intake-axds

Release 0.3.0

Axiom Data Science

Sep 12, 2023

USER DOCUMENTATION

CHAPTER

ONE

INSTALLATION

To install from conda-forge:

```
>>> conda install -c conda-forge intake-axds
```

To install from PyPI:

```
>>> pip install intake-axds
```

1.1 Overview

Use `intake-axds` to create intake catalogs containing sources in Axiom databases representing datasets. You can search in time and space as well as by variable and text to narrow to datasets for your project, then easily read in the data.

```
import intake
```

1.1.1 Datatypes

The default page size is 10, so requesting a datatype without any other input arguments will return the first 10 datasets of that datatype. The input argument `page_size` controls the maximum number of entries in the catalog.

Sensors (fixed location dataset like buoys)

Access sensor datasets by creating an AXDS catalog with `datatype="sensor_station"`. Note that webcam data is ignored.

```
cat = intake.open_axds_cat(datatype="sensor_station", page_size=10)
len(cat)
```

```
6
```

See what search was performed with `.get_search_urls()`.

```
cat.get_search_urls()
```

```
['https://search.axds.co/v2/search?portalId=-1&page=1&pageSize=10&verbose=true&
↪type=sensor_station']
```

See catalog-level metadata:

```
cat
```

```
catalog:
  args:
    datatype: sensor_station
    page_size: 10
  description: Catalog of Axiom assets.
  driver: intake_axds.axds_cat.AXDSCatalog
  metadata:
    kwargs_search:
      search_for:
      - null
    pgids:
    - null
    pglabels:
    - null
    query_type: union
```

What sources make up the catalog?

```
list(cat)
```

```
['org_mxak_naked_island',
 'ward-cove',
 'urn:ioos:station:gov.usgs.waterdata:02312600',
 'grave-point',
 'org_mxak_mary_island',
 'org_mxak_portland_island']
```

See source-level metadata for first source in catalog:

```
cat[list(cat)[0]]
```

```
org_mxak_naked_island:
  args:
    bin_interval: null
    binned: false
    end_time: null
    internal_id: 5
    only_pgids: null
    qartod: false
    start_time: null
    use_units: true
    uuid: org_mxak_naked_island
  description: AXDS dataset_id org_mxak_naked_island of datatype sensor_station
  driver: intake_axds.axds.AXDSSensorSource
  metadata:
    catalog_dir: ''
    datumConversions: []
    foreignNames:
    - null
```

(continues on next page)

(continued from previous page)

```
    - NAKED_ISLAND
    - NKXA2
    internal_id: 5
    maxLatitude: 58.255308
    maxLongitude: -134.945049
    maxTime: '2023-09-12T19:20:00Z'
    metadata_url: https://sensors.axds.co/api/metadata/filter/custom?filter=%7B
↪%22stations%22:%5B%225%22%5D%7D
    minLatitude: 58.255308
    minLongitude: -134.945049
    minTime: '2015-05-05T14:10:00.000Z'
    summary: Check that values are within reasonable bounds.
    title: 'Atmospheric Pressure: Barometric Pressure'
    uuid: org_mxak_naked_island
    variables:
    - wind_gust_from_direction
    - wind_from_direction
    - dew_point_temperature
    - wind_speed_of_gust
    - relative_humidity
    - air_temperature
    - air_pressure
    - wind_speed
    variables_details:
    - annotations: []
      label: 'Atmospheric Pressure: Barometric Pressure'
      parameterGroupId: 9
      plots:
      - label: '[default]'
        subPlots:
        - availableZ:
          - 0.0
          availableZBins: []
          datasetVariableId: air_pressure
          deviceId: 1014658
          discriminant: null
          endDate: '2023-09-12T19:20:00Z'
          feeds:
          - 1014760
          hasQc: true
          instrument: {}
          label: Barometric Pressure
          maxVal: 1043.4
          maxZ: 0.0
          medianTimeIntervalSecs: 600
          minVal: 965.8
          minZ: 0.0
          numObservations: 414369
          parameterGroupId: 9
          parameterId: 14
          plotLabel: '[default]'
          qcConfigId: 28
```

(continues on next page)

(continued from previous page)

```
        sensorParameterId: 14
        startDate: '2015-05-05T14:10:00Z'
        unitId: 24
        units: millibars
    - annotations: []
      label: Dew Point
      parameterGroupId: 29
      plots:
      - label: '[default]'
        subPlots:
        - availableZ:
          - 0.0
          availableZBins: []
          datasetVariableId: dew_point_temperature
          deviceId: 1014654
          discriminant: null
          endDate: '2023-09-12T19:20:00Z'
          feeds:
          - 1014756
          hasQc: true
          instrument: {}
          label: Dew Point
          maxVal: 17.22
          maxZ: 0.0
          medianTimeIntervalSecs: 600
          minVal: -22.89
          minZ: 0.0
          numObservations: 414018
          parameterGroupId: 29
          parameterId: 16
          plotLabel: '[default]'
          qcConfigId: 72
          sensorParameterId: 16
          startDate: '2015-05-05T14:10:00Z'
          unitId: 8
          units: degree_Celsius
    - annotations: []
      label: 'Humidity: Relative Humidity'
      parameterGroupId: 22
      plots:
      - label: '[default]'
        subPlots:
        - availableZ:
          - 0.0
          availableZBins: []
          datasetVariableId: relative_humidity
          deviceId: 1014652
          discriminant: null
          endDate: '2023-09-12T19:20:00Z'
          feeds:
          - 1014754
          hasQc: true
```

(continues on next page)

(continued from previous page)

```
        instrument: {}
        label: Relative Humidity
        maxVal: 99.9
        maxZ: 0.0
        medianTimeIntervalSecs: 600
        minVal: 21.1
        minZ: 0.0
        numObservations: 413936
        parameterGroupId: 22
        parameterId: 4
        plotLabel: '[default]'
        qcConfigId: 24
        sensorParameterId: 4
        startDate: '2015-05-05T14:10:00Z'
        unitId: 1
        units: '%'
  - annotations: []
    label: 'Temperature: Air Temperature'
    parameterGroupId: 6
    plots:
    - label: '[default]'
      subPlots:
      - availableZ:
        - 0.0
        availableZBins: []
        datasetVariableId: air_temperature
        deviceId: 1014651
        discriminant: null
        endDate: '2023-09-12T19:20:00Z'
        feeds:
        - 1014753
        hasQc: true
        instrument: {}
        label: Air Temperature
        maxVal: 26.5
        maxZ: 0.0
        medianTimeIntervalSecs: 600
        minVal: -14.0
        minZ: 0.0
        numObservations: 414491
        parameterGroupId: 6
        parameterId: 3
        plotLabel: '[default]'
        qcConfigId: 5
        sensorParameterId: 3
        startDate: '2015-05-05T14:10:00Z'
        unitId: 8
        units: degree_Celsius
  - annotations: []
    label: 'Winds: Gusts'
    parameterGroupId: 186
    plots:
```

(continues on next page)

