000000&
datetime=2019-08-01/2019-08-31&
variables=TMAX,TMIN,PRCP&
function=min,max,mean,count,std-dev
```

11.4.15.2. Sample responses

11.4.15.2.1. CSV

```
PRCP_count,PRCP_max,PRCP_mean,PRCP_min,PRCP_std-
dev,TMAX_count,TMAX_max,TMAX_mean,TMAX_min,TMAX_std-
dev,TMIN_count,TMIN_max,TMIN_mean,TMIN_min,TMIN_std-dev
1333,443.0,18.745686,0.0,42.16499,1178,350.0,251.22072,139.0,39.133766,1178,218.0,128.
9236,46.0,29.123257
```

11.4.15.2.2. GeoJSON

The geometry has been shortened.

```
{
  "type" : "FeatureCollection",
  "features" : [ {
    "type" : "Feature",
    "geometry" : {
      "type" : "Polygon",
      "coordinates" : [ [ [ 6.094259257869, 51.616130588555 ], [ 6.093923825039,
51.622147469348 ], ... ] ]
    },
    "properties" : {
      "phenomenonTime" : "2019-08-01/2019-08-31",
      "TMIN_count" : 1178,
      "TMAX_count" : 1178,
      "PRCP_mean" : 18.745686,
      "TMAX_max" : 350.0,
      "PRCP_min" : 0.0,
      "TMAX_min" : 139.0,
      "PRCP_max" : 443.0,
      "TMAX_std-dev" : 39.133766,
      "TMIN_mean" : 128.9236,
      "PRCP_count" : 1333,
      "TMIN_min" : 46.0,
      "PRCP_std-dev" : 42.16499,
      "TMAX_mean" : 251.22072,
      "TMIN_std-dev" : 29.123257,
      "TMIN_max" : 218.0
    }
  } ]
}
```

11.4.16. Retrieve a time series for selected variables for each station in an area, resample the observations to a time series in a 2D grid and on the values of each time series

This DAPA endpoint retrieves a time series for each station in an area (parameter `box`, `coords` or `coordsRef`) in the selected time interval or at the selected time instant (parameter `datetime`). Each time series contains daily observation values for each selected variable (parameter `variables`) for which a value has been observed at the station during the time interval.

The data is then resampled to a time series for each cell in a 2D spatial grid and all values in each time series are aggregated and each of the requested statistical functions (parameter `functions`) is applied to the values.

11.4.16.1. Sample request

Generate a grid of the maximum temperatures in Germany during 2019. The grid has 20 columns.

```
https://t16.ldproxy.net/ghcnd/collections/observation/processes/grid:aggregate-time?
bbox=5.8,47.2,15.1,55.1&
datetime=2019-01-01/2019-12-31&
variables=TMAX&
width=20&
functions=max
```

11.4.16.2. Sample responses

11.4.16.2.1. CSV

The response has been shortened to the first column.

```
longitude,latitude,TMAX_max
5.8,54.17058823529412,302.4419
5.8,53.70588235294118,346.7035
5.8,53.241176470588236,358.9652
5.8,52.7764705882353,386.4441
5.8,52.311764705882354,396.17108
5.8,51.847058823529416,403.03824
5.8,51.38235294117647,395.76236
5.8,50.91764705882353,386.07953
5.8,50.45294117647059,390.37183
5.8,49.98823529411765,392.23697
5.8,49.523529411764706,393.08337
5.8,49.05882352941177,394.32608
5.8,48.59411764705882,396.6705
5.8,48.129411764705885,391.1083
5.8,47.66470588235295,384.11874
...
```

11.4.16.2.2. GeoTIFF

GeoTIFF file with maximum temperatures over Germany in 2019 [12-TMAX-Germany-2019.tiff]

11.4.16.2.3. GeoJSON

The response is shortened to two cells.

```
{
  "type" : "FeatureCollection",
  "features" : [ {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 5.8, 54.17058823529412 ]
    },
    "properties" : {
      "phenomenonTime" : "2019-01-01/2019-12-31",
      "TMAX_max" : 302.4419
    }
  }, ... {
    "type" : "Feature",
    "geometry" : {
      "type" : "Point",
      "coordinates" : [ 14.635000000000002, 47.66470588235295 ]
    },
    "properties" : {
      "phenomenonTime" : "2019-01-01/2019-12-31",
      "TMAX_max" : 346.3743
    }
  } ]
}
```

11.5. API Endpoint 3 (terradue - D167)

Terradue implemented the DAPA endpoint as described in the figure below:

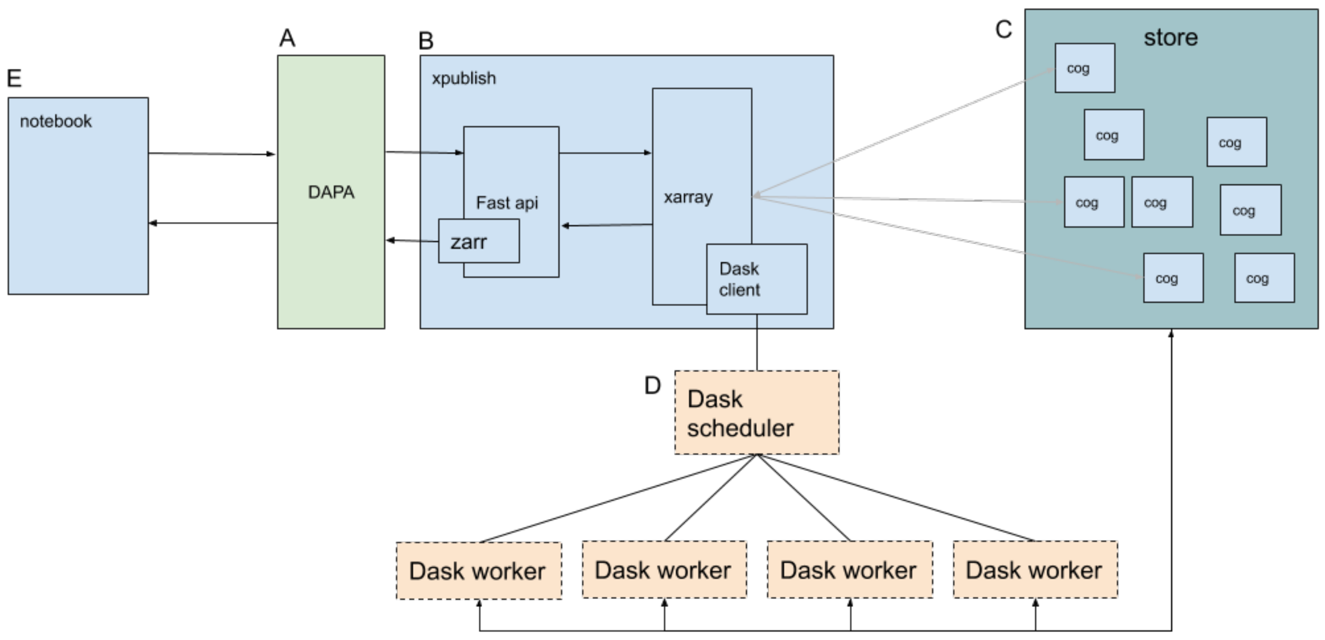


Figure 10. DAPA endpoint with x-publish service Sentinel-5 NO2 dataset

The roles of the components are explained below:

A - DAPA endpoint

The DAPA endpoint is a web service exposing the DAPA OpenAPI.

It is implemented in Python using the FastAPI framework.

B - xpublish

xpublish is an OpenAPI that serves Xarray Datasets via a web application. The data and/or metadata in the Xarray Datasets are exposed in various forms through pluggable REST API endpoints. Furthermore, and enabled with Dask on the server-side, xpublish supports on-demand delivery of large datasets.

xpublish uses a FastAPI web app that is exposed as a REST-like API with builtin and/or user-defined endpoints. For the DAPA endpoint, xpublish provides a Zarr compatible REST-like API.

C - store

The store component exposes an HTTP 1.1 interface to the Sentinel-5 NO2 dataset. This dataset is made on one Cloud Optimized GeoTIFF file per observed area and month.

xpublish exposes an Xarray via its REST-like API and the Xarray accesses the Sentinel-5 NO2 dataset using HTTP 1.1 range requests and thus only accessing the chunks of data required for processing the DAPA request.

D - Dask

As introduced when describing the xpublish element, a Dask cluster can be used at the server side to parallelize the processing of the chunks of data required for processing the DAPA request.

E - Notebook

The Notebook component is a JupyterLab instance that, using the DAPA API, creates and issues GET requests to analyze the Sentinel-5 NO2 dataset.

11.5.1. /collections

Description: this method returns the list of collections exposed by the DAPA endpoint

Method: GET

Parameters: None

Response format:

```
{
    "collections": [{
        "id": "S5P-NO2",
        "title": "Sentinel-5P NO2 column number density over Europe",
        "extent": {
            "spatial": {
                "bbox": [[-13, 57, 33, 33]],
                "crs": "EPSG:4326"
                },
            "temporal": {
                "interval": [["2018-05-01T00:00:00/2020-04-01T00:00:00"]]
            }
        }
    }]
}
```

11.5.2. /collections/{collection_id}/variables

Description: this method returns the list of variables for the collection with the id {collection_id}

Method: GET

Parameters: collection_id - the collection id

Response format:

```
{
    "variables": [{
        "id": "tropospheric_NO2_column_number_density",
        "title": "Troposheric NO2 column number density",
        "uom": "umol/m2"
    }]
}
```

11.5.3. /collections/{collection_id}/processes/position:retrieve

Description: this method retrieves data at a location (point sampling) for a time instant, interval or series the collection with the id {collection_id}

Method: GET

Parameters:

- collection_id - the collection id
- location - a location provided as: longitude & latitude pair, bounding box, a Well-Known Text or free-text
- variable - the variable id
- datetime - the date/time interval or instant

Response format: xarray serialized to JSON

```
[[[87.39471435546875,
    92.51070404052734,
    98.10374450683594,
    99.78125,
   ....
    65.9665756225586,
    66.09892272949219,
    65.9913558959961,
    63.43209457397461,
    59.1694221496582]]]
```

11.5.4. /collections/{collection_id}/processes/position:aggregate-time:

Description: this method retrieves observation data at a location (point sampling) for a time interval or series and aggregate over time the collection with the id {collection_id}

Method: GET

Parameters:

- collection_id - the collection id
- location - a location provided as: longitude & latitude pair, bounding box, Well-Known Text or free-text
- variable - the variable id
- datetime - the date/time interval or instant

Response format: xarray serialized to JSON

```
[[[87.39471435546875,
    92.51070404052734,
    98.10374450683594,
    99.78125,
   ....
    65.9665756225586,
    66.09892272949219,
    65.9913558959961,
    63.43209457397461,
    59.1694221496582]]]
```

11.5.5. /collections/{collection_id}/processes/area:retrieve

*Description: this method retrieves data in an area for a time instant, interval or series the collection with the id {collection_id}

Method: GET

Parameters:

- collection_id - the collection id
- location - a location provided as: longitude & latitude pair, bounding box, Well-Known Text or free-text
- variable - the variable id
- datetime - the date/time interval or instant

Response format: xarray serialized to JSON

```
[[[87.39471435546875,
    92.51070404052734,
    98.10374450683594,
    99.78125,
   ....
    65.9665756225586,
    66.09892272949219,
    65.9913558959961,
    63.43209457397461,
    59.1694221496582]]]
```

11.5.6. /collections/{collection_id}/processes/area:aggregate-space

Description: this method retrieves observation data in an area for a time instant, interval or series and aggregate over space the collection with the id {collection_id}

Method: GET

Parameters:

- collection_id - the collection id
- location - a location provided as: longitude & latitude pair, bounding box, Well-Known Text or free-text
- variable - the variable id
- datetime - the date/time interval or instant
- function - the function applied to the aggregation. One of:
    - min: minimum value
    - max: maximum value
    - mean: average/mean value
    - std-dev: standard deviation

Response format: xarray serialized to JSON

