A COMMON, OPEN SOURCE INTERFACE BETWEEN EARTH OBSERVATION DATA INFRASTRUCTURES AND FRONT-END APPLICATIONS

Deliverable 15

Version 1.0 from 2019/05/03

openEO dataset and process descriptions

Change history

Issue	Date	Author(s)	Description
0.1	2019/03/12	Matthias Mohr, WWU	First draft.
0.2	2019/03/18	Claudio Navacchi, TU Wien Milutin Milenkovic, WUR	Review of chapter 5.
0.3	2019/03/25	Luca Foresta, EODC	Internal review.
0.4	2019/04/25	Matthias Mohr, WWU	Finalised version of processes
1.0	2019/05/03	Matthias Schramm, TU Wien	Final review and adaptations for submission.

Number of pages: **126**

Disclaimer

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 776242. Any dissemination of results reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Table of Contents

List of Acronyms

JSON JavaScript Object Notation

MSI Multi-Spectral Instrument

OLCI Ocean and Land Colour Instrument

SAR Synthetic-Aperture Radar

STAC SpatioTemporal Asset Catalog

1 Executive summary

This report covers the description frameworks we use in the openEO API to communicate the metadata for the datasets and processes between back-ends and clients.

The description frameworks are designed to be most useful for the openEO API. Therefore, we reviewed existing standards for both process and dataset descriptions. None of the standards fully suited our requirements and therefore we had to specify a new process description framework, which was influenced by existing standards. For the dataset description framework we joined the SpatioTemporal Asset Catalog (STAC) initiative to align with them and to push the still evolving STAC specification towards a specification that is not only useful for data providers, but also for data processing platforms.

The datasets required by the use cases were defined in the projects proposal and in Deliverable 09 [1]. To solve the use cases, processes were specified by the openEO consortium. We have specified more processes than required by our use cases to cover a wider range of applications and thus create a useful specification for a wider user base. Several project meetings were held to specify the processes. This report lists and describes the result of the work undertaken to specify dataset and process catalogues. This report represents therefore also a more detailed, evolved and general version of Deliverable 09.

2 Specification for processes

The openEO processes' specification describes a workflow that can be used to process data on a cloud back–end with client code. As the openEO API uses JSON as content encoding, the encoding should be based on JSON, too. The specification must allow the openEO project to standardise interoperable processes, but also make sure that back–end providers can add custom processes to their process catalogue. It must ensure that back–end implementers, client libraries and client users can understand the process descriptions correctly. This also means that client libraries should make openEO processes available to client users as if they were working with native functions in the respective programming language.

As none of the related specifications met our criteria, we had to come up with a new specification. It is influenced by OGC WPS[1] 1.0 and 2.0, JSON Schema[2] and OpenAPI 3.0[3]. Each process can be described with the following properties:

- `id` (string, required): A unique identifier, the process name used by users and for communication.
- `summary` (string): A short summary to get a quick overview.
- `description` (string, required): A full human-readable description of the process, which is mainly targeted towards client users and back-end implementers. It should describe precisely how the process works.

[1] https://www.opengeospatial.org/standards/wps
[2] https://json-schema.org/
[3] https://github.com/OAI/OpenAPI-Specification

- `categories` (array<string>, required): A list of categories the process belongs to, in order to simplify finding related processes.
- `parameters` (object, required): Describes the input of the process as a set of named parameters. Each parameter must be described with a human-oriented description and machine-oriented JSON schema. Each parameter can be specified to be required, deprecated and/or experimental. The latter two share the same definition as for processes described below.
- `parameter_order` (array<string>): Describes the order of the parameter for any programming languages that don't support named parameters. This property is required for all processes with two or more parameters.
- `returns` (object, required): Describes the output of the process with a human-oriented description and machine-oriented JSON schema.
- `deprecated` (boolean): Specifies that a process is deprecated with the potential to be removed in any of the next versions. The process should be transitioned out of usage as soon as possible and users should refrain from using the process in new implementations.
- `experimental` (boolean): Declares this process to be experimental, which means that it is likely to change or may produce unpredictable behaviour. Users should refrain from using the process in production, but still feel encouraged to try it out and give feedback.
- `exceptions` (object): Declares potential exceptions / errors that might occur during execution of this process and abort the processing chain. A name for the exception and a message intended to be displayed to the client user are required. The message may contain variables, enclosed by curly brackets. A detailed description and a HTTP status code can optionally be added.
- `examples` (array<object>): A list of examples for client users, but not specific to any programming language. Can be either a process graph or just the arguments passed to the parameters. Both allow a return value to be specified so that the examples could potentially also be used as unit tests for the processes. A title and description can be specified, too.
- `links` (array<object>): A list of related links, e.g. additional external documentation for this process. Consists of a URL and optionally a relation type[4], a media type and a human-readable title.

The full specification for process discovery is available in the OpenAPI document[5].

3 List of openEO processes

In the following chapters all predefined openEO processes are listed in alphabetical order. The processes are converted to a human-readable representation of the JSON files. The original files are available in the openEO GitHub repository[6]. The processes described here will be

[4] https://www.iana.org/assignments/link-relations/link-relations.xml
[5] https://open-eo.github.io/openeo-api/apireference/#tag/Process-Discovery
[6] https://github.com/Open-EO/openeo-processes

released with openEO API version 0.4. Since their description is a living document, this Deliverable only describes their actual state. The most recent version of the processes is always available at processes.openeo.org[7]. Back-end providers are recommended to implement these standard processes whenever feasible on their infrastructure, and may choose to specify custom ones. Custom processes could also be added to the standardised openEO processes in a next version if they are found to be useful for a broader audience. With each new version of the openEO API it is likely that the processes will evolve to incorporate feedback from partners and externals. We plan to add more processes once required by new use cases.

3.1 absolute: Absolute value

Computes the absolute value of a real number `x`, which is the "unsigned" portion of x and often denoted as *|x|*.

The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number.

- **Data type**: number / null

Return Value

The computed absolute value.

- **Data type**: number / null
- **Minimum value**: 0

Examples

1. `absolute(x = 0) => 0`
2. `absolute(x = 3.5) => 3.5`
3. `absolute(x = -0.4) => 0.4`
4. `absolute(x = -3.5) => 3.5`

See Also

- Absolute value explained by Wolfram MathWorld[8]

[7] http://processes.openeo.org
[8] http://mathworld.wolfram.com/AbsoluteValue.html

3.2 add_dimension: Add a new dimension

Adds a new named dimension to the data cube.

Afterwards, the dimension can be referenced with the specified `name`. If a dimension with the specified name already exists, a `DimensionExists` exception is thrown. The dimension value of the dimension is set to the specified `value`.

Parameters

data (required) A data cube to add the dimension to.

- **Data type**: raster-cube (object)

name (required) Name for the dimension.

- **Data type**: string

value (required) A dimension value (**not** a pixel value).

- **Data types**:
  - **number**
  - **string**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

type The type of dimension, defaults to `other`.

- **Data type**: string
- **Allowed values**:
  1. spatial
  2. temporal
  3. bands
  4. other
- **Default value**: other

Return Value

The data cube with a newly added dimension.

- **Data type**: raster-cube (object)

Exceptions

- `DimensionExists`: A dimension with the specified name already exists.

3.3 aggregate_polygon: Compute zonal statistics for polygons

Aggregates zonal statistics for one or multiple polygons over the spatial dimensions.

The process considers all pixels for which the point at the pixel center intersects with the corresponding polygon (as defined in the Simple Features standard by the OGC).

The data cube must have been reduced to only contain two raster dimensions and a third dimension the values are aggregated for, for example the temporal dimension to get a time series. Otherwise this process fails with the `TooManyDimensions` error.

The number of total and valid pixels is returned together with the calculated values.

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

polygons (required) One or more polygons to calculate zonal statistics for. Either specified as GeoJSON or vector data cube.

For GeoJSON this can be one of the following GeoJSON types:

- A `Polygon` geometry,
- a `GeometryCollection` containing Polygons,
- a `Feature` with a `Polygon` geometry or
- a `FeatureCollection` containing `Feature`s with a `Polygon` geometry.
- **Data types**:
  - **geojson (object)**
  - **vector-cube (object)**

reducer (required) A reducer to be applied on all values of each geometry. The reducer must be a callable process (or a set of processes as process graph) such as `mean` that accepts by default array as input. The process can also work on two values by setting the parameter `binary` to `true`.

- **Data types**:
  - **callback (object)**: Passes two values to the reducer.
    - **Callback parameters**:
      - **x**: The first value. Any data type could be passed.
      - **y**: The second value. Any data type could be passed.

name The property name (for GeoJSON) or the new dimension name (for vector cubes) to be used for storing the results. Defaults to `result`.

- **Data type**: string
- **Default value**: result

binary Specifies whether the process should pass two values to the reducer or a list of values (default).

If the process passes two values, the reducer must be both associative and commutative as the execution may be executed in parallel and therefore the order of execution is arbitrary.

This parameter is especially useful for UDFs passed as reducers. Back-ends may still optimize and parallelize processes that work on list of values.

- **Data type**: boolean
- **Default value**: *false*

Return Value

A vector data cube or a GeoJSON object, depending on the input of the `polygons` parameter.

The computed value is stored in a property (GeoJSON) or dimension (vector cube) with the name that was specified in the parameter `name`.

The computation also stores information about the total count of pixels (valid + invalid pixels) and the number of valid pixels (see `is_valid`) in each geometry. In GeoJSON these are stored as properties with the names `{name}_total_count` and `{name}_valid_count` (replace `{name}` with the value of the `name` parameter). In a vector data cube, these values are stored as attributes of the result value with the attribute names `total_count` and `valid_count`.

If the input was GeoJSON and the therefore the return value is also a GeoJSON object, the geometries (`Polygon` or `GeometryCollection`) get wrapped in a `Feature` or `FeatureCollection` respectively. The results of the computations are stored in the `properties` of each GeoJSON `Feature`.

- **Data types**:
  - **geojson (object)**
  - **vector-cube (object)**

Exceptions

- `TooManyDimensions`: The number of dimensions must be reduced to three for 'aggregate_polygon'.

See Also

- More information about the experimental status of the process[9]
- Aggregation explained in the openEO glossary[10]
- Simple Features standard by the OGC[11]
- Background information on reduction operators (binary reducers) by Wikipedia[12]

3.4 aggregate_temporal: Temporal aggregations

Computes a temporal aggregation based on an array of date and/or time intervals.

Calendar hierarchies such as year, month, week etc. must be transformed into specific intervals by the clients. For each interval, all data along the dimension will be passed through the reducer. The computed values will be projected to the labels, so the number of labels and the number of intervals need to be equal.

If the dimension is not set or is set to `null`, the data cube is expected to only have one temporal dimension.

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

intervals (required) Left-closed temporal intervals, which are allowed to overlap. Each temporal interval in the array has exactly two elements:

1. The first element is the start of the date and/or time interval. The specified instance in time is **included** in the interval.
2. The second element is the end of the date and/or time interval. The specified instance in time is **excluded** from the interval.

[9] https://github.com/Open-EO/openeo-processes/issues/2
[10] https://open-eo.github.io/openeo-api/v/0.4.0/glossary/#aggregation-and-resampling
[11] http://www.opengeospatial.org/standards/sfa
[12] https://en.wikipedia.org/wiki/Reduction_Operator

The specified temporal strings follow RFC 3339[13]. Although RFC 3339 prohibits the hour to be '24'[14], **this process allows the value '24' for the hour** of an end time in order to make it possible that left-closed time intervals can fully cover the day.

- **Data type**: temporal-intervals (array<temporal-interval:array>)
- **Array items**:
  - **Data type**: temporal-interval (array)
  - **Min. number of items**: 2
  - **Max. number of items**: 2
  - **Array items**:
    - **Data types**:
      - **date-time (string)**
      - **date (string)**
      - **time (string)**
      - **null**
- **Examples**:
  1. `[["2015-01-01","2016-01-01"],["2016-01-01","2017-01-01"],["2017-01-01","2018-01-01"]]`
  2. `[["00:00:00Z","12:00:00Z"],["12:00:00Z","24:00:00Z"]]`

labels (required) Labels for the intervals, which can contain dates and/or times. The number of labels and the number of groups need to be equal.

- **Data type**: array
- **Array items**:
  - **Data types**:
    - **date-time (string)**
    - **date (string)**
    - **time (string)**
    - **string**

reducer (required) A reducer to be applied on all values along the specified dimension. The reducer must be a callable process (or a set of processes as process graph) such as `mean` that accepts by default array as input. The process can also work on two values by setting the parameter `binary` to `true`.

[13] https://tools.ietf.org/html/rfc3339
[14] https://tools.ietf.org/html/rfc3339#section-5.7

- **Data types**:
  - **callback (object)**: Passes two values to the reducer.
    - **Callback parameters**:
      - **x**: The first value. Any data type could be passed.
      - **y**: The second value. Any data type could be passed.

dimension The temporal dimension for aggregation. All data along the dimension will be passed through the specified reducer. If the dimension is not set or set to `null`, the data cube is expected to only have one temporal dimension.

Note: The default dimensions a data cube provides are described in the collection's metadata field `cube:dimensions`.

- **Data type**: string / null
- **Default value**: *null*

binary Specifies whether the process should pass two values to the reducer or a list of values (default).

If the process passes two values, the reducer must be both associative and commutative as the execution may be executed in parallel and therefore the order of execution is arbitrary.

This parameter is especially useful for UDFs passed as reducers. Back-ends may still optimize and parallelize processes that work on list of values.

- **Data type**: boolean
- **Default value**: *false*

Return Value

A data cube with potentially lower resolution and potentially lower cardinality, but the same number of dimensions as the original data cube.

- **Data type**: raster-cube (object)

See Also

- Information about the supported temporal formats.[15]
- Aggregation explained in the openEO glossary[16]
- Background information on reduction operators (binary reducers) by Wikipedia[17]

[15] https://open-eo.github.io/openeo-api/v/0.4.0/processes/#openeo-specific-formats
[16] https://open-eo.github.io/openeo-api/v/0.4.0/glossary/#aggregation-and-resampling
[17] https://en.wikipedia.org/wiki/Reduction_Operator

3.5 and: Are all of the values true?

Checks if **all** of the values are true. Evaluates each expression from the first to the last element and stops once the outcome is unambiguous.

If only one value is given the process evaluates to the given value. If no value is given (i.e. the array is empty) the process returns `null`.

By default all no-data values are ignored so that the process returns `true` if all other values are true and otherwise returns `false`. Setting the `ignore_nodata` flag to `false` considers no-data values so that `null` is a valid logical object. If a component is `null`, the result will be `null` if the outcome is ambiguous. See the following truth table:

	null	false	true
null	null	false	null
false	false	false	false
true	null	false	true

Parameters

expressions (required) A set of boolean values.

- **Data type**: array<boolean|null>

ignore_nodata Indicates whether no-data values are ignored or not and ignores them by default.

- **Data type**: boolean
- **Default value**: *true*

Return Value

Boolean result of the logical expressions.

- **Data type**: boolean / null

Examples

1. `and(expressions = [false,null]) => false`
2. `and(expressions = [true,null]) => true`
3. `and(expressions = [false,null], ignore_nodata = false) => false`
4. `and(expressions = [true,null], ignore_nodata = false) => null`
5. `and(expressions = [true,false,true,false]) => false`

6. `and(expressions = [true,false]) => false`
7. `and(expressions = [true,true]) => true`
8. `and(expressions = [true]) => true`
9. `and(expressions = []) => null`

3.6 apply_dimension: Applies an n-ary process to all pixels

Applies an **n-ary** process (i.e. takes an array of pixel values instead of a single pixel value) to a raster data cube. In contrast, the process `apply` applies an unary process to all pixel values.

By default, `apply_dimension` applies the process on all pixel values in the data cube as `apply` does, but the parameter `dimension` can be specified to work only on a particular dimension only. For example, if the temporal dimension is specified the process will work on a time series of pixel values.

The n-ary process must return as many elements in the returned array as there are in the input array. Otherwise a `CardinalityChanged` error must be returned.

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

process (required) A process (callback) to be applied on each dimension. The specified process needs to accept an array as parameter and must return as many elements in the returned array as there are in the input array.

- **Data type**: callback (object)
- **Callback parameters**:
  - **data**: An array with elements of any data type.
    - **Data type**: array
    - **Array items**: Any data type.

dimension The name of the dimension to apply the process on. By default, applies the process on all pixel values (as `apply` does).

- **Data type**: string / null
- **Default value**: *null*

Return Value

A data cube with the newly computed values for the specified . The resolution, cardinality and the number of dimensions are the same as for the original data cube.

- **Data type**: raster-cube (object)

Exceptions

- `CardinalityChanged`: The callback returned less or more elements than it received.

t

Applies a focal operation based on a weighted kernel to each value of the specified dimensions in the data cube.

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

kernel (required) The kernel to be applied on the data cube. The kernel has to be as many dimensions as the data cube has dimensions.

- **Data type**: kernel (array)
- **Array items**: Usually a multi-dimensional array of numbers.

factor A factor that is multiplied to each value computed by the focal operation.

This is basically a shortcut for explicitly multiplying each value by a factor afterwards, which is often required for some kernel-based algorithms such as the Gaussian blur.