(continued from previous page)


```
    - label: '[default]'
      subPlots:
      - availableZ:
        - 0.0
        availableZBins: []
        datasetVariableId: wind_speed_of_gust
        deviceId: 1014656
        discriminant: null
        endDate: '2023-09-12T19:20:00Z'
        feeds:
        - 1014758
        hasQc: true
        instrument: {}
        label: Wind Gust
        maxVal: 114.96
        maxZ: 0.0
        medianTimeIntervalSecs: 600
        minVal: 0.58
        minZ: 0.0
        numObservations: 415753
        parameterGroupId: 186
        parameterId: 7
        plotLabel: '[default]'
        qcConfigId: null
        sensorParameterId: 7
        startDate: '2015-05-05T14:10:00Z'
        unitId: 23
        units: mile.hour-1
      - availableZ:
        - 0.0
        availableZBins: []
        datasetVariableId: wind_gust_from_direction
        deviceId: 1014657
        discriminant: null
        endDate: '2023-09-12T19:20:00Z'
        feeds:
        - 1014759
        hasQc: true
        instrument: {}
        label: Wind Gust From Direction
        maxVal: 359.9
        maxZ: 0.0
        medianTimeIntervalSecs: 600
        minVal: 0.0
        minZ: 0.0
        numObservations: 414353
        parameterGroupId: 186
        parameterId: 75
        plotLabel: '[default]'
        qcConfigId: 11
        sensorParameterId: 75
        startDate: '2015-05-05T14:10:00Z'
```


(continues on next page)

(continued from previous page)


```
        unitId: 10
        units: degrees
    - annotations: []
      label: 'Winds: Speed and Direction'
      parameterGroupId: 8
      plots:
      - label: '[default]'
        subPlots:
        - availableZ:
          - 0.0
          availableZBins: []
          datasetVariableId: wind_speed
          deviceId: 1014653
          discriminant: null
          endDate: '2023-09-12T19:20:00Z'
          feeds:
          - 1014755
          hasQc: true
          instrument: {}
          label: Wind Speed
          maxVal: 48.92
          maxZ: 0.0
          medianTimeIntervalSecs: 600
          minVal: 0.1
          minZ: 0.0
          numObservations: 413810
          parameterGroupId: 8
          parameterId: 5
          plotLabel: '[default]'
          qcConfigId: 29
          sensorParameterId: 5
          startDate: '2015-05-05T14:10:00Z'
          unitId: 27
          units: m.s-1
        - availableZ:
          - 0.0
          availableZBins: []
          datasetVariableId: wind_from_direction
          deviceId: 1014655
          discriminant: null
          endDate: '2023-09-12T19:20:00Z'
          feeds:
          - 1014757
          hasQc: true
          instrument: {}
          label: Wind From Direction
          maxVal: 359.9
          maxZ: 0.0
          medianTimeIntervalSecs: 600
          minVal: 0.0
          minZ: 0.0
          numObservations: 417118
```


(continues on next page)

(continued from previous page)

```
        parameterGroupId: 8
        parameterId: 6
        plotLabel: '[default]'
        qcConfigId: 6
        sensorParameterId: 6
        startDate: '2015-05-05T14:10:00Z'
        unitId: 10
        units: degrees
    version: 2
```

Read data from first source in catalog. Note that since no start time or stop time was entered, the full data range will be read in, along with all available variables. The output is a DataFrame.

```
cat[list(cat)[0]].read()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[7], line 1
----> 1 cat[list(cat)[0]].read()

File ~/checkouts/readthedocs.org/user_builds/intake-axds/checkouts/latest/intake_axds/
↪axds.py:404, in AXDSSensorSource.read(self)
    402 def read(self):
    403     """read data in"""
--> 404     return self._get_partition(None)

File ~/checkouts/readthedocs.org/user_builds/intake-axds/checkouts/latest/intake_axds/
↪axds.py:399, in AXDSSensorSource._get_partition(self, _)
    397 """get partition"""
    398 if self._dataframe is None:
--> 399     self._load_metadata()
    400 return self._dataframe

File ~/checkouts/readthedocs.org/user_builds/intake-axds/conda/latest/lib/python3.9/site-
↪packages/intake/source/base.py:283, in DataSourceBase._load_metadata(self)
    281 """load metadata only if needed"""
    282 if self._schema is None:
--> 283     self._schema = self._get_schema()
    284     self.dtype = self._schema.dtype
    285     self.shape = self._schema.shape

File ~/checkouts/readthedocs.org/user_builds/intake-axds/checkouts/latest/intake_axds/
↪axds.py:387, in AXDSSensorSource._get_schema(self)
    383 """get schema"""
    384 if self._dataframe is None:
    385     # TODO: could do partial read with chunksize to get likely schema from
    386     # first few records, rather than loading the whole thing
--> 387     self._load()
    388 return base.Schema(
    389     datashape=None,
    390     dtype=self._dataframe.dtypes,
   (...)
```

(continues on next page)

(continued from previous page)

```
    393     extra_metadata={},
    394 )

File ~/checkouts/readthedocs.org/user_builds/intake-axds/checkouts/latest/intake_axds/
↪axds.py:372, in AXDSSensorSource._load(self)
    369 def _load(self):
    370     """How to load in a specific station once you know it by uuid"""
--> 372     dfs = [self._load_to_dataframe(url) for url in self.data_urls]
    374     df = dfs[0]
    375     # this gets different and I think better results than dfs[0].join(dfs[1:],␣
↪how="outer", sort=True)
    376     # even though they should probably return the same thing.

File ~/checkouts/readthedocs.org/user_builds/intake-axds/checkouts/latest/intake_axds/
↪axds.py:372, in <listcomp>(.0)
    369 def _load(self):
    370     """How to load in a specific station once you know it by uuid"""
--> 372     dfs = [self._load_to_dataframe(url) for url in self.data_urls]
    374     df = dfs[0]
    375     # this gets different and I think better results than dfs[0].join(dfs[1:],␣
↪how="outer", sort=True)
    376     # even though they should probably return the same thing.

File ~/checkouts/readthedocs.org/user_builds/intake-axds/checkouts/latest/intake_axds/
↪axds.py:204, in AXDSSensorSource._load_to_dataframe(self, url)
    202 if len(data_raw["data"]["groupedFeeds"]) == 0:
    203     self._dataframe = None
--> 204     raise ValueError(f"No data found for url {url}.")
    206 # loop over the data feeds and read the data into DataFrames
    207 # link to other metadata as needed
    208 dfs = []

ValueError: No data found for url https://sensors.axds.co/api/observations/filter/custom?
↪filter=%7B%22stations%22%3A%5B%225%22%5D%7D&start=2015-05-05T14:10:00Z&end=2023-09-
↪12T19:20:00Z.
```

Sensor-specific options

Options that are specific to sensors are QARTOD, units, and binning.

QARTOD

All time series available for sensors optionally come with an aggregate QARTOD flag time series.

By default, QARTOD flags are not returned, but will be returned if `qartod=True` is input to the call for catalog. Alternatively, a user can select that values that correspond to specific flags should be returned (with other values nan'ed out) with an input like `qartod=[1,2]` to only return the values that either pass the QARTOD tests or were not tested. Is not available if `binned==True`.

Flags are:

- 1: Pass

- 2: Not Evaluated
- 3: Suspect
- 4: Fail
- 9: Missing Data

More information on QARTOD is available here.

Units

By defaults units will be returned, syntax is "standard_name [units]". If False, no units will be included and then the syntax for column names is "standard_name".

Binning

By default, raw data for sensors is returned. However, binned data can instead by returned by entering `binned=True` and `bin_interval` options of hourly, daily, weekly, monthly, yearly. If `bin_interval` is input, binned is set to True.

Examples

For example, the following would return data columns as well as associated QARTOD columns, without units in the column names:

```
cat = intake.open_axds_cat(datatype="sensor_station", qartod=True, use_units=False)
```

This example would return data columns binned monthly:

```
cat = intake.open_axds_cat(datatype="sensor_station", bin_interval="monthly")
```

Platforms (traveling sensor, like gliders)

Access platforms datasets by creating an AXDS catalog with `datatype="platform2"`. Everything should work the same as demonstrated for sensors.