```
[[[87.39471435546875,
    92.51070404052734,
    98.10374450683594,
    99.78125,
   ....
    65.9665756225586,
    66.09892272949219,
    65.9913558959961,
    63.43209457397461,
    59.1694221496582]]]
```

11.5.7. /collections/{collection_id}/processes/area:aggregate-time-space

Description: this method retrieves observation data in an area (polygonal sampling) for a time instant, interval or series and aggregates over space and time the collection with the id {collection_id}

Method: GET

Parameters:

- collection_id - the collection id
- location - a location provided as: longitude & latitude pair, bounding box, Well-Known Text or free-text
- variable - the variable id
- datetime - the date/time interval or instant
- function - the function applied to the aggregation. One of:
    - min: minimum value
    - max: maximum value

  - mean: average/mean value
  - std-dev: standard deviation

Response format:

```
{"mean": 124.00251770019531}
```

11.5.8. /collections/{collection_id}/processes/area:aggregate-time

Description: this method retrieves observation data in an area (polygonal sampling) for a time interval or series and aggregate over time the collection with the id {collection_id}

Method: GET

Parameters:

- collection_id - the collection id
- location - a location provided as: longitude & latitude pair, bounding box, Well-Known Text or free-text
- variable - the variable id
- datetime - the date/time interval or instant
- function - the function applied to the aggregation. One of:
  - min: minimum value
  - max: maximum value
  - mean: average/mean value
  - std-dev: standard deviation

Response format:

```
{"mean": 124.00251770019531}
```

11.5.9. /geothesaurus - location

Description: this method takes a location expressed as a longitude & latitude pair, a bounding box, a Well-Known Text or free-text to return a GeoJSON feature.

Method: GET

Parameters:

- location - a location provided as: longitude & latitude pair, bounding box, Well-Known Text or free-text

Response format: GeoJSON feature