- **Data type**: number
- **Default value**: 1

Return Value

A data cube with the newly computed values. The resolution, cardinality and the number of dimensions are the same as for the original data cube.

- **Data type**: raster-cube (object)

3.7 apply: Applies a unary process to each pixel

Applies a **unary** process which takes a single value such as `abs` or `sqrt` to each pixel value in the data cube (i.e. a local operation). In contrast, the process `apply_dimension` applies an n-ary process to a particular dimension.

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

process (required) A process (callback) to be applied on each value. The specified process must be unary meaning that it must work on a single value.

- **Data type**: callback (object)
- **Callback parameters**:
  - **x**: A value of any type could be passed.

Return Value

A data cube with the newly computed values. The resolution, cardinality and the number of dimensions are the same as for the original data cube.

- **Data type**: raster-cube (object)

3.8 arccos: Inverse cosine

Computes the arc cosine of `x`. The arc cosine is the inverse function of the cosine so that *arccos(cos(x)) = x*.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number.

- **Data type**: number / null

Return Value

The computed angle in radians.

- **Data type**: number / null

Examples

1. `arccos(x = 1) => 0`

See Also

- Inverse cosine explained by Wolfram MathWorld[18]

3.9 arcosh: Inverse hyperbolic cosine

Computes the inverse hyperbolic cosine of `x`. It is the inverse function of the hyperbolic cosine so that *arcosh(cosh(x)) = x*.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number.

- **Data type**: number / null

Return Value

The computed angle in radians.

- **Data type**: number / null

Examples

1. `arcosh(x = 1) => 0`

See Also

- Inverse hyperbolic cosine explained by Wolfram MathWorld[19]

[18] http://mathworld.wolfram.com/InverseCosine.html
[19] http://mathworld.wolfram.com/InverseHyperbolicCosine.html

3.10 arcsin: Inverse sine

Computes the arc sine of `x`. The arc sine is the inverse function of the sine so that *arcsin(sin(x)) = x*.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number.

- **Data type**: number / null

Return Value

The computed angle in radians.

- **Data type**: number / null

Examples

1. `arcsin(x = 0) => 0`

See Also

- Inverse sine explained by Wolfram MathWorld[20]

3.11 arctan: Inverse tangent

Computes the arc tangent of `x`. The arc tangent is the inverse function of the tangent so that *arctan(tan(x)) = x*.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number.

- **Data type**: number / null

[20] http://mathworld.wolfram.com/InverseSine.html

Return Value

The computed angle in radians.

- **Data type**: number / null

Examples

1. `arctan(x = 0) => 0`

See Also

- Inverse tangent explained by Wolfram MathWorld[21]

3.12 arctan2: Inverse tangent of two numbers.

Computes the arc tangent of two numbers `x` and `y`. It is similar to calculating the arc tangent of *y / x*, except that the signs of both arguments are used to determine the quadrant of the result.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated if any of the arguments is `null`.

Parameters

y (required) A number to be used as dividend.

- **Data type**: number / null

x (required) A number to be used as divisor.

- **Data type**: number / null

Return Value

The computed angle in radians.

- **Data type**: number / null

Examples

1. `arctan2(y = 0, x = 0) => 0`
2. `arctan2(y = null, x = 1.5) => null`

[21] http://mathworld.wolfram.com/InverseTangent.html

See Also

- Two-argument inverse tangent explained by Wikipedia[22]

3.13 array_contains: List contains an element

Checks whether the list (also known as *array*) specified for `data` contains the value specified in `element`.

Remarks:

- Data types MUST be checked strictly, for example a string with the content *1* is not equal to the number *1*.
- An integer *1* is equal to a floating point number *1.0* as `integer` is a sub-type of `number`. Still, this process may return unexpectedly `false` when comparing floating point numbers due to floating point inaccuracy in machine-based computation.
- Temporal strings are treated as normal strings and MUST NOT be interpreted.

Parameters

data (required) List in which to find a value in.

- **Data type**: array
- **Array items**: Any data type is allowed.

element (required) Value to find in `data`.

Return Value

Returns `true` if the list contains the value, false‘ otherwise.

- **Data type**: boolean

Examples

1. `array_contains(data = [1,2,3], element = 2) => true`
2. `array_contains(data = ["A","B","C"], element = "b") => false`
3. `array_contains(data = [1,2,3], element = "2") => false`
4. `array_contains(data = [1,2,3], element = 2) => true`
5. `array_contains(data = [1,2,null], element = null) => true`

[22] https://en.wikipedia.org/wiki/Atan2

6. `array_contains(data = [[1,2],[3,4]], element = [1,2]) => true`
7. `array_contains(data = [[1,2],[3,4]], element = 2) => false`
8. `array_contains(data = [{"a":"b"},{"c":"d"}], element = {"a":"b"}) => true`

3.14 array_element: Get an element from an array

Returns the element at the specified index from the array.

Parameters

data (required) An array.

- **Data type**: array
- **Array items**: Any data type is allowed.

index (required) The zero-based index of the element to retrieve.

- **Data type**: integer

return_nodata By default this process throws an `IndexOutOfBounds` exception if the index is invalid. If you want to return `null` instead, set this flag to `true`.

- **Data type**: boolean
- **Default value**: *false*

Return Value

The value of the requested element.

Exceptions

- `IndexOutOfBounds`: The array has no element with the specified index.

Examples

1. `array_element(data = [9,8,7,6,5], index = 2) => 7`
2. `array_element(data = ["A","B","C"], index = 0) => "A"`
3. `array_element(data = [], index = 0, return_nodata = true) => null`

3.15 arsinh: Inverse hyperbolic sine

Computes the inverse hyperbolic sine of `x`. It is the inverse function of the hyperbolic sine so that *arsinh(sinh(x)) = x*.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number.

- **Data type**: number / null

Return Value

The computed angle in radians.

- **Data type**: number / null

Examples

1. `arsinh(x = 0) => 0`

See Also

- Inverse hyperbolic sine explained by Wolfram MathWorld[23]

3.16 artanh: Inverse hyperbolic tangent

Computes the inverse hyperbolic tangent of `x`. It is the inverse function of the hyperbolic tangent so that *artanh(tanh(x)) = x*.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number.

- **Data type**: number / null

[23] http://mathworld.wolfram.com/InverseHyperbolicSine.html

Return Value

The computed angle in radians.

- **Data type**: number / null

Examples

1. `artanh(x = 0) => 0`

See Also

- Inverse hyperbolic tangent explained by Wolfram MathWorld[24]

3.17 between: Between comparison

By default this process checks whether `x` is greater than or equal to `min` and lower than or equal to `max`. Therefore, this process is an alias for `and([gte(x, min), lte(x, max)])` and all definitions from these processes apply here as well.

If `exclude_max` is set to `true` the upper bound is excluded so that the process checks whether `x` is greater than or equal to `min` and lower than `max`. In this case the process works the same as executing `and([gte(x, min), lt(x, max)])`.

Lower and upper bounds are not allowed to be swapped. So `min` MUST be lower than or equal to `max` or otherwise the process always returns `false`.

Parameters

x (required)

- **Data types**:
  - **number**
  - **null**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

min (required) Lower boundary (inclusive) to check against.

- **Data types**:

[24] http://mathworld.wolfram.com/InverseHyperbolicTangent.html

    - **number**
    - **date-time (string)**
    - **date (string)**
    - **time (string)**

max (required) Upper boundary (inclusive) to check against.

- **Data types**:
    - **number**
    - **date-time (string)**
    - **date (string)**
    - **time (string)**

exclude_max Exclude the upper boundary `max` if set to `true`. Defaults to `false`.

- **Data type**: boolean
- **Default value**: *false*

Return Value

`true` if `x` is between the specified bounds, otherwise `false`.

- **Data type**: boolean

Examples

1. `between(x = null, min = 0, max = 1) => null`
2. `between(x = 1, min = 0, max = 1) => true`
3. `between(x = 1, min = 0, max = 1, exclude_max = true) => false`
4. `between(x = 0.5, min = 1, max = 0) => false`
5. `between(x = -0.5, min = -1, max = 0) => true`
6. `between(x = "00:59:59Z", min = "01:00:00+01:00", max = "01:00:00Z") => true`
7. `between(x = "2018-07-23T17:22:45Z", min = "2018-01-01T00:00:00Z", max = "2018-12-31T23:59:59Z") => true`
8. `between(x = "2000-01-01", min = "2018-01-01", max = "2020-01-01") => false`
9. `between(x = "2018-12-31T17:22:45Z", min = "2018-01-01", max = "2018-12-31") => true`

10. `between(x = "2018-12-31T17:22:45Z", min = "2018-01-01", max = "2018-12-31", exclude_max = true) => false`

3.18 ceil: Round fractions up

The least integer greater than or equal to the number `x`.

The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number to round up.

- **Data type**: number / null

Return Value

The number rounded up.

- **Data type**: integer / null

Examples

1. `ceil(x = 0) => 0`
2. `ceil(x = 3.5) => 4`
3. `ceil(x = -0.4) => 0`
4. `ceil(x = -3.5) => -3`

See Also

- Ceiling explained by Wolfram MathWorld[25]

3.19 clip: Clips values between minimum and maximum values.

Clips an array of numbers between specified minimum and maximum values. All values larger than the maximal value will have the maximal value, all values lower than minimal value will have the minimal value.

[25] http://mathworld.wolfram.com/CeilingFunction.html

Parameters

data (required) An array of numbers.

- **Data type**: array<number|null>

min (required) Minimum value. All values lower than this value will be set to the value of this parameter.

- **Data type**: number

max (required) Maximum value. All values greater than this value will be set to the value of this parameter.

- **Data type**: number

Return Value

An array with the values clipped to the specified range.

- **Data type**: array<number|null>

Examples

1. `clip(data = [-2,-1,0,1,2], min = -1, max = 1) => [-1,-1,0,1,1]`
2. `clip(data = [-0.1,-0.001,null,0,0.25,0.75,1.001,null], min = 0, max = 1) => [0,0,null,0,0.25,0.75,1,null]`

3.20 cos: Cosine

Computes the cosine of `x`.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) An angle in radians.

- **Data type**: number / null

Return Value

The computed cosine of `x`.

- **Data type**: number / null

Examples

1. `cos(x = 0) => 1`

See Also

- Cosine explained by Wolfram MathWorld[26]

3.21 cosh: Hyperbolic cosine

Computes the hyperbolic cosine of `x`.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) An angle in radians.

- **Data type**: number / null

Return Value

The computed hyperbolic cosine of `x`.

- **Data type**: number / null

Examples

1. `cosh(x = 0) => 1`

See Also

- Hyperbolic cosine explained by Wolfram MathWorld[27]

[26] http://mathworld.wolfram.com/Cosine.html
[27] http://mathworld.wolfram.com/HyperbolicCosine.html

3.22 count: Count the number of elements

Gives the number of elements in an array that matches a certain criterion / expression.

Remarks:

- By default counts the number of valid elements. A valid element is every element for which `is_valid` returns `true`.
- To count all elements in a list set the `expression` parameter to boolean `true`.

Parameters

data (required) An array with elements of any data type.

- **Data type**: array
- **Array items**: Any data type is allowed.

expression An expression that is evaluated against each element in the array. An element is counted only if the expression returns `true`. Defaults to count valid elements in a list (see `is_valid`). Setting this parameter to boolean `true` counts all elements in the list.

- **Data types**:
  - **callback (object)**: An expression that is evaluated against each element in the array.
    - **Callback parameters**:
      - **x**: A single value from the array. Any data type could be passed.
  - **boolean**: Boolean 'true' counts all elements in the list.
    - **Constant value**: *true*
  - **null**: 'null' counts valid elements in the list.

Return Value

The counted number of elements.

- **Data type**: number

Examples

1. `count(data = []) => 0`
2. `count(data = [1,0,3,2]) => 4`
3. `count(data = ["ABC",null]) => 1`
4. `count(data = [false,null], expression = true) => 2`

5. `count(data = [0,1,2,3,4,5,null], expression = {"gt":{"process_id":"gt","arguments":{"x"`
   `=> 3`

3.23 create_raster_cube: Create an empty raster data cube

Creates a new raster data cube without dimensions.

Parameters

Return Value

An empty raster data cube.

- **Data type**: raster-cube (object)

3.24 cummax: Cumulative maxima

Finds cumulative maxima of an array of numbers. Every computed element is equal to the bigger one between current element and the previously computed element. The returned array and the input array have always the same length.

By default, no-data values are skipped, but stay in the result. Setting the `ignore_nodata` flag to `true` makes that once a no-data value / `null` is reached all following elements are set to `null` in the result.

Parameters

data (required) An array of numbers.

- **Data type**: array<number|null>

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is set for all the following elements.

- **Data type**: boolean
- **Default value**: *true*

Return Value

An array with the computed cumulative maxima.

- **Data type**: array<number|null>

Examples

1. `cummax(data = [1,3,5,3,1]) => [1,3,5,5,5]`
2. `cummax(data = [1,3,null,5,1]) => [1,3,null,5,5]`
3. `cummax(data = [1,3,null,5,1], ignore_nodata = false) => [1,3,null,null,null]`

3.25 cummin: Cumulative minima

Finds cumulative minima of an array of numbers. Every computed element is equal to the smaller one between current element and the previously computed element. The returned array and the input array have always the same length.

By default, no-data values are skipped, but stay in the result. Setting the `ignore_nodata` flag to `true` makes that once a no-data value / `null` is reached all following elements are set to `null` in the result.

Parameters

data (required) An array of numbers.

- **Data type**: array<number|null>

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is set for all the following elements.

- **Data type**: boolean
- **Default value**: *true*

Return Value

An array with the computed cumulative minima.

- **Data type**: array<number|null>

Examples

1. `cummin(data = [5,3,1,3,5]) => [5,3,1,1,1]`
2. `cummin(data = [5,3,null,1,5]) => [5,3,null,1,1]`
3. `cummin(data = [5,3,null,1,5], ignore_nodata = false) => [5,3,null,null,null]`

3.26 cumproduct: Cumulative products

Computes cumulative products of an array of numbers. Every computed element is equal to the product of current and all previous values. The returned array and the input array have always the same length.

By default, no-data values are skipped, but stay in the result. Setting the `ignore_nodata` flag to `true` makes that once a no-data value / `null` is reached all following elements are set to `null` in the result.

Parameters

data (required) An array of numbers.

- **Data type**: array<number|null>

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is set for all the following elements.

- **Data type**: boolean
- **Default value**: *true*

Return Value

An array with the computed cumulative products.

- **Data type**: array<number|null>

Examples

1. `cumproduct(data = [1,3,5,3,1]) => [1,3,15,45,45]`
2. `cumproduct(data = [1,2,3,null,3,1]) => [1,2,6,null,18,18]`
3. `cumproduct(data = [1,2,3,null,3,1], ignore_nodata = false) => [1,2,6,null,null,null]`

3.27 cumsum: Cumulative sums

Computes cumulative sums of an array of numbers. Every computed element is equal to the sum of current and all previous values. The returned array and the input array have always the same length.

By default, no-data values are skipped, but stay in the result. Setting the `ignore_nodata` flag to `true` makes that once a no-data value / `null` is reached all following elements are set to `null` in the result.

Parameters

data (required) An array of numbers.

- **Data type**: array<number|null>

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is set for all the following elements.

- **Data type**: boolean
- **Default value**: *true*

Return Value

An array with the computed cumulative sums.

- **Data type**: array<number|null>

Examples

1. `cumsum(data = [1,3,5,3,1]) => [1,4,9,12,13]`
2. `cumsum(data = [1,3,null,3,1]) => [1,4,null,7,8]`
3. `cumsum(data = [1,3,null,3,1], ignore_nodata = false) => [1,4,null,null,null]`

3.28 debug: Send debugging information to subscribed clients

Sends debugging information about the data to clients, which are subscribed to the topic `openeo.jobs.debug`.

Parameters

data (required) Data to send.

id An identifier to help identify the message in a bunch of other messages.

- **Data type**: string

Return Value

`false` if the information could not be sent, `true` otherwise.

- **Data type**: boolean

See Also

- Information about the openEO API for Subscriptions[28]

3.29 divide: Division of a sequence of numbers

Divides the first element in a sequential array of numbers by all other elements.

The computations should follow IEEE Standard 754[29] so that for example a division by zero should result in śinfinity if the processing environment supports it. Otherwise an exception must the thrown for incomputable results.

By default no-data values are ignored. Setting `ignore_nodata` to `false` considers no-data values so that `null` is returned if any element is such a value.

Parameters

data (required) An array of numbers with at least two elements.

- **Data type**: array<number|null>
- **Min. number of items**: 2

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if any value is such a value.

- **Data type**: boolean
- **Default value**: *true*

Return Value

The computed result of the sequence of numbers.

- **Data type**: number / null

Exceptions

- `DivisorMissing`: Division requires at least two numbers (a dividend and one or more divisors).