Data is output into a DataFrame for platforms. It is accessed by parquet file if available and otherwise by csv.

```
cat = intake.open_axds_cat(datatype="platform2")
list(cat)
```

See source-level metadata for first source in catalog:

```
cat[list(cat)[0]]
```

1.1.2 Filter in time and space

When setting up an AXDS intake catalog, you can narrow your search in time and space. The longitude values `min_lon` and `max_lon` should be in the range -180 to 180. You can search through the `kwargs_search` keyword or you can search explicitly using `bbox` (min_lon, min_lat, max_lon, max_lat) and `start_time` and `end_time`.

```
kw = {
    "min_lon": -180,
    "max_lon": -158,
    "min_lat": 50,
    "max_lat": 66,
    "min_time": '2015-1-1',
    "max_time": '2015-1-2',
}

cat = intake.open_axds_cat(datatype='sensor_station', kwargs_search=kw, page_size=5)
len(cat)
```

```
cat[list(cat)[0]]
```

1.1.3 Filter with keyword(s)

You can also narrow your search by one or more keywords, by passing a string or list of strings with `kwargs_search["search_for"]` or explicitly using `search_for`. If you input more than one string, be aware that the multiple searches required will be combined according to `query_type`, either as a logical OR if `query_type=="union"` or as a logical AND if `query_type=="intersection"`.

```
cat = intake.open_axds_cat(datatype='platform2', search_for=["whale", "bering"],
                           query_type="intersection", page_size=1000)
len(cat)
```

```
cat = intake.open_axds_cat(datatype='platform2', search_for=["whale", "bering"],
                           query_type="union", page_size=1000)
len(cat)
```

1.1.4 Filter by variable

This section describes two approaches for searching by variable. As with `search_for`, how multiple variable requests are combined depends on the input choice of `query_type`. However, in the case of variables there are three options for `query_type`:

- `query_type=="union"` logical OR
- `query_type=="intersection"` as a logical AND
- `query_type=="intersection_constrained"` as a logical AND but also only the requested variables are returned.

Select variable(s) to search for by standard_name

Check available standard names with:

```
import intake_axds

standard_names = intake_axds.utils.available_names()
len(standard_names), standard_names[:5]
```

Make a catalog of sensors that contain either of the standard_names input.

```
std_names = ["sea_water_practical_salinity", "sea_water_temperature"]
cat = intake.open_axds_cat(datatype="sensor_station", standard_names=std_names,
                           query_type="union")
cat[list(cat)[0]].metadata["variables"]
```

Make a catalog of sensors that contain both of the standard_names input.

```
std_names = ["sea_water_practical_salinity", "sea_water_temperature"]
cat = intake.open_axds_cat(datatype="sensor_station", standard_names=std_names,
                           query_type="intersection", page_size=100)
cat[list(cat)[0]].metadata["variables"]
```

Make a catalog of sensors that contain both of the standard_names input but then also only return those two variable types. All variables available in the dataset will still be present in the metadata, but only values for those requested will be returned in the DataFrame. We can look at the catalog metadata to see the parameterGroupIds and parameterGroupLables that will be used in data collection.

```
std_names = ["sea_water_practical_salinity", "sea_water_temperature"]
cat = intake.open_axds_cat(datatype="sensor_station", standard_names=std_names,
                           query_type="intersection_constrained", page_size=100)
cat
```

If you request standard_names that aren't present in the system, you will be told (cell commented out but will return exception and say that they aren't present).

```
# std_names = "sea_water_surface_salinity"
# cat = intake.open_axds_cat(datatype="sensor_station", standard_names=std_names)
```

Select variable(s) to search for by custom vocabulary

Instead of selecting the exact standard_names to search on, you can set up a collections of regular expressions to match on the variables you want. This is particularly useful if you are running with several different searches and ultimately will need to select data variables from datasets using a generic name.

Set up vocabulary

One way to set up a custom vocabulary is with a helper class from `cf-pandas` (see more information in the docs). Choose a nickname for each variable you want to be able to match on, like "temp" for matching sea water temperature variables, then set up the regular expressions you want to "count" as your variable "temp" — you can use the "Reg" class from `cf-pandas` to write these expressions easily. The following example shows setting up a custom vocabulary for identifying variables of "temp", "salt", and "ssh".

```
import cf_pandas as cfp

nickname = "temp"
vocab = cfp.Vocab()

# define a regular expression to represent your variable
reg = cfp.Reg(include="temp", exclude=["air","qc","status","atmospheric"])

# Make an entry to add to your vocabulary
vocab.make_entry(nickname, reg.pattern(), attr="name")

vocab.make_entry("salt", cfp.Reg(include="sal", exclude=["soil","qc","status"]).
↪pattern(), attr="name")
vocab.make_entry("ssh", cfp.Reg(include=["sea_surface_height","surface_elevation"],␣
↪exclude=["qc","status"]).pattern(), attr="name")

# what does the vocabulary look like?
vocab.vocab
```

You can use your custom vocab with a context manager, as in the following example. Alternatively, you can set the vocabulary up so all commands will know about it:

```
cf_xarray.set_options(custom_criteria=vocab.vocab)  # for cf-xarray
cfp.set_options(custom_criteria=vocab.vocab)  # for cf-pandas
```

```
with cfp.set_options(custom_criteria=vocab.vocab):
    cat = intake.open_axds_cat(datatype="platform2", keys_to_match=["temp","salt"])
cat[list(cat)[0]].metadata["variables"]
```

1.1.5 Catalog metadata and options

Can provide metadata at the catalog level with input arguments `name`, `description`, and `metadata` to override the defaults.

```
cat = intake.open_axds_cat(datatype="platform2", name="Catalog name", description="This␣
↪is the catalog.", page_size=1,
                           metadata={"special entry": "platforms"})
cat
```

ttl

The default `ttl` argument, or time before force-reloading the catalog, is `None`, but can be overridden by inputting a value:

```
cat.ttl is None
```

```
cat = intake.open_axds_cat(datatype="platform2", page_size=1, ttl=60)
cat.ttl
```

Verbose

Get information as the catalog function runs.

```
cat = intake.open_axds_cat(datatype="sensor_station", verbose=True, page_size=1)
```

1.1.6 Sensor Source

You can use the intake `AXDSSensorSource` directly with `intake.open_axds_sensor` if you know the `dataset_id` (UUID) or the `internal_id` (Axiom station id). Alternatively, you can search using `intake.open_axds_cat` for a sensor if you know the dataset_id and search for it with "search_for".

Note that only some metadata will be available until the dataset is read in, at which point the full metadata is also read in.

```
source = intake.open_axds_sensor(internal_id=110532, bin_interval="monthly")
source
```

```
source.read()
```

If you prefer a catalog approach for a known dataset_id, you can do that like this:

```
cat = intake.open_axds_cat(datatype="sensor_station", search_for="ism-aoos-noaa_nos_co_
↪ops_9469439",
                           verbose=True)
```

```
cat[list(cat)[0]]
```

You can request only specific data variable(s) be returned directly in the Sensor Source, though you need to know the parameterGroupId. You could access this by running the desired source once and looking at the metadata to select the

IDs you want to use. For example using the information from the previous catalog listed immediately above, we could set up the following:

```
source = intake.open_axds_sensor(internal_id=110532, bin_interval="monthly", only_
↪pgids=[47])
source
```

1.2 API

1.2.1 intake-axds Python API

intake-axds catalog

Set up a catalog for Axiom assets.