```
 [{
    "type": "Feature",
    "properties": {},
    "geometry": {
        "type": "Point",
        "coordinates": [8.68248, 50.11159]
    }
}]
```

Chapter 12. Data Formats

12.1. Overview

This section provides a survey of which data formats work best in which situations of data retrieval informed by the different scenarios and user groups addressed in this ER. The result is a better understanding of how to encode data for transaction and exchange.

Chapter 13. API Evaluation

13.1. Overview

This clause documents the evaluation of the components developed for this task from an end-user point of view.

13.2. Evaluation workshop

13.2.1. Overview

In July 2020, the OGC facilitated a five day virtual workshop to provide an opportunity for hands-on evaluation of technologies and draft specifications that facilitate ad hoc processing and analysis of large-scale observational datasets on cloud computing platforms “close to the data”. The workshop was a collaborative effort between OGC members participating in the EOS thread of Testbed-16 and those engaged in the EOApps Pilot activity. Each endpoint was evaluated based on the following high-level criteria:

- Learning curve to use the API.
- Richness of accessible functionality.
- Amount of code needed to execute some common analyses.

EOS participants and other expert workshop attendees evaluated the new draft Data Access and Processing API (DAPA API). The API is intended to provide simple access to processed samples of large datasets for subsequent analysis within interactive Jupyter Notebook (JPYNB) environments. They also examined procedures and tools for moving a local DAPA - JPYNB environment to the cloud as a Docker-based application package (AP) for sharing and close-to-data execution in complete and reproducible form.

EOApps Pilot participants then demonstrated and evaluated the maturity of several Earth Observation Applications-to-the-Data specifications such as ADES (Application Deployment and Execution Service) developed over the last two years as part of various OGC Innovation Program (IP) initiatives. ADES-enabled exploitation platforms are designed to convert and deploy an application package such as produced by the Testbed-16 tools for batch execution and further orchestration as an independently invocable processing service.

13.2.2. Summary agenda

The following table provides a summary of the agenda.

Session	Topic	Presenter	Notes
Day 1 [https://gitlab.ogc.org/ogc/t16-d005-data-access-and-processing-er/-/wikis/Notes/Workshop200727]	Introduction to the DAPA API and tutorials on interactive development of analytical workflows in Jupyter Notebooks		
Day 2 [https://gitlab.ogc.org/ogc/t16-d005-data-access-and-processing-er/-/wikis/Notes/Workshop200728]	Hands-on evaluation of DAPA services and Notebook-based workflows by remote sensing and observational data experts Notebooks		
Day 3 [https://gitlab.ogc.org/ogc/t16-d005-data-access-and-processing-er/-/wikis/Notes/Workshop200729]	Presentation and discussion of EO exploitation platforms, application package tools, and ADES services implemented in both the EOApps Pilot and Testbed-16. Similarities and differences, how EOAP continuing EOApps.		
Day 4 [https://gitlab.ogc.org/ogc/t16-d005-data-access-and-processing-er/-/wikis/Notes/Workshop200730]	Evaluation and discussion of DAPA API as a standalone service, as JPYNB component, and as an AP component (2 hours). Wrap-up discussion of EO Apps Pilot outcomes (1 hour)		
Day 5	Further discussion of EO Apps Pilot outcomes as needed		

13.2.3. Detailed agenda

Monday, 27th of July – 15:00 - 18:00 CEST / 09 AM - noon EDT
Introduction to the DAPA API and tutorials on interactive development of analytical workflows in Jupyter Notebooks.

- Welcome, Logistics, Agenda

- Introduction to OGC Testbed 16 EOS Thread and DAPA objectives and aims of the workshop
    - Participant introductions
    - Introduction to DAPA as an OGC API family member
- Introduction to DAPA instances (how to use their services, available data)
    - Terradue
    - Interactive Instruments
    - EOX
- Coffee break
- Introduction to Jupyter Notebooks (how to work with Notebooks)
    - Python (Important libraries: Requests, Pandas, Xarray)
    - R (does any Jupyter Notebook we provide support R currently?)
- Questions & Answers
- Summary (Overview of DAPA instances, Evaluation criteria)

Tuesday, 28th of July – 15:00 - 18:00 CEST / 09 AM - noon EDT
Hands-on evaluation of DAPA services and Notebook-based workflows by remote sensing and observational data experts

- Introduction to Hands-On
    - Logistics, Schedule for today
    - Demonstrate how we do the evaluation(?)
- Hands-on evaluation
    - Break-out room for each DAPA instance to answer questions?
    - Main room for general questions?
- Summary of the hands-on session and outlook for next days
    - What went well? What didn't go well? General questions
    - EO Apps presentations
    - Further hands-on DAPA on their own

Wednesday, 29th of July – 15:00 - 18:00 CEST / 09 AM - noon EDT
Presentation and discussion of EO exploitation platforms, application package tools, and ADES services implemented in both the EOApps Pilot and Testbed-16. Similarities and differences, how EOAP continuing EOApps.

- Introduction to the workshop - progress to date
- Overview of EOApps components: Earth Observation Exploitation Platforms, Applications, and Services
- Presentations by EOApps Pilot participants
- Overview of Testbed 16 EOAP task

- Presentations by EOAP participants
- Discussion and Generalization
- Summary and next steps

Thursday, 30th of July – 15:00 - 18:00 CEST / 09 AM - noon EDT
Evaluation and discussion of DAPA API as a standalone service, as JPYNB component, and as an AP component (2 hours). Wrap-up discussion of EO Apps Pilot outcomes (1 hour).

- DAPA: Evaluation and discussion
    - API
    - Jupyter Notebooks
- EO Apps Pilot: Wrap-up discussion

13.2.4. Evaluators

- Tom Landry - CRIM
    - Bennett Kanuka - Habitat One
    - Carsten Ehbrecht - DKRZ
    - Long Vu - Ouranos
    - Nils Hempelmann
    - Trevor James Smith - Ouranos
    - David Byrns - CRIM
    - David Landry - CRIM
    - Francis Charette-Migneault - CRIM
    - Jonas Eberle - DLR & EOX
    - Julian Zeidler - DLR
    - John Truckenbrodt - Uni Jena
    - Marcel Urban - Uni Jena
- Sreten Svetojevic - GMU
    - Eugene Yu - GMU
    - Zhiqi Yu - GMU
    - Md Shahinoor Rahman - GMU

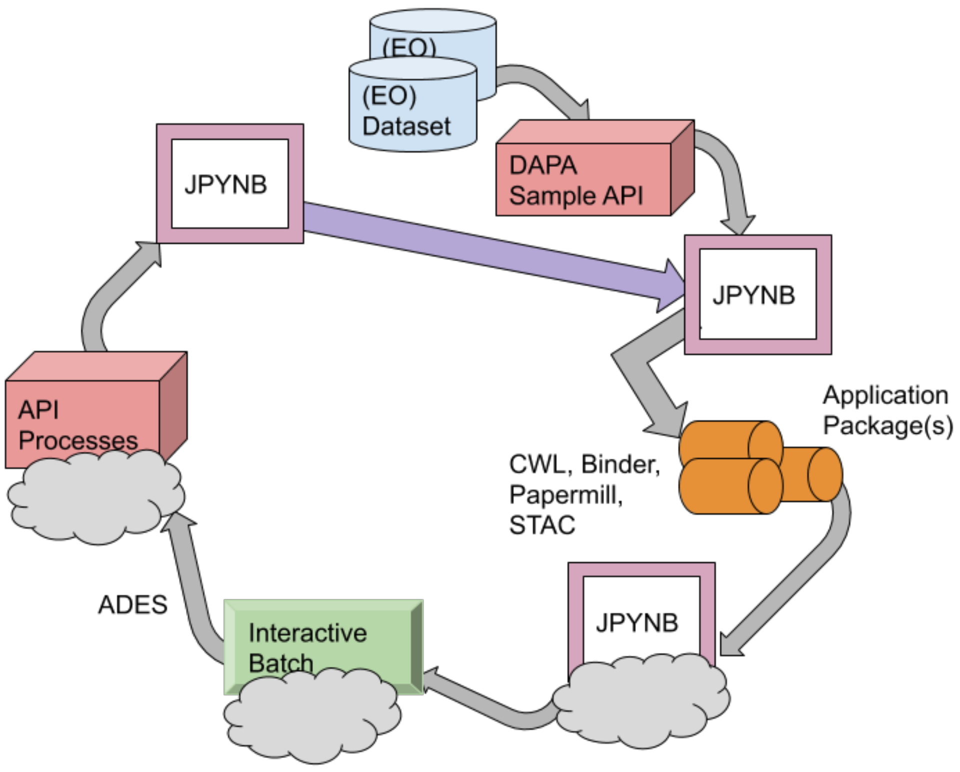
13.2.5. Evaluation Questions

The following questions were submitted to each evaluator:

1. Learning curve to use the API.
2. Richness of accessible functionality.
3. Amount of code needed to execute some common analyses.

4. What remains unclear about the APIs in general?
5. What, in your opinion, still needs to be considered or integrated?
6. What output formats do you want for easy further processing?
7. What functions/parameters (e.g. aggregation methods, calculation options) are missing?
8. What do you like about DAPA?
9. What have you used to learn about the APIs (e.g., the OpenAPI definition, examples, the additional documentation, all of it)? Any suggestions how to improve the API documentation?
10. How could use of a DAPA API impact / change / improve analytical procedures and protocols, e.g. reducing “data in motion"?
11. How simple could this be for non-expert users to learn and use, and how? For example, are defaults and named processes enough or should / could there be shortcuts for common “questions of the data”?

13.2.6. EO Applications Lifecycle Diagram



13.3. Evaluation of Expert 1 (EOX - D110)

13.3.1. Introduction

The API evaluation was coordinated and conducted by experts of the German Aerospace Center

(DLR) - German Remote Sensing Data Center in coordination with EOX. Furthermore, Earth scientists from the Friedrich Schiller University Jena - Department for Earth Observation participated in the evaluation.

Besides conducting the API evaluation workshop, DLR participated in this activity by providing use case scenarios and feedback to the API developers based on own experiences in the work with Earth scientists and the feedback after the API evaluation workshop.

13.3.2. Evaluation

In addition to the above questions regarding "learning curve to use the API", "Richness of accessible functionality", and "Amount of code needed to execute some common analyses", further questions have been drawn to get a more overall feedback from scientists:

- What remains unclear about the APIs in general?
- What, in your opinion, still needs to be considered or integrated?
- What output formats do you want for easy further processing?
- What functions/parameters (e.g. aggregation methods, calculation options) are missing?
- What do you like about DAPA?
- Execution
- What have you used to learn about the APIs (e.g., the OpenAPI definition, examples, the additional documentation, all of it)? Any suggestions how to improve the API documentation?
- How could use of a DAPA API impact / change / improve analytical procedures and protocols, e.g. reducing “data in motion"?
- How simple could this be for non-expert users to learn and use, and how?

After the API evaluation workshop, the invited experts from German Aerospace Center and Friedrich Schiller University Jena were interviewed regarding the above questions.

13.3.3. Feedback responses

1. Learning curve to use the API
   a. Super easy with requests module from Python (pass parameters as key-value pairs).
   b. Easy with Jupyter Notebooks
2. Richness of accessible functionality
   a. Good basis to start with.
   b. In combination with Jupyter Notebooks very good.
3. Amount of code needed to execute some common analyses
   a. Only less code necessary.
   b. Straight forward.
   c. Very good.
4. What remains unclear about the APIs in general?

    a. Not clear that Swagger UI (e.g., http://t16.ldproxy.net/ghcnd/api/?f=html) belongs to the API, so an overview with descriptions of parameters available (human readable) was missed.
    b. What do the requests in detail? Can there be a detailed description about that?

5. What, in your opinion, still needs to be considered or integrated?
    a. **Up/Downsampling** would be an interesting feature (e.g., downsampling to reduce the amount of data processing and data transfer for larger areas when you do not need the full resolution this is also done automatically on Google Earth Engine for the interactive visualization).
    b. Provide **more information about all parameters** and what they do in detail.
    c. Provide or link **more information about datasets** in general but also for further processing (e.g., there are artifacts/outliers in the S5p data provided by Terradue).
    d. Maybe very advanced: **Combine different datasets** in one request to get an equally sampled raster back. Example: Fusion of Sentinel-1 and Sentinel-2 for a specific point in time – the response would be a single raster image with bands for S1-VV, S1-VH, S2-B04, S2-NDVI. So users do not need to handle with matching these individual data from different data sources.
    e. **Test query**: Conduct a real request but with an "information" response requested. This response in human-readable format provides a description of what happens in doing this request (e.g., what bands are loaded, how much data will be processed and transferred back to the user, how much time could this take). As an example: If you prepare a request for a larger area (maybe by mistake), the server can tell you that maybe several hundreds of GB or maybe a TB will be transferred. Is this really what you want? Does the query make any sense at all and what will I get if I make this query. That would also make it much easier for me to learn how to use the API.
    f. Provide information about request limits (e.g., can I extract a mosaic from the whole world?)
    g. **Ease the access to API documentation**: Users do not always read documentation first (e.g., when a script is shared from a colleague). Can the API provide additional information about requests (e.g., paths and parameters available with descriptions, possible values included).
    h. Look at **functions from Google Earth Engine** if more functionality is necessary.
    i. Currently known which **projections** are provided/used for outputs. This could be a parameter.

6. What output formats do you want for easy further processing?
    a. **Binary stream better than JSON for raster data** as in many cases JSON needs to be converted in other data formats first.
    b. For direct additional processing, **data streaming** would be good.
    c. **GDAL-compatible data formats** and direct access via **GDAL Virtual File Systems** (GeoTIFFs should be COG, **alternatives to netCDF** need to be found for multi-temporal raster outputs).
    d. **"User-friendly" data formats**, which make the life of users easy to get a first overview of the data (e.g., not a CSV/JSON for time-series data, but a plot as image).
    e. **GDAL VRT** (or similar) would be very elegant for a 3d-raster stack with any band included is

a URL to a DAPA URL (useful for large data requests, server can respond very fast)

   f. **“Usual” data formats**, for spatial data are fine.

   g. Also **support for other programming languages** than only Python.

7. What functions/parameters (e.g. aggregation methods, calculation options) are missing?

   a. Not everything needs to be integrated in the API if users can do this in Notebooks or own scripts.

8. What do you like about DAPA?

   a. Simple access to extract data from large EO collections.

   b. All good, easy access to data is always important.

   c. **No need to worry about the data storage and preprocessing.**

   d. Focus on the thematic issue and no longer on data preprocessing.

   e. Error handling: **If parameter is wrong, the service lists what is available** (tested with Interactive Instruments).

   f. Can be very good used with commonly used tools (especially Python libraries).

9. Execution

   a. Good examples provided.

   b. Local execution on own computer (e.g., with API from interactive instruments).

   c. Also Jupyter Notebooks for on the fly testing much appreciated.

10. What have you used to learn about the APIs (e.g., the OpenAPI definition, examples, the additional documentation, all of it)? Any suggestions how to improve the API documentation?

   a. Examples (both in text and Jupyter Notebook).

   b. Additional documentation.

   c. Short overview with list of examples: The interactive instruments documentation page, with its live links, is a very good example of this.

11. How could use of a DAPA API impact / change / improve analytical procedures and protocols, e.g. reducing “data in motion"?

   a. More focus on **thematic/analytical issues** than on data (pre-) processing.

   b. More focus on **Capacity Building**.

   c. You can **pick people up faster** and push them in the right direction.

12. How simple could this be for non-expert users to learn and use, and how? For example, are defaults and named processes enough or should / could there be shortcuts for common “questions of the data”?

   a. When you address people who are at the local level (e.g., instances of agricultural management), you need to **provide an easy user interface to learn and interact with the API**.

   b. **Provision of notebooks** is very good for “non-expert” users.

   c. When you really want to use DAPA, you need to learn it. Striving for simplicity leads to a fixed path and tight functionality.

13. Other comments

    a. Setting up EOX DAPA app took several hours.

    b. Who is the **target group of DAPA**? If you aim for user from authorities, simple user interfaces are necessary. But also scientists are not always used to work with web services Build a requests from a simple user interface and copy this to an own script?

    c. At the end of the Testbed, it would be **interesting to see the final results**, maybe again with a small workshop.

    d. Very good/interesting **EO Apps Pilot** presentations and discussions.

13.3.4. Summary and Recommendations

All of the main evaluation criteria (learning curve, richness of functionality, and amount of code needed) were proved successful by the science experts. The idea of DAPA was well accepted. A lot of additional ideas to enrich the functionality and how to improve it for non-experts have been mentioned. Possible output formats for DAPA have been widely discussed among different scientists, such as GDAL-compatible formats as well as "user-friendly" data formats (e.g., CSV, PNG).

An open question is about the target group for DAPA and how to make DAPA more comfortable for users to learn the API. An idea came up to provide a web-based API request builder. Scientists often mentioned the need for additional documentation, especially about what exactly happens in terms of processing when conducting a request.

Finally, there was no feedback regarding which API specification is preferred. The different approaches from all three API implementations were used from the examples provided. As such, a possible conclusion is that many Earth scientists are not experts in designing APIs. They will use DAPA as it is provided. A simple execution is definitely necessary (e.g., no XML posts, no XML parsing).

The following recommendations can be summarized:

- A detailed description is necessary what the API offers and what the requests do in detail.
- More information about datasets and parameters are necessary.
- More "user-friendly" output formats (e.g., GDAL-compatible) - make it especially easier for non-experts (e.g., CSV, PNG).
- Provide an easy-to-use interface to learn how to use the API.

13.4. Evaluation of Expert 2 (GMU - D111)

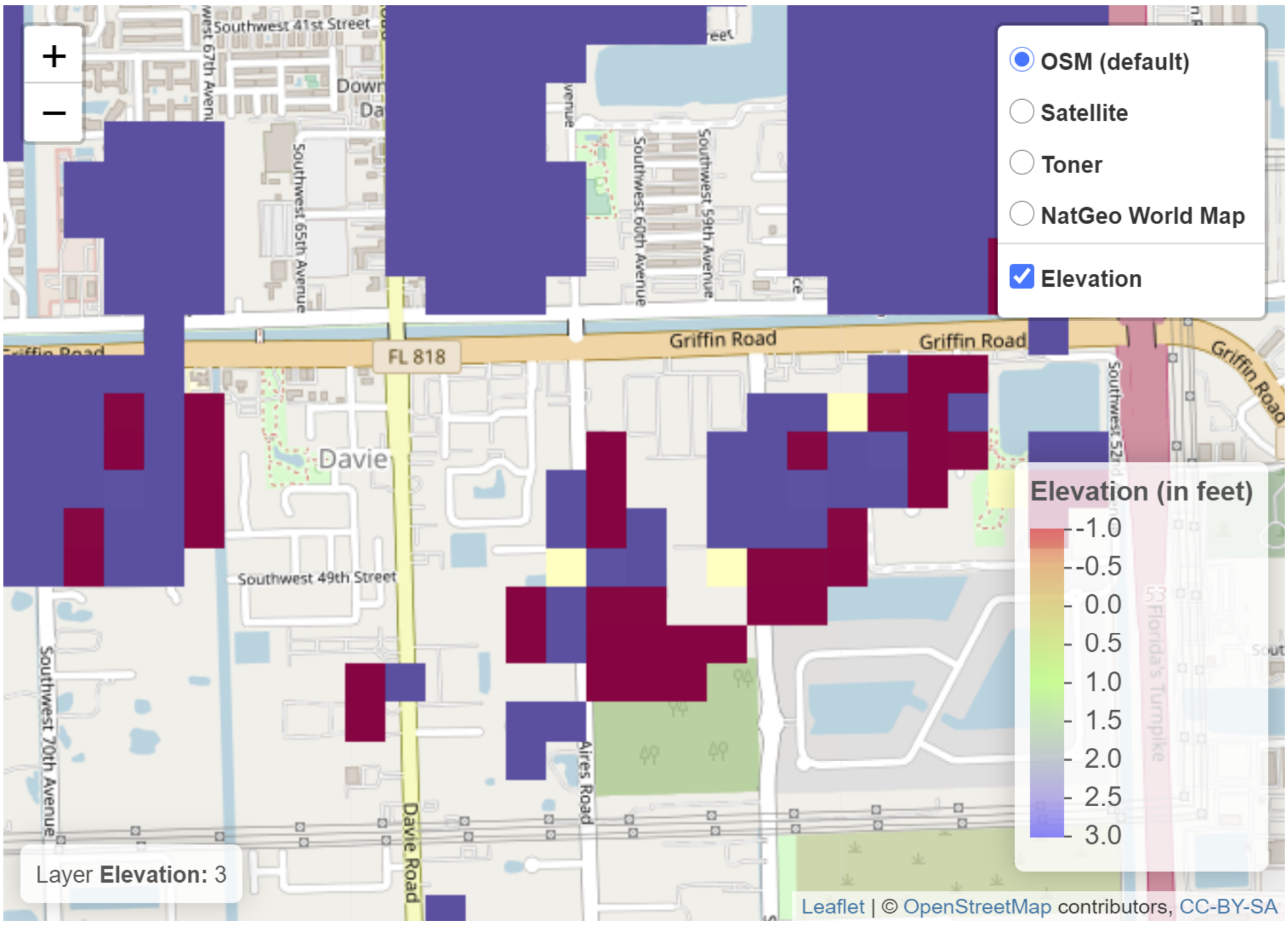
13.4.1. Introduction

Center for Spatial Information Systems and Science (CSISS) at George Mason University has conducted a DAPA API evaluation along the proposed scenario as well as general usability of the API.

13.4.2. Evaluation

1. Evaluation use case

Our use case scenario involved a basic flood modelling which required raster algebra and sampling of raster layers based on either standard administrative polygons such as U.S. states or zip codes. As a result, areas of low elevation and a higher risk of flooding were identified.



2. Learning curve

Provided Jupyter notebooks were an excellent starting point and facilitated the use of the API significantly. Some DAPA providers required an account set up but provided free access to evaluators. Patterns of providing parameters are immediately clear and straightforward.

3. The richness of accessible functionality

DAPA shows the tremendous value and potential to be a one-stop-shop for geographic data access and processing. Even though the API was used only as a source for DEM data set along the proposed scenario it reduced the time needed for finding and clipping a DEM raster to the study area. The API, however, lacks filtering of raster datasets by custom polygons or administrative divisions (countries) and/or subdivisions (states/provinces).

4. Amount of code needed to execute some common analysis

The API was tested along the proposed use case scenario in R programming language and a result is published at https://rpubs.com/scvetojevic/660301 DAPA API was easily integrated into the existing analysis script and brought significant value as a data source.

13.4.3. The R Notebook

13.4.3.1. About R notebooks

An R notebook [https://rmarkdown.rstudio.com/lesson-10.html] is similar to the above-described Jupyter notebooks. The notebook is an R markdown document that is fully reproducible and contains chunks of R code that can be executed independently. The notebook can be exported to a variety of output formats such as HTML, PDF, etc. using R studio.

13.4.3.2. Introduction

The R notebook analyzes a simple flooding scenario due to heavy rainfall in residential areas of a zip code in the state of Florida. The original version of the notebook is available at https://rpubs.com/scvetojevic/635933. Initially, the DEM raster is clipped using vector layers for the given zip code and land use classification. Further, DEM elevation values are converted from meters to feet and several feet of water are added to identify potentially affected residential areas for various water depths. Finally, an interactive map is produced using Leaflet for R package. The notebook could use only use the EOX Data Access and Processing API implementation since it was the only one that provided a Digital Elevation Model data set. The version of the notebook using EOX DAPA is available at https://rpubs.com/scvetojevic/660301. R package Raster was used to read the GeoTIFF file provided by DAPA. The URL was used from a provided EOX Jupyter notebook example on how to work with the API and it has the following format:

```
https://dapa-eval-gmu.USER-UUID.hub.eox.at/oapi/collections/DEM/dapa/area?bbox=-
80.4765,25.9567,-80.0153,26.3555&fields=DEM
```

13.4.3.3. DAPA Benefits and Further Considerations

DAPA provided easy access to a worldwide DEM data set that could be easily filtered by providing a bounding box for the study area. For similar use cases, DAPA would be even more useful if it provided filtering of raster data sets by custom vector polygons. The land use data set, which covered the entire state of Florida was particularly large and slow for R to work with. This is where DAPA could also help speed up analysis and eliminate the need for handling the land use vector files that are several GB in size.

13.5. Evaluation of Expert 3 (CRIM - D112)

13.5.1. Introduction

CRIM is grateful for being part of the OGC Testbed 16 DAPA thread. Through its role as an expert participant of the DAPA thread, CRIM actively followed the discussions concerning the design of DAPA throughout the testbed. This includes an active participation to the DAPA workshop, which was held virtually from July 27th to July 30th, 2020. The current document contains our overall assessment of the DAPA, as observed through these activities.

13.5.2. Evaluation

This section first describes the CRIM point of view adopted for this evaluation. This was done before making the assessment itself.

13.5.2.1. Evaluation use case

For the purpose of this evaluation, CRIM focused mainly on use cases that emerged from the Canadian Climate Data Portal. This use case was described in more detail during the DAPA weekly meetings.

The data portal is an interesting evaluation axis for two reasons. The portal involves running DAPA over climate model outputs, which are a type of data worthy of consideration for the DAPA project. Furthermore, this portal involves a lot of specific aggregation functions, which may illustrate need for further improvements to the DAPA.

13.5.2.2. Evaluation along the proposed axes

The OGC suggested three primary evaluation criteria to the participants:

- Learning curve,
- Richness of functionality,
- Volume of code needed to perform analysis.

13.5.2.3. Learning curve

The implementers were successful in providing an easy to learn API. This success is more related to the documentation surrounding the API than the API design itself. The execution environment provided by the implementers, which was propelled by tools such as Binder or Eurodatacube, made it exceptionally easy for the evaluators to get acquainted with the implementations. This is certainly a lesson to remember for further experimentation projects at the OGC, where appropriate.

As for the API itself, there were few surprises, which made it easy to use. One caveat is that it was easy to confuse DAPA processes from other collection properties. For instance, in the EOX implementation, the fields endpoint describes a collection property (the layers that are available in the collection), whereas the other endpoints such as area or cube are processes which need to be parameterized. This makes it slightly more difficult to answer the question “What are the operations that can executed on this collection?” when studying a DAPA for the first time. This question is, in our opinion, very common and could be answered trivially.

13.5.2.4. Richness of accessible functionality

With regards to DAPA’s objectives, the richness of accessible functionality was, overall, satisfactory. CRIM notes, however, the daily or monthly aggregation scenarios prescribed by the Canadian Climate Portal were not possible in the provided implementations. In the current state of affairs, a user would have to make one HTTP request per aggregation unit. Based on the existing implementations this issue does not seem difficult to fix.

This raises the question of whether the service providers should be able to customize DAPA functionality according to the needs of their users. CRIM is interested to see the level of specification in the evolving DAPA standard. DAPA could prescribe the operations that should be possible on a given collection, or it could allow service providers to add as many convenience functions as they wish on top of their collections.

13.5.2.5. Amount of code needed to execute some common analyses

The volume of code needed to execute some common analyses seems reasonable in the provided DAPA implementations. The volume of code required increases if the desired access pattern is not directly supported by DAPA. This is illustrated by the daily or monthly aggregation pattern mentioned in the preceding subsection. As noted there, some communities might have different common usage patterns. This reinforces the idea that DAPA should be a medium for service providers to add convenience functions on top of existing data collections, instead of prescribing access patterns.

13.5.2.6. Recommendations

Following the evaluation, CRIM would like to provide a few suggestions and other considerations regarding the DAPA project.

Have a specification strong enough to build interactive DAPA clients:

1. The first consideration is related to an interactive use of DAPA. The few demonstrations with SwaggerHub indicate that DAPA has a good potential for interactive usage. Another possibility that emerged from discussions is building an interactive client through JupyterLab widgets. Interactive clients are, after all, a large step forward in term of ease of use. Ease of use is one of the main motivations behind DAPA.
2. With that in mind, CRIM believes that the DAPA specification should be strong enough to allow building an interactive client easily. This implies that there should be self-descriptive API calls that provide basic information about how to call DAPA functionality. With this information in hand, a client could automatically build GUIs that call DAPA on behalf of the user.

No way for an immediate polygon filtering:

1. In the same spirit of simplicity, CRIM believes it should be possible for a user to provide an “immediate” polygon for area filtering. That is, it should be possible to provide a polygon directly in the request.
2. This recommendation stems from a very simple use case. If a user has the desired geometry stored locally, the current DAPA implementations do not allow the user to use this geometry directly. The user must either expose the geometry to the internet through some means, or download more data than he needs and then filter it manually. This process feels sub-optimal for the user. The immediate use case feels more basic than the remote geometry use case. In that regard, DAPA may not fulfill user expectations.

Consider using POST requests for parameterizable queries:

1. This suggestion stems partly from the previous one. The understanding is that providing immediate polygons is not possible in the current implementations because of concerns that

they could quickly hit the limits of what can be provided in a URL. CRIM proposes that this issue could be solved, and that some clarity could be gained, through the use of POST queries for parameterizable requests. Because they allow populating the request with a request body, POST queries allow for a much more expressive querying of the DAPA.

2. POST requests can indeed feel more complicated that what is aimed for with DAPA. They do require the user to build a JSON document describing the query they want to execute. However, if one studies the example notebooks provided by the API implementers, notice that all of them have query building code that construct the query URL. Transitioning this logic to one that constructs a JSON document is trivial. We could even argue that it is easier, because users could then rely on the JSON module to convert Python native objects directly into a format that is accepted by the server.
3. Note that POST requests do not entail having a multi-call relationship with the server. The server can respond to a POST request immediately, the same way it responds to a GET.
4. If POST is not too complicated for DAPA, POST can yield important benefits. It circumvents the size limit of URLs, which CRIM had to fight against in the past. POST also removes some ambiguities in the calls. Lists or coordinates can be provided in a JSON list instead of being flattened to a string in an address bar. Sets of queries can be provided in a streamlined manner. Consequently, it trivially allows immediate polygons for filtering. More generally, aggregation and filtering can be specified in separate sub-dictionaries, instead of having their parameters expressed in mangled key-value pairs in the query parameters. Lastly, in a POST request, the user does not have to worry as much about the encoding of the query. The user does not have to replace the space character by an URL-encoded one, for instance.
5. For these reasons, CRIM advocates for the use of POST queries in DAPA.

13.5.3. Evaluation questions from the participants

The API implementers had further questions for the participants. The following section summarizes the comments made previously in a way that answers those questions.

1. What remains unclear about the APIs in general?
   Finding which processes can be applied to what data could be slightly easier. This is going to be even more important when DAPA is applied to larger collections of resources. In that situation, all DAPA functionality may not be applicable to all collections.
2. What, in your opinion, still needs to be considered or integrated?
   There are two considerations when designing DAPA. The first is to make DAPA extensible in a way that service providers can provide convenience functions that are tailored to the needs of their community. The second is to make the DAPA specification strong enough to build an interactive client on top of it.
3. What output formats do you want for easy further processing?
   The current output formats seemed convenient, especially when notebooks are provided which contain the code to handle the output formats. CRIM advises against using text-based output to return gridded data.
4. What functions/parameters (e.g. aggregation methods, calculation options) are missing?
   The user should have finer control over aggregation. For instance they should be able to specify monthly aggregation over a year.

5. What do you like about DAPA?
   DAPA is, overall, successful in its objective to be easy to use.
6. What have you used to learn about the APIs (e.g., the OpenAPI definition, examples, the additional documentation, all of it)? Any suggestions on how to improve the API documentation?
   The example notebooks were used as a primary source of information. The API documentation was used when more detailed information was needed about a specific call. The API documentation was also used to get an overview of what was possible with DACCS on the API endpoints.
7. How could use of a DAPA API impact / change / improve analytical procedures and protocols, e.g. reducing “data in motion"?
   The basic concept of DAPA largely accomplishes this objective by performing common aggregation and filtering patterns close to the data. To maximize this benefit, DAPA should be flexible enough and allow service providers to design their own convenience functions according to the needs of their users.
8. How simple could this be for non-expert users to learn and use, and how? For example, are defaults and named processes enough or should / could there be shortcuts for common “questions of the data”?
   The most accessible way to present DAPA to an end-user is through an interactive client. In that regard, the SwaggerHub pages are a great success. The idea of building DAPA widgets for JupyterLab also shows a lot of potential.

13.5.4. Conclusion

CRIM affirms that DAPA largely fulfills its objective of being an easy to use entry point to OGC APIs. CRIM did provide some suggestions to evolve DAPA into an even more robust specification. Those included a better attention to the needs of interactive clients, as well as an improved design of the API requests themselves through the use of the POST HTTP method. With that in mind, CRIM has great hope about the success of the DAPA specification in the future.

Appendix A: Comparing DAPA and ADES

A.1. Overview

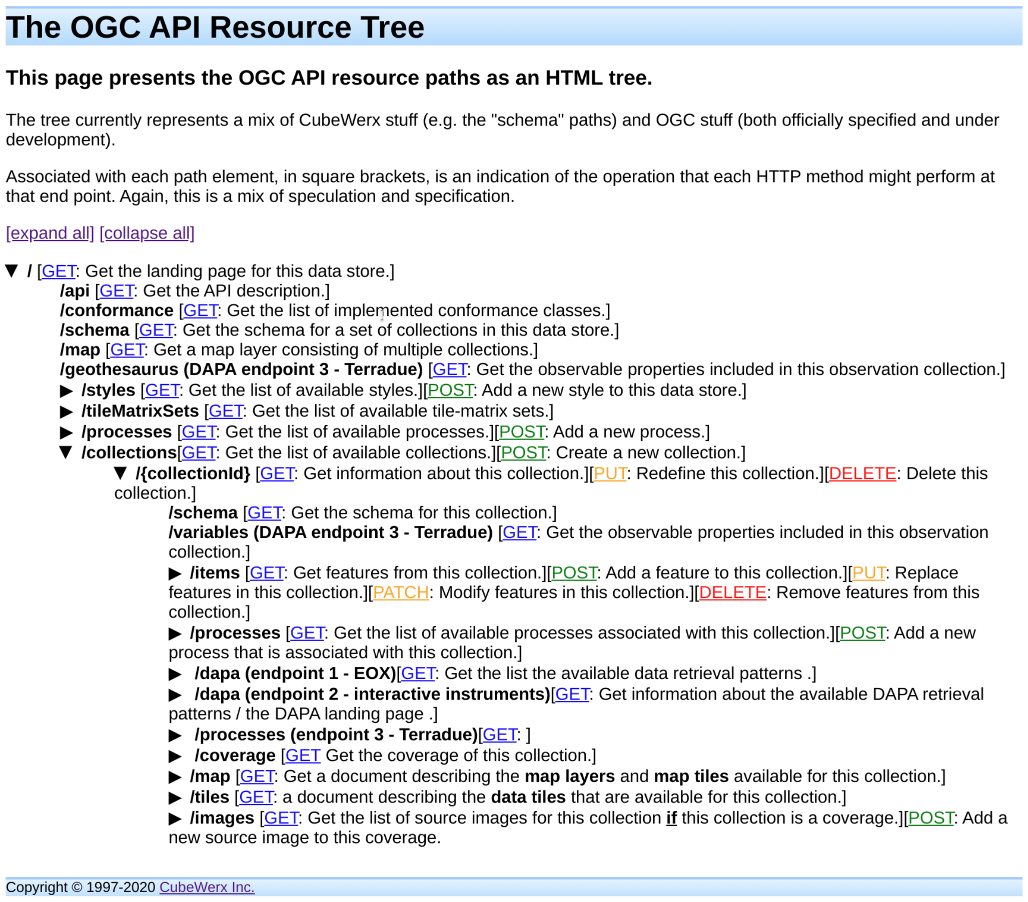
This section discusses the relationship between DAPA and the ADES and explores if there is any convergence that might be achieved between DATA and the OGC API - Processes specification (upon which the ADES is based).

This section illustrates the relationship between DAPA and the currently-defined Application Deployment and Execution Service (ADES) API, which is based on the OGC API - Processes specification, by way of a series of DAPA examples recast into ADES examples.

This section also explores avenues of convergence between DAPA and ADES and specifically if elements of DAPA could be absorbed into the OGC API - Processes specification as extensions.

The general conclusion in this section is that many elements of DAPA can be merged into the OGC API - Processes specification as extensions. Specifically, the following elements from DAPA would need to be integrated into an extension of the OGC API - Processes specification:

1. The current requirement that anchors the processes resource sub-tree at the root would need be relaxed to allow a processes subtree to be anchored under a collection resource. The implication is that the processes in this sub-tree implicitly operate on the parent collection resource. This organization is illustrated in the following figure:



The OGC API Resource Tree

This page presents the OGC API resource paths as an HTML tree.

The tree currently represents a mix of CubeWerx stuff (e.g. the "schema" paths) and OGC stuff (both officially specified and under development).

Associated with each path element, in square brackets, is an indication of the operation that each HTTP method might perform at that end point. Again, this is a mix of speculation and specification.

[expand all] [collapse all]

▼ **/** [GET: Get the landing page for this data store.]
- **/api** [GET: Get the API description.]
- **/conformance** [GET: Get the list of implemented conformance classes.]
- **/schema** [GET: Get the schema for a set of collections in this data store.]
- **/map** [GET: Get a map layer consisting of multiple collections.]
- **/geothesaurus (DAPA endpoint 3 - Terradue)** [GET: Get the observable properties included in this observation collection.]
- ▶ **/styles** [GET: Get the list of available styles.][POST: Add a new style to this data store.]
- ▶ **/tileMatrixSets** [GET: Get the list of available tile-matrix sets.]
- ▶ **/processes** [GET: Get the list of available processes.][POST: Add a new process.]
- ▼ **/collections**[GET: Get the list of available collections.][POST: Create a new collection.]
  - ▼ **/{collectionId}** [GET: Get information about this collection.][PUT: Redefine this collection.][DELETE: Delete this collection.]
    - **/schema** [GET: Get the schema for this collection.]
    - **/variables (DAPA endpoint 3 - Terradue)** [GET: Get the observable properties included in this observation collection.]
    - ▶ **/items** [GET: Get features from this collection.][POST: Add a feature to this collection.][PUT: Replace features in this collection.][PATCH: Modify features in this collection.][DELETE: Remove features from this collection.]
    - ▶ **/processes** [GET: Get the list of available processes associated with this collection.][POST: Add a new process that is associated with this collection.]
    - ▶ **/dapa (endpoint 1 - EOX)**[GET: Get the list the available data retrieval patterns .]
    - ▶ **/dapa (endpoint 2 - interactive instruments)**[GET: Get information about the available DAPA retrieval patterns / the DAPA landing page .]
    - ▶ **/processes (endpoint 3 - Terradue)**[GET:]
    - ▶ **/coverage** [GET Get the coverage of this collection.]
    - ▶ **/map** [GET: Get a document describing the **map layers** and **map tiles** available for this collection.]
    - ▶ **/tiles** [GET: a document describing the **data tiles** that are available for this collection.]
    - ▶ **/images** [GET: Get the list of source images for this collection **if** this collection is a coverage.][POST: Add a new source image to this coverage.

Copyright © 1997-2020 CubeWerx Inc.

Figure 11. OGC API Resource Tree

2. The current method of invoking a process in the OGC API - Processes specification is to compose a JSON document containing an execute request and then post that document to an execution endpoint using the HTTP POST method. In order to offer the same ease-of-use as DAPA, the OGC API - Processes specification would need to also offer a keyword-value pair encoded method of invoking a process.

   Since the ADES uses the HTTP POST method, this section uses sequence diagrams to illustrate the interactions between the client and the server. In all cases the responses from the ADES would be identical to those from a DAPA endpoint.

   NOTE The specific process description and execution encodings for the ADES used in this section are a hybrid between the work done in OGC Testbed 14 and the current draft OGC API - Processes specification.

A.2. Discovery

DAPA processes are discovered by inspecting the OpenAPI document.

The ADES defines the /processes resource that provides a list of available processes.

A.3. Process description

DAPA uses OpenAPI to fully defines each process, its inputs and its outputs.

The ADES uses a bespoke schema for describing each processes, its input and its outputs.

A.4. Encoding

DAPA processes are tied to a specific collection and are encoded as a subpath of the collection. For example:

```
http://.../ghcnd/collections/observations/area
```

where the collection is `observations` and the process is `area`.

ADES processes are independent of collection and their resource path is defined by the OGC API - Processes specification. For example:

```
http://.../processes/area
```

The `collection` would, in the ADES approach, be an argument or parameter of the `area` process.

A.5. Job Control

DAPA has no job control per se and uses a standard, synchronous, request and response model;

although adding asynchronous execution would be trivial.

The ADES, based on OGC API - Processes, has more rigorous job control and supports both synchronous and asynchronous processes execution with the ability to monitor and manipulate (to a certain extent) job execution.

A.6. Invocation

DAPA processes are invoked using the GET method with query parameters.

ADES processes are invoked by using the HTTP POST method with a body that contains an execute request encoded as a JSON document.

A.7. Responses

At the moment it appears that the ADES offer a richer ability to define responses. Multiple responses may be generated by a process with the ability to retrieve the responses by value or be reference.

By virtue of the fact that DAPA uses OpenAPI, such capabilities are possible as well although they have not been explored in sufficient detail yet to comment further.

A.8. Examples

A.9. Process `position`

Deployment

The following is a process description of the position process. In the ADES this would be posted to the /processes endpoint to deploy the process.

```
  CLIENT                                                          SERVER
    |                                                                |
    |   POST /processes   HTTP/1.1                                   |
    |   Host: www.someserver.com                                     |
    |   Content-Type: application/json                               |
    |                                                                |
    |   {                                                            |
    |     "processDescription": {                                    |
    |       "process": {                                             |
    |         "id": "position",                                      |
    |         "title": "Retrieve information about observations at an|
    |                   (interpolated) position.",                   |
    |         "abstract": "",                                        |
    |         "inputs": [                                            |
    |           {                                                    |
    |             "id": "collection",                                |
    |             "title": "Collection",                             |
    |               "abstract": "Name of the collection to process",   |
    |               "literalDataDomains": [                            |
    |                 {                                                |
    |                    "anyValue": true,                             |
    |                    "dataType": { "name": "string" }              |
    |                 }                                                |
    |               ],                                                 |
    |               "formats": [                                       |
    |                 {                                                |
    |                   "mimeType": "text/plain",                      |
    |                   "default": true                                |
    |                 }                                                |
    |               ],                                                 |
    |               "minOccurs": "1",                                  |
    |               "maxOccurs": "1"                                   |
    |             },                                                   |
    |             {                                                    |
    |               "id": "operation ",                                |
    |               "title": "Collection",                             |
    |               "abstract": "Name of the collection to process",   |
    |               "literalDataDomains": [                            |
    |                  {                                               |
    |                     "allowedValues": ["retrieve","aggregate-time"],
    |                     "defaultValue": "retrieve",                  |
    |                     "dataType": { "name": "string" }             |
    |                  }                                               |
    |               ],                                                 |
    |               "formats": [                                       |
    |                 {                                                |
    |                   "mimeType": "text/plain",                      |
    |                   "default": true                                |
    |                 }                                                |
    |               ],                                                 |
    |               "minOccurs": "1",                                  |
    |               "maxOccurs": "1"                                   |
    |             },                                                   |
    |             {                                                    |
    |               "id": "coord",                                     |
    |               "title": "Point Coordinates",                      |
    |               "abstract": "WKT representation of the point of    |
    |                            interest.",                           |
    |               "literalDataDomains": [                            |
    |                 {                                                |
    |                    "anyValue": true,                             |
    |                    "dataType": { "name": "string" }              |
    |                 }                                                |
    |               ],                                                 |
    |               "formats": [                                       |
    |                 {                                                |
    |                   "mimeType": "text/plain",                      |
    |                   "default": true                                |
    |                 }                                                  |
    |               ],                                                   |
    |               "minOccurs": "1",                                    |
    |               "maxOccurs": "1"                                     |
    |             },                                                     |
    |             {                                                      |
    |                "id": "datetime",                                   |
    |                "title": "Date Time",                               |
    |                "abstract": "Time instance or period.",             |
    |                "literalDataDomains": [                             |
    |                  {                                                 |
    |                     "anyValue": true,                              |
    |                     "dataType": { "name": "string" }               |
    |                  }                                                 |
    |                ],                                                  |
    |                "formats": [                                        |
    |                   {                                                |
    |                      "mimeType": "text/plain",                     |
    |                      "default": true                               |
    |                    }                                               |
    |                ],                                                  |
    |                "minOccurs": "1",                                   |
    |                "maxOccurs": "1"                                    |
    |             },                                                     |
    |             {                                                      |
    |                "id": "variable",                                   |
    |                "title": "Variable",                                |
    |                "abstract": "A variable to be included in the       |
    |                             response.",                            |
    |                "literalDataDomains": [                             |
    |                  {                                                 |
    |                     "anyValue": true,                              |
    |                     "dataType": { "name": "string" }               |
    |                  }                                                 |
    |                ],                                                  |
    |                "formats": [                                        |
    |                   {                                                |
    |                      "mimeType": "text/plain",                     |
    |                      "default": true                               |
    |                   }                                                |
    |                ],                                                  |
    |                "minOccurs": "1",                                   |
    |                "maxOccurs": "unbounded"                            |
    |             },                                                     |
    |             {                                                      |
    |                "id": "function",                                   |
    |                "title": "Function",                                |
    |                "abstract": "The name of function to apply to the   |
    |                             requested variables.                   |
    |                "literalDataDomains": [                             |
    |                  {                                                 |
    |                   "anyValue": true,                                 |
    |                   "dataType": { "name": "string" }                  |
    |                }                                                    |
    |              ],                                                     |
    |              "formats": [                                           |
    |                 {                                                   |
    |                    "mimeType": "text/plain",                        |
    |                    "default": true                                  |
    |                 }                                                   |
    |              ],                                                     |
    |              "minOccurs": "1",                                      |
    |              "maxOccurs": "unbounded"                               |
    |           }                                                         |
    |         ],                                                          |
    |         "outputs": [                                                |
    |           {                                                         |
    |             "id": "output",                                         |
    |             "title": "Interpolation response",                      |
    |             "formats": [                                            |
    |               {                                                     |
    |                 "mimeType": "application/json",                     |
    |                 "default": true                                     |
    |               }                                                     |
    |             ]                                                       |
    |           }                                                         |
    |         ]                                                           |
    |       },                                                            |
    |       "processVersion": "1.0.0",                                    |
    |       "jobControlOptions": [                                        |
    |         "sync-execute"                                              |
    |       ],                                                            |
    |       "outputTransmission": [                                       |
    |         "value"                                                     |
    |       ]                                                             |
    |     },                                                              |
    |     "immediateDeployment": true,                                    |
    |     "executionUnit": [                                              |
    |       {                                                             |
    |         "href": "image/position"                                    |
    |       }                                                             |
    |     ],                                                              |
    |     "deploymentProfileName": "dockerizedApplication"                |
    |   }                                                                 |
    |-------------------------------------------------------------------->|
    |                                                                     |
    |   HTTP/1.1 200 OK                                                   |
    |   Content-Type: application/json                                    |
    |                                                                     |
    | {                                                                   |
    |    "processSummary": {                                              |
    |      "id": "position",                                              |
    |      "title": "Retrieve information about observations at an   |
    |                (interpolated) position.",                       |
    |      "abstract": "",                                            |
    |      "processVersion": "1.0.0",                                 |
    |      "jobControlOptions": [                                     |
    |         "sync-execute"                                          |
    |      ],                                                         |
    |      "outputTransmission": [                                    |
    |        "value"                                                  |
    |      ],                                                         |
    |      "processDescriptionURL": "http://.../processes/position"  |
    |    }                                                            |
    | }                                                               |
    |<----------------------------------------------------------------|
```

Execution

1. Retrieve information about observations at an (interpolated) position at an instant. To execute the position processes, an execute request would be POSTed to the jobs endpoint of the position process as illustrated in the following sequence diagram.