[28] https://open-eo.github.io/openeo-api/v/0.4.0/apireference-subscriptions/#publish-openeojobsdebug
[29] https://ieeexplore.ieee.org/document/4610935

Examples

1. `divide(data = [15,5]) => 3`
2. `divide(data = [-2,4,2.5]) => -0.2`
3. `divide(data = [1,null], ignore_nodata = false) => null`

See Also

- Division explained by Wolfram MathWorld[30]
- IEEE Standard 754-2008 for Floating-Point Arithmetic[31]

3.30 e: Euler's number (e)

The real number *e* is a mathematical constant that is the base of the natural logarithm such that *ln(e) = 1*. The numerical value is approximately *2.71828*.

Parameters

Return Value

The numerical value of Euler's number.

- **Data type**: number

See Also

- Mathematical constant e explained by Wolfram MathWorld[32]

3.31 eq: Equal to comparison

Compares whether `x` is strictly equal to `y`.

Remarks:

- Data types MUST be checked strictly, for example a string with the content *1* is not equal to the number *1*. Nevertheless, an integer *1* is equal to a floating point number *1.0* as `integer` is a sub-type of `number`.
- If any of the operands is `null`, the return value is `null`.

[30] http://mathworld.wolfram.com/Division.html
[31] https://ieeexplore.ieee.org/document/4610935
[32] http://mathworld.wolfram.com/e.html

- Strings are expected to be encoded in UTF-8 by default.
- Temporal strings MUST be compared differently than other strings and MUST NOT be compared based on their string representation due to different possible representations. For example, the UTC time zone representation `Z` has the same meaning as `+00:00`.

Parameters

x (required) First operand.

- **Data types**:
  - **number**
  - **boolean**
  - **null**
  - **string**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

y (required) Second operand.

- **Data types**:
  - **number**
  - **boolean**
  - **null**
  - **string**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

delta Only applicable for comparing two numbers. If this optional parameter is set to a positive non-zero number the equality of two numbers is checked against a delta value. This is especially useful to circumvent problems with floating point inaccuracy in machine-based computation.

This option is basically an alias for the following computation: `lte(abs(minus([x, y]), delta)`

- **Data type**: number / null
- **Default value**: *null*

case_sensitive Only applicable for comparing two strings. Case sensitive comparison can be disabled by setting this parameter to `false`.

- **Data type**: boolean
- **Default value**: *true*

Return Value

Returns `true` if `x` is equal to `y`, `null` if any of the operands is `null`, otherwise `false`.

- **Data type**: boolean / null

Examples

1. `eq(x = 1, y = null) => null`
2. `eq(x = 1, y = 1) => true`
3. `eq(x = 1, y = "1") => false`
4. `eq(x = 1.02, y = 1, delta = 0.01) => false`
5. `eq(x = -1, y = -1.001, delta = 0.01) => true`
6. `eq(x = 115, y = 110, delta = 10) => true`
7. `eq(x = "Test", y = "test") => false`
8. `eq(x = "Test", y = "test", case_sensitive = false) => true`
9. `eq(x = "Ä", y = "ä", case_sensitive = false) => true`
10. `eq(x = "00:00:00+00:00", y = "00:00:00Z") => true`
11. `eq(x = "2018-01-01T12:00:00Z", y = "2018-01-01T12:00:00") => false`
12. `eq(x = "2018-01-01T00:00:00Z", y = "2018-01-01T01:00:00+01:00") => true`

See Also

- Information about the supported temporal formats.[33]

3.32 exp: Exponentiation to the base e

Exponential function to the base *e* raised to the power of `p`. This process is an alias for e^p / `power(e(), p)`.

The no-data value `null` is passed through and therefore gets propagated.

[33] https://open-eo.github.io/openeo-api/v/0.4.0/processes/#openeo-specific-formats

Parameters

p (required) The numerical exponent.

- **Data type**: number / null

Return Value

The computed value for *e* raised to the power of `p`.

- **Data type**: number / null

Examples

1. `exp(p = 0) => 1`
2. `exp(p = null) => null`

See Also

- Exponential function explained by Wolfram MathWorld[34]

3.33 extrema: Minimum and maximum values

Two element array containing the minimum and the maximum values of `data`.

This process is basically an alias for calling both `min` and `max`, but may be implemented more performant by back-ends as it only needs to iterate over the data once instead of twice.

Parameters

data (required) An array of numbers.

- **Data type**: array<number|null>

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that an array with two `null` values is returned if any value is such a value.

- **Data type**: boolean
- **Default value**: *true*

[34] http://mathworld.wolfram.com/ExponentialFunction.html

Return Value

An array containing the minimum and maximum values for the specified numbers. The first element is the minimum, the second element is the maximum. If the input array is empty both elements are set to `null`.

- **Data types**:
  - **array<number>**:
    - **Min. number of items**: 2
    - **Max. number of items**: 2
  - **array<null>**:
    - **Min. number of items**: 2
    - **Max. number of items**: 2

Examples

1. `extrema(data = [1,0,3,2]) => [0,3]`
2. `extrema(data = [5,2.5,null,-0.7]) => [-0.7,5]`
3. `extrema(data = [1,0,3,null,2], ignore_nodata = false) => [null,null]`
4. `extrema(data = []) => [null,null]`

3.34 filter_bands: Filter the bands by name

Filters the bands in the data cube so that bands that don't match any of the criteria are dropped from the data cube. The data cube is expected to have only one dimension of type `bands`.

The following criteria can be used to select bands:

- `bands`: band name (e.g. `B01` or `B8A`)
- `common_names`: common band names (e.g. `red` or `nir`)
- `wavelengths`: ranges of wavelengths in micrometres (μm) (e.g. 0.5 - 0.6)

To keep algorithms interoperable it is recommended to prefer the common bands names or the wavelengths over collection and/or back-end specific band names.

If multiple criteria are specified, any of them must match and not all of them, i.e. they are combined with an OR-operation. If no criteria is specified, the `BandFilterParameterMissing` exception must be thrown.

Important: The order of the specified array defines the order of the bands in the data cube, which can be important for subsequent processes. If multiple bands are matched by a single criterion (e.g. a range of wavelengths), they are ordered alphabetically by band names. Bands without names have an arbitrary order.

Parameters

data (required) A data cube with bands.

- **Data type**: raster-cube (object)

bands A list of band names.

The order of the specified array defines the order of the bands in the data cube.

- **Data type**: array<string>

common_names A list of common band names.

The order of the specified array defines the order of the bands in the data cube.

- **Data type**: array<string>

wavelengths A list of sub-lists with each sub-list consisting of two elements. The first element is the minimum wavelength and the second element is the maximum wavelength. Wavelengths are specified in micrometres (μm).

The order of the specified array defines the order of the bands in the data cube.

- **Data type**: array<array<number»
- **Array items**:
  - **Data type**: array<number>
  - **Min. number of items**: 2
  - **Max. number of items**: 2
  - **Examples**:
    1. `[[0.45,0.5],[0.6,0.7]]`

Return Value

A data cube limited to a subset of its original bands. Therefore, the cardinality is potentially lower, but the resolution and the number of dimensions are the same as for the original data cube.

- **Data type**: raster-cube (object)

Exceptions

- `BandFilterParameterMissing`: The process 'filter_bands' requires any of the parameters 'bands', 'common_names' or 'wavelengths' to be set.

See Also

- List of common band names as specified by the STAC specification[35]

3.35 filter_bbox: Spatial filter using a bounding box

Limits the data cube to the specified bounding box.

The filter retains a pixel in the data cube if the point at the pixel center intersects with the bounding box (as defined in the Simple Features standard by the OGC).

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

extent (required) A bounding box, which may include a vertical axis (see `base` and `height`).

The coordinate reference system of the extent must be specified as EPSG[36] code or PROJ[37] definition.

- **Data type**: bounding-box (object)
- **Required**:
  1. west
  2. south
  3. east
  4. north
- **Properties**:
  - **west**: West (lower left corner, coordinate axis 1).
    - **Data type**: number
  - **south**: South (lower left corner, coordinate axis 2).
    - **Data type**: number
  - **east**: East (upper right corner, coordinate axis 1).
    - **Data type**: number
  - **north**: North (upper right corner, coordinate axis 2).
    - **Data type**: number

[35] https://github.com/radiantearth/stac-spec/tree/master/extensions/eo#common-band-names
[36] http://www.epsg.org
[37] https://proj4.org

- **base**: Base (optional, lower left corner, coordinate axis 3).
  - **Data type**: number / null
  - **Default value**: *null*
- **height**: Height (optional, upper right corner, coordinate axis 3).
  - **Data type**: number / null
  - **Default value**: *null*
- **crs**: Coordinate reference system of the extent specified as EPSG code or PROJ definition. Whenever possible, it is recommended to use EPSG codes instead of PROJ definitions. Defaults to '4326' (EPSG code 4326) unless the client explicitly requests a different coordinate reference system.
  - **Schema**:
    - **Data types**:
      - **epsg-code (integer)**:
        - **Examples**:
          1. `7099`
      - **proj-definition (string)**:
        - **Examples**:
          1. `"+proj=moll +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs"`

Return Value

A data cube restricted to the bounding box. Therefore, the cardinality is potentially lower, but the resolution and the number of dimensions are the same as for the original data cube.

- **Data type**: raster-cube (object)

See Also

- PROJ parameters for cartographic projections[38]
- Official EPSG code registry[39]
- Unofficial EPSG code database[40]
- Simple Features standard by the OGC[41]

[38] https://proj4.org/usage/projections.html
[39] http://www.epsg-registry.org
[40] http://www.epsg.io
[41] http://www.opengeospatial.org/standards/sfa

3.36 filter_polygon: Spatial filter using polygons

Limits the data cube over the spatial dimensions to the specified polygons.

The filter retains a pixel in the data cube if the point at the pixel center intersects with at least one of the polygons (as defined in the Simple Features standard by the OGC).

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

polygons (required) One or more polygons used for filtering, either specified as GeoJSON or vector data cube.

For GeoJSON this can be one of the following GeoJSON types:

- A `Polygon` geometry,
- a `GeometryCollection` containing Polygons,
- a `Feature` with a `Polygon` geometry or
- a `FeatureCollection` containing `Feature`s with a `Polygon` geometry.
- **Data types**:
  - **geojson (object)**
  - **vector-cube (object)**

Return Value

A data cube restricted to the specified polygons. Therefore, the cardinality is potentially lower, but the resolution and the number of dimensions are the same as for the original data cube.

- **Data type**: raster-cube (object)

See Also

- Simple Features standard by the OGC[42]

3.37 filter_temporal: Temporal filter for a date and/or time intervals

Limits the data cube to the specified interval of dates and/or times.

[42] http://www.opengeospatial.org/standards/sfa

More precisely, the filter checks whether the temporal dimension value is greater than or equal to the lower boundary (start date/time) and the temporal dimension value is less than the value of the upper boundary (end date/time). This corresponds to a left-closed interval, which contains the lower boundary but not the upper boundary.

If the dimension is set to `null` (it's the default value), the data cube is expected to only have one temporal dimension.

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

extent (required) Left-closed temporal interval, i.e. an array with exactly two elements:

1. The first element is the start of the date and/or time interval. The specified instance in time is **included** in the interval.
2. The second element is the end of the date and/or time interval. The specified instance in time is **excluded** from the interval.

The specified temporal strings follow RFC 3339[43]. Although RFC 3339 prohibits the hour to be '24'[44], **this process allows the value '24' for the hour** of an end time in order to make it possible that left-closed time intervals can fully cover the day.

Also supports open intervals by setting one of the boundaries to `null`, but never both.

- **Data type**: temporal-interval (array)
- **Min. number of items**: 2
- **Max. number of items**: 2
- **Array items**:
  - **Data types**:
    - **date-time (string)**
    - **date (string)**
    - **time (string)**
    - **null**
- **Examples**:
  1. `["2015-01-01","2016-01-01"]`
  2. `["12:00:00Z","24:00:00Z"]`

[43] https://tools.ietf.org/html/rfc3339
[44] https://tools.ietf.org/html/rfc3339#section-5.7

dimension The temporal dimension to filter on. If the dimension is not set or is set to `null`, the data cube is expected to only have one temporal dimension.

Note: The default dimensions a data cube provides are described in the collection's metadata field `cube:dimensions`.

- **Data type**: string / null
- **Default value**: *null*

Return Value

A data cube restricted to the specified temporal extent. Therefore, the cardinality is potentially lower, but the resolution and the number of dimensions are the same as for the original data cube.

- **Data type**: raster-cube (object)

See Also

- Information about the supported temporal formats.[45]

3.38 filter: Filter based on a logical expression.

Filters the dimension values based on a logical expression so that afterwards each dimension value in the data cube conforms to the expression.

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

expression (required) An expression that is evaluated against each dimension value in the specified dimension. A dimension value is dropped from the data cube if the expression returns `false`. The data type of the parameter depends on the dimension values stored for the dimension.

- **Data type**: callback (object)
- **Callback parameters**:
  - **value**: A single dimension value to compare against.
    - **Data types**:

[45] https://open-eo.github.io/openeo-api/v/0.4.0/processes/#openeo-specific-formats

- **number**
- **string**
- **date-time (string)**
- **date (string)**
- **time (string)**

dimension (required) The dimension to filter on.

Remarks:

- The default dimensions a data cube provides are described in the collection's metadata field `cube:dimensions`.
- There could be multiple spatial dimensions such as `x`, `y` or `z`.
- For multi-spectral imagery there is usually a separate dimension of type `bands` for the bands.
- **Data type**: string

Return Value

A data cube restricted by the specified expression. Therefore, the cardinality is potentially lower, but the resolution and the number of dimensions are the same as for the original data cube.

- **Data type**: raster-cube (object)

3.39 find_collections: Search for collections by metadata properties

Searches for collections available on the current back-end by metadata properties and returns ids of collections that match all criteria. `property` can be used to get metadata properties.

A single collection can be selected using array operations such as `first`, `last` and `array_element` and afterwards be loaded using `load_collection`.

Parameters

expression (required) An expression that is evaluated against each collection the back-end offers. The expression filters the collections to include only collection ids which the given expression returns `true` for. `property` can be used to retrieve metadata properties.

- **Data type**: callback (object)
- **Callback parameters**:
  - **id**: The collection id.

- **Data type**: collection-id (string)
- **Pattern**: `^[A-Za-z0-9_\-\.~/]+$`

Return Value

An array of collection ids.

- **Data type**: array<collection-id:string>
- **Array items**:
  - **Data type**: collection-id (string)
  - **Pattern**: `^[A-Za-z0-9_\-\.~/]+$`

3.40 first: First element

Gives the first element of an array. For an empty array `null` is returned.

Parameters

data (required) An array with elements of any data type. An empty array resolves always with `null`.

- **Data type**: array
- **Array items**: Any data type is allowed.

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if the first value is such a value.

- **Data type**: boolean
- **Default value**: *true*

Return Value

The first element of the input array.

Examples

1. `first(data = [1,0,3,2]) => 1`
2. `first(data = [null,"A","B"]) => "A"`

3. `first(data = [null,2,3], ignore_nodata = false) => null`
4. `first(data = []) => null`

3.41 floor: Round fractions down

The greatest integer less than or equal to the number `x`.

This process is *not* an alias for the `int` process as defined by some mathematicians, see the examples for negative numbers in both processes for differences.

The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number to round down.

- **Data type**: number / null

Return Value

The number rounded down.

- **Data type**: integer / null

Examples

1. `floor(x = 0) => 0`
2. `floor(x = 3.5) => 3`
3. `floor(x = -0.4) => -1`
4. `floor(x = -3.5) => -4`

See Also

- Floor explained by Wolfram MathWorld[46]

3.42 gt: Greater than comparison

Compares whether `x` is strictly greater than `y`.

Remarks:

[46] http://mathworld.wolfram.com/FloorFunction.html

- If any of the operands is `null`, the return value is `null`.
- Temporal strings can *not* be compared based on their string representation due to the time zone / time-offset representations.
- Comparing strings is currently not supported, but is planned to be added in the future.

Parameters

x (required) First operand.

- **Data types**:
  - **number**
  - **null**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

y (required) Second operand.

- **Data types**:
  - **number**
  - **null**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

Return Value

`true` if `x` is strictly greater than `y` or `null` if any of the operands is `null`, otherwise `false`.

- **Data type**: boolean / null

Examples

1. `gt(x = 1, y = null) => null`
2. `gt(x = 0, y = 0) => false`
3. `gt(x = 2, y = 1) => true`
4. `gt(x = -0.5, y = -0.6) => true`
5. `gt(x = "00:00:00Z", y = "00:00:00+01:00") => true`

6. `gt(x = "1950-01-01T00:00:00Z", y = "2018-01-01T12:00:00Z") => false`
7. `gt(x = "2018-01-01T12:00:00+00:00", y = "2018-01-01T12:00:00Z") => false`

See Also

- Information about the supported temporal formats.[47]

3.43 gte: Greater than or equal to comparison

Compares whether `x` is greater than or equal to `y`.

Remarks:

- If any of the operands is `null`, the return value is `null`.
- Temporal strings can *not* be compared based on their string representation due to the time zone / time-offset representations.
- Comparing strings is currently not supported, but is planned to be added in the future.

Parameters

x (required) First operand.

- **Data types**:
  - **number**
  - **null**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

y (required) Second operand.

- **Data types**:
  - **number**
  - **null**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

[47] https://open-eo.github.io/openeo-api/v/0.4.0/processes/#openeo-specific-formats

Return Value

`true` if `x` is greater than or equal to `y` or `null` if any of the operands is `null`, otherwise `false`.