AXDSCatalog

class intake_axds.axds_cat.**AXDSCatalog**(**args*, ***kwargs*)

Makes data sources out of all datasets for a given AXDS data type.

pglabels

If `keys_to_match` or `standard_names` is input to search on, they are converted to parameterGroupLabels and saved to the catalog metadata.

Type
list[str]

pgids

If `keys_to_match` or `standard_names` is input to search on, they are converted to parameterGroupIds and saved to the catalog metadata. In the case that `query_type=="intersection_constrained"` and `datatype=="platform2"`, the pgids are passed to the sensor source so that only data from variables corresponding to those pgids are returned.

Type
list[int]

Parameters

- **datatype** (*str*) – Axiom data type. Currently "platform2" or "sensor_station" but eventually also "module". Platforms and sensors are returned as dataframe containers.
- **keys_to_match** (*str, list, optional*) – Name of keys to match with system-available variable parameterNames using criteria. To filter search by variables, either input keys_to_match and a vocabulary or input standard_names. Results from multiple values will be combined according to `query_type`.
- **standard_names** (*str, list, optional*) – Standard names to select from Axiom search parameterNames. If more than one is input, the search is for a logical OR of datasets containing the standard_names. To filter search by variables, either input keys_to_match and a vocabulary or input standard_names. Results from multiple values will be combined according to `query_type`.
- **bbox** (*tuple of 4 floats, optional*) – For explicit geographic search queries, pass a tuple of four floats in the *bbox* argument. The bounding box parameters are *(min_lon, min_lat, max_lon, max_lat)*.

- **start_time** (*str, optional*) – For explicit search queries for datasets that contain data after *start_time*. Must include end_time if include start_time.
- **end_time** (*str, optional*) – For explicit search queries for datasets that contain data before *end_time*. Must include start_time if include end_time.
- **search_for** (*str, list of strings, optional*) – For explicit search queries for datasets that any contain of the terms specified in this keyword argument. Results from multiple values will be combined according to `query_type`.
- **kwargs_search** (*dict, optional*) – Keyword arguments to input to search on the server before making the catalog. Options are:
  - to search by bounding box: include all of min_lon, max_lon, min_lat, max_lat: (int, float). Longitudes must be between -180 to +180.
  - to search within a datetime range: include both of min_time, max_time: interpretable datetime string, e.g., "2021-1-1"
  - to search using a textual keyword: include *search_for* as a string or list of strings. Results from multiple values will be combined according to `query_type`.
- **query_type** (*str, default "union"*) – Specifies how the catalog should apply the query parameters. Choices are:
  - `"union"`: the results will be the union of each resulting dataset. This is equivalent to a logical OR.
  - `"intersection"`: the set of results will be the intersection of each individual query made to the server. This is equivalent to a logical AND of the results.
  - `"intersection_constrained"`: the set of results will be the intersection of queries but also only the variables requested (using either `keys_to_match` or `standard_names`) will be returned in the DataFrame, instead of all available variables. This only applies to `datatype=="sensor_station"`.
- **qartod** (*bool, int, list, optional*) – Whether to return QARTOD agg flags when available, which is only for sensor_stations. Can instead input an int or a list of ints representing the _qa_agg flags for which to return data values. More information about QARTOD testing and flags can be found here: https://cdn.ioos.noaa.gov/media/2020/07/QARTOD-Data-Flags-Manual_version1.2final.pdf. Only used by datatype "sensor_station". Is not available if `binned==True`.

  Examples of ways to use this input are:
  - `qartod=True`: Return aggregate QARTOD flags as a column for each data variable.
  - `qartod=False`: Do not return any QARTOD flag columns.
  - `qartod=1`: nan any data values for which the aggregated QARTOD flags are not equal to 1.
  - `qartod=[1,3]`: nan any data values for which the aggregated QARTOD flags are not equal to 1 or 3.

  Flags are:
  - 1: Pass
  - 2: Not Evaluated
  - 3: Suspect
  - 4: Fail

  - 9: Missing Data

- **use_units** (*bool, optional*) – If True include units in column names. Syntax is "standard_name [units]". If False, no units. Then syntax for column names is "standard_name". This is currently specific to sensor_station only. Only used by datatype "sensor_station".
- **binned** (*bool, optional*) – True for binned data, False for raw, by default False. Only used by datatype "sensor_station".
- **bin_interval** (*Optional[str], optional*) – If `binned=True`, input the binning interval to return. Options are hourly, daily, weekly, monthly, yearly. If bin_interval is input, binned is set to True. Only used by datatype "sensor_station".
- **page_size** (*int, optional*) – Number of results. Fewer is faster. Note that default is 10. Note that if you want to make sure you get all available datasets, you should input a large number like 50000.
- **verbose** (*bool, optional*) – Set to True for helpful information.
- **ttl** (*int, optional*) – Time to live for catalog (in seconds). How long before force-reloading catalog. Set to None to not do this.
- **name** (*str, optional*) – Name for catalog.
- **description** (*str, optional*) – Description for catalog.
- **metadata** (*dict, optional*) – Metadata for catalog.
- **kwargs** – Other input arguments are passed to the intake Catalog class. They can includegetenv, getshell, persist_mode, storage_options, and user_parameters, in addition to some that are surfaced directly in this class.

Notes

only datatype sensor_station uses the following parameters: qartod, use_units, binned, bin_interval

datatype of sensor_station skips webcam data.

Attributes

auth
cache
cache_dirs
cat
classname
description
dtype
entry
gui
Source GUI, with parameter selection and plotting

has_been_persisted
hvplot
alias for `DataSource.plot`

is_persisted
kwargs
plot
Plot API accessor

plots
List custom associated quick-plots

shape

Methods

`__call__`(**kwargs)	Create a new instance of this source with altered arguments
close()	Close open resources corresponding to this data source.
configure_new(**kwargs)	Create a new instance of this source with altered arguments
describe()	Description from the entry spec
discover()	Open resource and populate the source attributes.
export(path, **kwargs)	Save this data for sharing with other people
filter(func)	Create a Catalog of a subset of entries based on a condition
force_reload()	Imperative reload data now
from_dict(entries, **kwargs)	Create Catalog from the given set of entries
get(**kwargs)	Create a new instance of this source with altered arguments
get_search_urls()	Gather all search urls for catalog.
items()	Get an iterator over (key, source) tuples for the catalog entries.
keys()	Entry names in this catalog as an iterator (alias for __iter__)
persist([ttl])	Save data from this source to local persistent storage
pop(key)	Remove entry from catalog and return it
read()	Load entire dataset into a container and return it
read_chunked()	Return iterator over container fragments of data source
read_partition(i)	Return a part of the data corresponding to i-th partition.
reload()	Reload catalog if sufficient time has passed
save(url[, storage_options])	Output this catalog to a file as YAML
search_url([pglabel, text_search])	Set up one url for searching.
serialize()	Produce YAML version of this catalog.
to_dask()	Return a dask container for this data source
to_spark()	Provide an equivalent data object in Apache Spark
values()	Get an iterator over the sources for catalog entries.
walk([sofar, prefix, depth])	Get all entries in this catalog and sub-catalogs
yaml()	Return YAML representation of this data-source

get_persisted	
search	
set_cache_dir	

`get_search_urls`() → list

Gather all search urls for catalog.