```
 CLIENT                                                          SERVER
   |                                                                |
   |   POST /processes/position/jobs   HTTP/1.1                     |
   |   Host: www.someserver.com                                     |
   |   Content-Type: application/json                               |
   |                                                                |
   |   {                                                            |
   |      "inputs": [                                               |
   |         {                                                      |
   |            "id": "collection",                                 |
   |            "input": { "value": "observations" }                |
   |         },                                                     |
   |         {                                                      |
   |            "id": "operation",                                  |
   |            "input": { "value": "retrieve" }                    |
   |         },                                                     |
   |         {                                                      |
   |            "id": "coord",                                      |
   |            "input": { "value": "POINT(6.9299617 50.000008)" }  |
   |         },                                                     |
   |         {                                                      |
   |            "id": "datetime",                                   |
   |            "input": { "value": "2019-07-01" }                  |
   |         },                                                     |
   |         {                                                      |
   |            "id": "variable",                                   |
   |            "input": { "value": "TMAX" }                        |
   |         },                                                     |
   |         {                                                      |
   |             "id": "variable",                              |
   |             "input": { "value": "TMIN" }                   |
   |          },                                                |
   |          {                                                 |
   |             "id": "variable",                              |
   |             "input": { "value": "TAVG" }                   |
   |          },                                                |
   |          {                                                 |
   |             "id": "variable",                              |
   |             "input": { "value": "PRCP" }                   |
   |          }                                                 |
   |       ],                                                   |
   |       "outputs": [                                         |
   |          {                                                 |
   |             "id": "output",                                |
   |             "transmissionMode": "value"                    |
   |          }                                                 |
   |       ],                                                   |
   |       "mode": "sync",                                      |
   |       "response": "document"                               |
   |    }                                                       |
   |----------------------------------------------------------->|
   |                                                            |
   |    HTTP/1.1 200 OK                                         |
   |    Content-Type: application/json                          |
   |                                                            |
   |    { ... }                                                 |
   |<-----------------------------------------------------------|
```

2. Retrieve information about observations at an (interpolated) position for an interval.

```
CLIENT                                                      SERVER
  |                                                            |
  |    POST /processes/position/jobs   HTTP/1.1                |
  |    Host: www.someserver.com                                |
  |    Content-Type: application/json                          |
  |                                                            |
  |    {                                                       |
  |       "inputs": [                                          |
  |          {                                                 |
  |             "id": "collection",                            |
  |             "input": { "value": "observations" }           |
  |          },                                                |
  |          {                                                 |
  |             "id": "operation",                             |
  |             "input": { "value": "retrieve" }               |
  |          },                                                |
  |          {                                                 |
  |             "id": "coord",                                 |
  |             "input": { "value": "POINT(6.9299617 50.000008)" }  |
  |         },                                                   |
  |         {                                                    |
  |            "id": "datetime",                                 |
  |            "input": { "value": "2019-07-01/2019-07-31" }     |
  |         },                                                   |
  |         {                                                    |
  |            "id": "variable",                                 |
  |            "input": { "value": "TMAX" }                      |
  |         },                                                   |
  |         {                                                    |
  |            "id": "variable",                                 |
  |            "input": { "value": "TMIN" }                      |
  |         },                                                   |
  |         {                                                    |
  |            "id": "variable",                                 |
  |            "input": { "value": "TAVG" }                      |
  |         },                                                   |
  |         {                                                    |
  |            "id": "variable",                                 |
  |            "input": { "value": "PRCP" }                      |
  |         }                                                    |
  |      ],                                                      |
  |      "outputs": [                                            |
  |         {                                                    |
  |            "id": "output",                                   |
  |            "transmissionMode": "value"                       |
  |         }                                                    |
  |      ],                                                      |
  |      "mode": "sync",                                         |
  |      "response": "document"                                  |
  |   }                                                          |
  |------------------------------------------------------------->|
  |                                                              |
  |   HTTP/1.1 200 OK                                            |
  |   Content-Type: application/json                             |
  |                                                              |
  |   { ... }                                                    |
  |<-------------------------------------------------------------|
```

3. Retrieve information about observations at an (interpolated) position, aggregated over an interval

```
  CLIENT                                                      SERVER
    |                                                              |
    |   POST /processes/position/jobs   HTTP/1.1                   |
    |   Host: www.someserver.com                                   |
    |   Content-Type: application/json                             |
    |                                                              |
    |   {                                                          |
    |      "inputs": [                                             |
    |           {                                                        |
    |              "id": "collection",                                   |
    |              "input": { "value": "observations" }                  |
    |           },                                                       |
    |           {                                                        |
    |              "id": "operation",                                    |
    |              "input": { "value": "aggregate-time" }                |
    |           },                                                       |
    |           {                                                        |
    |              "id": "coord",                                        |
    |              "input": { "value": "POINT(6.9299617 50.000008)" }    |
    |           },                                                       |
    |           {                                                        |
    |              "id": "datetime",                                     |
    |              "input": { "value": "2019-07-01/2019-07-31" }         |
    |           },                                                       |
    |           {                                                        |
    |              "id": "variable",                                     |
    |              "input": { "value": "TMAX" }                          |
    |           },                                                       |
    |           {                                                        |
    |              "id": "variable",                                     |
    |              "input": { "value": "TMIN" }                          |
    |           },                                                       |
    |           {                                                        |
    |              "id": "variable",                                     |
    |              "input": { "value": "TAVG" }                          |
    |           },                                                       |
    |           {                                                        |
    |              "id": "variable",                                     |
    |              "input": { "value": "PRCP" }                          |
    |           },                                                       |
    |           {                                                        |
    |              "id": "function",                                     |
    |              "input": { "value": "min" }                           |
    |           },                                                       |
    |           {                                                        |
    |              "id": "function",                                     |
    |              "input": { "value": "max" }                           |
    |           },                                                       |
    |           {                                                        |
    |              "id": "function",                                     |
    |              "input": { "value": "average" }                       |
    |           },                                                       |
    |           {                                                        |
    |              "id": "function",                                     |
    |              "input": { "value": "count" }                         |
    |           },                                                       |
    |        ],                                                          |
    |        "outputs": [                                                |
    |           {                                                        |
    |              "id": "output",                                      |
    |              "transmissionMode": "value"                          |
    |           }                                                       |
    |        ],                                                         |
    |        "mode": "sync",                                            |
    |        "response": "document"                                     |
    |   }                                                               |
    |------------------------------------------------------------------>|
    |                                                                   |
    |   HTTP/1.1 200 OK                                                 |
    |   Content-Type: application/json                                  |
    |                                                                   |
    |   { ... }                                                         |
    |<------------------------------------------------------------------|
```

A.10. Process `area`

Deployment

The following is a process description of the area process. In the ADES this would be posted to the /processes endpoint to deploy the process.

```
CLIENT                                                              SERVER
  |                                                                   |
  |   POST /processes   HTTP/1.1                                      |
  |   Host: www.someserver.com                                        |
  |   Content-Type: application/json                                  |
  |                                                                   |
  |   {                                                               |
  |     "processDescription": {                                       |
  |       "process": {                                                |
  |         "id": "area",                                             |
  |         "title": "Retrieve information about observations in an|
  |                   area.",                                         |
  |         "abstract": "",                                           |
  |         "inputs": [                                               |
  |           {                                                       |
  |             "id": "collection",                                   |
  |             "title": "Collection",                                |
  |             "abstract": "Name of the collection to process",     |
  |             "literalDataDomains": [                               |
  |               {                                                   |
  |                  "anyValue": true,                                |
  |                  "dataType": { "name": "string" }                 |
  |               }                                                   |
  |             ],                                                    |
  |             "formats": [                                          |
  |               {                                                   |
  |                 "mimeType": "text/plain",                         |
  |                   "default": true                              |
  |                 }                                              |
  |               ],                                               |
  |               "minOccurs": "1",                                |
  |               "maxOccurs": "1"                                 |
  |             },                                                 |
  |             {                                                  |
  |               "id": "operation ",                              |
  |               "title": "Collection",                           |
  |               "abstract": "Name of the collection to process", |
  |               "literalDataDomains": [                          |
  |                  {                                             |
  |                     "allowedValues": ["retrieve",              |
  |                                       "aggregate-space",       |
  |                                       "aggregate-time",        |
  |                                       "aggregate-space-time"], |
  |                     "defaultValue": "retrieve",                |
  |                     "dataType": { "name": "string" }           |
  |                  }                                             |
  |               ],                                               |
  |               "formats": [                                     |
  |                 {                                              |
  |                   "mimeType": "text/plain",                    |
  |                   "default": true                              |
  |                 }                                              |
  |               ],                                               |
  |               "minOccurs": "1",                                |
  |               "maxOccurs": "1"                                 |
  |             },                                                 |
  |             {                                                  |
  |               "id": "coord",                                   |
  |               "title": "Geometry",                             |
  |               "abstract": "WKT representation of the geometry  |
  |                            of the area of interest.",          |
  |               "literalDataDomains": [                          |
  |                 {                                              |
  |                    "anyValue": true,                           |
  |                    "dataType": { "name": "string" }            |
  |                 }                                              |
  |               ],                                               |
  |               "formats": [                                     |
  |                 {                                              |
  |                   "mimeType": "text/plain",                    |
  |                   "default": true                              |
  |                 }                                              |
  |               ],                                               |
  |               "minOccurs": "1",                                |
  |               "maxOccurs": "1"                                 |
  |             },                                                 |
  |             {                                                  |
  |                "id": "datetime",                               |
  |                 "title": "Date Time",                                  |
  |                 "abstract": "Time instance or period.",                |
  |                 "literalDataDomains": [                                |
  |                   {                                                    |
  |                      "anyValue": true,                                 |
  |                      "dataType": { "name": "string" }                  |
  |                   }                                                    |
  |                 ],                                                     |
  |                 "formats": [                                           |
  |                    {                                                   |
  |                       "mimeType": "text/plain",                        |
  |                       "default": true                                  |
  |                     }                                                  |
  |                 ],                                                     |
  |                 "minOccurs": "1",                                      |
  |                 "maxOccurs": "1"                                       |
  |              },                                                        |
  |              {                                                         |
  |                 "id": "variable",                                      |
  |                 "title": "Variable",                                   |
  |                 "abstract": "A variable to be included in the response.",
  |                 "literalDataDomains": [                                |
  |                   {                                                    |
  |                      "anyValue": true,                                 |
  |                      "dataType": { "name": "string" }                  |
  |                   }                                                    |
  |                 ],                                                     |
  |                 "formats": [                                           |
  |                    {                                                   |
  |                       "mimeType": "text/plain",                        |
  |                       "default": true                                  |
  |                    }                                                   |
  |                 ],                                                     |
  |                 "minOccurs": "1",                                      |
  |                 "maxOccurs": "unbounded"                               |
  |              },                                                        |
  |              {                                                         |
  |                 "id": "function",                                      |
  |                 "title": "Function",                                   |
  |                 "abstract": "The name of function to apply to the |
  |                              requested variables.                      |
  |                 "literalDataDomains": [                                |
  |                   {                                                    |
  |                      "anyValue": true,                                 |
  |                      "dataType": { "name": "string" }                  |
  |                   }                                                    |
  |                 ],                                                     |
  |                 "formats": [                                           |
  |                    {                                                   |
  |                       "mimeType": "text/plain",                        |
  |                       "default": true                                  |
  |                 }                                                  |
  |              ],                                                    |
  |              "minOccurs": "1",                                     |
  |              "maxOccurs": "unbounded"                              |
  |           }                                                        |
  |         ],                                                         |
  |         "outputs": [                                               |
  |           {                                                        |
  |             "id": "output",                                        |
  |             "title": "Interpolation response",                     |
  |             "formats": [                                           |
  |               {                                                    |
  |                 "mimeType": "application/json",                    |
  |                 "default": true                                    |
  |               }                                                    |
  |             ]                                                      |
  |           }                                                        |
  |         ]                                                          |
  |       },                                                           |
  |       "processVersion": "1.0.0",                                   |
  |       "jobControlOptions": [                                       |
  |         "sync-execute"                                             |
  |       ],                                                           |
  |       "outputTransmission": [                                      |
  |         "value"                                                    |
  |       ]                                                            |
  |     },                                                             |
  |     "immediateDeployment": true,                                   |
  |     "executionUnit": [                                             |
  |       {                                                            |
  |         "href": "image/area"                                       |
  |       }                                                            |
  |     ],                                                             |
  |     "deploymentProfileName": "dockerizedApplication"               |
  |   }                                                                |
  |----------------------------------------------------------------->|
  |                                                                    |
  |   HTTP/1.1 200 OK                                                  |
  |   Content-Type: application/json                                   |
  |                                                                    |
  | {                                                                  |
  |    "processSummary": {                                             |
  |      "id": "area",                                                 |
  |      "title": "Retrieve information about observations in an     |
  |               area.",                                              |
  |      "abstract": "",                                               |
  |      "processVersion": "1.0.0",                                    |
  |      "jobControlOptions": [                                        |
  |         "sync-execute"                                             |
  |      ],                                                            |
  |      "outputTransmission": [                                       |
  |         "value"                                                |
  |      ],                                                        |
  |      "processDescriptionURL": "http://.../processes/area"      |
  |    }                                                           |
  | }                                                              |
  |<---------------------------------------------------------------|
```

Execution

1. Retrieve information about observations in an area at an instant