- **Data type**: boolean / null

Examples

1. `gte(x = 1, y = null) => null`
2. `gte(x = 0, y = 0) => true`
3. `gte(x = 1, y = 2) => false`
4. `gte(x = -0.5, y = -0.6) => true`
5. `gte(x = "00:00:00Z", y = "00:00:00+01:00") => true`
6. `gte(x = "1950-01-01T00:00:00Z", y = "2018-01-01T12:00:00Z") => false`
7. `gte(x = "2018-01-01T12:00:00+00:00", y = "2018-01-01T12:00:00Z") => true`

See Also

- Information about the supported temporal formats.[48]

3.44 if: If-Then-Else conditional

Returns the value of the `accept` parameter if the expression is `true` or the value of the `reject` parameter if the expression is `false`. This works similar to an if-then-else construct.

The no-data value `null` is passed through and therefore gets propagated.

Parameters

expression (required) A boolean value.

- **Data type**: boolean / null

accept A value that is returned if the boolean expression is `true`. Defaults to `true`.

- **Default value**: *true*

reject A value that is returned if the boolean expression is `false`. Defaults to `false`.

- **Default value**: *false*

[48] https://open-eo.github.io/openeo-api/v/0.4.0/processes/#openeo-specific-formats

Return Value

Either the `accept` or `reject` argument depending on the given boolean expression.

Examples

1. `if(expression = true) => true`
2. `if(expression = null) => null`
3. `if(expression = false) => false`
4. `if(expression = true, accept = "A") => "A"`
5. `if(expression = false, accept = [1,2,3], reject = [4,5,6]) => [4,5,6]`

3.45 int: Integer part of a number

The integer part of the real number `x`.

This process is *not* an alias for the `floor` process as defined by some mathematicians, see the examples for negative numbers in both processes for differences.

The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number.

- **Data type**: number / null

Return Value

Integer part of the number.

- **Data type**: integer / null

Examples

1. `int(x = 0) => 0`
2. `int(x = 3.5) => 3`
3. `int(x = -0.4) => 0`
4. `int(x = -3.5) => -3`

See Also

- Integer Part explained by Wolfram MathWorld[49]

3.46 is_nan: Value is not a number

Checks whether the specified value `x` is not a number (often abbreviated as `NaN`). The definition of `NaN` follows the IEEE Standard 754[50]. All non-numeric data types MUST also return `true`.

Parameters

x (required) The data to check.

Return Value

`true` if the data is not a number, otherwise `false`

- **Data type**: boolean

Examples

1. `is_nan(x = 1) => false`
2. `is_nan(x = "Test") => true`

See Also

- IEEE Standard 754-2008 for Floating-Point Arithmetic[51]

3.47 is_nodata: Value is not a no-data value

Checks whether the specified data is a missing data, i.e. equals to any of the no-data values / `null`.

Parameters

x (required) The data to check.

[49] http://mathworld.wolfram.com/IntegerPart.html
[50] https://ieeexplore.ieee.org/document/4610935
[51] https://ieeexplore.ieee.org/document/4610935

Return Value

`true` if the data is a no-data value, otherwise `false`

- **Data type**: boolean

Examples

1. `is_nodata(x = 1) => false`
2. `is_nodata(x = "Test") => false`
3. `is_nodata(x = null) => true`

3.48 is_valid: Value is valid data

Checks whether the specified value `x` is valid. A value is considered valid if it is

1. not a no-data value (`null`) and
2. a finite number (only if `x` is a number). The definition of finite and infinite numbers follows the IEEE Standard 754[52].

Parameters

x (required) The data to check.

Return Value

`true` if the data is valid, otherwise `false`.

- **Data type**: boolean

Examples

1. `is_valid(x = 1) => true`
2. `is_valid(x = "Test") => true`
3. `is_valid(x = null) => false`

See Also

- IEEE Standard 754-2008 for Floating-Point Arithmetic[53]

[52] https://ieeexplore.ieee.org/document/4610935
[53] https://ieeexplore.ieee.org/document/4610935

3.49 last: Last element

Gives the last element of an array. For an empty array `null` is returned.

Parameters

data (required) An array with elements of any data type. An empty array resolves always with `null`.

- **Data type**: array
- **Array items**: Any data type is allowed.

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if the last value is such a value.

- **Data type**: boolean
- **Default value**: *true*

Return Value

The last element of the input array.

Examples

1. `last(data = [1,0,3,2]) => 2`
2. `last(data = ["A","B",null]) => "B"`
3. `last(data = [0,1,null], ignore_nodata = false) => null`
4. `last(data = []) => null`

3.50 linear_scale_range: Linear transformation between two ranges

Performs a linear transformation between the input and output range.

The underlying formula is: `((x - inputMin) / (inputMax - inputMin)) * (outputMax - outputMin) + outputMin`.

Potential use case include

- scaling values to the 8-bit range (0 - 255) often used for numeric representation of values in one of the channels of the RGB colour model[54] or

[54] https://en.wikipedia.org/wiki/RGB_color_model#Numeric_representations

- calculating percentages (0 - 100).

The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number to transform.

- **Data type**: number / null

inputMin (required) Minimum value the input can obtain.

- **Data type**: number

inputMax (required) Maximum value the input can obtain.

- **Data type**: number

outputMin Minimum value of the desired output range.

- **Data type**: number
- **Default value**: 0

outputMax Maximum value of the desired output range.

- **Data type**: number
- **Default value**: 1

Return Value

The transformed number.

- **Data type**: number / null

Examples

1. `linear_scale_range(x = 0.3, inputMin = -1, inputMax = 1, outputMin = 0, outputMax = 255) => 165.75`
2. `linear_scale_range(x = 25.5, inputMin = 0, inputMax = 255) => 0.1`
3. `linear_scale_range(x = null, inputMin = 0, inputMax = 100) => null`

3.51 ln: Natural logarithm

The natural logarithm is the logarithm to the base *e* of the number `x`. This process is an alias for the *log* process with the base set to *e*: `log(x, e())`. The natural logarithm is the inverse function of taking *e* to the power x.

The computations should follow IEEE Standard 754[55] so that for example `ln(0)` should result in śinfinity if the processing environment supports it. Otherwise an exception must the thrown for incomputable results.

The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number to compute the natural logarithm for.

- **Data type**: number / null

Return Value

The computed natural logarithm.

- **Data type**: number / null

Examples

1. `ln(x = 1) => 0`

See Also

- Natural logarithm explained by Wolfram MathWorld[56]
- IEEE Standard 754-2008 for Floating-Point Arithmetic[57]

3.52 load_collection: Load a collection

Loads a collection from the current back-end by its id and returns it as processable data cube. The data that is added to the data cube can be restricted with the additional `spatial_extent`, `temporal_extent`, `bands` and `properties`.

Remarks:

[55] https://ieeexplore.ieee.org/document/4610935
[56] http://mathworld.wolfram.com/NaturalLogarithm.html
[57] https://ieeexplore.ieee.org/document/4610935

- The bands (and all dimensions that specify nominal dimension values) are expected to be ordered as specified in the metadata if the `bands` parameter is set to `null`.
- If no additional parameter is specified this would imply that the whole data set is expected to be loaded. Due to the large size of many data sets this is not recommended and may be optimized by back-ends to only load the data that is actually required after evaluating subsequent processes such as filters. This means that the pixel values should be processed only after the data has been limited to the required extents and as a consequence also to a manageable size.

Parameters

id (required) The collection id.

- **Data type**: collection-id (string)
- **Pattern**: `^[A-Za-z0-9_\-\.~/]+$`

spatial_extent (required) Limits the data to load from the collection to the specified bounding box or polygons.

The process puts a pixel into the data cube if the point at the pixel center intersects with the bounding box or any of the polygons (as defined in the Simple Features standard by the OGC).

The coordinate reference system of the bounding box must be specified as EPSG[58] code or PROJ[59] definition.

The GeoJSON can be one of the following GeoJSON types:

- A `Polygon` geometry,
- a `GeometryCollection` containing Polygons,
- a `Feature` with a `Polygon` geometry or
- a `FeatureCollection` containing `Feature`s with a `Polygon` geometry.

Set this parameter to `null` to set no limit for the spatial extent. Be careful with this when loading large datasets!

- **Data types**:
  - **bounding-box (object)**:
    - **Required**:
      1. west
      2. south
      3. east

[58] http://www.epsg.org
[59] https://proj4.org

4. north

- **Properties**:
  - **west**: West (lower left corner, coordinate axis 1).
    - **Data type**: number
  - **south**: South (lower left corner, coordinate axis 2).
    - **Data type**: number
  - **east**: East (upper right corner, coordinate axis 1).
    - **Data type**: number
  - **north**: North (upper right corner, coordinate axis 2).
    - **Data type**: number
  - **base**: Base (optional, lower left corner, coordinate axis 3).
    - **Data type**: number / null
    - **Default value**: *null*
  - **height**: Height (optional, upper right corner, coordinate axis 3).
    - **Data type**: number / null
    - **Default value**: *null*
  - **crs**: Coordinate reference system of the extent specified as EPSG code or PROJ definition. Whenever possible, it is recommended to use EPSG codes instead of PROJ definitions. Defaults to '4326' (EPSG code 4326) unless the client explicitly requests a different coordinate reference system.
    - **Schema**:
      - **Data types**:
        - **epsg-code (integer)**:
          - **Examples**:
            1. `7099`
        - **proj-definition (string)**:
          - **Examples**:
            1. `"+proj=moll +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs"`
- **geojson (object)**
- **null**

temporal_extent (required) Limits the data to load from the collection to the specified left-closed temporal interval. Applies to all temporal dimensions if there are multiple of them. The interval has to be specified as an array with exactly two elements:

1. The first element is the start of the date and/or time interval. The specified instance in time is **included** in the interval.
2. The second element is the end of the date and/or time interval. The specified instance in time is **excluded** from the interval.

The specified temporal strings follow RFC 3339[60]. Although RFC 3339 prohibits the hour to be '24'[61], **this process allows the value '24' for the hour** of an end time in order to make it possible that left-closed time intervals can fully cover the day.

Also supports open intervals by setting one of the boundaries to `null`, but never both.

Set this parameter to `null` to set no limit for the spatial extent. Be careful with this when loading large datasets!

- **Data types**:
  - **temporal-interval (array)**:
    - **Min. number of items**: 2
    - **Max. number of items**: 2
    - **Array items**:
      - **Data types**:
        - **date-time (string)**
        - **date (string)**
        - **time (string)**
        - **null**
    - **Examples**:
      1. `["2015-01-01","2016-01-01"]`
      2. `["12:00:00Z","24:00:00Z"]`
  - **null**

bands Only adds the specified bands into the data cube so that bands that don't match the list of band names are not available. Applies to all dimensions of type `bands` if there are multiple of them.

The order of the specified array defines the order of the bands in the data cube.

- **Data types**:
  - **array<string>**
  - **null**

[60] https://tools.ietf.org/html/rfc3339
[61] https://tools.ietf.org/html/rfc3339#section-5.7

properties Limits the data by metadata properties to include only data in the data cube which all given expressions return `true` for (AND operation).

Specify key-value-pairs with the keys being the name of the metadata property, which can be retrieved with the openEO Data Discovery for Collections. The values must be expressions to be evaluated against the collection metadata, see the example.

Note: Back-ends may not pass the actual value to the expressions, but pass a proprietary index or a placeholder so that they can use the expressions to query against another data source. So debugging on the callback parameter `value` may lead to unexpected results.

- **Data types**:
  - **object**:
    - **AdditionalProperties**:
      - **Data type**: callback (object)
      - **Callback parameters**:
        - **value**: The property value. Any data type could be passed.
  - **null**

Return Value

A data cube for further processing.

- **Data types**:
  - **raster-cube (object)**
  - **vector-cube (object)**

See Also

- PROJ parameters for cartographic projections[62]
- Official EPSG code registry[63]
- Unofficial EPSG code database[64]
- Simple Features standard by the OGC[65]
- Information about the supported temporal formats.[66]

[62] https://proj4.org/usage/projections.html
[63] http://www.epsg-registry.org
[64] http://www.epsg.io
[65] http://www.opengeospatial.org/standards/sfa
[66] https://open-eo.github.io/openeo-api/v/0.4.0/processes/#openeo-specific-formats

3.53 load_result: Load batch job results

Loads batch job results by job id from the local user workspace / data store. The job must have been stored by the authenticated user on the back-end currently connected to.

Parameters

id (required) The id of a batch job with results.

- **Data type**: job-id (string)
- **Pattern**: `^[A-Za-z0-9_\-\.~]+$`

Return Value

A data cube for further processing.

- **Data types**:
  - **raster-cube (object)**
  - **vector-cube (object)**

3.54 log: Logarithm to a base

Logarithm to the base `base` of the number `x` is defined to be the inverse function of taking b to the power of x.

The computations should follow IEEE Standard 754[67] so that for example `log(0, 2)` should result in śinfinity if the processing environment supports it. Otherwise an exception must the thrown for incomputable results.

The no-data value `null` is passed through and therefore gets propagated if any of the arguments is `null`.

Parameters

x (required) A number to compute the logarithm for.

- **Data type**: number / null

base (required) The numerical base.

- **Data type**: number / null

[67] https://ieeexplore.ieee.org/document/4610935

Return Value

The computed logarithm.

- **Data type**: number / null

Examples

1. `log(x = 10, base = 10) => 1`
2. `log(x = 2, base = 2) => 1`
3. `log(x = 4, base = 2) => 2`
4. `log(x = 1, base = 16) => 0`

See Also

- Logarithm explained by Wolfram MathWorld[68]
- IEEE Standard 754-2008 for Floating-Point Arithmetic[69]

3.55 lt: Less than comparison

Compares whether `x` is strictly less than `y`.

Remarks:

- If any of the operands is `null`, the return value is `null`.
- Temporal strings can *not* be compared based on their string representation due to the time zone / time-offset representations.
- Comparing strings is currently not supported, but is planned to be added in the future.

Parameters

x (required) First operand.

- **Data types**:
  - **number**
  - **null**
  - **date-time (string)**
  - **date (string)**

[68] http://mathworld.wolfram.com/Logarithm.html
[69] https://ieeexplore.ieee.org/document/4610935

- **time (string)**

y (required) Second operand.

- **Data types**:
  - **number**
  - **null**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

Return Value

`true` if `x` is strictly less than `y`, `null` if any of the operands is `null`, otherwise `false`.

- **Data type**: boolean / null

Examples

1. `lt(x = 1, y = null) => null`
2. `lt(x = 0, y = 0) => false`
3. `lt(x = 1, y = 2) => true`
4. `lt(x = -0.5, y = -0.6) => false`
5. `lt(x = "00:00:00+01:00", y = "00:00:00Z") => true`
6. `lt(x = "1950-01-01T00:00:00Z", y = "2018-01-01T12:00:00Z") => true`
7. `lt(x = "2018-01-01T12:00:00+00:00", y = "2018-01-01T12:00:00Z") => false`

See Also

- Information about the supported temporal formats.[70]

3.56 lte: Less than or equal to comparison

Compares whether `x` is less than or equal to `y`.

Remarks:

- If any of the operands is `null`, the return value is `null`.

[70] https://open-eo.github.io/openeo-api/v/0.4.0/processes/#openeo-specific-formats

- Temporal strings can *not* be compared based on their string representation due to the time zone / time-offset representations.
- Comparing strings is currently not supported, but is planned to be added in the future.

Parameters

x (required) First operand.

- **Data types**:
  - **number**
  - **null**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

y (required) Second operand.

- **Data types**:
  - **number**
  - **null**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

Return Value

`true` if `x` is less than or equal to `y`, `null` if any of the operands is `null`, otherwise `false`.

- **Data type**: boolean / null

Examples

1. `lte(x = 1, y = null) => null`
2. `lte(x = 0, y = 0) => true`
3. `lte(x = 1, y = 2) => true`
4. `lte(x = -0.5, y = -0.6) => false`
5. `lte(x = "00:00:00+01:00", y = "00:00:00Z") => true`
6. `lte(x = "1950-01-01T00:00:00Z", y = "2018-01-01T12:00:00Z") => true`

7. `lte(x = "2018-01-01T12:00:00+00:00", y = "2018-01-01T12:00:00Z") => true`

See Also

- Information about the supported temporal formats.[71]

3.57 mask: Apply a mask

Applies a mask to a raster data cube. A mask can either be specified as:

- **Raster data cube** for which parallel pixels among `data` and `mask` are compared and those pixels in `data` are replaced, which are non-zero (for numbers) or `true` (for boolean values) in `mask`.
- **GeoJSON or vector data cube** containing one or more polygons. All pixels for which the point at the pixel center intersects with the corresponding polygon (as defined in the Simple Features standard by the OGC) are replaced.

The pixel values are replaced with the value specified for `replacement`, which defaults to `null` (no data). No data values will be left untouched by the masking operation.

Parameters

data (required) A raster data cube.

- **Data type**: raster-cube (object)

mask (required) Either a raster data cube, a GeoJSON object or a vector data cube.

For **raster data cubes**, both data cubes must be compatible so that every pixel in `data` has a parallel element in `mask`.