Inputs that can have more than one search_url are pglabels and search_for list.

Returns
List of search urls.

Return type
list

search_url(*pglabel: Optional[str] = None*, *text_search: Optional[str] = None*) → str

Set up one url for searching.

Parameters

- **pglabel** (*Optional[str], optional*) – Parameter Group Label (not ID), by default None
- **text_search** (*Optional[str], optional*) – free text search, by default None

Returns
URL to use to search Axiom systems.

Return type
str

intake-axds sensor source

AXDSSensorSource

class intake_axds.axds.**AXDSSensorSource**(**args*, ***kwargs*)

Intake Source for AXDS sensor

Parameters

- **internal_id** (*Optional[int], optional*) – Internal station id for Axiom, by default None. Not the UUID. Need to input internal_id or UUID. If both are input, be sure they are for the same station.
- **uuid** (*Optional[str], optional*) – The UUID for the station, by default None. Not the internal_id. Need to input internal_id or UUID. If both are input, be sure they are for the same station. Note that there may also be a "datasetId" parameter which is sometimes but not always the same as the UUID.
- **start_time** (*Optional[str], optional*) – At what datetime for data to start, by default None. Must be interpretable by pandas `Timestamp`. If not input, the datetime at which the dataset starts will be used.
- **end_time** (*Optional[str], optional*) – At what datetime for data to end, by default None. Must be interpretable by pandas `Timestamp`. If not input, the datetime at which the dataset ends will be used.
- **qartod** (*bool, int, list, optional*) – Whether to return QARTOD agg flags when available, which is only for sensor_stations. Can instead input an int or a list of ints representing the _qa_agg flags for which to return data values. More information about QARTOD testing and flags can be found here: https://cdn.ioos.noaa.gov/media/2020/07/QARTOD-Data-Flags-Manual_version1.2final.pdf. Only used by datatype "sensor_station". Is not available if *binned==True*.

  Examples of ways to use this input are:

  – `qartod=True`: Return aggregate QARTOD flags as a column for each data variable.

  – `qartod=False`: Do not return any QARTOD flag columns.

- qartod=1: nan any data values for which the aggregated QARTOD flags are not equal to 1.
- qartod=[1,3]: nan any data values for which the aggregated QARTOD flags are not equal to 1 or 3.

Flags are:

- 1: Pass
- 2: Not Evaluated
- 3: Suspect
- 4: Fail
- 9: Missing Data

- **use_units** (*bool, optional*) – If True include units in column names. Syntax is "standard_name [units]". If False, no units. Then syntax for column names is "standard_name". This is currently specific to sensor_station only. Only used by datatype "sensor_station".
- **metadata** (*dict, optional*) – Metadata for catalog.
- **binned** (*bool, optional*) – True for binned data, False for raw, by default False. Only used by datatype "sensor_station".
- **bin_interval** (*Optional[str], optional*) – If binned=True, input the binning interval to return. Options are hourly, daily, weekly, monthly, yearly. If bin_interval is input, binned is set to True. Only used by datatype "sensor_station".
- **only_pgids** (*list, optional*) – If input, only return data associated with these parameterGroupIds. This is separate from parameterGroupLabels and parameterGroupIds that might be present in the metadata.

Raises

ValueError – _description_

Attributes

cache
cache_dirs
cat
classname
data_urls
Prepare to load in data by getting data_urls.

description
dtype
entry
gui
Source GUI, with parameter selection and plotting

has_been_persisted
hvplot
alias for DataSource.plot

is_persisted
plot
Plot API accessor

plots
List custom associated quick-plots

shape

Methods

`__call__`(**kwargs)	Create a new instance of this source with altered arguments
close()	Close open resources corresponding to this data source.
configure_new(**kwargs)	Create a new instance of this source with altered arguments
describe()	Description from the entry spec
discover()	Open resource and populate the source attributes.
export(path, **kwargs)	Save this data for sharing with other people
get(**kwargs)	Create a new instance of this source with altered arguments
get_filters()	Return appropriate filter for stationid.
persist([ttl])	Save data from this source to local persistent storage
read()	read data in
read_chunked()	Return iterator over container fragments of data source
read_partition(i)	Return a part of the data corresponding to i-th partition.
to_dask()	Return a dask container for this data source
to_spark()	Provide an equivalent data object in Apache Spark
yaml()	Return YAML representation of this data-source

get_persisted	
set_cache_dir	

property data_urls

Prepare to load in data by getting data_urls.

For V1 sources there will be a data_url per parameterGroupId but not for V2 sources.

get_filters()

Return appropriate filter for stationid.

What filter form to use depends on if V1 or V2.

For V1, use each parameterGroupId only once to make a filter since all data of that type will be read in together.

Following Sensor API https://admin.axds.co/#!/sensors/api/overview

read()

read data in

intake-axds utilities

Utils to run.

intake_axds.utils.**available_names**() → list

Return available parameterNames for variables.

Returns
parametersNames, which are a superset of standard_names.

Return type
list

intake_axds.utils.**check_station**(*metadata: dict*, *verbose: bool*) → bool

Whether to keep station or not.

Parameters

- **metadata** (*dict*) – metadata about station.
- **verbose** (*bool, optional*) – Set to True for helpful information.

Returns
True to keep station, False to skip.

Return type
bool

intake_axds.utils.**load_metadata**(*datatype: str*, *results: dict*) → dict

Load metadata for catalog entry.

Parameters
results (*dict*) – Returned results from call to server for a single dataset.

Returns
Metadata to store with catalog entry.

Return type
dict

intake_axds.utils.**make_data_url**(*filter: str*, *start_time: str*, *end_time: str*, *binned: bool = False*, *bin_interval: Optional[str] = None*) → str

Create url for accessing sensor data, raw or binned.

Parameters

- **filter** (*str*) – get this from `make_filter()`; contains station and potentially variable info.
- **start_time** (*str*) – e.g. "2022-1-1". Needs to be interpretable by pandas `Timestamp`.
- **end_time** (*str*) – e.g. "2022-1-2". Needs to be interpretable by pandas `Timestamp`.
- **binned** (*bool, optional*) – True for binned data, False for raw, by default False.
- **bin_interval** (*Optional[str], optional*) – If `binned=True`, input the binning interval to return. Options are hourly, daily, weekly, monthly, yearly.

Returns
URL from which to access data.

Return type
str

intake_axds.utils.**make_filter**(*internal_id: int*, *parameterGroupId: Optional[int] = None*) → str

Make filter for Axiom Sensors API.

Parameters

- **internal_id** (*int*) – internal id for station. Not the uuid.
- **parameterGroupId** (*Optional[int], optional*) – Parameter Group ID to narrow search, by default None

Returns
filter to use in station metadata and data access

Return type
str

intake_axds.utils.**make_label**(*label: str*, *units: Optional[str] = None*, *use_units: bool = True*) → str

making column name

Parameters

- **label** (*str*) – variable label to use in column header
- **units** (*Optional[str], optional*) – units to use in column name, if not None, by default None
- **use_units** (*bool, optional*) – Users can choose not to include units in column name, by default True

Returns
string to use as column name

Return type
str

intake_axds.utils.**make_metadata_url**(*filter: str*) → str

Make url for finding metadata

Parameters
filter (*str*) – filter for Sensors API. Use `make_filter` to make this.

Returns
url for metadata.

Return type
str

intake_axds.utils.**make_search_docs_url**(*internal_id: Optional[int] = None*, *uuid: Optional[str] = None*) → str

Url for Axiom Search docs.