```
     CLIENT                                                         SERVER
       |                                                              |
       |   POST /processes/area/jobs   HTTP/1.1                       |
       |   Host: www.someserver.com                                   |
       |   Content-Type: application/json                             |
       |                                                              |
       |   {                                                          |
       |      "inputs": [                                             |
       |         {                                                    |
       |            "id": "collection",                               |
       |            "input": { "value": "observations" }              |
       |         },                                                   |
       |         {                                                    |
       |            "id": "operation",                                |
       |            "input": { "value": "retrieve" }                  |
       |         },                                                   |
       |         {                                                    |
       |            "id": "coord",                                    |
       |            "input": { "value": "POLYGON((5.8 47.2),(15.1 47.2),|
       |                       (15.1 55.1),(5.8 55.1),(5.8, 47.2))" } |
       |         },                                                   |
       |         {                                                    |
       |            "id": "datetime",                                 |
       |            "input": { "value": "2019-07-01" }                |
       |         },                                                   |
       |         {                                                    |
       |            "id": "variable",                                 |
       |            "input": { "value": "TMAX" }                      |
       |         },                                                   |
       |         {                                                    |
       |            "id": "variable",                                 |
       |            "input": { "value": "TMIN" }                      |
       |         },                                                   |
       |         {                                                    |
       |            "id": "variable",                                 |
       |            "input": { "value": "TAVG" }                      |
       |         },                                                   |
       |         {                                                    |
       |            "id": "variable",                                 |
       |             "input": { "value": "PRCP" }                        |
       |          }                                                      |
       |       ],                                                        |
       |       "outputs": [                                              |
       |          {                                                      |
       |             "id": "output",                                     |
       |             "transmissionMode": "value"                         |
       |          }                                                      |
       |       ],                                                        |
       |       "mode": "sync",                                           |
       |       "response": "document"                                    |
       |    }                                                            |
       |---------------------------------------------------------------->|
       |                                                                 |
       |    HTTP/1.1 200 OK                                              |
       |    Content-Type: application/json                               |
       |                                                                 |
       |    { ... }                                                      |
       |<----------------------------------------------------------------|
```

2. Retrieve information about observations in an area at an instant, aggregated over all stations

```
CLIENT                                                          SERVER
  |                                                                 |
  |    POST /processes/area/jobs   HTTP/1.1                         |
  |    Host: www.someserver.com                                     |
  |    Content-Type: application/json                               |
  |                                                                 |
  |    {                                                            |
  |       "inputs": [                                               |
  |          {                                                      |
  |             "id": "collection",                                 |
  |             "input": { "value": "observations" }                |
  |          },                                                     |
  |          {                                                      |
  |             "id": "operation",                                  |
  |             "input": { "value": "aggregate-space" }             |
  |          },                                                     |
  |          {                                                      |
  |             "id": "coord",                                      |
  |             "input": { "value": "POLYGON((5.8 47.2),(15.1 47.2),|
  |                        (15.1 55.1),(5.8 55.1),(5.8, 47.2))" }   |
  |          },                                                     |
  |          {                                                      |
  |             "id": "datetime",                                   |
  |             "input": { "value": "2019-07-01" }                  |
  |          },                                                     |
  |          {                                                      |
  |             "id": "variable",                                   |
  |             "input": { "value": "TMAX" }                        |
  |         },                                                  |
  |         {                                                   |
  |            "id": "variable",                                |
  |            "input": { "value": "TMIN" }                     |
  |         },                                                  |
  |         {                                                   |
  |            "id": "variable",                                |
  |            "input": { "value": "TAVG" }                     |
  |         },                                                  |
  |         {                                                   |
  |            "id": "variable",                                |
  |            "input": { "value": "PRCP" }                     |
  |         },                                                  |
  |         {                                                   |
  |            "id": "function",                                |
  |            "input": { "value": "min" }                      |
  |         },                                                  |
  |         {                                                   |
  |            "id": "function",                                |
  |            "input": { "value": "max" }                      |
  |         },                                                  |
  |         {                                                   |
  |            "id": "function",                                |
  |            "input": { "value": "average" }                  |
  |         },                                                  |
  |         {                                                   |
  |            "id": "function",                                |
  |            "input": { "value": "count" }                    |
  |         },                                                  |
  |      ],                                                     |
  |      "outputs": [                                           |
  |         {                                                   |
  |            "id": "output",                                  |
  |            "transmissionMode": "value"                      |
  |         }                                                   |
  |      ],                                                     |
  |      "mode": "sync",                                        |
  |      "response": "document"                                 |
  |   }                                                         |
  |------------------------------------------------------------>|
  |                                                             |
  |   HTTP/1.1 200 OK                                           |
  |   Content-Type: application/json                            |
  |                                                             |
  |   { ... }                                                   |
  |<------------------------------------------------------------|
```

3. Retrieve information about observations in an area for an interval

```
    CLIENT                                                      SERVER
```

```
|                                                                    |
|   POST /processes/area/jobs   HTTP/1.1                             |
|   Host: www.someserver.com                                         |
|   Content-Type: application/json                                   |
|                                                                    |
|   {                                                                |
|      "inputs": [                                                   |
|         {                                                          |
|            "id": "collection",                                     |
|            "input": { "value": "observations" }                    |
|         },                                                         |
|         {                                                          |
|            "id": "operation",                                      |
|            "input": { "value": "retrieve" }                        |
|         },                                                         |
|         {                                                          |
|            "id": "coord",                                          |
|            "input": { "value": "POLYGON((5.8 47.2),(15.1 47.2),|
|                       (15.1 55.1),(5.8 55.1),(5.8, 47.2))" }   |
|         },                                                         |
|         {                                                          |
|            "id": "datetime",                                       |
|            "input": { "value": "2019-07-01/2019-07-31" }           |
|         },                                                         |
|         {                                                          |
|            "id": "variable",                                       |
|            "input": { "value": "TMAX" }                            |
|         },                                                         |
|         {                                                          |
|            "id": "variable",                                       |
|            "input": { "value": "TMIN" }                            |
|         },                                                         |
|         {                                                          |
|            "id": "variable",                                       |
|            "input": { "value": "TAVG" }                            |
|         },                                                         |
|         {                                                          |
|            "id": "variable",                                       |
|            "input": { "value": "PRCP" }                            |
|         },                                                         |
|      ],                                                            |
|      "outputs": [                                                  |
|         {                                                          |
|            "id": "output",                                         |
|            "transmissionMode": "value"                             |
|         }                                                          |
|      ],                                                            |
|      "mode": "sync",                                               |
|      "response": "document"                                        |
|   }                                                                |
|------------------------------------------------------------------->|
|                                                                 |
|   HTTP/1.1 200 OK                                               |
|   Content-Type: application/json                                |
|                                                                 |
|   { ... }                                                       |
|<----------------------------------------------------------------|
```

4. Retrieve information about observations in an area for an interval, aggregated over all stations

```
  CLIENT                                                          SERVER
     |                                                                 |
     |   POST /processes/area/jobs   HTTP/1.1                          |
     |   Host: www.someserver.com                                      |
     |   Content-Type: application/json                                |
     |                                                                 |
     |   {                                                             |
     |      "inputs": [                                                |
     |         {                                                       |
     |            "id": "collection",                                  |
     |            "input": { "value": "observations" }                 |
     |         },                                                      |
     |         {                                                       |
     |            "id": "operation",                                   |
     |            "input": { "value": "aggregate-space" }              |
     |         },                                                      |
     |         {                                                       |
     |            "id": "coord",                                       |
     |            "input": { "value": "POLYGON((5.8 47.2),(15.1 47.2),|
     |                       (15.1 55.1),(5.8 55.1),(5.8, 47.2))" }    |
     |         },                                                      |
     |         {                                                       |
     |            "id": "datetime",                                    |
     |            "input": { "value": "2019-07-01/2019-07-31" }        |
     |         },                                                      |
     |         {                                                       |
     |            "id": "variable",                                    |
     |            "input": { "value": "TMAX" }                         |
     |         },                                                      |
     |         {                                                       |
     |            "id": "variable",                                    |
     |            "input": { "value": "TMIN" }                         |
     |         },                                                      |
     |         {                                                       |
     |            "id": "variable",                                    |
     |            "input": { "value": "TAVG" }                         |
     |         },                                                      |
     |         {                                                       |
     |            "id": "variable",                                    |
     |            "input": { "value": "PRCP" }                         |
     |         },                                                      |
     |        {                                                    |
     |           "id": "function",                                 |
     |           "input": { "value": "min" }                       |
     |        },                                                   |
     |        {                                                    |
     |           "id": "function",                                 |
     |           "input": { "value": "max" }                       |
     |        },                                                   |
     |        {                                                    |
     |           "id": "function",                                 |
     |           "input": { "value": "average" }                   |
     |        },                                                   |
     |        {                                                    |
     |           "id": "function",                                 |
     |           "input": { "value": "count" }                     |
     |        },                                                   |
     |     ],                                                      |
     |     "outputs": [                                            |
     |        {                                                    |
     |           "id": "output",                                   |
     |           "transmissionMode": "value"                       |
     |        }                                                    |
     |     ],                                                      |
     |     "mode": "sync",                                         |
     |     "response": "document"                                  |
     |  }                                                          |
     |------------------------------------------------------------>|
     |                                                             |
     |  HTTP/1.1 200 OK                                            |
     |  Content-Type: application/json                             |
     |                                                             |
     |  { ... }                                                    |
     |<------------------------------------------------------------|
```

5. Retrieve information about observations in an area for an interval, aggregated over time.