For **GeoJSON** the provided types can be one of the following:

- A `Polygon` geometry,
- a `GeometryCollection` containing Polygons,
- a `Feature` with a `Polygon` geometry or
- a `FeatureCollection` containing `Features` with a `Polygon` geometry.
- **Data types**:
  - **geojson (object)**
  - **vector-cube (object)**
  - **raster-cube (object)**

[71] https://open-eo.github.io/openeo-api/v/0.4.0/processes/#openeo-specific-formats

replacement The value used to replace non-zero and `true` values with.

- **Data type**: number / boolean / string / null
- **Default value**: *null*

Return Value

The masked raster data cube.

- **Data type**: raster-cube (object)

See Also

- Simple Features standard by the OGC[72]

3.58 max: Maximum value

Computes the largest value of an array of numbers, which is is equal to the first element of a sorted (i.e., ordered) version the array.

Parameters

data (required) An array of numbers. An empty array resolves always with `null`.

- **Data type**: array<number|null>

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if any value is such a value.

- **Data type**: boolean
- **Default value**: *true*

Return Value

The maximum value.

- **Data type**: number / null

[72] http://www.opengeospatial.org/standards/sfa

Examples

1. `max(data = [1,0,3,2]) => 3`
2. `max(data = [5,2.5,null,-0.7]) => 5`
3. `max(data = [1,0,3,null,2], ignore_nodata = false) => null`
4. `max(data = []) => null`

See Also

- Maximum explained by Wolfram MathWorld[73]

3.59 mean: Arithmetic mean (average)

The arithmetic mean of an array of numbers is the quantity commonly called the average. It is defined as the sum of all elements divided by the number of elements.

Parameters

data (required) An array of numbers. An empty array resolves always with `null`.

- **Data type**: array<number|null>

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if any value is such a value.

- **Data type**: boolean
- **Default value**: *true*

Return Value

The computed arithmetic mean.

- **Data type**: number / null

Examples

1. `mean(data = [1,0,3,2]) => 1.5`
2. `mean(data = [9,2.5,null,-2.5]) => 3`

[73] http://mathworld.wolfram.com/Maximum.html

3. `mean(data = [1,null], ignore_nodata = false) => null`
4. `mean(data = []) => null`

See Also

- Arithmetic mean explained by Wolfram MathWorld[74]

3.60 median: Statistical median

The statistical median of an array of numbers is the value separating the higher half from the lower half of the data.

Remarks:

- For a symmetric arrays, the result is equal to the `mean`.
- The median can also be calculated by computing the quantile (see process `quantiles`) with the probability of *0.5*: `quantiles(data, [0.5])`.
- An empty input array returns `null`.

Parameters

data (required) An array of numbers. An empty array resolves always with `null`.

- **Data type**: array<number|null>

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if any value is such a value.

- **Data type**: boolean
- **Default value**: *true*

Return Value

The computed statistical median.

- **Data type**: number / null

Examples

1. `median(data = [1,3,3,6,7,8,9]) => 6`

[74] http://mathworld.wolfram.com/ArithmeticMean.html

2. `median(data = [1,2,3,4,5,6,8,9]) => 4.5`
3. `median(data = [-1,-0.5,null,1]) => -0.5`
4. `median(data = [-1,0,null,1], ignore_nodata = false) => null`
5. `median(data = []) => null`

See Also

- Statistical Median explained by Wolfram MathWorld[75]

3.61 merge_cubes: Merging two data cubes

The data cubes have to be compatible. A merge is the inverse of a split if there is no overlap. If data overlaps the parameter `overlap_resolver` must be specified to resolve the overlap. It doesn't add dimensions.

Parameters

cube1 (required) The first data cube.

- **Data type**: raster-cube (object)

cube2 (required) The second data cube.

- **Data type**: raster-cube (object)

overlap_resolver A reducer that resolves the conflict if the data overlaps. The reducer must be a callable process (or a set of processes as process graph) such as `mean` that accepts by default array as input. The process can also work on two values by setting the parameter `binary` to `true`. The reducer must return a value of the same data type as the input values in the array. `null` (default) can be specified if no overlap resolver is required.

- **Data types**:
  - **callback (object)**: Passes two values to the reducer.
    - **Callback parameters**:
      - **x**: The first value. Any data type could be passed.
      - **y**: The second value. Any data type could be passed.
  - **null**

[75] http://mathworld.wolfram.com/StatisticalMedian.html

binary Specifies whether the process should pass two values to the reducer or a list of values (default).

If the process passes two values, the reducer must be both associative and commutative as the execution may be executed in parallel and therefore the order of execution is arbitrary.

This parameter is especially useful for UDFs passed as reducers. Back-ends may still optimize and parallelize processes that work on list of values.

- **Data type**: boolean
- **Default value**: *false*

Return Value

The merged data cube.

- **Data type**: raster-cube (object)

Exceptions

- `OverlapResolverMissing`: Two data cubes with overlap but without an overlap resolver have been specified.

See Also

- Background information on reduction operators (binary reducers) by Wikipedia[76]

3.62 min: Minimum value

Computes the smallest value of an array of numbers, which is is equal to the last element of a sorted (i.e., ordered) version the array.

Parameters

data (required) An array of numbers. An empty array resolves always with `null`.

- **Data type**: array<number|null>

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if any value is such a value.

[76] https://en.wikipedia.org/wiki/Reduction_Operator

- **Data type**: boolean
- **Default value**: *true*

Return Value

The minimum value.

- **Data type**: number / null

Examples

1. `min(data = [1,0,3,2]) => 0`
2. `min(data = [5,2.5,null,-0.7]) => -0.7`
3. `min(data = [1,0,3,null,2], ignore_nodata = false) => null`
4. `min(data = []) => null`

See Also

- Minimum explained by Wolfram MathWorld[77]

3.63 mod: Modulo

Remainder after division of `x` by `y`.

The result of a modulo operation has the sign of the divisor. The handling regarding the sign of the result differs between programming languages[78] and needs careful consideration while implementing this process.

The no-data value `null` is passed through and therefore gets propagated if any of the arguments is `null`.

Parameters

x (required) A number to be used as dividend.

- **Data type**: number / null

y (required) A number to be used as divisor.

- **Data type**: number / null

[77] http://mathworld.wolfram.com/Minimum.html
[78] https://en.wikipedia.org/wiki/Modulo_operation

Return Value

The remainder after division.

- **Data type**: number / null

Examples

1. `mod(x = 27, y = 5) => 2`
2. `mod(x = -27, y = 5) => 3`
3. `mod(x = 27, y = -5) => -3`
4. `mod(x = -27, y = -5) => -2`
5. `mod(x = 27, y = null) => null`
6. `mod(x = null, y = 5) => null`

See Also

- Modulo explained by Wikipedia[79]

3.64 multiply: Multiplication of a sequence of numbers

Multiplies all elements in a sequential array of numbers and returns the computed product.

The computations should follow IEEE Standard 754[80] whenever the processing environment supports it. Otherwise an exception must the thrown for incomputable results.

By default no-data values are ignored. Setting `ignore_nodata` to `false` considers no-data values so that `null` is returned if any element is such a value.

Parameters

data (required) An array of numbers with at least two elements.

- **Data type**: array<number|null>
- **Min. number of items**: 2

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if any value is such a value.

[79] https://en.wikipedia.org/wiki/Modulo_operation
[80] https://ieeexplore.ieee.org/document/4610935

- **Data type**: boolean
- **Default value**: *true*

Return Value

The computed product of the sequence of numbers.

- **Data type**: number / null

Exceptions

- `MultiplicandMissing`: Multiplication requires at least two numbers.

Examples

1. `multiply(data = [5,0]) => 0`
2. `multiply(data = [-2,4,2.5]) => -20`
3. `multiply(data = [1,null], ignore_nodata = false) => null`

See Also

- Product explained by Wolfram MathWorld[81]
- IEEE Standard 754-2008 for Floating-Point Arithmetic[82]

3.65 ndvi: Normalized Difference Vegetation Index

Computes the Normalized Difference Vegetation Index (NDVI). The NDVI is computed as *(nir - red) / (nir + red)*.

The `data` parameter expects a raster data cube with two bands that have the common names `red` and `nir` assigned. The process returns a raster data cube with two bands being replaced with a new band that holds the computed values. The newly created band is named `ndvi` by default. This name can be changed with the `name` parameter.

This process is very similar to the process `normalized_difference`, but determines the bands automatically based on the common name (`red`/`nir`) specified in the metadata.

[81] http://mathworld.wolfram.com/Product.html
[82] https://ieeexplore.ieee.org/document/4610935

Parameters

data (required) A raster data cube with two bands that have the common names `red` and `nir` assigned.

- **Data type**: raster-cube (object)

name Name of the newly created band with the computed values. Defaults to `normalized_difference`.

- **Data type**: string
- **Default value**: normalized_difference
- **Pattern**: `^[A-Za-z0-9_]+$`

Return Value

A raster data cube with the two bands being replaced with a new band that holds the computed values.

- **Data type**: raster-cube (object)

See Also

- NDVI explained by Wikipedia[83]
- NDVI explained by NASA[84]

3.66 neq: Not equal to comparison

Compares whether `x` is *not* strictly equal to `y`. This process is an alias for: `not(eq(val1, val2))`.

Remarks:

- Data types MUST be checked strictly, for example a string with the content *1* is not equal to the number *1*. Nevertheless, an integer *1* is equal to a floating point number *1.0* as `integer` is a sub-type of `number`.
- If any of the operands is `null`, the return value is `null`.
- Strings are expected to be encoded in UTF-8 by default.
- Temporal strings MUST be compared differently than other strings and MUST NOT be compared based on their string representation due to different possible representations. For example, the UTC time zone representation `Z` has the same meaning as `+00:00`.

[83] https://en.wikipedia.org/wiki/Normalized_difference_vegetation_index
[84] https://earthobservatory.nasa.gov/features/MeasuringVegetation/measuring_vegetation_2.php

Parameters

x (required) First operand.

- **Data types**:
  - **number**
  - **boolean**
  - **null**
  - **string**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

y (required) Second operand.

- **Data types**:
  - **number**
  - **boolean**
  - **null**
  - **string**
  - **date-time (string)**
  - **date (string)**
  - **time (string)**

delta Only applicable for comparing two numbers. If this optional parameter is set to a positive non-zero number the non-equality of two numbers is checked against a delta value. This is especially useful to circumvent problems with floating point inaccuracy in machine-based computation.

This option is basically an alias for the following computation: `gt(abs(minus([x, y]), delta)`

- **Data type**: number / null
- **Default value**: *null*

case_sensitive Only applicable for comparing two strings. Case sensitive comparison can be disabled by setting this parameter to `false`.

- **Data type**: boolean
- **Default value**: *true*

Return Value

Returns `true` if `x` is *not* equal to `y`, `null` if any of the operands is `null`, otherwise `false`.

- **Data type**: boolean / null

Examples

1. `neq(x = 1, y = null) => null`
2. `neq(x = 1, y = 1) => false`
3. `neq(x = 1, y = "1") => true`
4. `neq(x = 1.02, y = 1, delta = 0.01) => true`
5. `neq(x = -1, y = -1.001, delta = 0.01) => false`
6. `neq(x = 115, y = 110, delta = 10) => false`
7. `neq(x = "Test", y = "test") => true`
8. `neq(x = "Test", y = "test", case_sensitive = false) => false`
9. `neq(x = "Ä", y = "ä", case_sensitive = false) => false`
10. `neq(x = "00:00:00+00:00", y = "00:00:00Z") => false`
11. `neq(x = "2018-01-01T12:00:00Z", y = "2018-01-01T12:00:00") => true`
12. `neq(x = "2018-01-01T00:00:00Z", y = "2018-01-01T01:00:00+01:00") => false`

See Also

- Information about the supported temporal formats.[85]

3.67 normalized_difference: Normalized difference for two bands

Computes the normalized difference for two bands. The normalized difference is computed as *(band1 - band2) / (band1 + band2)*.

Each of the parameters expects a raster data cube with exactly one band. The process returns a raster data cube with exactly one band that holds the computed values. The newly created band is named `normalized_difference` by default. This name can be changed with the `name` parameter.

This process could be used for a number of remote sensing indices such as:

- NDVI[86]

[85] https://open-eo.github.io/openeo-api/v/0.4.0/processes/#openeo-specific-formats
[86] https://eos.com/ndvi/

- NDWI[87]
- NDSI[88]

Please note that some back-ends may have native processes available for convenience such as the `ndvi`.

Parameters

band1 (required) A raster data cube with exactly one band to be used as first band.

- **Data type**: raster-cube (object)

band2 (required) A raster data cube with exactly one band to be used as second band.

- **Data type**: raster-cube (object)

name Name of the newly created band with the computed values. Defaults to `normalized_difference`.

- **Data type**: string
- **Default value**: normalized_difference
- **Pattern**: `^[A-Za-z0-9_]+$`

Return Value

A raster data cube with exactly one band that holds the computed values.

- **Data type**: raster-cube (object)

See Also

- NDVI explained by EOS[89]
- NDWI explained by EOS[90]
- NDSI explained by EOS[91]

3.68 not: Inverting a boolean

Inverts a single boolean so that `true` gets `false` and `false` gets `true`.

[87] https://eos.com/ndwi/
[88] https://eos.com/ndsi/
[89] https://eos.com/ndvi/
[90] https://eos.com/ndwi/
[91] https://eos.com/ndsi/

The no-data value `null` is passed through and therefore gets propagated.

Parameters

expression (required) Boolean value to invert.

- **Data type**: boolean / null

Return Value

Inverted boolean value.

- **Data type**: boolean / null

Examples

1. `not(expression = null) => null`
2. `not(expression = false) => true`
3. `not(expression = true) => false`

3.69 or: Is at least one value true?

Checks if **at least one** of the values is true. Evaluates each expression from the first to the last element and stops once the outcome is unambiguous.

If only one value is given the process evaluates to the given value. If no value is given (i.e. the array is empty) the process returns `null`.

By default all no-data values are ignored so that the process returns `true` if at least one of the other values is true and otherwise returns `false`. Setting the `ignore_nodata` flag to `false` considers no-data values so that `null` is a valid logical object. If a component is `null`, the result will be `null` if the outcome is ambiguous. See the following truth table:

	null	false	true
null	null	null	true
false	null	false	true
true	true	true	true

Parameters

expressions (required) A set of boolean values.

- **Data type**: array<boolean|null>

ignore_nodata Indicates whether no-data values are ignored or not and ignores them by default.

- **Data type**: boolean
- **Default value**: *true*

Return Value

Boolean result of the logical expressions.

- **Data type**: boolean / null

Examples

1. `or(expressions = [false,null]) => false`
2. `or(expressions = [true,null]) => true`
3. `or(expressions = [false,null], ignore_nodata = false) => null`
4. `or(expressions = [true,null], ignore_nodata = false) => true`
5. `or(expressions = [true,false,true,false]) => true`
6. `or(expressions = [true,false]) => true`
7. `or(expressions = [false,false]) => false`
8. `or(expressions = [true]) => true`
9. `or(expressions = []) => null`

3.70 order: Create a permutation

Computes a permutation which allows rearranging the data into ascending or descending order. In other words, this process computes the ranked (sorted) element positions in the original list.

Remarks:

- The positions in the result are zero-based.
- Ties will be left in their original ordering.
- Temporal strings can *not* be compared based on their string representation due to the time zone / time-offset representations.

Parameters

data (required) An array to compute the order for.

- **Data type**: array

- **Array items**:
  - **Data types**:
    - **number**
    - **null**
    - **date-time (string)**
    - **date (string)**
    - **time (string)**

asc The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, set this parameter to `false`.

- **Data type**: boolean
- **Default value**: *true*

nodata Controls the handling of no-data values (`null`). By default they are removed. If `true`, missing values in the data are put last; if `false`, they are put first.

- **Data type**: boolean / null
- **Default value**: *null*

Return Value

The computed permutation.

- **Data type**: array<integer>
- **Array items**:
  - **Data type**: integer
  - **Minimum value**: 0

Examples

1. `order(data = [6,-1,2,null,7,4,null,8,3,9,9]) => [1,2,8,5,0,4,7,9,10]`
2. `order(data = [6,-1,2,null,7,4,null,8,3,9,9], nodata = true) => [1,2,8,5,0,4,7,9,10,3,6]`
3. `order(data = [6,-1,2,null,7,4,null,8,3,9,9], asc = false, nodata = true) => [9,10,7,4,0,5,8,2,1,3,6]`
4. `order(data = [6,-1,2,null,7,4,null,8,3,9,9], asc = false, nodata = false) => [3,6,9,10,7,4,0,5,8,2,1]`

See Also

- Information about the supported temporal formats.[92]
- Permutation explained by Wolfram MathWorld[93]

3.71 output: Send data to subscribed clients

Outputs the data to clients, which are subscribed to the topic `openeo.jobs.output`.

Parameters

data (required) Data to send.

id An identifier to help identify the message in a bunch of other messages.

- **Data type**: string

Return Value

`false` if the information could not be sent, `true` otherwise.

- **Data type**: boolean

See Also

- Information about the openEO API for Subscriptions[94]

3.72 pi: Pi (π)

The real number Pi (π) is a mathematical constant that is the ratio of the circumference of a circle to its diameter. The numerical value is approximately *3.14159*.