Uses whichever of internal_id and uuid is not None to formulate url.

Parameters

- **internal_id** (*Optional[int], optional*) – Internal station id for Axiom. Not the UUID.
- **uuid** (*str*) – uuid for station.

Returns
Url for finding Axiom Search docs

Return type
str

intake_axds.utils.**match_key_to_parameter**(*keys_to_match: list*, *criteria: Optional[dict] = None*) → list

Find Parameter Group values that match keys_to_match.

Parameters

- **keys_to_match** (*list*) – The custom_criteria key to narrow the search, which will be matched to the category results using the custom_criteria that must be set up ahead of time with *cf-pandas*.
- **criteria** (*dict, optional*) – Criteria to use to map from variable to attributes describing the variable. If user has defined custom_criteria, this will be used by default.

Returns
Parameter Group values that match key, according to the custom criteria.

Return type
list

intake_axds.utils.**match_std_names_to_parameter**(*standard_names: list*) → list

Find Parameter Group values that match standard_names.

Parameters
standard_names (*list*) – standard_names values to narrow the search.

Returns
Parameter Group values that match standard_names.

Return type
list

intake_axds.utils.**response_from_url**(*url: str*) → Union[list, dict]

Return response from url.

Parameters
url (*str*) – URL to check.

Returns
should be a list or dict depending on the url

Return type
list, dict

Inherited from intake

intake-axds catalog

Set up a catalog for Axiom assets.

class intake_axds.axds_cat.**AXDSCatalog**(**args*, ***kwargs*)

Bases: Catalog

Makes data sources out of all datasets for a given AXDS data type.

pglabels

If keys_to_match or standard_names is input to search on, they are converted to parameterGroupLabels and saved to the catalog metadata.

Type
list[str]

pgids

If `keys_to_match` or `standard_names` is input to search on, they are converted to parameterGroupIds and saved to the catalog metadata. In the case that `query_type=="intersection_constrained"` and `datatype=="platform2"`, the pgids are passed to the sensor source so that only data from variables corresponding to those pgids are returned.

Type
list[int]

Parameters

- **datatype** (*str*) – Axiom data type. Currently "platform2" or "sensor_station" but eventually also "module". Platforms and sensors are returned as dataframe containers.
- **keys_to_match** (*str, list, optional*) – Name of keys to match with system-available variable parameterNames using criteria. To filter search by variables, either input keys_to_match and a vocabulary or input standard_names. Results from multiple values will be combined according to `query_type`.
- **standard_names** (*str, list, optional*) – Standard names to select from Axiom search parameterNames. If more than one is input, the search is for a logical OR of datasets containing the standard_names. To filter search by variables, either input keys_to_match and a vocabulary or input standard_names. Results from multiple values will be combined according to `query_type`.
- **bbox** (*tuple of 4 floats, optional*) – For explicit geographic search queries, pass a tuple of four floats in the *bbox* argument. The bounding box parameters are *(min_lon, min_lat, max_lon, max_lat)*.
- **start_time** (*str, optional*) – For explicit search queries for datasets that contain data after *start_time*. Must include end_time if include start_time.
- **end_time** (*str, optional*) – For explicit search queries for datasets that contain data before *end_time*. Must include start_time if include end_time.
- **search_for** (*str, list of strings, optional*) – For explicit search queries for datasets that any contain of the terms specified in this keyword argument. Results from multiple values will be combined according to `query_type`.
- **kwargs_search** (*dict, optional*) – Keyword arguments to input to search on the server before making the catalog. Options are:
  - to search by bounding box: include all of min_lon, max_lon, min_lat, max_lat: (int, float). Longitudes must be between -180 to +180.
  - to search within a datetime range: include both of min_time, max_time: interpretable datetime string, e.g., "2021-1-1"
  - to search using a textual keyword: include *search_for* as a string or list of strings. Results from multiple values will be combined according to `query_type`.
- **query_type** (*str, default "union"*) – Specifies how the catalog should apply the query parameters. Choices are:
  - `"union"`: the results will be the union of each resulting dataset. This is equivalent to a logical OR.
  - `"intersection"`: the set of results will be the intersection of each individual query made to the server. This is equivalent to a logical AND of the results.

  - "intersection_constrained": the set of results will be the intersection of queries but also only the variables requested (using either keys_to_match or standard_names) will be returned in the DataFrame, instead of all available variables. This only applies to datatype=="sensor_station".

- **qartod** (*bool, int, list, optional*) – Whether to return QARTOD agg flags when available, which is only for sensor_stations. Can instead input an int or a list of ints representing the _qa_agg flags for which to return data values. More information about QARTOD testing and flags can be found here: https://cdn.ioos.noaa.gov/media/2020/07/QARTOD-Data-Flags-Manual_version1.2final.pdf. Only used by datatype "sensor_station". Is not available if binned==True.

  Examples of ways to use this input are:

  - qartod=True: Return aggregate QARTOD flags as a column for each data variable.
  - qartod=False: Do not return any QARTOD flag columns.
  - qartod=1: nan any data values for which the aggregated QARTOD flags are not equal to 1.
  - qartod=[1,3]: nan any data values for which the aggregated QARTOD flags are not equal to 1 or 3.

  Flags are:

  - 1: Pass
  - 2: Not Evaluated
  - 3: Suspect
  - 4: Fail
  - 9: Missing Data

- **use_units** (*bool, optional*) – If True include units in column names. Syntax is "standard_name [units]". If False, no units. Then syntax for column names is "standard_name". This is currently specific to sensor_station only. Only used by datatype "sensor_station".
- **binned** (*bool, optional*) – True for binned data, False for raw, by default False. Only used by datatype "sensor_station".
- **bin_interval** (*Optional[str], optional*) – If binned=True, input the binning interval to return. Options are hourly, daily, weekly, monthly, yearly. If bin_interval is input, binned is set to True. Only used by datatype "sensor_station".
- **page_size** (*int, optional*) – Number of results. Fewer is faster. Note that default is 10. Note that if you want to make sure you get all available datasets, you should input a large number like 50000.
- **verbose** (*bool, optional*) – Set to True for helpful information.
- **ttl** (*int, optional*) – Time to live for catalog (in seconds). How long before force-reloading catalog. Set to None to not do this.
- **name** (*str, optional*) – Name for catalog.
- **description** (*str, optional*) – Description for catalog.
- **metadata** (*dict, optional*) – Metadata for catalog.
- **kwargs** – Other input arguments are passed to the intake Catalog class. They can includegetenv, getshell, persist_mode, storage_options, and user_parameters, in addition to some that are surfaced directly in this class.

Notes

only datatype sensor_station uses the following parameters: qartod, use_units, binned, bin_interval

datatype of sensor_station skips webcam data.