```
 CLIENT                                                      SERVER
   |                                                             |
   |  POST /processes/area/jobs   HTTP/1.1                       |
   |  Host: www.someserver.com                                   |
   |  Content-Type: application/json                             |
   |                                                             |
   |  {                                                          |
   |     "inputs": [                                             |
   |        {                                                    |
   |           "id": "collection",                               |
   |           "input": { "value": "observations" }              |
   |        },                                                   |
   |        {                                                    |
   |           "id": "operation",                                |
   |             "input": { "value": "aggregate-time" }                         |
   |          },                                                                |
   |          {                                                                 |
   |             "id": "coord",                                                 |
   |             "input": { "value": "POLYGON((5.8 47.2),(15.1 47.2),|
   |                        (15.1 55.1),(5.8 55.1),(5.8, 47.2))" }   |
   |          },                                                                |
   |          {                                                                 |
   |             "id": "datetime",                                              |
   |             "input": { "value": "2019-07-01/2019-07-31" }                  |
   |          },                                                                |
   |          {                                                                 |
   |             "id": "variable",                                              |
   |             "input": { "value": "TMAX" }                                   |
   |          },                                                                |
   |          {                                                                 |
   |             "id": "variable",                                              |
   |             "input": { "value": "TMIN" }                                   |
   |          },                                                                |
   |          {                                                                 |
   |             "id": "variable",                                              |
   |             "input": { "value": "TAVG" }                                   |
   |          },                                                                |
   |          {                                                                 |
   |             "id": "variable",                                              |
   |             "input": { "value": "PRCP" }                                   |
   |          },                                                                |
   |          {                                                                 |
   |             "id": "function",                                              |
   |             "input": { "value": "min" }                                    |
   |          },                                                                |
   |          {                                                                 |
   |             "id": "function",                                              |
   |             "input": { "value": "max" }                                    |
   |          },                                                                |
   |          {                                                                 |
   |             "id": "function",                                              |
   |             "input": { "value": "average" }                                |
   |          },                                                                |
   |          {                                                                 |
   |             "id": "function",                                              |
   |             "input": { "value": "count" }                                  |
   |          },                                                                |
   |       ],                                                                   |
   |       "outputs": [                                                         |
   |          {                                                                 |
   |             "id": "output",                                                |
   |             "transmissionMode": "value"                                    |
   |          }                                                                 |
   |       ],                                                                   |
   |       "mode": "sync",                                                      |
   |       "response": "document"                               |
   |   }                                                        |
   |----------------------------------------------------------->|
   |                                                            |
   |   HTTP/1.1 200 OK                                          |
   |   Content-Type: application/json                           |
   |                                                            |
   |   { ... }                                                  |
   |<-----------------------------------------------------------|
```

6. Retrieve information about observations in an area for an interval, aggregated over all stations and the time interval.

```
   CLIENT                                                     SERVER
     |                                                            |
     |   POST /processes/area/jobs   HTTP/1.1                     |
     |   Host: www.someserver.com                                 |
     |   Content-Type: application/json                           |
     |                                                            |
     |   {                                                        |
     |      "inputs": [                                           |
     |         {                                                  |
     |            "id": "collection",                             |
     |            "input": { "value": "observations" }            |
     |         },                                                 |
     |         {                                                  |
     |            "id": "operation",                              |
     |            "input": { "value": "aggregate-space-time" }    |
     |         },                                                 |
     |         {                                                  |
     |            "id": "coord",                                  |
     |            "input": { "value": "POLYGON((5.8 47.2),(15.1 47.2),|
     |                       (15.1 55.1),(5.8 55.1),(5.8, 47.2))" }   |
     |         },                                                 |
     |         {                                                  |
     |            "id": "datetime",                               |
     |            "input": { "value": "2019-07-01/2019-07-31" }   |
     |         },                                                 |
     |         {                                                  |
     |            "id": "variable",                               |
     |            "input": { "value": "TMAX" }                    |
     |         },                                                 |
     |         {                                                  |
     |            "id": "variable",                               |
     |            "input": { "value": "TMIN" }                    |
     |         },                                                 |
     |         {                                                  |
     |            "id": "variable",                               |
     |            "input": { "value": "TAVG" }                    |
     |         },                                                 |
     |         {                                                   |
     |            "id": "variable",                                |
     |            "input": { "value": "PRCP" }                     |
     |         },                                                  |
     |         {                                                   |
     |            "id": "function",                                |
     |            "input": { "value": "min" }                      |
     |         },                                                  |
     |         {                                                   |
     |            "id": "function",                                |
     |            "input": { "value": "max" }                      |
     |         },                                                  |
     |         {                                                   |
     |            "id": "function",                                |
     |            "input": { "value": "average" }                  |
     |         },                                                  |
     |         {                                                   |
     |            "id": "function",                                |
     |            "input": { "value": "count" }                    |
     |         },                                                  |
     |      ],                                                     |
     |      "outputs": [                                           |
     |         {                                                   |
     |            "id": "output",                                  |
     |            "transmissionMode": "value"                      |
     |         }                                                   |
     |      ],                                                     |
     |      "mode": "sync",                                        |
     |      "response": "document"                                 |
     |   }                                                         |
     |------------------------------------------------------------>|
     |                                                             |
     |   HTTP/1.1 200 OK                                           |
     |   Content-Type: application/json                            |
     |                                                             |
     |   { ... }                                                   |
     |<------------------------------------------------------------|
```

A.11. A KVP-encoding for execute ADES processes

Currently, the only way to execute an ADES process is to post and JSON-encoded execute request to a process' jobs endpoint using the HTTP POST method. This section proposes an alternative method of executing an ADES processes using the HTTP GET method with KVP-encoded inputs.

A.12. Execution endpoint

To be compatible with the current OGC API - Processes specification, the execute endpoint is:

```
/processes/{processId}/jobs.
```

Although there is nothing wrong with using the endpoint, the `/processes/{processId}` endpoint seems to be a more natural endpoint for invoking as process. Of course, the current behavior of doing a GET at the `/processes/{processId}` endpoint is to get a process description. In retrospect it might have been more convenient if the OGC API - Processes specification had defined a dedicated endpoint for getting a process description; perhaps something like `/processes/{processId}/api`.

A.13. Basic form of the URL

The basic form of a URL that can be used to invoke a process is:

```
http://www.someserver.com/.../processes/{processId}/jobs?
   mode=sync|async|auto&
   response=raw|document&
   {id}=<input value>&{id}=<input value>&...
   output=<output reference>&output=<output reference>&...
```

Each of mode, response and output parameter can be prefixed by the string `oapip:` if it is necessary to discriminate the parameter from an input of the same name (e.g. oapip:mode as the mode parameter versus mode as the name of a process input).

A.14. Process Inputs (<input value>)

Each process input maps to a query parameter of the same name on the URL. The basic forms of process inputs as query parameters are:

- COMPLEX (by value):

```
{inputId}=<value>[,<format facets>]
```

- COMPLEX (be reference):

```
{inputId}=<uri>[,<format facets>]]
```

- LITERAL:

```
{inputId}="string"[,<data type and uom facets>]
```

- BBOX DATA:

```
{inputId}=1,2,3,4[[,5,6][,<crs ref>]]
```

- GEOMETRY DATA:

```
{inputId}=<wkt string>[,<crs ref>]
```

NOTE:

- {inputId} is the process input identifier.
- <value> is the value of the input.
- <uri> is a reference to a complex value.
- "string" is a literal value.
- <format facets> is a comma-separated triple of the form {mimeType}[,{schema}[,utf-8]] where the commas are literal commas. Embedded commas must be appropriately encoded.
- <data type and uom facets> is a comma-separated quad of the form {dataType}[,{dataTypeRef}[,{uom}[,{uomRef}]]] where the commas are literal commas and embedded commas must be appropriately encoded.
- <crs ref> is a reference to a coordinate reference system

CAUTION Further investigation needs to be done into how arrays of values are specified given that the input parameter is already an array to accommodate format facets, data type specification and uom facets. Perhaps the default behavior should be that an array of values is interpreted as an array of values UNLESS format facets are specified in which case the array must be delimited by "[ ]" (i.e. input1=val,val,val,val versus input1=%5Bval%2Cval%2Cval%2Cval%5D,{mimeType},{schema},utf-8).

A specific Input can be repeated as a query parameter (i.e. type=array and collectionFormat=multi in OpenAPI parlance) if the cardinality of the input is greater than one.

NOTE See note above about arrays of values. This approach supports inputs with cardinality>1 but it is questionable whether this is the "most" natural way to handle this.

In its most basic form, however, it is important to note that a process input looks like any other query parameter on a URL (i.e. {inputId}=value).

A.15. Process outputs (<output reference>)

Each process output can be specified using the output query parameter. Multiple output parameters can be specified to request multiple outputs. The basic form of the output parameter is:

```
output={outputId}[,value|reference][,{mimeType}[,{schema}[,{encoding}]]]
```

A.16. Examples

The following set of examples are identical to the JSON-POST example except that the processes are invoked using the GET method and the processes inputs are encoded as query parameter on the process URL.

1. Retrieve information about observations at an (interpolated) position at an instant.

```
/collections/observations/processes/area:retrieve/jobs?coord=POINT(6.929617
50.000008)&datetime=2019-07-
01&variables=TMAX,TMIN,TAVG,PRCP&output=output&transmissionMode=value&mode=sync&res
ponse=document
```

2. Retrieve information about observations at an (interpolated) position for an interval.

```
/collections/observations/processes/area:retrieve/jobs?operation=retrieve&coord=POI
NT(6.929617 50.000008)&datetime=2019-07-01/2019-07-
31&variables=TMAX,TMIN,TAVG,PRCP&output=output&transmissionMode=value&mode=sync&res
ponse=document
```

3. Retrieve information about observations at an (interpolated) position, aggregated over an interval

```
/collections/observations/processes/area:aggregate-time/jobs?coord=POINT(6.929617
50.000008)&datetime=2019-07-01/2019-07-
31&variables=TMAX,TMIN,TAVG,PRCP&functions=max,average,count&output=output&transmis
sionMode=value&mode=sync&response=document
```

4. Retrieve information about observations in an area at an instant

```
/collections/observations/processes/area_retrieve/jobs?coord=POLYGON((5.8
47.2),(15.1 47.2),(15.1 55.1),(5.8 55.1),(5.8, 47.2))&datetime=2019-07-
01&variables=TMAX,TMIN,TAVG,PRCP&output=output&transmissionMode=value&mode=sync&res
ponse=document
```

5. Retrieve information about observations in an area at an instant, aggregated over all stations

```
/collections/observations/processes/area:aggregate-space/jobs?coord=POLYGON((5.8
47.2),(15.1 47.2),(15.1 55.1),(5.8 55.1),(5.8, 47.2))&datetime=2019-07-
01&variables=TMAX,TMIN,TAVG,PRCP&functions=min,max,average,count&output=output&tran
smissionMode=value&mode=sync&response=document
```

6. Retrieve information about observations in an area for an interval

```
/collections/observations/processes/area:retrieve/jobs?coord=POLYGON((5.8
47.2),(15.1 47.2),(15.1 55.1),(5.8 55.1),(5.8, 47.2))&datetime=2019-07-01/2019-07-
31&variables=TMAX,TMIN,TAVG,PRCP&output=output&transmissionMode=value&mode=sync&res
ponse=document
```

7. Retrieve information about observations in an area for an interval, aggregated over all stations

```
/collections/observations/processes/area:aggregate-space/jobs?coord=POLYGON((5.8
47.2),(15.1 47.2),(15.1 55.1),(5.8 55.1),(5.8, 47.2))&datetime=2019-07-01/2019-07-
31&variables=TMAX,TMIN,TAVG,PRCP&function=min,max,average,count&output=output&trans
missionMode=value&mode=sync&response=document
```

8. Retrieve information about observations in an area for an interval, aggregated over time.

```
/collections/observations/processes/area:aggregate-time/jobs?coord=POLYGON((5.8
47.2),(15.1 47.2),(15.1 55.1),(5.8 55.1),(5.8, 47.2))&datetime=2019-07-01/2019-07-
31&variables=TMAX,TMIN,TAVG,PRCP&function=min,max,average,count&output=output&trans
missionMode=value&mode=sync&response=document
```

9. Retrieve information about observations in an area for an interval, aggregated over all stations and the time interval.

```
/collections/observations/processes/area:aggregate-space-
time/jobs?coord=POLYGON((5.8 47.2),(15.1 47.2),(15.1 55.1),(5.8 55.1),(5.8,
47.2))&datetime=2019-07-01/2019-07-
31&variables=TMAX,TMIN,TAVG,PRCP&function=min,max,average,count&output=output&trans
missionMode=value&mode=sync&response=document
```

Appendix B: Revision History

Table 1. Revision History

Date	**Editor**	**Release**	**Primary clauses modified**	**Descriptions**
November 20, 2020	P. Vretanos	1.0	multiple	submitted version