Parameters

Return Value

The numerical value of Pi.

- **Data type**: number

[92] https://open-eo.github.io/openeo-api/v/0.4.0/processes/#openeo-specific-formats
[93] http://mathworld.wolfram.com/Permutation.html
[94] https://open-eo.github.io/openeo-api/v/0.4.0/apireference-subscriptions/#publish-openeojobsoutput

See Also

- Mathematical constant Pi explained by Wolfram MathWorld[95]

3.73 power: Exponentiation

Computes the exponentiation for the base `base` raised to the power of `p`.

The no-data value `null` is passed through and therefore gets propagated if any of the arguments is `null`.

Parameters

base (required) The numerical base.

- **Data type**: number / null

p (required) The numerical exponent.

- **Data type**: number / null

Return Value

The computed value for `base` raised to the power of `p`.

- **Data type**: number / null

Examples

1. `power(base = 0, p = 2) => 0`
2. `power(base = 2.5, p = 0) => 1`
3. `power(base = 3, p = 3) => 27`
4. `power(base = 5, p = -1) => 0.2`
5. `power(base = 1, p = 0.5) => 1`
6. `power(base = 1, p = null) => null`
7. `power(base = null, p = 2) => null`

[95] http://mathworld.wolfram.com/Pi.html

See Also

- Power explained by Wolfram MathWorld[96]

3.74 product: Multiplication of a sequence of numbers

This process is an exact alias for the `multiply` process. See `multiply` for more information.

Parameters

data (required) See `multiply` for more information.

- **Data type**: array<number|null>

ignore_nodata See `multiply` for more information.

- **Data type**: boolean
- **Default value**: *true*

Return Value

See `multiply` for more information.

- **Data type**: number / null

3.75 property: Get metadata for data cubes or collections

Retrieves metadata properties for a data cube (if a `raster-cube` or `vector-cube` is passed) or a collection (if a collection id is passed). Properties for a single dimension can be queried with the `dimension` parameter.

Available metadata properties can be retrieved with the openEO Data Discovery for Collections, but for data cubes only a subset may be available after processing.

Parameters

data (required) A data cube.

- **Data types**:
  - **raster-cube (object)**
  - **vector-cube (object)**

[96] http://mathworld.wolfram.com/Power.html

- **collection-id (string)**: A collection id.
  - **Pattern**: `^[A-Za-z0-9_\-\.~/]+$`

name (required) Name of the metadata property.

- **Data type**: string

dimension Optionally, a dimension to get the property for. `null` by default, which gets a property globally for the collection/data cube.

- **Data type**: string / null
- **Default value**: *null*

Return Value

The value for the metadata property or `null` if no information are available.

3.76 quantiles: Quantiles

Calculates quantiles, which are cut points dividing the range of a probability distribution into either

- intervals corresponding to the given `probabilities` or
- (nearly) equal-sized intervals (q-quantiles based on the parameter `q`).

Either the parameter `probabilites` or `q` must be specified, otherwise the `QuantilesParameterMissing` exception must be thrown. If both parameters are set the `QuantilesParameterConflict` exception must be thrown.

Parameters

data (required) An array of numbers.

- **Data type**: array<number|null>

probabilities A list of probabilities to calculate quantiles for. The probabilities must be between 0 and 1.

- **Data type**: array<number>
- **Array items**:
  - **Data type**: number

- **Minimum value**: 0
- **Maximum value**: 1

q A number of intervals to calculate quantiles for. Calculates q-quantiles with (nearly) equal-sized intervals.

- **Data type**: integer
- **Minimum value**: 2

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that an array with `null` values is returned if any element is such a value.

- **Data type**: boolean
- **Default value**: *true*

Return Value

An array with the computed quantiles. The list has either

- as many elements as the given list of `probabilities` had or
- *q-1* elements.

If the input array is empty the resulting array is filled with as many `null` values as required according to the list above. For an example, see the 'Empty array example'.

- **Data type**: array<number|null>

Exceptions

- `QuantilesParameterMissing`: The process 'quantiles' requires either the 'probabilities' or 'q' parameter to be set.
- `QuantilesParameterConflict`: The process 'quantiles' only allows that either the 'probabilities' or the 'q' parameter is set.

Examples

1. `quantiles(data = [2,4,4,4,5,5,7,9], probabilities = [0.005,0.01,0.02,0.05,0.1,0.5]) => [2.07,2.14,2.28,2.7,3.4,4.5]`
2. `quantiles(data = [2,4,4,4,5,5,7,9], q = 4) => [4,4.5,5.5]`
3. `quantiles(data = [-1,-0.5,null,1], q = 2) => [-0.5]`

4. `quantiles(data = [-1,-0.5,null,1], q = 4, ignore_nodata = false) => [null,null,null]`
5. `quantiles(data = [], probabilities = [0.1,0.5]) => [null,null]`

See Also

- Quantiles explained by Wikipedia[97]

3.77 rearrange: Rearranges an array based on a permutation

Rearranges an array based on a permutation.

Parameters

data (required) An array to rearrange.

- **Data type**: array
- **Array items**: Any data type is allowed.

order (required) A permutation used for rearranging, i.e. a ranked list of element positions in the original list. The positions must be zero-based.

- **Data type**: array<integer>
- **Array items**:
  - **Data type**: integer
  - **Minimum value**: 0

Return Value

The rearranged array.

- **Data type**: array
- **Array items**: Any data type is allowed.

Examples

1. `rearrange(data = [5,4,3], order = [2,1,0]) => [3,4,5]`
2. `rearrange(data = [5,4,3,2], order = [1,3]) => [4,2]`
3. `rearrange(data = [5,4,3,2], order = [0,2,1,3]) => [5,3,4,2]`

[97] https://en.wikipedia.org/wiki/Quantile

See Also

- Permutation explained by Wolfram MathWorld[98]

3.78 reduce: Reduce dimensions

Applies a reducer to a data cube dimension by collapsing all the input values along the specified dimension into an output value computed by the reducer.

The reducer must be a callable process (or a set of processes as process graph) that accepts by default array as input. The process can also work on two values by setting the parameter `binary` to `true`. The reducer must compute a single or multiple return values of the same type as the input values were. Multiple values must be wrapped in an array. An example for a process returning a single value is `median`. In this case the specified dimension would be removed. If a callback such as `extrema` returns multiple values, a new dimension with the specified name in `target_dimension` is created (see the description of the parameter for more information).

A special case is that the reducer can be set to `null`, which is the default if no reducer is specified. It acts as a no-operation reducer so that the remaining value is treated like a reduction result and the dimension gets dropped. This only works on dimensions with a single dimension value left (e.g. after filtering for a single band), otherwise the process fails with a `TooManyDimensionValues` error.

Nominal values can be reduced too, but need to be mapped. For example date strings to numeric timestamps since 1970 etc.

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

reducer A reducer to be applied on the specified dimension (see the process description for more details).

- **Data types**:
  - **callback (object)**: Passes two values to the reducer.
    - **Callback parameters**:
      - **x**: The first value. Any data type could be passed.
      - **y**: The second value. Any data type could be passed.

[98] http://mathworld.wolfram.com/Permutation.html

- **null**: Specifying ‘null‘ works only on dimensions with a single dimension value left. In this case the remaining value is treated like a reduction result and the dimension gets dropped.

dimension (required) The dimension over which to reduce.

Remarks:

- The default dimensions a data cube provides are described in the collection’s metadata field `cube:dimensions`.
- There could be multiple spatial dimensions such as `x`, `y` or `z`.
- For multi-spectral imagery there is usually a separate dimension of type `bands` for the bands.
- **Data type**: string

target_dimension The name of the target dimension. Only required if the reducer returns multiple values, otherwise ignored. By default creates a new dimension with the specified name and the type `other` (see `add_dimension`). If a dimension with the specified name exists, the dimension is replaced, but keeps the original type.

- **Data type**: string / null
- **Default value**: *null*

binary Specifies whether the process should pass two values to the reducer or a list of values (default).

If the process passes two values, the reducer must be both associative and commutative as the execution may be executed in parallel and therefore the order of execution is arbitrary.

This parameter is especially useful for UDFs passed as reducers. Back-ends may still optimize and parallelize processes that work on list of values.

This parameter can’t be used with the reducer set to `null`. If a reducer is specified but only a single value is available, the reducer doesn’t get executed.

- **Data type**: boolean
- **Default value**: *false*

Return Value

A data cube with the newly computed values. The number of dimensions is reduced for callbacks returning a single value or doesn’t change if the callback returns multiple values. The resolution and cardinality are the same as for the original data cube.

- **Data type**: raster-cube (object)

Exceptions

- `TooManyDimensionValues`: The number of dimension values exceeds one, which requires a reducer.

See Also

- Background information on reduction operators (binary reducers) by Wikipedia[99]

3.79 rename_dimension: Renames a dimension

Renames a dimension in the data cube.

Afterwards, the dimension can be referenced with the new name. If a dimension with the specified name already exists, a `DimensionExists` exception is thrown.

Parameters

data (required) The data cube.

- **Data types**:
  - **raster-cube (object)**
  - **vector-cube (object)**

old (required) The current name of the dimension.

- **Data type**: string

new (required) A new Name for the dimension.

- **Data type**: string

Return Value

The data cube with the renamed dimension. The old name can not be referenced any longer.

- **Data types**:
  - **raster-cube (object)**
  - **vector-cube (object)**

[99] https://en.wikipedia.org/wiki/Reduction_Operator

Exceptions

- `DimensionExists`: A dimension with the specified name already exists.

3.80 resample_cube_spatial: Resample the spatial dimensions to a target data cube

Resamples the spatial dimensions (x,y) from a source data cube to a target data cube and return the results as a new data cube.

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

target (required) A data cube that describes the spatial target resolution.

- **Data type**: raster-cube (object)

method Resampling method. Methods are inspired by GDAL, see gdalwarp[100] for more information.

- **Data type**: string
- **Default value**: near
- **Allowed values**:
  1. near
  2. bilinear
  3. cubic
  4. cubicspline
  5. lanczos
  6. average
  7. mode
  8. max
  9. min
  10. med
  11. q1
  12. q3

[100] https://www.gdal.org/gdalwarp.html

Return Value

A data cube with potentially lower spatial resolution and potentially lower cardinality, but the same number of dimensions as the original data cube.

- **Data type**: raster-cube (object)

See Also

- Resampling explained in the openEO glossary[101]

3.81 resample_cube_temporal: Resample a temporal dimension to a target data cube

Resamples a temporal dimension from a source data cube to a target data cube and return the results as a new data cube.

If the dimension is not set or is set to `null`, the data cube is expected to only have one temporal dimension.

Parameters

data (required) A data cube.

- **Data type**: raster-cube (object)

target (required) A data cube that describes the temporal target resolution.

- **Data type**: raster-cube (object)

method (required) A resampling method to be applied, could be a reducer for downsampling or other methods for upsampling. The reducer must be a callable process (or a set of processes as process graph) such as `mean` that accepts by default array as input. The process can also work on two values by setting the parameter `binary` to `true`.

- **Data types**:
  - **callback (object)**: Passes two values to the reducer.
    - **Callback parameters**:
      - **x**: The first value. Any data type could be passed.
      - **y**: The second value. Any data type could be passed.

[101] https://open-eo.github.io/openeo-api/v/0.4.0/glossary/#aggregation-and-resampling

dimension The temporal dimension to resample, which must exist with this name in both data cubes. If the dimension is not set or is set to `null`, the data cube is expected to only have one temporal dimension.

Note: The default dimensions a data cube provides are described in the collection's metadata field `cube:dimensions`.

- **Data type**: string / null
- **Default value**: *null*

binary Specifies whether the process should pass two values to the reducer specified as resampling method or a list of values (default).

If the process passes two values, the reducer must be both associative and commutative as the execution may be executed in parallel and therefore the order of execution is arbitrary.

This parameter is especially useful for UDFs passed as reducers. Back-ends may still optimize and parallelize processes that work on list of values.

- **Data type**: boolean
- **Default value**: *false*

Return Value

A data cube with potentially lower temporal resolution and potentially lower cardinality, but the same number of dimensions as the original data cube.

- **Data type**: raster-cube (object)

See Also

- Information about the supported temporal formats.[102]
- Resampling explained in the openEO glossary[103]
- Background information on reduction operators (binary reducers) by Wikipedia[104]

3.82 resample_spatial: Resample and warp the spatial dimensions

Resamples the spatial dimensions (x,y) of the data cube to a specified resolution and/or warps the data cube to the target projection. At least `resolution` or `projection` must be specified.

Use `filter_bbox` to set the target spatial extent.

[102] https://open-eo.github.io/openeo-api/v/0.4.0/processes/#openeo-specific-formats
[103] https://open-eo.github.io/openeo-api/v/0.4.0/glossary/#aggregation-and-resampling
[104] https://en.wikipedia.org/wiki/Reduction_Operator

Parameters

data (required) A raster data cube.

- **Data type**: raster-cube (object)

resolution Resamples the data cube to the target resolution, which can be specified either as separate values for x and y or as a single value for both axes. Specified in the units of the target projection. Doesn't change the resolution by default (`0`).

- **Data types**:
  - **number**: A single number used as resolution for both x and y.
    - **Minimum value**: 0
  - **array<number>**: A two-element array to specify separate resolutions for x (first element) and y (second element).
    - **Min. number of items**: 2
    - **Max. number of items**: 2
    - **Array items**:
      - **Data type**: number
      - **Minimum value**: 0

projection Warps the data cube to the target projection. Target projection specified as EPSG[105] code or PROJ[106] definition. Doesn't change the projection by default (`null`).

- **Data types**:
  - **epsg-code (integer)**:
    - **Examples**:
      1. `7099`
  - **proj-definition (string)**:
    - **Examples**:
      1. `"+proj=moll +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs"`
  - **null**

method Resampling method. Methods are inspired by GDAL, see gdalwarp[107] for more information.

[105] http://www.epsg.org
[106] https://proj4.org
[107] https://www.gdal.org/gdalwarp.html

- **Data type**: string
- **Default value**: near
- **Allowed values**:
  1. near
  2. bilinear
  3. cubic
  4. cubicspline
  5. lanczos
  6. average
  7. mode
  8. max
  9. min
  10. med
  11. q1
  12. q3

align Specifies to which corner of the spatial extent the new resampled data is aligned to.

- **Data type**: string
- **Allowed values**:
  1. lower-left
  2. upper-left
  3. lower-right
  4. upper-right
- **Default value**: lower-left

Return Value

A raster data cube with values warped onto the new projection.

- **Data type**: raster-cube (object)

See Also

- PROJ parameters for cartographic projections[108]

[108] https://proj4.org/usage/projections.html

- Official EPSG code registry[109]
- Unofficial EPSG code database[110]

3.83 round: Rounds to a specified precision

Rounds a real number `x` to specified precision `p`.

If the fractional part of `x` is halfway between two integers, one of which is even and the other odd, then the even number is returned. This behaviour follows IEEE Standard 754[111]. This kind of rounding is also called “rounding to nearest” or “banker’s rounding”. It minimizes rounding errors that result from consistently rounding a midpoint value in a single direction.

The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number to round.

- **Data type**: number / null

p A positive number specifies the number of digits after the decimal point to round to. A negative number means rounding to a power of ten, so for example *-2* rounds to the nearest hundred. Defaults to *0*.

- **Data type**: integer
- **Default value**: 0

Return Value

The rounded number.

- **Data type**: number / null

Examples

1. `round(x = 0) => 0`
2. `round(x = 3.56, p = 1) => 3.6`
3. `round(x = -0.4444444, p = 2) => -0.44`
4. `round(x = -2.5) => -2`

[109] http://www.epsg-registry.org
[110] http://www.epsg.io
[111] https://ieeexplore.ieee.org/document/4610935

5. `round(x = -3.5) => -4`
6. `round(x = 1234.5, p = -2) => 1200`

See Also

- Absolute value explained by Wolfram MathWorld[112]
- IEEE Standard 754-2008 for Floating-Point Arithmetic[113]

3.84 run_process_graph: Load and run a stored process graph

Loads and executes a stored process graph.

The process can either load

- a locally stored process graph by id, which is stored by the authenticated user on the back-end currently connected to or
- a remotely stored and published process graph by absolute URI, for example from openEO Hub[114].

Parameters

id (required) A process graph id or an absolute URI to an externally hosted process graph.

- **Data types**:
  - **uri (string)**: URI
  - **process-graph-id (string)**: Process graph id
    - **Pattern**: `^[A-Za-z0-9_\-\.~]+$`

variables Key-value-pairs with values for variables that are defined by the process graph. The key of the pair has to be the corresponding `variable_id` for the value specified. The replacement for the variable is the value of the pair.

- **Data type**: process-graph-variables (object)
- **Default value**: `{}` (Empty object)

Return Value

The result of processing the process graph.

[112] http://mathworld.wolfram.com/AbsoluteValue.html
[113] https://ieeexplore.ieee.org/document/4610935
[114] https://hub.openeo.org

See Also

- More information about the experimental status of the process[115]

3.85 run_udf_externally: Run an externally hosted UDF container

Runs a compatible UDF container that is either externally hosted by a service provider or running on a local machine of the user. The UDF container must follow the openEO UDF specification[116].