Attributes

auth
cache
cache_dirs
cat
classname
description
dtype
entry
gui
Source GUI, with parameter selection and plotting

has_been_persisted
hvplot
alias for `DataSource.plot`

is_persisted
kwargs
plot
Plot API accessor

plots
List custom associated quick-plots

shape

Methods

`__call__`(**kwargs)	Create a new instance of this source with altered arguments
close()	Close open resources corresponding to this data source.
configure_new(**kwargs)	Create a new instance of this source with altered arguments
describe()	Description from the entry spec
discover()	Open resource and populate the source attributes.
export(path, **kwargs)	Save this data for sharing with other people
filter(func)	Create a Catalog of a subset of entries based on a condition
force_reload()	Imperative reload data now
from_dict(entries, **kwargs)	Create Catalog from the given set of entries
get(**kwargs)	Create a new instance of this source with altered arguments
get_search_urls()	Gather all search urls for catalog.
items()	Get an iterator over (key, source) tuples for the catalog entries.
keys()	Entry names in this catalog as an iterator (alias for __iter__)
persist([ttl])	Save data from this source to local persistent storage
pop(key)	Remove entry from catalog and return it
read()	Load entire dataset into a container and return it
read_chunked()	Return iterator over container fragments of data source
read_partition(i)	Return a part of the data corresponding to i-th partition.
reload()	Reload catalog if sufficient time has passed
save(url[, storage_options])	Output this catalog to a file as YAML
search_url([pglabel, text_search])	Set up one url for searching.
serialize()	Produce YAML version of this catalog.
to_dask()	Return a dask container for this data source
to_spark()	Provide an equivalent data object in Apache Spark
values()	Get an iterator over the sources for catalog entries.
walk([sofar, prefix, depth])	Get all entries in this catalog and sub-catalogs
yaml()	Return YAML representation of this data-source

get_persisted	
search	
set_cache_dir	

close()

Close open resources corresponding to this data source.

configure_new(***kwargs*)

Create a new instance of this source with altered arguments

Enables the picking of options and re-evaluating templates from any user-parameters associated with this source, or overriding any of the init arguments.

Returns a new data source instance. The instance will be recreated from the original entry definition in a catalog **if** this source was originally created from a catalog.

`describe`()

Description from the entry spec

`discover`()

Open resource and populate the source attributes.

`export`(*path*, ***kwargs*)

Save this data for sharing with other people

Creates a copy of the data in a format appropriate for its container, in the location specified (which can be remote, e.g., s3).

Returns the resultant source object, so that you can, for instance, add it to a catalog (`catalog.add(source)`) or get its YAML representation (`.yaml()`).

`filter`(*func*)

Create a Catalog of a subset of entries based on a condition

Warning: This function operates on CatalogEntry objects not DataSource objects.

Note: Note that, whatever specific class this is performed on, the return instance is a Catalog. The entries are passed unmodified, so they will still reference the original catalog instance and include its details such as directory,.

Parameters

func (*function*) – This should take a CatalogEntry and return True or False. Those items returning True will be included in the new Catalog, with the same entry names

Returns

New catalog with Entries that still refer to their parents

Return type

Catalog

`force_reload`()

Imperative reload data now

`classmethod from_dict`(*entries*, ***kwargs*)

Create Catalog from the given set of entries

Parameters

- **entries** (*dict-like*) – A mapping of name:entry which supports dict-like functionality, e.g., is derived from `collections.abc.Mapping`.
- **kwargs** (*passed on the constructor*) – Things like metadata, name; see `__init__`.

Return type

Catalog instance

get(***kwargs*)

Create a new instance of this source with altered arguments

Enables the picking of options and re-evaluating templates from any user-parameters associated with this source, or overriding any of the init arguments.

Returns a new data source instance. The instance will be recreated from the original entry definition in a catalog **if** this source was originally created from a catalog.

get_search_urls() → list

Gather all search urls for catalog.

Inputs that can have more than one search_url are pglabels and search_for list.

Returns
List of search urls.

Return type
list

property gui

Source GUI, with parameter selection and plotting

property has_been_persisted

The base class does not interact with persistence

property hvplot

alias for `DataSource.plot`

property is_persisted

The base class does not interact with persistence

items()

Get an iterator over (key, source) tuples for the catalog entries.

keys()

Entry names in this catalog as an iterator (alias for __iter__)

persist(*ttl=None*, ***kwargs*)

Save data from this source to local persistent storage

Parameters

- **ttl** (*numeric, optional*) – Time to live in seconds. If provided, the original source will be accessed and a new persisted version written transparently when more than `ttl` seconds have passed since the old persisted version was written.
- **kargs** (*passed to the _persist method on the base container.*) –

property plot

Plot API accessor

This property exposes both predefined plots (described in the source metadata) and general-purpose plotting via the hvPlot library. Supported containers are: array, dataframe and xarray,

To display in a notebook, be sure to run `intake.output_notebook()` first.

The set of plots defined for this source can be found by

```
>>> source.plots
["plot1", "plot2"]
```

and to display one of these:

```
>>> source.plot.plot1()
<holoviews/panel output>
```

To create new plot types and supply custom configuration, use one of the methods of `hvplot.hvPlot`:

```
>>> source.plot.line(x="fieldX", y="fieldY")
```

The full set of arguments that can be passed, and the types of plot they refer to, can be found in the doc and attributes of `hvplot.HoloViewsConverter`.

Once you have found a suitable plot, you may wish to update the plots definitions of the source. Simply add the `plotname=` optional argument (this will overwrite any existing plot of that name). The source's YAML representation will include the new plot, and it could be saved into a catalog with this new definition.

```
>>> source.plot.line(plotname="new", x="fieldX", y="fieldY");
>>> source.plots
["plot1", "plot2", "new"]
```

property plots

List custom associated quick-plots

pop(*key*)

Remove entry from catalog and return it

This relies on the *_entries* attribute being mutable, which it normally is. Note that if a catalog automatically reloads, any entry removed here may soon reappear

Parameters

key (*str*) – Key to give the entry in the cat

read()

Load entire dataset into a container and return it

read_chunked()

Return iterator over container fragments of data source

read_partition(*i*)

Return a part of the data corresponding to i-th partition.

By default, assumes i should be an integer between zero and npartitions; override for more complex indexing schemes.

reload()

Reload catalog if sufficient time has passed

save(*url*, *storage_options=None*)

Output this catalog to a file as YAML

Parameters

- **url** (*str*) – Location to save to, perhaps remote
- **storage_options** (*dict*) – Extra arguments for the file-system

search_url(*pglabel: Optional[str] = None*, *text_search: Optional[str] = None*) → str

Set up one url for searching.

Parameters

- **pglabel** (*Optional[str], optional*) – Parameter Group Label (not ID), by default None
- **text_search** (*Optional[str], optional*) – free text search, by default None

Returns
URL to use to search Axiom systems.

Return type
str

serialize()

Produce YAML version of this catalog.

Note that this is not the same as `.yaml()`, which produces a YAML block referring to this catalog.

to_dask()

Return a dask container for this data source

to_spark()

Provide an equivalent data object in Apache Spark

The mapping of python-oriented data containers to Spark ones will be imperfect, and only a small number of drivers are expected to be able to produce Spark objects. The standard arguments may b translated, unsupported or ignored, depending on the specific driver.

This method requires the package intake-spark

values()

Get an iterator over the sources for catalog entries.

walk(*sofar=None*, *prefix=None*, *depth=2*)

Get all entries in this catalog and sub-catalogs

Parameters

- **sofar** (*dict or None*) – Within recursion, use this dict for output
- **prefix** (*list of str or None*) – Names of levels already visited
- **depth** (*int*) – Number of levels to descend; needed to truncate circular references and for cleaner output

Returns

- *Dict where the keys are the entry names in dotted syntax, and the*
- *values are entry instances.*

yaml()

Return YAML representation of this data-source

The output may be roughly appropriate for inclusion in a YAML catalog. This is a best-effort implementation

intake-axds sensor source

class intake_axds.axds.**AXDSSensorSource**(**args*, ***kwargs*)

Bases: DataSource

Intake Source for AXDS sensor

Parameters

- **internal_id** (*Optional[int], optional*) – Internal station id for Axiom, by default None. Not the UUID. Need to input internal_id or UUID. If both are input, be sure they are for the same station.
- **uuid** (*Optional[str], optional*) – The UUID for the station, by default None. Not the internal_id. Need to input internal_id or UUID. If both are input, be sure they are for the same station. Note that there may also be a "datasetId" parameter which is sometimes but not always the same as the UUID.
- **start_time** (*Optional[str], optional*) – At what datetime for data to start, by default None. Must be interpretable by pandas Timestamp. If not input, the datetime at which the dataset starts will be used.
- **end_time** (*Optional[str], optional*) – At what datetime for data to end, by default None. Must be interpretable by pandas Timestamp. If not input, the datetime at which the dataset ends will be used.
- **qartod** (*bool, int, list, optional*) – Whether to return QARTOD agg flags when available, which is only for sensor_stations. Can instead input an int or a list of ints representing the _qa_agg flags for which to return data values. More information about QARTOD testing and flags can be found here: https://cdn.ioos.noaa.gov/media/2020/07/QARTOD-Data-Flags-Manual_version1.2final.pdf. Only used by datatype "sensor_station". Is not available if *binned==True*.

  Examples of ways to use this input are:

  - qartod=True: Return aggregate QARTOD flags as a column for each data variable.
  - qartod=False: Do not return any QARTOD flag columns.
  - qartod=1: nan any data values for which the aggregated QARTOD flags are not equal to 1.
  - qartod=[1,3]: nan any data values for which the aggregated QARTOD flags are not equal to 1 or 3.

  Flags are:

  - 1: Pass
  - 2: Not Evaluated
  - 3: Suspect
  - 4: Fail
  - 9: Missing Data
- **use_units** (*bool, optional*) – If True include units in column names. Syntax is "standard_name [units]". If False, no units. Then syntax for column names is "standard_name". This is currently specific to sensor_station only. Only used by datatype "sensor_station".
- **metadata** (*dict, optional*) – Metadata for catalog.

- **binned** (*bool, optional*) – True for binned data, False for raw, by default False. Only used by datatype "sensor_station".
- **bin_interval** (*Optional[str], optional*) – If `binned=True`, input the binning interval to return. Options are hourly, daily, weekly, monthly, yearly. If bin_interval is input, binned is set to True. Only used by datatype "sensor_station".
- **only_pgids** (*list, optional*) – If input, only return data associated with these parameterGroupIds. This is separate from parameterGroupLabels and parameterGroupIds that might be present in the metadata.

Raises

ValueError – _description_

Attributes

cache
cache_dirs
cat
classname
data_urls
Prepare to load in data by getting data_urls.

description
dtype
entry
gui
Source GUI, with parameter selection and plotting

has_been_persisted
hvplot
alias for `DataSource.plot`

is_persisted
plot
Plot API accessor

plots
List custom associated quick-plots

shape

Methods

`__call__`(**kwargs)	Create a new instance of this source with altered arguments
close()	Close open resources corresponding to this data source.
configure_new(**kwargs)	Create a new instance of this source with altered arguments
describe()	Description from the entry spec
discover()	Open resource and populate the source attributes.
export(path, **kwargs)	Save this data for sharing with other people
get(**kwargs)	Create a new instance of this source with altered arguments
get_filters()	Return appropriate filter for stationid.
persist([ttl])	Save data from this source to local persistent storage
read()	read data in
read_chunked()	Return iterator over container fragments of data source
read_partition(i)	Return a part of the data corresponding to i-th partition.
to_dask()	Return a dask container for this data source
to_spark()	Provide an equivalent data object in Apache Spark
yaml()	Return YAML representation of this data-source

get_persisted	
set_cache_dir	

close()

Close open resources corresponding to this data source.

configure_new(***kwargs*)

Create a new instance of this source with altered arguments

Enables the picking of options and re-evaluating templates from any user-parameters associated with this source, or overriding any of the init arguments.

Returns a new data source instance. The instance will be recreated from the original entry definition in a catalog **if** this source was originally created from a catalog.

property data_urls

Prepare to load in data by getting data_urls.

For V1 sources there will be a data_url per parameterGroupId but not for V2 sources.

describe()

Description from the entry spec

discover()

Open resource and populate the source attributes.

export(*path*, ***kwargs*)

Save this data for sharing with other people

Creates a copy of the data in a format appropriate for its container, in the location specified (which can be remote, e.g., s3).

Returns the resultant source object, so that you can, for instance, add it to a catalog (`catalog.add(source)`) or get its YAML representation (`.yaml()`).

`get`(***kwargs*)

Create a new instance of this source with altered arguments

Enables the picking of options and re-evaluating templates from any user-parameters associated with this source, or overriding any of the init arguments.

Returns a new data source instance. The instance will be recreated from the original entry definition in a catalog **if** this source was originally created from a catalog.

`get_filters`()

Return appropriate filter for stationid.

What filter form to use depends on if V1 or V2.

For V1, use each parameterGroupId only once to make a filter since all data of that type will be read in together.

Following Sensor API https://admin.axds.co/#!/sensors/api/overview

`property gui`

Source GUI, with parameter selection and plotting

`property has_been_persisted`

The base class does not interact with persistence

`property hvplot`

alias for `DataSource.plot`

`property is_persisted`

The base class does not interact with persistence

`persist`(*ttl=None*, ***kwargs*)

Save data from this source to local persistent storage

Parameters

- **ttl** (*numeric, optional*) – Time to live in seconds. If provided, the original source will be accessed and a new persisted version written transparently when more than `ttl` seconds have passed since the old persisted version was written.
- **kargs** (*passed to the _persist method on the base container.*) –

`property plot`

Plot API accessor

This property exposes both predefined plots (described in the source metadata) and general-purpose plotting via the hvPlot library. Supported containers are: array, dataframe and xarray,

To display in a notebook, be sure to run `intake.output_notebook()` first.

The set of plots defined for this source can be found by

```
>>> source.plots
["plot1", "plot2"]
```

and to display one of these:

```
>>> source.plot.plot1()
<holoviews/panel output>
```

To create new plot types and supply custom configuration, use one of the methods of `hvplot.hvPlot`:

```
>>> source.plot.line(x="fieldX", y="fieldY")
```

The full set of arguments that can be passed, and the types of plot they refer to, can be found in the doc and attributes of `hvplot.HoloViewsConverter`.

Once you have found a suitable plot, you may wish to update the plots definitions of the source. Simply add the `plotname=` optional argument (this will overwrite any existing plot of that name). The source's YAML representation will include the new plot, and it could be saved into a catalog with this new definition.

```
>>> source.plot.line(plotname="new", x="fieldX", y="fieldY");
>>> source.plots
["plot1", "plot2", "new"]
```

property plots

List custom associated quick-plots

read()

read data in

read_chunked()

Return iterator over container fragments of data source

read_partition(*i*)

Return a part of the data corresponding to i-th partition.

By default, assumes i should be an integer between zero and npartitions; override for more complex indexing schemes.

to_dask()

Return a dask container for this data source

to_spark()

Provide an equivalent data object in Apache Spark

The mapping of python-oriented data containers to Spark ones will be imperfect, and only a small number of drivers are expected to be able to produce Spark objects. The standard arguments may b translated, unsupported or ignored, depending on the specific driver.

This method requires the package intake-spark

yaml()

Return YAML representation of this data-source

The output may be roughly appropriate for inclusion in a YAML catalog. This is a best-effort implementation

PYTHON MODULE INDEX

i

INDEX

L

M

P

R

S

T

V

W

Y