The referenced UDF service can be executed as callback in several processes such as `aggregate_temporal`, `apply`, `apply_dimension`, `filter` and `reduce`. In this case an array is passed instead of a raster data cube. The user must ensure that the data is properly passed as an array so that the UDF can make sense of it.

Parameters

data (required) The data to be passed to the UDF as array or raster data cube.

- **Data types**:
  - **raster-cube (object)**
  - **array**:
    - **Min. number of items**: 1
    - **Array items**: Any data type.

url (required) URL to a remote UDF service.

- **Data type**: uri (string)

context Additional data such as configuration options that should be passed to the UDF.

- **Data type**: object
- **Default value**: `{}` (Empty object)

Return Value

The data processed by the UDF service. Returns a raster data cube if a raster data cube was passed for `data`. If an array was passed for `data`, the returned value is defined by the context and is exactly what the UDF returned.

[115] https://github.com/Open-EO/openeo-processes/issues/5
[116] https://open-eo.github.io/openeo-udf/

- **Data types**:
  - **raster-cube (object)**
  - **any**: Any data type.

See Also

- openEO UDF specification[117]
- openEO UDF repository[118]

3.86 run_udf: Run an UDF

Runs an UDF in one of the supported runtime environments.

The process can either:

1. load and run a locally stored UDF from a file in the workspace of the authenticated user. The path to the UDF file must be relative to the root directory of the user's workspace, so without the user id in the path.
2. fetch and run a remotely stored and published UDF by absolute URI, for example from openEO Hub[119]).
3. run the source code specified inline as string.

The loaded UDF can be executed as callback in several processes such as `aggregate_temporal`, `apply`, `apply_dimension`, `filter` and `reduce`. In this case an array is passed instead of a raster data cube. The user must ensure that the data is properly passed as an array so that the UDF can make sense of it.

Parameters

data (required) The data to be passed to the UDF as array or raster data cube.

- **Data types**:
  - **raster-cube (object)**
  - **array**:
    - **Min. number of items**: 1
    - **Array items**: Any data type.

udf (required) Either source code, an absolute URL or a path to an UDF script.

[117] https://open-eo.github.io/openeo-udf/
[118] https://github.com/Open-EO/openeo-udf
[119] https://hub.openeo.org

- **Data types**:
  - **uri (string)**: URI to an UDF
  - **string**: Source code as string

runtime (required) An UDF runtime identifier available at the back-end.

- **Data type**: string

version An UDF runtime version. If set to `null`, the default runtime version specified for each runtime is used.

- **Data type**: string / null
- **Default value**: *null*

context Additional data such as configuration options that should be passed to the UDF.

- **Data type**: object
- **Default value**: `{}` (Empty object)

Return Value

The data processed by the UDF. Returns a raster data cube if a raster data cube was passed for `data`. If an array was passed for `data`, the returned value is defined by the context and is exactly what the UDF returned.

- **Data types**:
  - **raster-cube (object)**
  - **any**: Any data type.

Exceptions

- `InvalidVersion`: The specified UDF runtime version is not supported.

3.87 save_result: Save processed data to storage

Saves processed data to the local user workspace / data store of the authenticated user. This process aims to be compatible to GDAL/OGR formats and options. STAC-compatible metadata should be stored with the processed data.

Calling this process may be rejected by back-ends in the context of secondary web services.

Parameters

data (required) The data to save.

- **Data types**:
  - **raster-cube (object)**
  - **vector-cube (object)**

format (required) The file format to save to. It must be one of the values that the server reports as supported output formats, which usually correspond to the short GDAL/OGR codes. This parameter is *case insensitive*.

- **Data type**: output-format (string)

options The file format options to be used to create the file(s). Must correspond to the options that the server reports as supported options for the chosen `format`. The option names and valid values usually correspond to the GDAL/OGR format options.

- **Data type**: output-format-options (object)
- **Default value**: `{}` (Empty object)

Return Value

`false` if saving failed, `true` otherwise.

- **Data type**: boolean

See Also

- GDAL Raster Formats[120]
- OGR Vector Formats[121]

3.88 sd: Standard deviation

Computes the sample standard deviation, which quantifies the amount of variation of an array of numbers. It is defined to be the square root of the corresponding variance (see `variance`).

A low standard deviation indicates that the values tend to be close to the expected value, while a high standard deviation indicates that the values are spread out over a wider range.

[120] https://www.gdal.org/formats_list.html
[121] https://www.gdal.org/ogr_formats.html

Parameters

data (required) An array of numbers. An empty array resolves always with `null`.

- **Data type**: array<number|null>

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if any value is such a value.

- **Data type**: boolean
- **Default value**: *true*

Return Value

The computed sample standard deviation.

- **Data type**: number / null

Examples

1. `sd(data = [-1,1,3,null]) => 2`
2. `sd(data = [-1,1,3,null], ignore_nodata = false) => null`
3. `sd(data = []) => null`

See Also

- Standard deviation explained by Wolfram MathWorld[122]

3.89 sgn: Signum

The signum (also known as *sign*) of `x` is defined as:

- *1* if *x > 0*
- *0* if *x = 0*
- *-1* if *x < 0*

The no-data value `null` is passed through and therefore gets propagated.

[122] http://mathworld.wolfram.com/StandardDeviation.html

Parameters

x (required) A number.

- **Data type**: number / null

Return Value

The computed signum value of `x`.

- **Data type**: number / null

Examples

1. `sgn(x = -2) => -1`
2. `sgn(x = 3.5) => 1`
3. `sgn(x = 0) => 0`
4. `sgn(x = null) => null`

See Also

- Sign explained by Wolfram MathWorld[123]

3.90 sin: Sine

Computes the sine of `x`.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) An angle in radians.

- **Data type**: number / null

Return Value

The computed sine of `x`.

- **Data type**: number / null

[123] http://mathworld.wolfram.com/Sign.html

Examples

1. `sin(x = 0) => 0`

See Also

- Sine explained by Wolfram MathWorld[124]

3.91 sinh: Hyperbolic sine

Computes the hyperbolic sine of `x`.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) An angle in radians.

- **Data type**: number / null

Return Value

The computed hyperbolic sine of `x`.

- **Data type**: number / null

Examples

1. `sinh(x = 0) => 0`

See Also

- Hyperbolic sine explained by Wolfram MathWorld[125]

3.92 sort: Sort data

Sorts an array into ascending (default) or descending order.

[124] http://mathworld.wolfram.com/Sine.html
[125] http://mathworld.wolfram.com/HyperbolicSine.html

This process is an alias to call `order` and `rearrange` consecutively: `rearrange(data, order(data, nodata))`. This process could be faster though. See `order` for more information on sorting behaviour.

Parameters

data (required) An array with data to sort.

- **Data type**: array
- **Array items**:
  - **Data types**:
    - **number**
    - **null**
    - **date-time (string)**
    - **date (string)**
    - **time (string)**

asc The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, set this parameter to `false`.

- **Data type**: boolean
- **Default value**: *true*

nodata Controls the handling of no-data values (`null`). By default they are removed. If `true`, missing values in the data are put last; if `false`, they are put first.

- **Data type**: boolean / null
- **Default value**: *null*

Return Value

The sorted array.

- **Data type**: array
- **Array items**:
  - **Data types**:
    - **number**
    - **null**
    - **date-time (string)**

- **date (string)**
- **time (string)**

Examples

1. `sort(data = [6,-1,2,null,7,4,null,8,3,9,9]) => [-1,2,3,4,6,7,8,9,9]`
2. `sort(data = [6,-1,2,null,7,4,null,8,3,9,9], asc = false, nodata = true) => [9,9,8,7,6,4`

3.93 sqrt: Square root

Computes the square root of a real number `x`. This process is an alias for `x` to the power of *0.5*: `power(x, 0.5)`.

A square root of x is a number a such that $a^2 = x$. Therefore, the square root is the inverse function of a to the power of 2, but only for $a >= 0$.

The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) A number.

- **Data type**: number / null

Return Value

The computed square root.

- **Data type**: number / null

Examples

1. `sqrt(x = 0) => 0`
2. `sqrt(x = 1) => 1`
3. `sqrt(x = 9) => 3`
4. `sqrt(x = null) => null`

See Also

- Square root explained by Wolfram MathWorld[126]

[126] http://mathworld.wolfram.com/SquareRoot.html

3.94 subtract: Subtraction of a sequence of numbers

Takes the first element of a sequential array of numbers and subtracts all other elements from it.

The computations should follow IEEE Standard 754[127] whenever the processing environment supports it. Otherwise an exception must the thrown for incomputable results.

By default no-data values are ignored. Setting `ignore_nodata` to `false` considers no-data values so that `null` is returned if any element is such a value.

Parameters

data (required) An array of numbers with at least two elements.

- **Data type**: array<number|null>
- **Min. number of items**: 2

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if any value is such a value.

- **Data type**: boolean
- **Default value**: *true*

Return Value

The computed result of the sequence of numbers.

- **Data type**: number / null

Exceptions

- `SubtrahendMissing`: Subtraction requires at least two numbers (a minuend and one or more subtrahends).

Examples

1. `subtract(data = [5,10]) => -5`
2. `subtract(data = [-2,4,-2]) => -4`
3. `subtract(data = [1,null], ignore_nodata = false) => null`

[127] https://ieeexplore.ieee.org/document/4610935

See Also

- Subtraction explained by Wolfram MathWorld[128]
- IEEE Standard 754-2008 for Floating-Point Arithmetic[129]

3.95 sum: Addition of a sequence of numbers

Sums up all elements in a sequential array of numbers and returns the computed sum.

The computations should follow IEEE Standard 754[130] whenever the processing environment supports it. Otherwise an exception must the thrown for incomputable results.

By default no-data values are ignored. Setting `ignore_nodata` to `false` considers no-data values so that `null` is returned if any element is such a value.

Parameters

data (required) An array of numbers with at least two elements.

- **Data type**: array<number|null>
- **Min. number of items**: 2

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if any value is such a value.

- **Data type**: boolean
- **Default value**: *true*

Return Value

The computed sum of the sequence of numbers.

- **Data type**: number / null

Exceptions

- `SummandMissing`: Addition requires at least two numbers.

[128] http://mathworld.wolfram.com/Subtraction.html
[129] https://ieeexplore.ieee.org/document/4610935
[130] https://ieeexplore.ieee.org/document/4610935

Examples

1. `sum(data = [5,1]) => 6`
2. `sum(data = [-2,4,2.5]) => 4.5`
3. `sum(data = [1,null], ignore_nodata = false) => null`

See Also

- Sum explained by Wolfram MathWorld[131]
- IEEE Standard 754-2008 for Floating-Point Arithmetic[132]

3.96 tan: Tangent

Computes the tangent of `x`. The tangent is defined to be the sine of x divided by the cosine of x.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) An angle in radians.

- **Data type**: number / null

Return Value

The computed tangent of `x`.

- **Data type**: number / null

Examples

1. `tan(x = 0) => 0`

See Also

- Tangent explained by Wolfram MathWorld[133]

[131] http://mathworld.wolfram.com/Sum.html
[132] https://ieeexplore.ieee.org/document/4610935
[133] http://mathworld.wolfram.com/Tangent.html

3.97 tanh: Hyperbolic tangent

Computes the hyperbolic tangent of `x`. The tangent is defined to be the hyperbolic sine of x divided by the hyperbolic cosine of x.

Works on radians only. The no-data value `null` is passed through and therefore gets propagated.

Parameters

x (required) An angle in radians.

- **Data type**: number / null

Return Value

The computed hyperbolic tangent of `x`.

- **Data type**: number / null

Examples

1. `tanh(x = 0) => 0`

See Also

- Hyperbolic tangent explained by Wolfram MathWorld[134]

3.98 text_begins: Text begins with another text

Checks whether the text (also known as *string*) specified for `data` contains the text specified for `pattern` at the very beginning. Both are expected to be encoded in UTF-8 by default. Regular expressions are not supported.

Parameters

data (required) Text in which to find something at the beginning.

- **Data type**: string

[134] http://mathworld.wolfram.com/HyperbolicTangent.html

pattern (required) Text to find at the beginning of `data`.

- **Data type**: string

case_sensitive Case sensitive comparison can be disabled by setting this parameter to `false`.

- **Data type**: boolean
- **Default value**: *true*

Return Value

`true` if `data` begins with `pattern`, false‘ otherwise.

- **Data type**: boolean

Examples

1. `text_begins(data = "Lorem ipsum dolor sit amet", pattern = "amet") => false`
2. `text_begins(data = "Lorem ipsum dolor sit amet", pattern = "Lorem") => true`
3. `text_begins(data = "Lorem ipsum dolor sit amet", pattern = "lorem") => false`
4. `text_begins(data = "Lorem ipsum dolor sit amet", pattern = "lorem", case_sensitive = false) => true`
5. `text_begins(data = "Ä", pattern = "ä", case_sensitive = false) => true`

3.99 text_contains: Text contains another text

Checks whether the text (also known as *string*) specified for `data` contains the text specified for `pattern`. Both are expected to be encoded in UTF-8 by default. Regular expressions are not supported.

Parameters

data (required) Text in which to find something in.

- **Data type**: string

pattern (required) Text to find in `data`.

- **Data type**: string

case_sensitive Case sensitive comparison can be disabled by setting this parameter to `false`.

- **Data type**: boolean
- **Default value**: *true*

Return Value

`true` if `data` contains the `pattern`, false‘ otherwise.

- **Data type**: boolean

Examples

1. `text_contains(data = "Lorem ipsum dolor sit amet", pattern = "openEO") => false`
2. `text_contains(data = "Lorem ipsum dolor sit amet", pattern = "ipsum dolor") => true`
3. `text_contains(data = "Lorem ipsum dolor sit amet", pattern = "Ipsum Dolor") => false`
4. `text_contains(data = "Lorem ipsum dolor sit amet", pattern = "SIT", case_sensitive = false) => true`
5. `text_contains(data = "ÄÖÜ", pattern = "ö", case_sensitive = false) => true`

3.100 text_ends: Text ends with another text

Checks whether the text (also known as *string*) specified for `data` contains the text specified for `pattern` at the very end. Both are expected to be encoded in UTF-8 by default. Regular expressions are not supported.

Parameters

data (required) Text in which to find something at the end.

- **Data type**: string

pattern (required) Text to find at the end of `data`.

- **Data type**: string

case_sensitive Case sensitive comparison can be disabled by setting this parameter to `false`.

- **Data type**: boolean
- **Default value**: *true*

Return Value

`true` if `data` ends with `pattern`, false‘ otherwise.

- **Data type**: boolean

Examples

1. `text_ends(data = "Lorem ipsum dolor sit amet", pattern = "amet") => true`
2. `text_ends(data = "Lorem ipsum dolor sit amet", pattern = "AMET") => false`
3. `text_ends(data = "Lorem ipsum dolor sit amet", pattern = "Lorem") => false`
4. `text_ends(data = "Lorem ipsum dolor sit amet", pattern = "AMET", case_sensitive = false) => true`
5. `text_ends(data = "Ä", pattern = "ä", case_sensitive = false) => true`

3.101 text_merge: Concatenate elements to a string

Merges string representations of a set of elements together to a single string, with the separator between each element.

Parameters

data (required) A set of elements. Numbers, boolean values and null values get converted to their (lower case) string representation. For example: `1` (integer), `-1.5` (number), `true` / `false` (boolean values)

- **Data type**: array<string|number|boolean|null>

separator A separator to put between each of the individual texts. Defaults to an empty string.

- **Data type**: string / number / boolean / null

Return Value

Returns a string containing a string representation of all the array elements in the same order, with the separator between each element.

- **Data type**: string

Examples

1. `text_merge(data = ["Hello","World"], separator = " ") => "Hello World"`
2. `text_merge(data = [1,2,3,4,5,6,7,8,9,0]) => "1234567890"`
3. `text_merge(data = [null,true,false,1,-1.5,""], separator = "\n") => "null\ntrue\nfalse\`
4. `text_merge(data = [2,0], separator = 1) => "210"`
5. `text_merge(data = []) => ""`

3.102 trim: Remove slices with no-data values

Removes slices solely containing no-data values. If the dimension is irregular categorical then slices in the middle can be removed.

Parameters

data (required) A raster data cube to trim.

- **Data type**: raster-cube (object)

Return Value

A trimmed raster data cube.

- **Data type**: raster-cube (object)

3.103 variance: Variance

Computes the sample variance of an array of numbers by calculating the square of the standard deviation (see `sd`). It is defined to be the expectation of the squared deviation of a random variable from its expected value. Basically, it measures how far the numbers in the array are spread out from their average value.

Parameters

data (required) An array of numbers. An empty array resolves always with `null`.

- **Data type**: array<number|null>

ignore_nodata Indicates whether no-data values are ignored or not. Ignores them by default. Setting this flag to `false` considers no-data values so that `null` is returned if any value is such a value.

- **Data type**: boolean
- **Default value**: *true*

Return Value

The computed sample variance.

- **Data type**: number / null

Examples

1. `variance(data = [-1,1,3]) => 4`
2. `variance(data = [2,3,3,null,4,4,5]) => 1.1`
3. `variance(data = [-1,1,null,3], ignore_nodata = false) => null`
4. `variance(data = []) => null`

See Also

- Variance explained by Wolfram MathWorld[135]

3.104 xor: Is exactly one value true?

Checks if **exactly one** of the values is true. Evaluates each expression from the first to the last element and stops once the outcome is unambiguous.

If only one value is given the process evaluates to the given value. If no value is given (i.e. the array is empty) the process returns `null`.

By default all no-data values are ignored so that the process returns `true` if exactly one of the other values is true and otherwise returns `false`. Setting the `ignore_nodata` flag to `false` considers no-data values so that `null` is a valid logical object. If a component is `null`, the result will be `null` if the outcome is ambiguous. See the following truth table:

	null	false	true
null	null	null	null
false	null	false	true
true	null	true	false

[135] http://mathworld.wolfram.com/Variance.html

Parameters

expressions (required) A set of boolean values.

- **Data type**: array<boolean|null>

ignore_nodata Indicates whether no-data values are ignored or not and ignores them by default.

- **Data type**: boolean
- **Default value**: *true*

Return Value

Boolean result of the logical expressions.

- **Data type**: boolean / null

Examples

1. `xor(expressions = [false,null]) => false`
2. `xor(expressions = [true,null]) => true`
3. `xor(expressions = [false,null], ignore_nodata = false) => null`
4. `xor(expressions = [true,null], ignore_nodata = false) => null`
5. `xor(expressions = [true,false,true,false]) => false`
6. `xor(expressions = [true,false]) => true`
7. `xor(expressions = [false,false]) => false`
8. `xor(expressions = [true]) => true`
9. `xor(expressions = []) => null`

4 Specification for datasets

For data discovery, openEO aimed for a lightweight and extensible approach based on JSON that is capable of describing data on the dataset[136] level. We reviewed existing standards for both dataset descriptions. Most of the standards use XML to encode the information and are often more focused on describing granules. None of the established standards fully suited our requirements. We found the STAC specification to be lightweight, but still extensible. The

[136] A *dataset* (called *collection* in the openEO specification) is an aggregation of granules sharing the same product specification. A *granule* typically refers to a limited area and a single overpass leading to a very short observation period (seconds) or a temporal aggregation of such data.

specification lead is *Chris Holmes* from the *Radiant Earth Foundation*, but in general it is an open-source specification with multiple organisations being involved. The specification had not matured to a stable specification yet and was also focused on granule[137] level metadata due to the early state of the project. This allowed the openEO project to join the STAC initiative to align with them and to push the specification towards also being useful for data processing platforms and not only data providers. In cooperation with the Google Earth Engine Team we introduced the STAC Collection specification. Since then, openEO also contributed several extensions to better describe several of our main datasets:

- Data Cube Extension[138]: Describe data Cube related metadata, especially their dimensions.
- Non-Common Properties Extension[139] (draft): Gives summaries of properties which are not common across collections in the collection metadata. This is very useful as backends doesn't necessarily provide information about individual granules and therefore users need to be able to request summaries of available metadata.
- SAR Extension[140]: Covers synthetic-aperture radar data..
- Scientific Extension[141]: Indicate from which publication data originates and how the data itself should be cited or referenced.

In addition, openEO integrated multiple other extensions into the openEO API:

- EO Extension[142]: Covers electro-optical data, which may consist of multiple spectral bands.
- STAC API[143]: A small API specification for data discovery, which is also closely bound to the evolving OGC WFS 3.0 specification[144] so that implementing STAC API leads to also implementing OGC WFS 3.0. Therefore, openEO implementations can also conform easily to STAC API and OGC WFS 3.0.

Therefore, the openEO API is able to describe all datasets that STAC currently supports, which includes Synthetic-Aperture Radar (SAR) datasets such as Sentinel-1, Multi-Spectral Instrument (MSI) datasets such as Sentinel-2 and Ocean and Land Colour Instrument (OLCI) datasets such as Sentinel-3. The fields described in the next paragraphs are ported over from the STAC specification to be explicitly available in the openEO API for dataset descriptions. We included the fields in a way that they are fully compatible to STAC, but may be restricted or forced to be always required. The field descriptions are ported over from STAC. For full definitions see the STAC specification and its extensions.

General metadata

[137] *Granules* are called *items* in STAC.
[138] https://github.com/radiantearth/stac-spec/blob/master/extensions/datacube/README.md
[139] https://github.com/radiantearth/stac-spec/pull/416
[140] https://github.com/radiantearth/stac-spec/blob/master/extensions/sar/README.md
[141] https://github.com/radiantearth/stac-spec/blob/master/extensions/scientific/README.md
[142] https://github.com/radiantearth/stac-spec/blob/master/extensions/eo/README.md
[143] https://github.com/radiantearth/stac-spec/blob/master/api-spec/README.md
[144] https://github.com/opengeospatial/WFS_FES

- `stac_version` (string, required): The STAC version the collection implements.
- `id` (string, required): Identifier for the collection that is unique across the back-end.
- `title` (string): A short descriptive one-line title for the collection.
- `description` (string, required): Detailed description to fully explain the collection.
- `keywords` (array<string>): List of keywords describing the collection.
- `version` (string): Version of the collection.
- `license` (string, required): Collection's license(s) as a SPDX License identifier[145] or expression[146], or `proprietary` if the license is not on the SPDX license list. Proprietary licensed data SHOULD add a link to the license text with the license relation in the links section.
- `providers` (array<object>): A list of providers, which may include all organisations capturing or processing the data or the hosting provider. Providers are listed in chronological order and each entry must provide the organisation name, any may add additional information such as a description, roles (any of `producer`, `licensor`, `processor` or `host`), and/or an URL.
- `extent` (object, required): Describes the spatio-temporal extents of the collection.
- `links` (array<object>): A list of related links, e.g. additional external documentation for this collection. Consists of a URL and optionally a relation type[147], a media type and a human-readable title.
- `properties` (object, required): A list of all metadata properties, which are common across the whole collection. Can be any of the fields in the following paragraphs.
- `other_properties` (object, required): A list of all metadata properties, which don't have common values across the whole collection. Can be any of the fields in the following paragraphs. It allows to specify a summary of the values as extent or set of values.

Data cubes

- `cube:dimensions` (object, required): Describes the dimensions that are available once the collection is loaded with the `load_collection` process. Each dimension consists of a type (`spatial`, `temporal`, `bands` or any custom type) and either an extent or a set of values. Optionally, a step, a unit, a reference system and, for spatial dimensions, the axis can be specified.

EO (Electro-Optical)

- `eo:gsd` (number): The nominal Ground Sample Distance for the data, as measured in meters on the ground. Since GSD can vary across a scene depending on projection, this should be the average or most commonly used GSD in the center of the image. If the

[145] https://spdx.org/licenses/
[146] https://spdx.org/spdx-specification-21-web-version#h.jxpfx0ykyb60
[147] https://www.iana.org/assignments/link-relations/link-relations.xml

data includes multiple bands with different GSD values, this should be the value for the greatest number or most common bands. For instance, Landsat optical and short-wave IR bands are all 30 meters, but the panchromatic band is 15 meters. The eo:gsd should be 30 meters in this case since those are the bands most commonly used.

- `eo:platform` (string): Unique name of the specific platform the instrument is attached to. For satellites this would be the name of the satellite (e.g., landsat-8, sentinel-2A).
- `eo:constellation` (string): The name of the group of satellites that have similar payloads and have their orbits arranged in a way to increase the temporal resolution of acquisitions of data with similar geometric and radiometric characteristics. Examples are the Sentinel-2 constellation, which has S2A and S2B and RapidEye. This field allows users to search for Sentinel-2 data, for example, without needing to specify which specific platform the data came from.
- `eo:instrument` (string): The name of the sensor used, although for Items which contain data from multiple sensors this could also name multiple sensors. For example, data from the Landsat-8 platform is collected with the OLI sensor as well as the TIRS sensor, but the data is distributed together and commonly referred to as OLI_TIRS.
- `eo:epsg` (number|null): EPSG code of the datasource, null if no EPSG code.
- `eo:bands` (array<object>): This is a list of the available bands. Each band may be described with a name, a common name, a description, the ground sample distance, the accuracy, a center wavelength and full width at half maximum (FWHM).

Not all fields from this extension have been ported over to openEO yet as they are usually only applicable on the item level. The missing fields are `eo:cloud_cover`, `eo:off_nadir`, `eo:azimuth`, `eo:sun_azimuth` and `eo:sun_elevation`, which may still be used in `other_properties` (see above).

SAR

- `sar:platform` (string): Unique name of the specific platform the instrument is attached to.
- `sar:constellation` (string): The name of the group of satellites that have similar payloads and have their orbits arranged in a way to increase the temporal resolution of acquisitions of data with similar geometric and radiometric characteristics. Examples are the Sentinel-1 constellation, entailing S1A, S1B and in future S1C and S1D as well as RADARSAT with RADARSAT-1 and RADARSAT-2. This field allows users to search for Sentinel-1 data, for example, without needing to specify which specific platform the data came from.
- `sar:instrument` (string): Name of the sensor used, although for Items which contain data from multiple sensors this could also name multiple sensors.
- `sar:instrument_mode` (string): The name of the sensor acquisition mode that is commonly used. This should be the short name, if available. For example, WV for "Wave mode" of Sentinel-1 and Envisat ASAR satellites.
- `sar:frequence_band` (string): The common name for the frequency band to make it eas-

ier to search for bands across instruments. One of `P`, `L`, `S`, `C`, `X`, `Ku`, `K` or `Ka`.

- `sar:center_wavelength` (number): The center wavelength of the instrument, in centimeters (cm).
- `sar:center_frequency` (number): The center frequency of the instrument, in gigahertz (GHz).
- `sar:polarization` (array<string>): A single polarization or a polarization combination specified as array. For single polarized radars one of `HH`, `VV`, `HV` or `VH` must be set. Fully polarimetric radars add all four polarizations to the array. Dual polarized radars and alternating polarization add the corresponding polarizations to the array, for instance for `HH+HV` add both `HH` and `HV`.
- `sar:bands` (array<object>): This is a list of the available bands. Each band may be described with a name, a description, a data_type, a unit and a polarization (either `HH`, `VV`, `HV`, `VH` or `null` if not applicable).
- `sar:type` (string): The product type, for example `RAW`, `GRD`, `OCN` or `SLC` for Sentinel-1.
- `sar:resolution` (array<number>): The maximum ability to distinguish two adjacent targets, in meters (m). The first element of the array is the range resolution, the second element is the azimuth resolution.
- `sar:pixel_spacing` (array<number>): The distance between adjacent pixels, in meters (m). The first element of the array is the range pixel spacing, the second element is the azimuth pixel spacing. Strongly RECOMMENDED to be specified for products of type `GRD`.
- `sar:looks` (array<number>): The number of groups of signal samples (looks). The first element of the array must be the number of range looks, the second element must be the number of azimuth looks, the optional third element is the equivalent number of looks (ENL).

Not all fields from this extension have been ported over to openEO yet as they are usually only applicable on the item level. The missing fields are `sar:pass_direction` and `sar:absolute_orbit`, which may still be used in `other_properties` (see above).

Scientific

- `sci:doi` (string): The DOI name of the collection.
- `sci:citation` (string): The recommended human-readable reference (citation) to be used by publications citing this collection.
- `sci:publications` (array<object>): A list of publications describing and referencing the collection. Each may include the DOI name and the human-readable reference (citation).

5 List of datasets

As discussed in Deliverable 09 [1] the project's use cases require Sentinel-1, -2 and -3 to be available at the back-ends. The following chapter will provide human-readable **examples** based on the STAC specification. The descriptions will vary between back-ends thus we can't provide more than examples. Two of the reasons are that back-ends provide different processing levels (e.g. Sentinel-2 L1C / L2A) of the data or cover different areas.

5.1 Sentinel 1 GRD

General Collection Metadata

- ID: Sentinel-1-GRD
- Title: Sentinel-1 SAR GRD
- Description: The dataset provides data from a dual-polarization C-band SAR instrument. It includes GRD[148] scenes, processed using the Sentinel-1 Toolbox. [...]
- License: proprietary[149]
- Provider(s):
  1. European Commission / ESA[150] (producer, licensor)
  2. openEO (processor, host)
- Temporal Extent: 2014-10-03 until present
- Spatial Extent (west, south, east, north): 180, -90, -180, 90 (WGS 84)
- STAC Version: 0.6.2

Common Properties

- Data Cube Dimensions:
  1. x (spatial, x): -180 to 180 (WGS 84)
  2. y (spatial, y): -90 to 90 (WGS 84)
  3. time (temporal): 2014-10-03 until present, irregular spaced steps
  4. bands (bands): amplitude_VV, amplitude_VH, amplitude_HV, amplitude_HH
- Constellation: Sentinel-1
- Instrument: C-SAR
- Frequency Band: C
- Center Wavelength: 5.546576466235 cm

[148] https://earth.esa.int/web/sentinel/user-guides/sentinel-1-sar/product-types-processing-levels/level-1
[149] https://scihub.copernicus.eu/twiki/pub/SciHubWebPortal/TermsConditions/Sentinel_Data_Terms_and_Conditions.pdf
[150] https://sentinel.esa.int/web/sentinel/missions/sentinel-2

- Center Frequency: 5.405 GHz
- Product Type: GRD
- Bands:
  1. amplitude_VV (Data type: amplitude, Polarization: VV)
  2. amplitude_VH (Data type: amplitude, Polarization: VH)
  3. amplitude_HV (Data type: amplitude, Polarization: HV)
  4. amplitude_HH (Data type: amplitude, Polarization: HH)

Non-Common Properties

- Platform: Sentinel-1A or Sentinel-1B
- Instrument Mode: SM, IW or EW
- Polarization: VV, HH, VV+VH or HH+HV
- Resolution: Range: 8.1 to 95.1 m, Azimuth: 8.1 to 90.6 m
- Pixel Spacing: 3.5 x 3.5 m, 10 x 10 m, 25 x 25 m or 40 x 40 m
- Looks: Range: 2 to 22, Azimuth: 1 to 22
- Equivalent Number of Looks (ENL): 2.7 to 398.4
- Pass Direction: ascending or descending

5.2 Sentinel 2

General Collection Metadata

- ID: Sentinel-2
- Title: Sentinel-2 MSI L1C
- Description: The dataset provides images from a high-resolution, multi-spectral imaging instrument. It includes Level-1C[151] scenes in UTM/WGS84 projection. [...]
- License: proprietary[152]
- Provider(s):
  1. European Commission / ESA[153] (producer, licensor)
  2. openEO (processor, host)
- Temporal Extent: 2015-06-23 until present
- Spatial Extent (west, south, east, north): 180, -56, -180, 83 (WGS 84)

[151] https://earth.esa.int/web/sentinel/user-guides/sentinel-2-msi/product-types/level-1c
[152] https://scihub.copernicus.eu/twiki/pub/SciHubWebPortal/TermsConditions/Sentinel_Data_Terms_and_Conditions.pdf
[153] https://sentinel.esa.int/web/sentinel/missions/sentinel-2

- STAC Version: 0.6.2

Common Properties

- Data Cube Dimensions:
  1. x (spatial, x): -180 to 180 (WGS 84)
  2. y (spatial, y): -56 to 83 (WGS 84)
  3. time (temporal): 2015-06-23 until present, irregular spaced steps
  4. spectral (bands): B1, B2, B3, B4, B5, B6, B7, B8, B8A, B9, B10, B11, B12
- Constellation: Sentinel-2
- Instrument: MSI
- Bands:
  1. B1 (coastal): 60 m
  2. B2 (blue): 10 m
  3. B3 (green): 10 m
  4. B4 (red): 10 m
  5. B5: 20 m
  6. B6: 20 m
  7. B7: 20 m
  8. B8 (nir): 10 m
  9. B8A (nir): 20 m
  10. B9: 60 m
  11. B10 (cirrus): 60 m
  12. B11 (swir16): 20 m
  13. B12 (swir22): 60 m

Non-Common Properties

- Platform: Sentinel-2A or Sentinel-2B
- EPSG Code(s): 32601, 32602, 32603, 32604, 32605, 32606, 32607, 32608, 32609, 32610, 32611, 32612, 32613, 32614, 32615, 32616, 32617, 32618, 32619, 32620, 32621, 32622, 32623, 32624, 32625, 32626, 32627, 32628, 32629, 32630, 32631, 32632, 32633, 32634, 32635, 32636, 32637, 32638, 32639, 32640, 32641, 32642, 32643, 32644, 32645, 32646, 32647, 32648, 32649, 32650, 32651, 32652, 32653, 32654, 32655, 32656, 32657, 32658, 32659, 32660
- Cloud Cover: 0 to 100 %
- Viewing Angle (off nadir): 0 to 90 (in degree)

- Viewing Azimuth Angle: 0 to 360 (in degree)
- Sun Azimuth Angle: 0 to 360 (in degree)
- Sun Elevation Angle: 0 to 90 (in degree)

6 References

[1] M. Schramm, W. Wagner, A. Jacob, J. Dries, A. Dostalova, S. Carter, J. Verbesselt, and M. Neteler, “openeo d09: First overview of needed processes from use cases,” May 2018. [Online]. Available: https://zenodo.org/record/2542822